Manual Testing

Contents

[Software Testing: 2](#_Toc533698309)

[Software Development Life Cycle 3](#_Toc533698310)

[Software Testing Life Cycle 5](#_Toc533698311)

[Bug Life Cycle or Defect Life Cycle 7](#_Toc533698312)

[Waterfall Model: 9](#_Toc533698313)

[Spiral Model: 10](#_Toc533698314)

[V Model: 11](#_Toc533698315)

[Manual Testing: 13](#_Toc533698316)

[Automation Testing: 14](#_Toc533698317)

[Test Case Template 15](#_Toc533698318)

[Verification and Validation: 17](#_Toc533698319)

[Test Scenario and Test Cases 17](#_Toc533698320)

[Regression and Retesting 18](#_Toc533698321)

[Smoke Testing 19](#_Toc533698322)

[Defect Severity and Priority in Software Testing 20](#_Toc533698323)

[Performance Testing 23](#_Toc533698324)

[Functional and Non-Functional Testing: 24](#_Toc533698325)

[Exploratory testing: 24](#_Toc533698326)

[Levels of testing: 24](#_Toc533698327)

[System Testing (End to End Testing) 25](#_Toc533698328)

[Black Box Testing: 26](#_Toc533698329)

[White Box Testing: 30](#_Toc533698330)

[Sample Test Strategy Document: 30](#_Toc533698331)

[Test plan 38](#_Toc533698332)

[Bug Report Template – Detailed Explanation 41](#_Toc533698333)

[PDCA Cycle (Plan Do Check Act) in Software Development Life Cycle 45](#_Toc533698334)

[Agile Methodology In Software Development 46](#_Toc533698335)

[Agile Scrum Methodology In Software Development 47](#_Toc533698337)

# Software 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 the quality product.

**Definition:**

According to **ANSI/IEEE 1059**standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

**Software Testing Types:**

**Manual Testing:**Manual testing is the process of testing the software manually to find the defects. Tester should have the perspective of an end users and to ensure all the features are working as mentioned in the requirement document. In this process, testers execute the test cases and generate the reports manually without using any automation tools.

**Automation Testing:**Automation testing is the process of testing the software using an automation tools to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. Some of the famous automation testing tools for functional testing are QTP/UFT and Selenium.

**Testing Methods:**

* Static Testing
* Dynamic Testing

**Testing Approaches:**

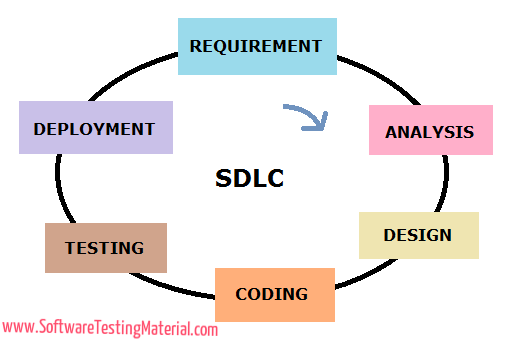
* White Box Testing
* Black Box Testing
* Grey Box Testing

**Testing Levels:**

* Unit Testing
* Integration Testing
* System Testing
* Acceptance Testing

# Software Development Life Cycle

**Software Development Life Cycle** (SDLC) aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.



**Detailed Explanation:**

A process followed in software projects is SDLC. Each phase of SDLC produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design. Testing should be done on a developed product based on requirement. Deployment should be done once the testing was completed. It aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

**Requirement Phase:**

Requirement gathering and analysis is the most important phase in software development lifecycle. Business Analyst collects the requirement from the Customer/Client as per the clients business needs and documents the requirements in the **Business Requirement Specification** (document name varies depends upon the Organization. Some examples are Customer Requirement Specification (CRS), Business Specification (BS) etc., and provides the same to Development Team.

**Analysis Phase:**

Once the requirement gathering and analysis is done the next step is to define and document the product requirements and get them approved by the customer. This is done through **SRS (Software Requirement Specification)** document. SRS consists of all the product requirements to be designed and developed during the project life cycle. Key people involved in this phase are Project Manager, Business Analysist and Senior members of the Team. The outcome of this phase is Software Requirement Specification.

**Design Phase:**

It has two steps:  
HLD – High Level Design – It gives the architecture of the software product to be developed and is done by architects and senior developers

LLD – Low Level Design – It is done by senior developers. It describes how each and every feature in the product should work and how every component should work. Here, only the design will be there and not the code

The outcome from this phase is **High Level Document and Low Level Document** which works as an input to the next phase

**Development Phase:**

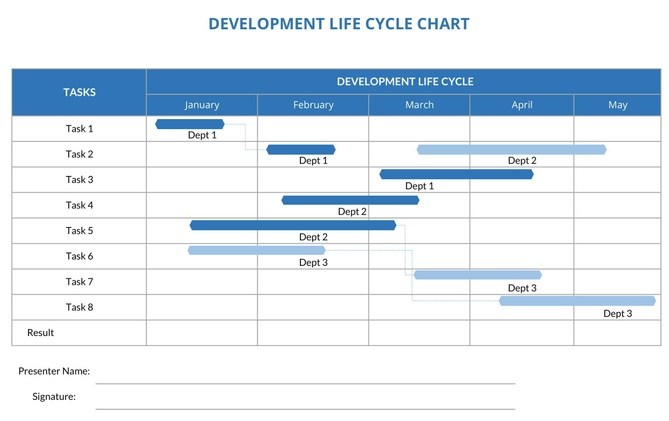
Developers of all levels (seniors, juniors, freshers) involved in this phase. This is the phase where we start building the software and start writing the code for the product. The outcome from this phase is **Source Code Document (SCD)** and the developed product.

**Testing Phase:**

When the software is ready, it is sent to the testing department where Test team tests it thoroughly for different defects. They either test the software manually or using automated testing tools depends on process defined in STLC (Software Testing Life Cycle) and ensure that each and every component of the software works fine. Once the QA makes sure that the software is error-free, it goes to the next stage, which is Implementation. The outcome of this phase is **the Quality Product and the Testing Artefacts.**

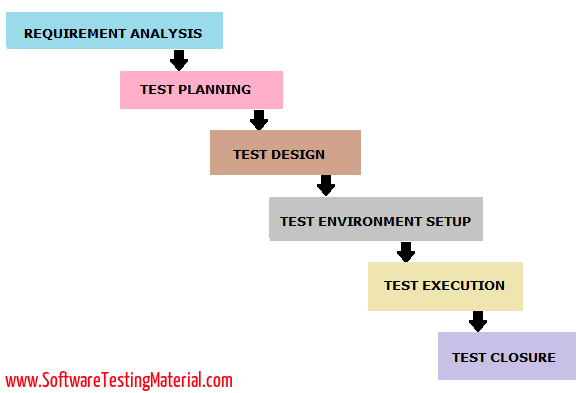
**Deployment & Maintenance Phase:**

After successful testing, the product is delivered/deployed to the customer for their use. Deployment is done by the Deployment/Implementation engineers. Once when the customers start using the developed system then the actual problems will come up and needs to be solved from time to time. Fixing the issues found by the customer comes in the maintenance phase. 100% testing is not possible – because, the way testers test the product is different from the way customers use the product. **Maintenance** should be done as per SLA (Service Level Agreement)



# Software Testing Life Cycle

Software Testing Life Cycle (**STLC**) identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between Organizations, there is a testing life cycle.



Every phase of STLC (Software Testing Life Cycle) has a definite Entry and Exit Criteria.

**Requirement Analysis:**

Entry criteria for this phase is BRS (Business Requirement Specification) document. During this phase, test team studies and analyzes the requirements from a testing perspective. This phase helps to identify whether the requirements are testable or not. If any requirement is not testable, test team can communicate with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) during this phase so that the mitigation strategy can be planned.

**Entry Criteria:** BRS (Business Requirement Specification)  
**Deliverables:** List of all testable requirements, Automation feasibility report (if applicable)

**Test Planning:**

Test planning is the first step of the testing process. In this phase typically Test Manager/Test Lead involves determining the effort and cost estimates for the entire project. Preparation of Test Plan will be done based on the requirement analysis. Activities like resource planning, determining roles and responsibilities, tool selection (if automation), training requirement etc., carried out in this phase. The deliverables of this phase are Test Plan & Effort estimation documents.

**Entry Criteria:** Requirements Documents  
**Deliverables:** Test Strategy, Test Plan, and Test Effort estimation document.

**Test Design:**

Test team starts with test cases development activity here in this phase. Test team prepares test cases, test scripts (if automation) and test data. Once the test cases are ready then these test cases are reviewed by peer members or team lead. Also, test team prepares the Requirement Traceability Matrix (RTM). RTM traces the requirements to the test cases that are needed to verify whether the requirements are fulfilled. The deliverables of this phase are Test Cases, Test Scripts, Test Data, Requirements Traceability Matrix

**Entry Criteria:** Requirements Documents (Updated version of unclear or missing requirement)  
**Deliverables:** Test cases, Test Scripts (if automation), Test data.

**Test Environment Setup:**

This phase can be started in parallel with Test design phase. Test environment setup is done based on the hardware and software requirement list. Some cases test team may not be involved in this phase. Development team or customer provides the test environment. Meanwhile, test team should prepare the smoke test cases to check the readiness of the given test environment.

**Entry Criteria:** Test Plan, Smoke Test cases, Test Data  
**Deliverables:** Test Environment. Smoke Test Results.

**Test Execution:**

Test team starts executing the test cases based on the planned test cases. If a test case result is Pass/Fail then the same should be updated in the test cases. Defect report should be prepared for failed test cases and should be reported to the Development Team through bug tracking tool (eg., Quality Center) for fixing the defects. Retesting will be performed once the defect was fixed. Click here to see the Bug Life Cycle.

**Entry Criteria:** Test Plan document, Test cases, Test data, Test Environment.  
**Deliverables:**Test case execution report, Defect report, RTM

**Test Closure:**

The final stage where we prepare Test Closure Report, Test Metrics.  
Testing team will be called out for a meeting to evaluate cycle completion criteria based on Test coverage, Quality, Time, Cost, Sm oftware, Business objectives. Test team analyses the test artifacts (such as Test cases, Defect reports etc.,) to identify strategies that have to be implemented in future, which will help to remove process bottlenecks in the upcoming projects. Test metrics and Test closure report will be prepared based on the above criteria.

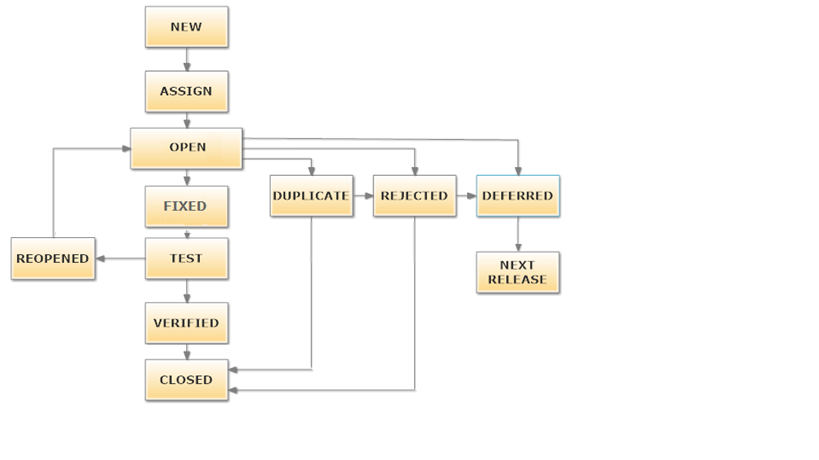
**Entry Criteria:** Test Case Execution report (make sure there are no high severity defects opened), Defect report  
**Deliverables:** Test Closure report, Test metrics

# Bug Life Cycle or Defect Life Cycle

**Bug life cycle** is also known as **Defect life cycle**. In Software Development process, the bug has a life cycle. The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, JIRA etc.,) used and the process followed in the organization.

**What is a Software Bug?**

Software bug can be defined as the abnormal behavior of the software. Bug starts when the defect is found and ends when a defect is closed, after ensuring it is not reproduced.



**The different states of a bug in the bug life cycle are as follows:**

**New:** When a tester finds a new defect. He should provide a proper Defect document to the Development team to reproduce and fix the defect. In this state, the status of the defect posted by tester is “New”

**Assigned:** Defects which are in the status of New will be approved (if valid) and assigned to the development team by Test Lead/Project Lead/Project Manager. Once the defect is assigned then the status of the bug changes to “Assigned”

**Open:** The development team starts analyzing and works on the defect fix

**Fixed:** When a developer makes the necessary code change and verifies the change, then the status of the bug will be changed as “Fixed” and the bug is passed to the testing team.

**Test:** If the status is “Test”, it means the defect is fixed and ready to do test whether it is fixed or not.

**Verified:** The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”

**Closed:** After verified the fix, if the bug is no longer exits then the status of bug will be assigned as “Closed.”

**Reopen:** If the defect remains same after the retest, then the tester posts the defect using defect retesting document and changes the status to “Reopen”. Again the bug goes through the life cycle to be fixed.

**Duplicate:** If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to “duplicate” by the development team.

**Deferred:**In some cases, Project Manager/Lead may set the bug status as deferred.  
If the bug found during end of release and the bug is minor or not important to fix immediately  
If the bug is not related to current build  
If it is expected to get fixed in the next release  
Customer is thinking to change the requirement  
In such cases the status will be changed as “deferred” and it will be fixed in the next release.

**Rejected:**If the system is working according to specifications and bug is just due to some misinterpretation (such as referring to old requirements or extra features) then Team lead or developers can mark such bugs as “Rejected”

**Cannot be fixed:** Technology not supporting, Root of the product issue, Cost of fixing bug is more

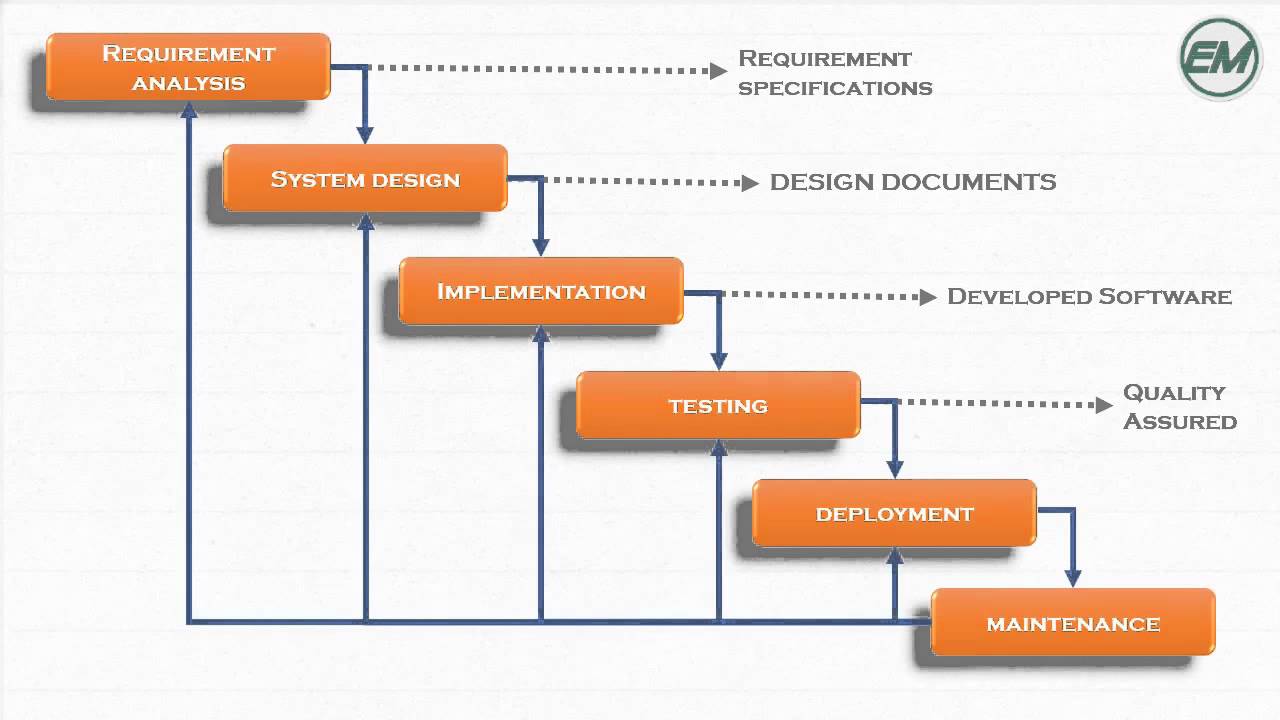
**Not Reproducible:** Platform mismatch, improper defect document, data mismatch, build mismatch, inconsistent defects

**Need more information:** If a developer is unable to reproduce the bug as per the steps provided by a tester then the developer can change the status as “Need more information’. In this case, the tester needs to add detailed reproducing steps and assign bug back to the development team for a fix. This won’t happen if the tester writes a good defect document.

This is all about Bug Life Cycle / Defect Life Cycle. Some companies use these bug id’s in  RTM to map with the test cases.

# Waterfall Model:

Waterfall Model is a traditional model. It is aka Sequential Design Process, often used in SDLC, in which the progress is seen as flowing downwards like a waterfall, through the different phases such as Requirement Gathering, Feasibility Study/Analysis, Design, Coding, Testing, Installation and Maintenance. Every next phase is begun only once the goal of previous phase is completed. This methodology is preferred in projects where quality is more important as compared to schedule or cost. This methodology is best suitable for short term projects where the requirements will not change. (E.g. Calculator, Attendance Management)



**Advantages:**

* Requirements do not change nor does design and code, so we get a stable product.
* This model is simple to implement. Requirements are finalized earlier in the life cycle. So there won’t be any chaos in the next phases.
* Required resources to implement this model are minimal compared to other methodologies
* Every phase has specific deliverable’s. It gives high visibility to the project manager and clients about the progress of the project.

**Disadvantages:**

* Backtracking is not possible i.e., we cannot go back and change requirements once the design stage is reached.
* Change in requirements leads to change in design and code which results defects in the project due to overlapping of phases.
* Customer may not be satisfied, if the changes they required are not incorporated in the product.
* The end result of waterfall model may not be a flexible product
* In terms of testing, testers involve only in the testing phase. Requirements are not tested in the requirement phase. It can’t be modified even though we identify that there is a bug in the requirement in the testing phase. It goes on till the end and leads to lot of re-work.
* It is not suitable for long term projects where requirements may change time to time
* Waterfall model can be used only when the requirements are very well known and fixed

Testing is not just finding bugs. As per the Waterfall Model, Testers involve only almost at the end of the SDLC. Ages ago the mantra of testing is just to finding bugs in the software. Things changed a lot now. There are some other SDLC models implemented. I would post other models in the upcoming posts in detail with their advantages and disadvantages. It is up to your team to choose the SDLC model depends on the project you are working.

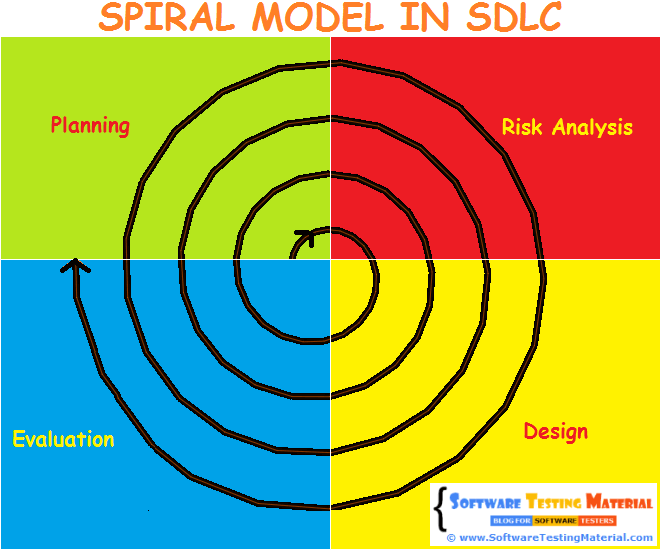
# Spiral Model:

Spiral Model was first described by Barry W. Boehm(American Software Engineer) in 1986.

Spiral model works in an iterative nature. It is a combination of both Prototype development process and Linear development process (waterfall model). This model place more emphasis on risk analysis. Mostly this model adpots to the large and complicated projects where risk is high. Every Iteration starts with a planning and ends with the product evaluation by client.

Let’s take an example of a product development team (like Microsoft). They know that there will be a high risk and they face lots of difficulties in the journey of developing and releasing the product

and also they know that they will release next version of product when the current version is in existence. They prefer Spiral Model to develop the product in an iterative nature. They could release one version of the product to the end user and start developing next version which includes new enhancements and improvements on previous version (based on the issues faced by the user in the previous version). Like Microsoft released Windows 8 and improved it based on user feedback and released the next version (Windows 8.1).



Spiral Model undergoes 4 phases.

**Planning Phase** – Requirement Gathering, Cost Estimation, Resource Allocation  
**Risk Analysis Phase** – Strengths and weaknesses of the project  
**Design Phase** – Coding, Internal Testing and deployment  
**Evaluation Phase** – Client Evaluation (Client side Testing) to get the feedback

**Advantages:**

* It allows requirement changes
* Suitable for large and complicated projects
* It allows better risk analysis
* Cost effective due to good risk management

**Disadvantages:**

* Not suitable for small projects
* Success of the project depends on risk analysis phase
* Have to hire more experienced resource especially for risk analysis

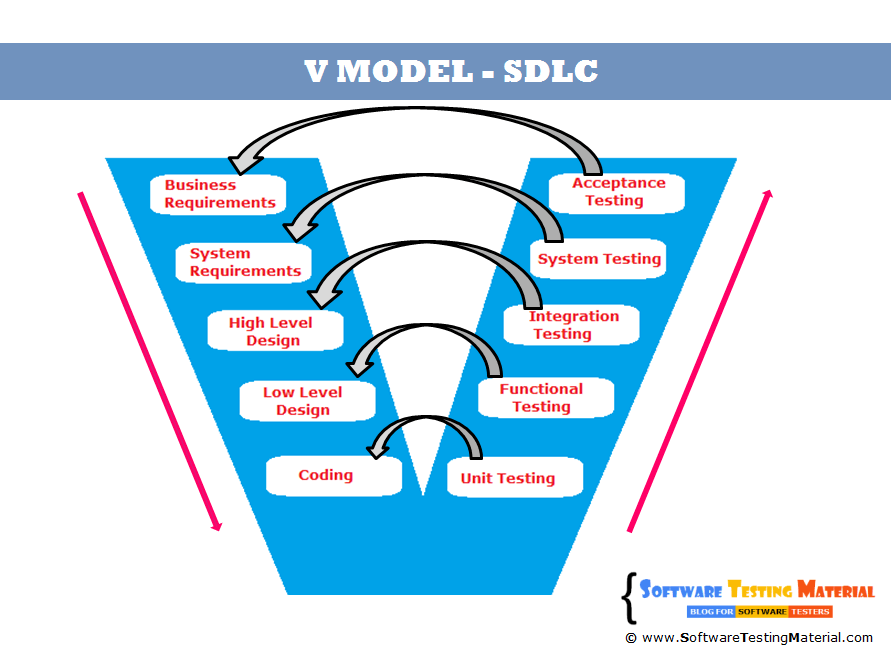
# V Model:

V-model is also known as **Verification and Validation (V&V) model**. In this each phase of SDLC must be completed before the next phase starts. It follows a sequential design process same like waterfall model.

It overcomes the disadvantages of waterfall model. In the waterfall model, we have seen that testers involve in the project only at the last phase of the development process.

In this, test team involves in the early phases of SDLC. Testing starts in early stages of product development which avoids downward flow of defects which in turns reduces lot of rework. Both teams (test and development) work in parallel. The test team works on various activities like preparing test strategy, test plan and test cases/scripts while the development team works on SRS, Design and Coding.

Once the requirements were received, both the development and test team start their activities.



Deliverables are parallel in this model. Whilst, developers are working on SRS (System Requirement Specification), testers work on BRS (Business Requirement Specification) and prepare ATP(Acceptance Test Plan) and ATC (Acceptance Test Cases) and so on.

Testers will be ready with all the required artifacts (such as Test Plan, Test Cases)  by the time developers release the finished product. It saves lots of time.

Let’s see the how the development team and test team involves in each phase of SDLC in V Model.

1. Once client sends BRS, both the teams (test and development) start their activities. The developers translate the BRS to SRS. The test team involves in reviewing the BRS to find the missing or wrong requirements and writes acceptance test plan and acceptance test cases.

2. In the next stage, the development team sends the SRS the testing team for review and the developers start building the HLD (High Level Design Document) of the product. The test team involves in reviewing the SRS against the BRS and writes system test plan and test cases.

3. In the next stage, the development team starts building the LLD (Low Level Design) of the product. The test team involves in reviewing the HLD (High Level Design) and writes Integration test plan and integration test cases.

4. In the next stage, the development team starts with the coding of the product. The test team involves in reviewing the LLD and writes functional test plan and functional test cases.

5. In the next stage, the development team releases the build to the test team once the unit testing was done. The test team carries out functional testing, integration testing, system testing and acceptance testing on the release build step by step.

**Advantages:**

* Testing starts in early stages of product development which avoids downward flow of defects and helps to find the defects in the early stages
* Test team will be ready with the test cases by the time developers release the software which in turns saves a lot of time
* Testing is involved in every stage of product development. It gives a quality product.
* Total investment is less due to less or no rework.

**Disadvantages:**

* Initial investment is more because test team involves right from the early stage.
* Whenever there is change in requirement, the same procedure continues. It leads more documentation work.

**Applications:**

Long term projects, complex applications, When customer is expecting a very high quality product with in stipulated time frame because every stage is tested and developers & testers are working in parallel

# Manual Testing:

Manual testing is the process of testing the software manually to find the defects. Testers should have the perspective of an end user and to ensure all the features are working as mentioned in the requirement document. In this process, testers execute the test cases and generate the reports manually without using any automation tools.

**Types of Manual Testing:**

1. Black Box Testing
2. White Box Testing
3. Unit Testing
4. System Testing
5. Integration Testing
6. Acceptance Testing

**Black Box Testing:** Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

**White Box Testing:** White Box Testing is also called as Glass Box, Clear Box, and Structural Testing. It is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually done at the unit level.

**Unit Testing:** Unit Testing is also called as Module Testing or Component Testing. It is done to check whether the individual unit or module of the source code is working properly. It is done by the developers in developer’s environment.

**Integration Testing:**Integration Testing is the process of testing the interface between the two software units. Integration testing is done by three ways. Big Bang Approach, Top Down Approach, Bottom-Up Approach

**Acceptance Testing:**It is also known as pre-production testing.  This is done by the end users along with the testers to validate the functionality of the application. After successful acceptance testing. Formal testing conducted to determine whether an application is developed as per the requirement. It allows customer to accept or reject the application. Types of acceptance testing are Alpha, Beta & Gamma.

Advantages:

1. Manual testing can be done on all kinds of applications
2. It is preferable for short life cycle products
3. Newly designed test cases should be executed manually
4. Application must be tested manually before it is automated
5. It is preferred in the projects where the requirements change frequently and for the products where the GUI changes constantly
6. It is cheaper in terms of initial investment compared to Automation testing
7. It requires less time and expense to begin productive manual testing
8. It allows tester to perform adhoc testing
9. There is no necessity to the tester to have knowledge on Automation Tools

Disadvantages:

1. Manual Testing is time-consuming mainly while doing regression testing.
2. Expensive over automation testing in the long run

# Automation Testing:

Automation testing is the process of testing the software using an automation tools to find the defects. In this process, executing the test scripts and generating the results are performed automatically by automation tools. Some most popular tools to do automation testing are HP QTP/UFT, [Selenium WebDriver](https://www.softwaretestingmaterial.com/install-selenium-webdriver/), etc.,

Advantages:

1. Automation testing is faster in execution
2. It is cheaper compared to manual testing in a long run
3. Automated testing is more reliable
4. Automated testing is more powerful and versatile
5. It is mostly used for regression testing
6. It does not require human intervention. Test scripts can be run unattended
7. It helps to increase the test coverage

Disadvantages:

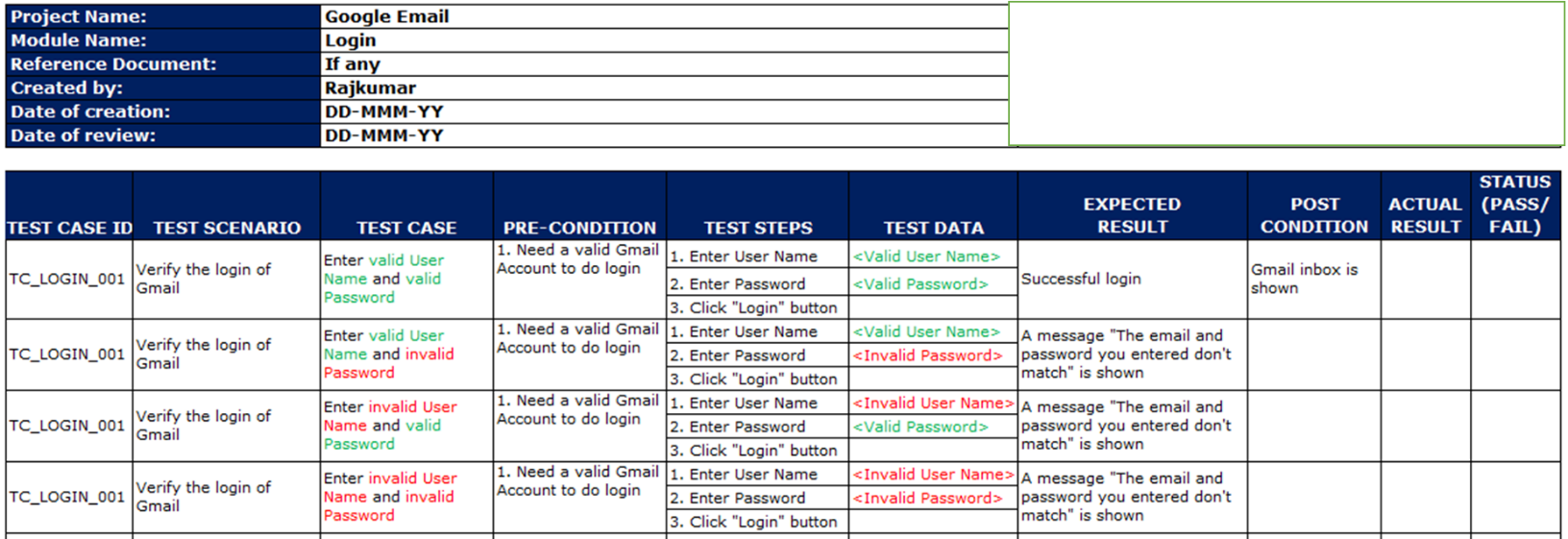
1. It is recommended only for stable products
2. Automation testing is expensive initially
3. Most of the automation tools are expensive
4. It has some limitations such as handling captcha, fonts, color
5. Huge maintenance in case of repeated changes in the requirements
6. Not all the tools support all kinds of testing. Such as windows, web, mobility, performance/load testing

# Test Case Template

A **test case template** is a document comes under one of the [test artifacts](https://www.softwaretestingmaterial.com/test-deliverables/), which allows testers to develop the test cases for a particular test scenario in order to verify whether the features of an application are working as intended or not. Test cases are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected result, post-conditions and actual results.

Ex: For login app

 1. Enter valid User Name and valid Password  
2. Enter valid User Name and invalid Password  
3. Enter invalid User Name and valid Password  
4. Enter invalid User Name and invalid Password



**PROJECT NAME:** Name of the project the test cases belong to  
**MODULE NAME:** Name of the module the test cases belong to  
**REFERENCE DOCUMENT:**Mention the path of the reference documents (if any such as Requirement Document, [Test Plan](https://www.softwaretestingmaterial.com/test-plan/), Test Scenarios etc.,)  
**CREATED BY:** Name of the Tester who created the test cases  
**DATE OF CREATION:** When the test cases were created  
**REVIEWED BY:**Name of the Tester who created the test cases  
**DATE OF REVIEW:**When the test cases were reviewed  
**EXECUTED BY:** Name of the Tester who executed the test case  
**DATE OF EXECUTION:** When the test case was executed  
**TEST CASE ID:** Each test case should be represented by a unique ID. It’s good practice to follow some naming convention for better understanding and discrimination purpose.  
**TEST SCENARIO:** Test Scenario ID or title of the test scenario.  
**TEST CASE:** Title of the test case  
**PRE-CONDITION:** Conditions which needs to meet before executing the test case.  
**TEST STEPS:** Mention all the test steps in detail and in the order how it could be executed.  
**TEST DATA:** The data which could be used an input for the test cases.  
**EXPECTED RESULT:** The result which we expect once the test cases were executed. It might be anything such as Home Page, Relevant screen, Error message etc.,  
**POST-CONDITION:** Conditions which needs to achieve when the test case was successfully executed.  
**ACTUAL RESULT:** The result which system shows once the test case was executed.  
**STATUS:** If the actual and expected results are same, mention it as Passed. Else make it as Failed. If a test fails, it has to go through the [bug life cycle](https://www.softwaretestingmaterial.com/bug-life-cycle/) to be fixed.

# Verification and Validation:

In software testing, verification and validation are the processes to check whether a software system meets the specifications and that it fulfills its intended purpose or not. Verification and validation is also known as V & V. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the Software Development Life Cycle.

**VERIFICATION: (Static Testing)**

Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.

Activities involved here are Inspections, Reviews, Walkthroughs

**VALIDATION: (Dynamic Testing)**

Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

Activities involved in this is Testing the software application

In simple words, Validation is to validate the actual and expected output of the software.

# Test Scenario and Test Cases

**Test Scenario** gives the idea of what we have to test. Test Scenario is like a high-level test case.

Test Scenario answers “**What to be tested**”

Assume that we need to test the functionality of a login page of Gmail application. Test scenario for the Gmail login page functionality as follows:

**Test Scenario Example:** Verify the login functionality

**Test cases** are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected result, post-conditions and actual results.

Test Case answers “**How to be tested**”

Assume that we need to test the functionality of a login page of Gmail application. Test cases for the above login page functionality as follows:

**Test Case Examples:**

**Test Case 1:** Enter valid User Name and valid Password  
**Test Case 2:** Enter valid User Name and invalid Password  
**Test Case 3:** Enter invalid User Name and valid Password  
**Test Case 4:** Enter invalid User Name and invalid Password

|  |  |
| --- | --- |
| **TEST CASE** | **TEST SCENARIO** |
| Test case consists of Test case name, Precondition, Test steps, Expected result and Post condition | Test scenario are one liner but it is associated with multiple test cases |
| Test case guide a user on “how to test” | Test scenario guide a user on “what to test” |
| Purpose of test case is to validate the test scenario by executing a set of steps | Purpose of test scenario is to test end to end functionality of a software application |
| Creating test cases is important when working with testers off-site | Creating test scenarios helps you in a time-sensitive situation (especially working in Agile) |
| Software applications change often. It leads to redesigning the pages and adding new functionalities. It hard to maintain test cases | Test scenarios are easy to maintain due to its high level design |
| More time consumption compared to test scenarios | Less time consumption compared to test cases |
| Required more resources to create and execute test cases | Relatively less resources enough to create and test using test scenarios |
| It helps in exhaustive testing of application | It helps in agile way of testing end to end functionality |
| Test cases are derived from test scenarios | Test scenarios are derived from use cases |
| Test cases are low level actions | Test scenarios are high level actions |

# Regression and Retesting

**REGRESSION TESTING:**

Repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes in the software being tested or in another related or unrelated software components.

Usually, we do regression testing in the following cases:

* New functionalities are added to the application
* Change Requirement (In organizations, we call it as CR)
* Defect Fixing
* Performance Issue Fix
* Environment change (E.g.. Updating the DB from MySQL to Oracle)

**RETESTING:**

To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build.

Say, Build 1.0 was released. Test team found some defects (Defect Id 1.0.1, 1.0.2) and posted.

Build 1.1 was released, now testing the defects 1.0.1 and 1.0.2 in this build is retesting.

# Smoke Testing

Smoke Testing is done to make sure if the build we received from the development team is testable or not. It is also called as “Day 0” check. It is done at the “build level”.

It helps not to waste the testing time to simply testing the whole application when the key features don’t work or the key bugs have not been fixed yet. Here our focus will be on primary and core application work flow.

**How to Conduct Smoke Testing?**

To conduct smoke testing, we don’t write test cases. We just pick the necessary test cases from already written test cases.

Do we really write test cases for all testing types? Here in this article, we have given clear idea on choosing testing types to write test cases.

As mentioned earlier, here in Smoke Testing, our main focus will be on core application work flow. So we pick the test cases from our test suite which cover major functionality of the application. In general, we pick minimal number of test cases that won’t take more than half an hour to execute.

**Sanity Testing**

Sanity Testing is done during the release phase to check for the main functionalities of the application without going deeper. It is also called as a subset of Regression testing. It is done at the “release level”.

At times due to release time constraints rigorous [regression testing](https://www.softwaretestingmaterial.com/difference-between-regression-and-retesting/) can’t be done to the build, sanity testing does that part by checking main functionalities.

**How to Conduct Sanity Testing?**

Same like Smoke testing, we don’t write separate test cases for Sanity testing. We just pick the necessary test cases from already written test cases.

As mentioned earlier, it is a subset of regression testing. When it comes to Sanity testing, the main focus is to make sure whether the planned functionality is working as expected.

|  |  |
| --- | --- |
| Smoke Test is done to make sure if the build we received from the development team is testable or not | Sanity Test is done during the release phase to check for the main functionalities of the application without going deeper |
| Smoke Testing is performed by both Developers and Testers | Sanity Testing is performed by Testers alone |
| Smoke Testing exercises the entire application from end to end | Sanity Testing exercises only the particular component of the entire application |
| Smoke Testing, build may be either stable or unstable | Sanity Testing, build is relatively stable |

# Defect Severity and Priority in Software Testing

Bug/Defect severity can be defined as the impact of the bug on the application. It can be Critical, Major or Minor. In simple words, how much effect will be there on the system because of a particular defect

**What are the types of Severity?**

Severity can be categorized into three types:

As mentioned above the type of severity are categorized as Critical, Major, and Minor

Let’s see how can we segregate a bug into these types:

**Critical:**

A critical severity issue is an issue where a large piece of functionality or major system component is completely broken and there is no workaround to move further.  
For example, Due to a bug in one module, we cannot test the other modules because that blocker bug has blocked the other modules. Bugs which affects the customers business are considered as critical

**Major:**

A major severity issue is an issue where a large piece of functionality or major system component is completely broken and there is a workaround to move further.

**Minor:**

A minor severity issue is an issue that imposes some loss of functionality, but for which there is an acceptable & easily reproducible workaround.

For example, font family or font size or color or spelling issue

**Trivial:**

A trivial severity defect is a defect which is related to enhancement of the system

**What is Priority?**

Defect priority can be defined as impact of the bug on the customers business. Main focus on how soon the defect should be fixed. It gives the order in which a defect should be resolved. Developers decide which defect they should take up next based on the priority. It can be High, Medium or Low.

Most of the times the priority status is set based on the customer requirement.

**What are the types of Priority?**

Priority can be categorized into three types:

As mentioned above the type of severity are categorized as High, Medium, and Low

**High:**

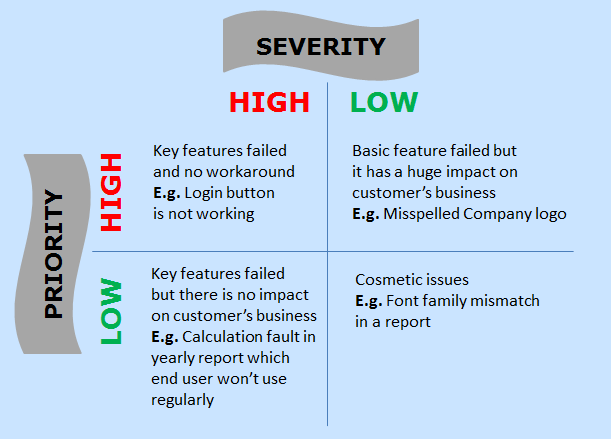
A high priority issue is an issue which has a high impact on the customers business or an issue which affects the system severely and the system cannot be used until the issue was fixed. These kinds of issues must be fixed immediately. Most of the cases as per the user perspective, the priority of the issue is set to high priority even though the severity of the issue is minor.

**Medium:**

Issues which can be released in the next build comes under medium priority. Such issues can be resolved along with other development activities.

**Low:**

An issue which has no impact on the customer business comes under low priority.



**High Priority & High Severity:**

A critical issue where a large piece of functionality or major system component is completely broken.

**Low Priority & High Severity:**

An issue which won’t affects customers business but it has a big impact in terms of functionality.

**High Priority & Low Severity:**

A minor issue that imposes some loss of functionality, but for which there is an acceptable & easily reproducible workaround. Testing can proceed without interruption but it affects customers reputation.

**Low Priority & Low Severity:**

A minor issue that imposes some loss of functionality, but for which there is an acceptable & easily reproducible workaround. Testing can proceed without interruption.

**What is a defect?**

The variation between the actual results and expected results is known as defect.

If a developer finds an issue and corrects it by himself in the development phase then it’s called a defect.

**What is a bug?**

If testers find any mismatch in the application/system in testing phase then they call it as Bug.

As I mentioned earlier, there is a contradiction in the usage of Bug and Defect. People widely say the bug is an informal name for the defect.

**What is an error?**

We can’t compile or run a program due to coding mistake in a program. If a developer unable to successfully compile or run a program then they call it as an **error**.

**What is a failure?**

Once the product is deployed and customers find any issues then they call the product as a failure product. After release, if an end user finds an issue then that particular issue is called as **failure**

# Performance Testing

Performance testing and types of performance testing such as Load Testing, Volume Testing, Stress Testing, Capacity Testing, Soak/Endurance Testing and Spike Testing come under Non-functional Testing

Performance testing determines or validates the speed, scalability, and/or stability characteristics of the system or application under test. Performance is concerned with achieving response times, throughput, and resource-utilization levels that meet the performance objectives for the project or product.

Capacity Testing:

Capacity Testing is to determine how many users the system/application can handle successfully before the performance goals become unacceptable. This allows us to avoid the potential problems in future such as increased user base or increased volume of data.

Load Testing:

Load Testing is to verify that the system/application can handle the expected number of transactions and to verify the system/application behaviour under both normal and peak load conditions (no. of users).

Volume Testing:

Volume Testing is to verify that the system/application can handle a large amount of data. This testing focuses on Data Base.

Stress Testing:

Stress Testing is to verify the behaviour of the system once the load increases more than the system’s design expectations. This testing addresses which components fail first when we stress the system by applying the load beyond the design expectations. So that we can design more robust system.

Soak Testing:

Soak Testing is aka Endurance Testing. Running a system at high load for a prolonged period of time to identify the performance problems is called Soak Testing. It is to make sure the software can handle the expected load over a long period of time.

Spike Testing:

Spike Testing is to determine the behaviour of the system under sudden increase of load (a large number of users) on the system.

# Functional and Non-Functional Testing:

Functional testing: In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black box testing and the testers need not concern about the source code of the application.

Non-functional testing: In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility etc., Main focus is to improve the user experience on how fast the system responds to a request.

# **Exploratory testing**:

Usually, this process will be carried out by domain experts. They perform testing just by exploring the functionalities of the application without having the knowledge of the requirements.

# Levels of testing:

**UNIT TESTING:**

Unit Testing is done to check whether the individual modules of the source code are working properly. i.e. testing each and every unit of the application separately by the developer in developer’s environment. It is AKA Module Testing or Component Testing

**INTEGRATION TESTING:**

Integration Testing is the process of testing the connectivity or data transfer between the couple of unit tested modules. It is AKA I&T Testing or String Testing

It is sub divided into Top Down Approach, Bottom Up Approach and Sandwich Approach (Combination of Top Down and Bottom Up). This process is carried out by using dummy programs called Stubs and Drivers. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

**Big Bang Integration Testing:**

In Big Bang Integration Testing, the individual modules are not integrated until all the modules are ready. Then they will run to check whether it is performing well. In this type of testing, some disadvantages might occur like, defects can be found at the later stage. It would be difficult to find out whether the defect arouse in interface or in module.

**Top Down Integration Testing**

In Top Down Integration Testing, the high level modules are integrated and tested first. i.e Testing from main module to sub module. In this type of testing, Stubs are used as temporary module if a module is not ready for integration testing.

**Bottom Up Integration Testing**

In Bottom Up Integration Testing, the low-level modules are integrated and tested first i.e Testing from sub-module to main module. Same like Stubs, here drivers are used as a temporary module for integration testing.

**Stub:**

It is called by the Module under Test.

**Driver:**

It calls the Module to be tested.

# System Testing (End to End Testing)

It’s a black box testing. Testing the fully integrated application this is also called as end to end scenario testing. To ensure that the software works in all intended target systems. Verify thorough testing of every input in the application to check for desired outputs. Testing of the users experience with the application.

**ACCEPTANCE TESTING:**

To obtain customer sign-off so that software can be delivered and payments received.

Types of Acceptance Testing are Alpha, Beta & Gamma Testing.

**Alpha Testing:**

Alpha testing is mostly like performing usability testing which is done by the in-house developers who developed the software. Sometimes this alpha testing is done by client or outsiders with the presence of developers or testers.

**Beta Testing:**

Beta testing is done by limited number of end users before delivery, the change request would be fixed if the user gives feedback or reports defect.

**Gamma Testing:**

Gamma testing is done when the software is ready for release with specified requirements; this testing is done directly by skipping all the in-house testing activities.

# Black Box Testing:

**Black box testing:** Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

**Glass box testing** – Refer white box testing

**White box testing:** White Box Testing is also called as Glass Box, Clear Box, and Structural Testing. It is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually was done at the unit level.

**Gray box testing:** Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

It is also called as Behavioral/Specification-Based/Input-Output Testing

Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

Testers create test scenarios/cases based on software requirements and specifications. So it is AKA Specification Based Testing.

Tester performs testing only on the functional part of an application to make sure the behavior of the software is as expected. So it is AKA Behavioral Based Testing.

The tester passes input data to make sure whether the actual output matches the expected output. So it is AKA Input-Output Testing.

Black Box Testing Techniques:

* Equivalence Partitioning
* Boundary Value Analysis
* Decision Table
* State Transition

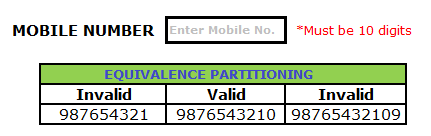
**Equivalence Partitioning:** Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases.

Each and every condition of particular partition (group) works as same as other. If a condition in a partition is valid, other conditions are valid too. If a condition in a partition is invalid, other conditions are invalid too.

It helps to reduce the total number of test cases from infinite to finite. The selected test cases from these groups ensure coverage of all possible scenarios.

Equivalence partitioning is applicable at all levels of testing.

Example on Equivalence Partitioning Test Case Design Technique:



Assume, we have to test a filed which accepts a Mobile Number of ten digits.

Valid input: 10 digits

Invalid Input: 9 digits, 11 digits

Valid Class: Enter 10 digit mobile number = 9876543210

Invalid Class Enter mobile number which has less than 10 digits = 987654321

Invalid Class Enter mobile number which has more than 11 digits = 98765432109

**Boundary Value Analysis:** Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.

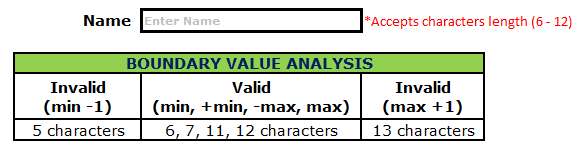
Tests can be designed to cover both valid and invalid boundary values. When designing test cases, a test for each boundary value is chosen.

For each boundary, we test +/-1 in the least significant digit of either side of the boundary.

Boundary value analysis can be applied at all test levels.

**Example on Boundary Value Analysis Test Case Design Technique:**

Assume we have to test a text field (Name) which accepts the length between 6-12 characters.



Minimum boundary value is 6

Maximum boundary value is 12

Valid text length is 6, 7, 11, 12

Invalid text length is 5, 13

Test case 1: Text length of 5 (min-1) = Invalid

Test case 2: Text length of exactly 6 (min) = Valid

Test case 3: Text length of 7 (min+1) = Valid

Test case 4: Text length of 11 (max-1) = Valid

Test case 5: Text length of exactly 12 (max) = Valid

Test case 6: Text length of 13 (max+1) = Invalid

**Decision Table:** Decision Table is aka Cause-Effect Table. This test technique is appropriate for functionalities which has logical relationships between inputs (if-else logic). In Decision table technique, we deal with combinations of inputs. To identify the test cases with decision table, we consider conditions and actions. We take conditions as inputs and actions as outputs.

**Examples on Decision Table Test Case Design Technique:**

login page validation. Allow user to login only when both the ‘User ID’ and ‘Password’ are entered correct.

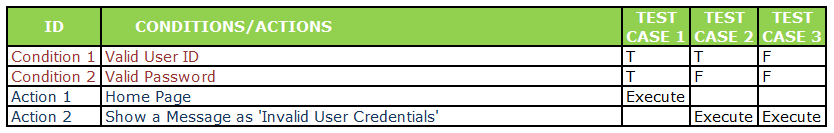
Here the Conditions to allow user to login are Enter Valid User Name and Enter Valid Password.

The Actions performed are Displaying home page and Displaying an error message that User ID or Password is wrong.



From the case 2 and case 3, we could identify that if one of the condition failed then the system will display an error message as Invalid User Credentials.

So I am eliminating one of the test case from case 2 and case 3 and concluding with the below tabular column.

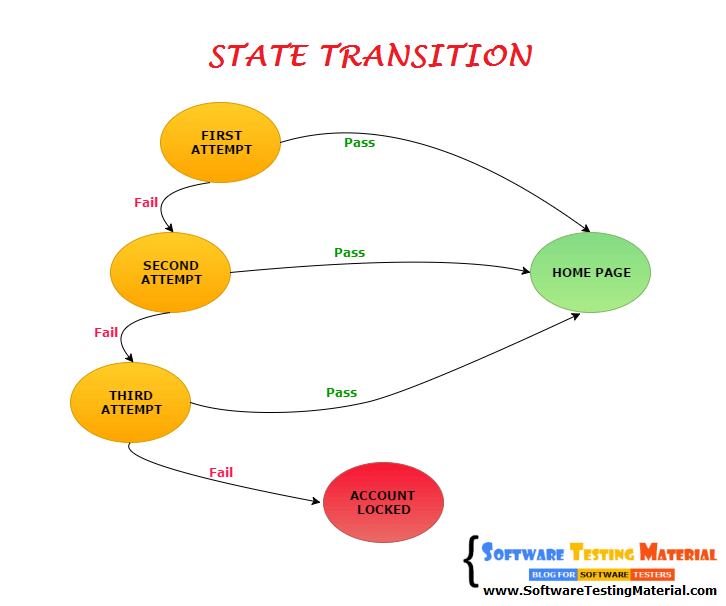


**State Transition:** Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state.

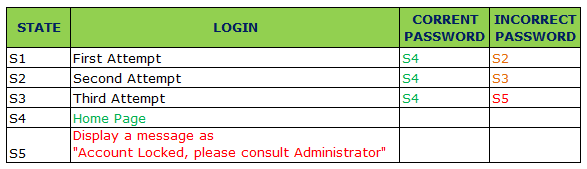
**Example on State Transition Test Case Design Technique:**

Take an example of login page of an application which locks the user name after three wrong attempts of password.

A finite state system is often shown as a state diagram



It works like a truth table. First determine the states, input data and output data.



Entering correct password in the first attempt or second attempt or third attempt, user will be redirected to the home page (i.e., State – S4).

Entering incorrect correct password in the first attempt, a message will be displayed as try again and user will be redirected to state S2 for the second attempt.

Entering incorrect correct password in the second attempt, a message will be displayed as try again and user will be redirected to state S3 for the third attempt.

Entering incorrect correct password in the third attempt, user will be redirected to state S5 and a message will be displayed as “Account locked. Consult Administrator”.

Types of Black Box Testing:

Functionality Testing: In simple words, what the system actually does is functional testing

Non-functionality Testing: In simple words, how well the system performs is non-functionality testing

# White Box Testing:

It is also called as Glass Box, Clear Box, Structural Testing.

White Box Testing is based on applications internal code structure. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually done at the unit level.

White Box Testing Techniques:

* Statement Coverage
* Branch Coverage
* Path Coverage

# Sample Test Strategy Document:

Test Strategy is a high level document (static document) and usually developed by project manager. It is a document which captures the approach on how we go about testing the product and achieve the goals. It is normally derived from the Business Requirement Specification (BRS). Documents like Test Plan are prepared by keeping this document as base.

Even though testing differs between organizations. Almost all the software development organizations follow Test Strategy document to achieve the goals and to follow the best practice.

Usually test team starts writing the detailed Test Plan and continue further phases of testing once the test strategy is ready. In Agile world, some of the companies are not spending time on test plan preparation due to the minimal time for each release but they maintain test strategy document. Maintaining this document for the entire project leads to mitigate the unforeseen risks.

This is one of the important documents in test deliverables. Like other test deliverables, test team shares this with the stakeholders for better understanding about the scope of the project, test approaches and other important aspects.

If you are a beginner, you may not get an opportunity to create a test strategy document but it’s good to know how to create test strategy document. It will be helpful when you are handling a QA Team. Once you become a Project Lead or Project manager you have to develop test strategy document. Creating an effective test strategy document is a skill which you must acquire. By writing a test strategy plan you can define the testing approach of your project. Test strategy document should be circulated to all the team members so that every team member will be consistent with the testing approach. Remember there is no rule to maintain all these sections in your Test Strategy document. It varies from company to company. This list gives a fair idea on how to write a good Test Strategy.

Sections of Test Strategy Document:

Following are the sections of test strategy document:

* Scope and overview
* Test Approach
* Testing tools
* Industry standards to follow
* Test deliverables
* Testing metrics
* Requirement Traceability Matrix
* Risk and mitigation
* Reporting tool
* Test summary

We have seen what is test strategy document and what it contains. Let’s discuss each section of Test Strategy in STLC briefly.

**Scope and overview:**

In this section, we will mention the scope of testing activities (what to test and why to test) and mention an overview of the AUT.

Example: Creating a new Application (Say Google Mail) which offers email services. Test the functionality of emails and make sure it gives value to the customer.

**Test Approach:**

In this section, we usually define the following

* Test levels
* Test types
* Roles and responsibilities
* Environment requirements

**Test Levels:**

This section lists out the levels of testing that will be performed during QA Testing. Levels of testing such as unit testing, integration testing, system testing and user acceptance testing. Testers are responsible for integration testing, system testing and user acceptance testing.

**Test Types:**

This section lists out the testing types that will be performed during QA Testing.

**Roles and responsibilities:**

This section describes the roles and responsibilities of Project Manager, Project Lead, individual testers.

**Environment requirements:**

This section lists out the hardware and software for the test environment in order to commence the testing activities.

**Testing tools:**

This section will describe the testing tools necessary to conduct the tests

Example: Name of Test Management Tool, Name of Bug Tracking Tool, Name of Automation Tool

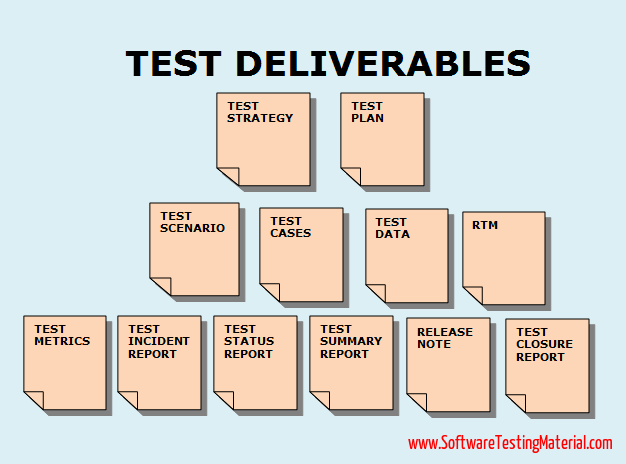
**Industry standards to follow:**

This section describes the industry standard to produce high quality system that meets or exceeds customer expectations. Usually, project manager decides the testing models and procedures which need to follow to achieve the goals of the project.

**Test deliverables:**

This section lists out the deliverables that need to produce before, during and at the end of testing.

Test Deliverables are the test artifacts which are given to the stakeholders of a software project during the SDLC (Software Development Life Cycle). A software project which follows SDLC undergoes the different phases before delivering to the customer. In this process there will be some deliverables in every phase. Some of the deliverables are provided before the testing phase commences and some are provided during the testing phase and rest after the testing phase is completed.



Every software application goes through different phases of SDLC and STLC. In the process of software application development, test teams prepare different documents to improve the communication among the team members and other stakeholders. These documents are also known as Test Deliverables, as they are delivered to client along with the final product of software application.

The following are list of test deliverables:

The test deliverables prepared during the process of software testing are as follows

1. **Test Strategy:** Test Strategy is a high-level document (static document) and usually developed by project manager. It is a document which captures the approach on how we go about testing the product and achieve the goals. It is normally derived from the Business Requirement Specification (BRS). Documents like Test Plan are prepared by keeping this document as a base. Click here for more details.

**2. Test Plan:** Test plan document is a document which contains the plan for all the testing activities to be done to deliver a quality product. Test Plan document is derived from the Product Description, SRS, or Use Case documents for all future activities of the project. It is usually prepared by the Test Lead or Test Manager. Click here for more details.

3. **Effort Estimation Report:** In this report, usually test teams mention the efforts put in to complete the testing process by the test team.

4. **Test Scenarios:** Test Scenario gives the idea of what we have to test. Test Scenario is like a high-level test case.

5**. Test Cases/Scripts:** Test cases are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected result, post-conditions and actual results.

6. **Test Data:** Test data is the data that is used by the testers to run the test cases. Whilst running the test cases, testers need to enter some input data. To do so, testers prepare test data. It can be prepared manually and also by using tools.

For example, To test a basic login functionality having a user id, password fields. We need to enter some data in the user id and password fields. So we need to collect some test data.

7. **Requirement Traceability Matrix (RTM**): Requirements Traceability Matrix (RTM) is used to trace the requirements to the tests that are needed to verify whether the requirements are fulfilled. Requirement Traceability Matrix AKA Traceability Matrix or Cross Reference Matrix. Click here for more details.

8. Defect Report/Bug Report: The purpose of using Defect report template or Bug report template is to convey the detailed information (like environment details, steps to reproduce etc.,) about the bug to the developers. It allows developers to replicate the bug easily. Click here for more details.

9. Test Execution Report: It contains the test results and the summary of test execution activities.

10. Graphs and Metrics: Software test metrics is to monitor and control process and product. It helps to drive the project towards our planned goals without deviation. Metrics answer different questions. It’s important to decide what questions you want answers to. Click here for more details.

11. Test summary report: It contains the summary of test activities and final test results.

12. **Test incident report:** It contains all the incidents such as resolved or unresolved incidents which are found while testing the software.

13. **Test closure report**: It gives a detailed analysis of the bugs found, bugs removed and discrepancies found in the software.

14. **Release Note:** Release notes will be sent to the client, customer or stakeholders along with the build. It contains list of new releases, bug fixes.

15. **Installation/configuration guide:** This guide helps to install or configure the components that make up the system and its hardware and software requirements.

16. **User guide**: This guide gives assistance to the end user on accessing the software application.

17. **Test status report**: It is to track the testing status. It is prepared on a periodic or weekly basis. It contains work done till date and work remains pending.

18. **Weekly status report** (Project manager to client): It is similar to the Test status report but generate weekly.

**Testing metrics:**

This section describes the metrics that should be used in the project to analyze the project status.

Software test metrics is to monitor and control process and product. It helps to drive the project towards our planned goals without deviation.

Metrics answer different questions. It’s important to decide what questions you want answers to.

Software test metrics are classified into two types

* Process metrics
* Product metrics

**Process Metrics:**

Software Test Metrics used in the process of test preparation and test execution phase of STLC.

The following are generated during the Test Preparation phase of STLC:

**Test Case Preparation Productivity:**

It is used to calculate the number of Test Cases prepared and the effort spent for the preparation of Test Cases.

Formula:

Test Case Preparation Productivity = (No of Test Case)/ (Effort spent for Test Case Preparation)

No. of Test cases = 240

Effort spent for Test case preparation (in hours) = 10

Test Case preparation productivity = 240/10 = 24 test cases/hour

**Test Design Coverage:**

It helps to measure the percentage of test case coverage against the number of requirements

Formula:

Test Design Coverage = ((Total number of requirements mapped to test cases) / (Total number of requirements)\*100

Total number of requirements: 100

Total number of requirements mapped to test cases: 98

Test Design Coverage = (98/100)\*100 = 98%

The following are generated during the Test Execution phase of STLC:

**Test Execution Productivity:**

It determines the number of Test Cases that can be executed per hour

Formula:

|  |  |
| --- | --- |
|  | (No of Test cases executed)/ (Effort spent for execution of test cases) |

E.g.:

No of Test cases executed = 180

Effort spent for execution of test cases = 10

Test Execution Productivity = 180/10 = 18 test cases/hour

**Test Execution Coverage:**

It is to measure the number of test cases executed against the number of test cases planed.

Formula:

Test Execution Coverage = (Total no. of test cases executed / Total no. of test cases planned to execute)\*100

Total no. of test cases planned to execute = 240

Total no. of test cases executed = 160

Test Execution Coverage = (180/240)\*100 = 75%

**Test Cases Passed:**

It is to measure the percentage no. of test cases passed

Formula:

Test Cases Pass = (Total no. of test cases passed) / (Total no. of test cases executed) \* 100

Test Cases Pass = (80/90)\*100 = 88.8 = 89%

**Test Cases Failed:**

It is to measure the percentage no. of test cases failed

Formula:

Test Cases Failed = (Total no. of test cases failed) / (Total no. of test cases executed) \* 100

Test Cases Failed= (10/90)\*100 = 11.1 = 11%

**Product metric:**

Software Test Metrics used in the process of defect analysis phase of STLC.

**Error Discovery Rate:**

It is to determine the effectiveness of the test cases.

Formula:

Error Discovery Rate = (Total number of defects found /Total no. of test cases executed)\*100

Total no. of test cases executed = 240

Total number of defects found = 60

Error Discovery Rate = (60/240)\*100 = 25%

**Defect Fix Rate:**

It helps to know the quality of a build in terms of defect fixing.

Formula:

Defect Fix Rate = (Total no of Defects reported as fixed - Total no. of defects reopened) / (Total no of Defects reported as fixed + Total no. of new Bugs due to fix)\*100

Total no of defects reported as fixed = 10

Total no. of defects reopened = 2

Total no. of new Bugs due to fix = 1

Defect Fix Rate = ((10 – 2)/(10 + 1))\*100 = (8/11)100 = 72.7 = 73%

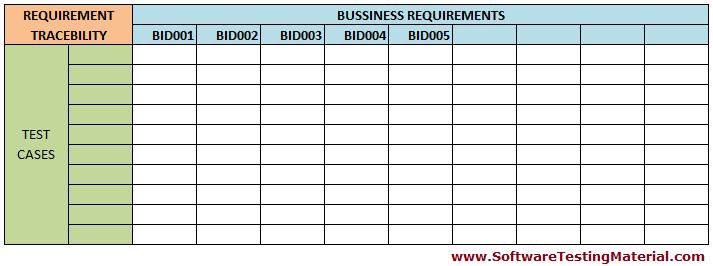
**Requirement Traceability Matrix:**

Requirement traceability matrix is used to trace the requirements to the tests that are needed to verify whether the requirements are fulfilled.

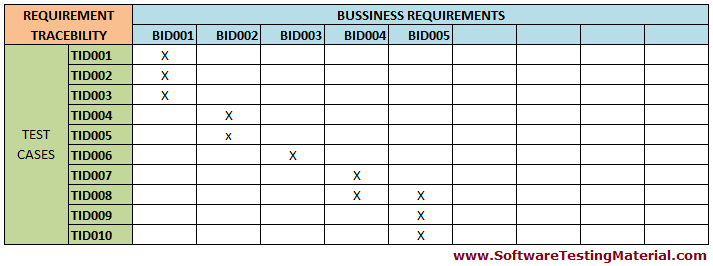
RTM too varies between organizations. Most of the organizations use just the Requirement Id’s and Test Case Id’s in the RTM. It is possible to make some other fields such as Requirement Description, Test Phase, Test case result, Document Owner etc., It is necessary to update the RTM whenever there is a change in requirement.

The following illustration gives you a basic idea about Requirement Traceability Matrix (RTM).

Assume we have 5 requirements

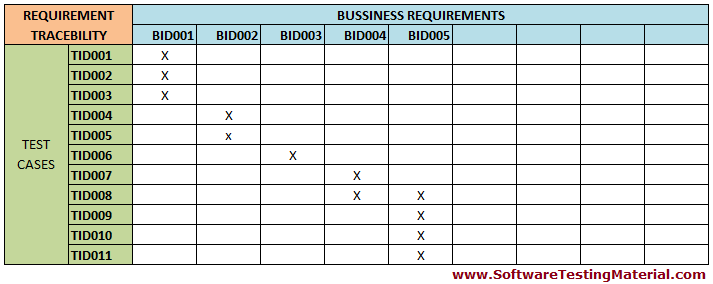


Assume total test cases identified are 10



Whenever we write new test cases, the same need to be updated in the RTM

Adding a new test case id TID011 and mapping it to the requirement id BID005



Types of Requirements Traceability Matrix (RTM):

Let’s see different types of Traceability Matrix:

**Forward Traceability:** Mapping requirements to test cases is called Forward Traceability Matrix. It is used to ensure whether the project progresses in the desired direction. It makes sure that each requirement is tested thoroughly.

**Backward or Reverse Traceability:** Mapping test cases to requirements is called Backward Traceability Matrix. It is used to ensure whether the current product remains on the right track. It makes sure that we are not expanding the scope of the project by adding functionality that is not specified in the requirements.

**Bi-directional traceability** (Forward + Backward): Mapping requirements to test cases (forward traceability) and test cases to requirements (backward traceability) is called Bi-directional Traceability Matrix. It is used to ensure that all the specified requirements have appropriate test cases and vice versa.

**Advantage of Requirements Traceability Matrix (RTM):**

* 100% test coverage
* It allows to identify the missing functionality easily
* It allows to identify the test cases which needs to be updated in case of change in requirement
* It is easy to track the overall test execution status

**Risk and mitigation:**

Identify all the testing risks that will affect the testing process and specify a plan to mitigate the risk.

**Reporting tool:**

This section outlines how defects and issues will be tracked using a reporting tool.

**Test Summary:**

This section lists out what kind of test summary reports will be produced along with the frequency. Test summary reports will be generated on a daily, weekly or monthly basis depends on how critical the project is.

**Conclusion:**

Test strategy document gives a clear vision of what the test team will do for the whole project. It is a static document means it wont change throughout the project life cycle. The one who prepares this document, must have good experience in the product domain, as this is the document that is going to drive the entire team and it won’t change throughout the project life cycle (it is a static document). Test strategy document should be circulated to entire testing team before the testing activities begin. Writing a good test strategy improves the complete testing process and leads to produce a high-quality system.

# Test plan

est plan is one of the documents in test deliverables. Like other test deliverables, the test plan document is also shared with the stakeholders. The stakeholders get to know the scope, approach, objectives, and schedule of software testing to be done.

**How To Prepare Effective Test Plan?**

Some of the measures are to start preparing the test plan early in the STLC, keep the test plan short and simple to understand, and keep the test plan up-to-date

**Who Prepare Test Plan Template?**

Usually, Test Lead prepares Test Plan and Testers involve in the process of preparing test plan document. Once the test plan is well prepared, then the testers write test scenarios and test cases based on test plan document.

**Sections of Test Plan Template:**

Following are the sections of test plan document as per IEEE 829 standards.

* Test Plan Identifier
* References
* Introduction
* Test Items
* Features To Be Tested
* Features Not To Be Tested
* Approach
* Pass/Fail Criteria
* Suspension Criteria
* Test Deliverables
* Testing Tasks
* Environmental Needs
* Responsibilities
* Staffing and Training Needs
* Schedule
* Risks and Contingencies
* Approvals

Let’s see each component of the Test Plan Document. We are going to present the Test Plan Document as per IEEE 829 Standards.

**Test Plan Identifier:**

Test Plan Identifier is a unique number to identify the test plan.

Example: ProjectName\_0001

**References**:

This section is to specify all the list of documents that support the test plan which you are currently creating.

Example: SRS (System Requirement Specification), Use Case Documents, Test Strategy, Project Plan, Project Guidelines etc.,

**Introduction:**

Introduction or summary includes the purpose and scope of the project

Example: The objective of this document is to test the functionality of the ‘ProjectName’

**Test Items:**

A list of test items which will be tested

Example: Testing should be done on both front end and back end of the application on the Windows/Linux environments.

**Features To Be Tested:**

In this section, we list out all the features that will be tested within the project.

Example: The features which are to be tested are Login Page, Dashboard, Reports.

**Features Not To Be Tested:**

In this section, we list out the features which are not included in the project.

Example: Payment using PayPal features is above to remove from the application. There is no need to test this feature.

**Approach**:

The overall strategy of how testing will be performed. It contains details such as Methodology, Test types, Test techniques etc.,

Example: We follow Agile Methodology in this project

**Pass/Fail Criteria:**

In this section, we specify the criteria that will be used to determine pass or fail percentage of test items.

Example: All the major functionality of the application should work as intended and the pass percentage of test cases should be more than 95% and there should not be any critical bugs.

**Suspension Criteria:**

In this section, we specify when to stop the testing.

Example: If any of the major functionalities are not functional or system experiences login issues then testing should suspend.

**Test Deliverables:**

List of documents need to be delivered at each phase of testing life cycle. The list of all test artifacts.

Examples: Test Cases, Bug Report

**Testing Tasks:**

In this section, we specify the list of testing tasks we need to complete in the current project.

Example: Test environment should be ready prior to test execution phase. Test summary report needs to be prepared.

**Environmental Needs:**

List of hardware, software and any other tools that are needed for a test environment.

**Responsibilities**:

We specify the list of roles and responsibilities of each test tasks.

Example: Test plan should be prepared by Test Lead. Preparation and execution of tests should be carried out by testers.

**Staffing and Training Needs:**

Plan training course to improve the skills of resources in the project to achieve the desired goals.

**Schedule**:

Complete details on when to start, finish and how much time each task should take place.

Example: Perform test execution – 120 man-hours, Test Reporting – 30 man-hours

**Risks and Contingencies:**

In this section, we specify the probability of risks and contingencies to overcome those risks.

Example: Risk – In case of a wrong budget estimation, the cost may overrun. Contingency Plan – Establish the scope before beginning the testing tasks and pay attention in the project planning and also track the budget estimates constantly.

**Approvals:**

Who should sign off and approve the testing project

Example: Project manager should agree on completion of the project and determine the steps to proceed further.

# Bug Report Template – Detailed Explanation

Defect report template or Bug report template is one of the test artifacts. It comes into picture when the test execution phase is started.

The purpose of using Defect report template or Bug report template is to convey the detailed information (like environment details, steps to reproduce etc.,) about the bug to the developers. It allows developers to replicate the bug easily.

**How to Write Good Bug Report!!**

Have you ever seen a rejected bug which has comments as “it is not reproducible”. Sometimes Dev Team rejects few bugs due to Bad Bug Report.

**Bad Bug Report?**

Imagine you are using Mozilla Firefox for testing (I am mentioning a sample case here). You found an issue that login button is not working. You have posted the same issue with all the steps except mentioning the name and version of Browser. One of the developers opened that report and tried to reproduce it based on the steps you mentioned in the report. Here, in this case, the Developer is using Internet Explorer. The Login button is working properly in their environment. So the Developer will reject the bug by mentioning the comments as the bug is not reproducible. You will find the same issue after you do retest. Again you will report the same issue and get the same comments from the Dev Team.

You forgot to mention the name and version of Browser in your bug report. If you forgot some key information to reproduce the bug in the Developers Environment, you will face the consequences like this.

It creates a bad impression on you. Action will be taken on you based on the company for wasting the time and effort.

There is an old saying: “You never get a second chance to make a first impression.”

Writing good bug report is a skill every tester should have. You have to give all necessary details to the Dev Team to get your issue fixed.

Earlier I have posted a detailed post on “Bug Life Cycle”, if you haven’t gone through it, you can browse “Bug Life Cycle” here

Do you want to get the bug fixed without rejection? So you have to report it by using a good bug report.

**How To Write Good Defect Report?**

Let me first mention what are the fields needed in a good bug report.

Defect ID, Reporter Name, Defect Reported Date, Who Detected, How Detected, Project Name, Release/Build Version, Defect/Enhancement, Environment, Priority, Severity, Status, Description, Steps To Reproduce, URL, Expected Result, Actual Result.

Earlier I have posted a detailed post on “Bug Report Template With Detailed Explanation”, click here to get the detailed explanation on each field and download a sample bug report.

The first thing we should do before writing a bug report is to reproduce the bug two to three times.

If you are sure that bug exists then ascertain whether the same bug was posted by someone else or not. Use some keywords related to your bug and search in the Defect Tracking Tool. If you did not find an issue which is related to the bug same like you found then you could start writing a bug report.

Why could not we ascertain whether the same issue is available in the related modules?. If you find that the same issue is available in the related modules then you could address those issues in the same bug report. It saves a lot of time in terms of fixing the issues and writing repeated bug reports for the same kind of issues.

Start writing a bug report by mentioning all the details in the fields as mentioned above and write detailed steps to reproduce

**Make a checklist and ensure whether you have passed all the points before reporting a bug.**

i. Have I reproduced the bug 2-3 times.

ii. Have I verified in the Defect Tracking Tool (using keywords) whether someone else already posted the same issue.

iii. Have I verified the similar issue in the related modules.

iv. Have I written the detailed steps to reproduce the bug.

v. Have I written proper defect summary.

vi. Have I attached relevant screenshots.

vii. Have I missed any necessary fields in the bug report?

**Consolidating all the points on how to write good bug report:**

i. Reproduce the bug 2-3 times.

ii. Use some keywords related to your bug and search in the Defect Tracking Tool.

iii. Check in similar modules.

iv. Report the problem immediately.

v. Write detailed steps to reproduce the bug.

vi. Write a good defect summary. Watch your language in the process of writing the bug report, your words should not offend people. Never use capital letter whilst explaining the issue.

vii. Advisable to Illustrate the issue by using proper screenshots.

viii. Proofread your bug report twice or thrice before posting it.

This is all about “writing a good bug report”. If you have any thoughts, please comment below. As promised, here is the sample defect report template.

**Defect ID**: Add a Defect ID using a naming convention followed by your team. The Defect ID will be generated automatically in case of defect management tool.

**Title/Summary**: Title should be short and simple. It should contain specific terms related to the actual issue. Be specific while writing the title.

Assume, you have found a bug in the registration page while uploading a profile picture that too a particular file format (i.e., JPEG file). System is crashing while uploading a JPEG file.

Note: I use this example, throughout this post.

Good: “Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system”.

Bad: “System crashes”.

Reporter Name: Name of the one who found the defect (Usually tester’s name but sometimes it might be Developer, Business Analyst, Subject Matter Expert (SME), Customer)

Defect Reported Date: Mention the date on which you have found the bug.

Who Detected: Specify the designation of the one who found the defect. E.g. QA, Developer, Business Analyst, SME, Customer

How Detected: In this field, you must specify on how you have detected such as while doing Testing or while doing Review or while giving Walkthrough etc.,

Project Name: Sometimes, we may work on multiple projects simultaneously. So, choose the project name correctly. Specify the name of the project (If it’s a product, specify the product name)

Release/Build Version: On which release this issue occurs. Mention the build version details clearly.

Defect/Enhancement: If the system is not behaving as intended then you need to specify it as a Defect. If its just a request for a new feature then you must specify it as Enhancement.

Environment: You must mention the details of Operation Systems, Browser Details and any other related to the test environment in which you have encountered the bug.

Example: Windows 8/Chrome 48.0.2564.103

Priority: Priority defines how soon the bug should be fixed. Usually, the priority of the bug is set by the Managers. Based on the priority, developers could understand how soon it must be fixed and set the order in which a bug should be resolved.

Description: In the description section, you must briefly explain what you have done before facing the bug.

**Steps to reproduce**: In this section, you should describe how to reproduce the bug in step by step manner. Easy to follow steps give room to the developers to fix the issue without any chaos. These steps should describe the bug well enough and allows developers to understand and act on the bug without discussing to the one who wrote the bug report. Start with “opening the application”, include “prerequisites” if any and write till the step which “causes the bug”.

Good:

i. Open URL “Your URL”

ii. Click on “Registration Page”

iii. Upload “JPEG” file in the profile photo field

Bad:

Upload a file in the registration page.

URL: Mention the URL of the application (If available)

Expected Result: What is the expected output from the application when you make an action which causes failure.

Good: A message should display “Profile picture uploaded successfully”

Bad: System should accept the profile picture.

Earlier I have posted a detailed post on “Test Case Template With Explanation”, if you haven’t gone through it, you can browse “Test Case Template With Explanation” here.

Actual Result: What is the expected output from the application when you make an action which causes failure.

Good: “Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system”.

Bad: System is not accepting profile picture.

Attachments: Attach the screenshots which you had captured when you faced the bug. It helps the developers to see the bug which you have faced.

Defect Close Date: The ‘Defect Close Date’ is the date which needs to be updated once you ensure that the defect is not reproducible.

This is all about Bug Report Template. Download a sample Bug Report / Defect Report Template for your reference.

# PDCA Cycle (Plan Do Check Act) in Software Development Life Cycle

What is PDCA Cycle?

PDCA Cycle is an iterative four-step management method used in business to focus on continuous improvement of processes. The PDCA cycle consists of four steps namely Plan, Do, Check, and Act. It is one of the key concepts of quality and it is also called the Deming circle/cycle/wheel.

**PLAN**:

Plan a change (either to solve a problem or to improve some areas) and decide what goal to achieve.

Here we define the objective, strategy and supporting methods to achieve the goal of our plan.

**DO:**

To design or revise the business requirement as planned

Here we implement the plan (in terms of putting the plan into an action) and test its performance

**CHECK:**

Evaluate the results to make sure whether we reach the goals as planned

Here we make a checklist to record what went well and what did not work (lessons learnt)

**ACT:**

If the changes are not as planned then continue the cycle to achieve the goal with a different plan.

Here we take action on what is not working as planned. Task is to keep trying to improve the process with different plan.

PDCA Cycle is a continuous process until we achieve our goals which we planned.

# Agile Methodology In Software Development

Agile is a software development methodology to build a software incrementally using short iterations of 1 to 4 weeks so that the development process is aligned with the changing business needs. Instead of a single-pass development of 6 to 18 months where all the requirements and risks are predicted upfront, Agile adopts a process of frequent feedback where a workable product is delivered after 1 to 4-week iteration.

# 

How an Agile Team Plans its Work?

An Agile team works in iterations to deliver user stories where each iteration is of 10 to 15 days. Each user story is planned based on its backlog prioritization and size. The team uses its capacity − how many hours are available with team to work on tasks − to decide how much scope they have to plan.

Point A Point defines how much a team can commit. A point usually refers to 8 hours. Each story is estimated in points.

Capacity defines how much an individual can commit. Capacity is estimated in hours

What is a User Story?

A user story is a requirement which defines what is required by the user as functionality.

A user story can be in two forms:

As a <User Role> I want <Functionality> so that <Business Value>In order to <Business value> as a <User Role> I want <Functionality>

During release planning, a rough estimate is given to a user story using relative scale as points. During iteration planning, the story is broken down into tasks.

Relationship of User Stories and Tasks:

• User story talks about what is to be done. It defines what a user needs.

• Task talks about how it is to be done. It defines how a functionality is to be implemented.

• Stories are implemented by tasks. Each story is a collection of tasks.

• User story is divided into tasks when it is planned in current iteration.

• Tasks are estimated in hours, typically from 2 to 12 hours.

**Agile Manifesto**

The Agile Manifesto was published by a team of software developers in 2001,

highlighting the importance that needs to be given to the development team,

accommodating changing requirements, customer involvement.

The Agile Manifesto is as follows:

“We are uncovering better ways of developing software by doing it and helping others

do it. Through this work, we have come to value:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left

more.”

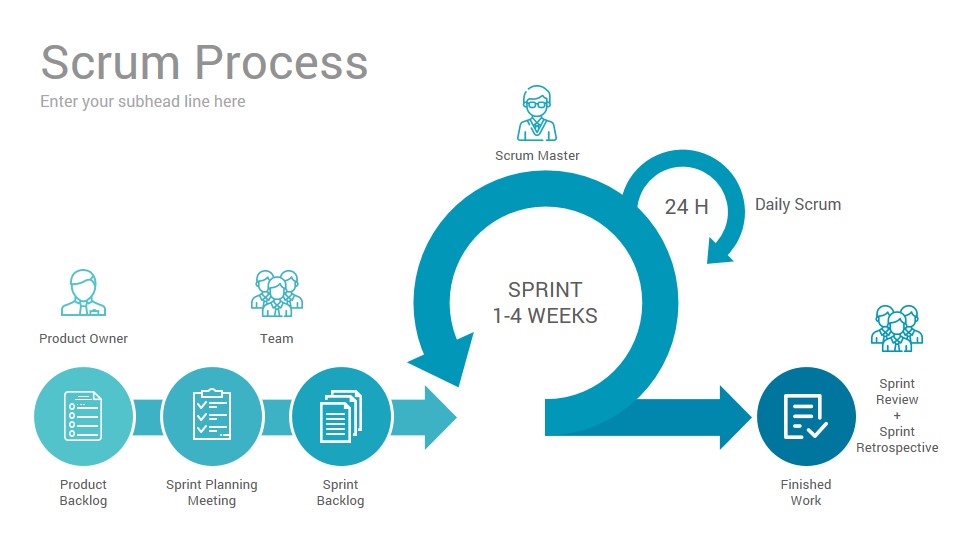
…Manifesto for Agile Software Development, Authors: Beck, Kent, et al. (2001)

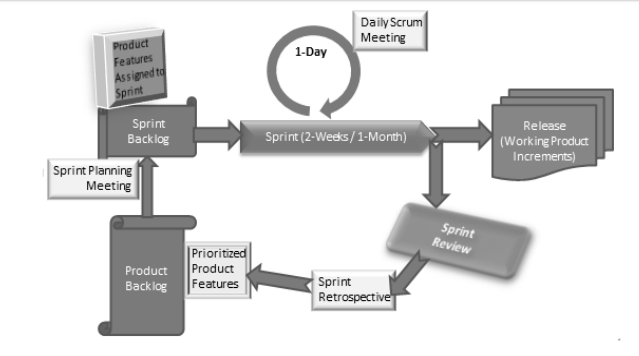
Agile methodologies include the following:

* Dynamic System Development Methodology
* Scrum
* Extreme Programming
* Test-driven Development
* Lean
* Kanban

# Agile Scrum Methodology In Software Development

Agile Scrum Methodology is one of the popular Agile software development methods. There are some other Agile software development methods but the popular one which is using widely is Agile Scrum Methodology. The Agile Scrum Methodology is a combination of both Incremental and Iterative model for managing product development.





In Scrum, the project is divided into Sprints.

Sprint: Each Sprint has a specified time line (2 weeks to 1 month). This time line will be agreed by a Scrum Team during the Sprint Planning Meeting. Here, User Stories are split in to different modules. End result of every Sprint should be potentially shippable product.

Number of story points delivered/demo in a Sprint is called velocity. For example, if team planned 30 story point(Business value) worth of user stories in a sprint and able to deliver as planned then team's velocity is 30. Total number of available hours for a sprint is called Team's Capacity.

The three important aspects involved in Scrum such as Roles, Artifacts and Meetings:

**ROLES IN AGILE SCRUM METHODOLOGY:**

**Product Owner:**

Product Owner usually represents the Client and acts as a point of contact from Client side. The one who prioritizes the list of Product Backlogs which Scrum Team should finish and release.

**Scrum Master:**

Scrum Master acts as a facilitator to the Scrum Development Team. Clarifies the queries and organizes the team from distractions and teach team how to use scrum and also concentrates on Return on Investment (ROI).

**Scrum Development Team:**

Developer’s, QA’s. Who develops the product. Scrum development team decides the effort estimation to complete a Product Backlog Item.

**Scrum Team:**

A cross-functional, self-organizing group of dedicated people (Group of Product Owner, Business Analyst, Developer’s and QA’s). Recommended size of a scrum team is 7 plus or minus 2 (i.e, between 5 to 9 members in a team)

**ARTIFACTS IN AGILE SCRUM METHODOLOGY:**

**User Stories:**

User Stories are not like a traditional requirement documents. In User Stories, stake holder mention what features they need and what they want to achieve.

**User Story Acceptance Criteria**

Each User Story also has Acceptance Criterion defined, so that correctness of

implementation of the user story is confirmed by passing the Acceptance Test that is

based on the Acceptance Criterion.

Following are the sample acceptance criterion for the example of User Story

Customer’s Withdrawal of Cash.

Acceptance Criterion 1:

Given that the account is creditworthy

And the card is valid

And the dispenser contains cash,

When the customer requests the cash

Then ensure the account is debited

And ensure cash is dispensed

And ensure the card is returned.

Acceptance Criterion 2:

Given that the account is overdrawn

And the card is valid

When the customer requests the cash

Then ensure the rejection message is displayed

And ensure cash is not dispensed

And ensure the card is returned.

**Estimation Techniques**

The Scrum Estimation of User Stories is in terms of the degree of difficulty for each

of the User Stories. To assess the degree of difficulty, a particular scale is used.

There are several types of scales that are used in Scrum Estimation. Following are

some examples:

* Numeric Sizing (1 through 10)
* T-shirt Sizes (XS, S, M, L, XL XXL, XXXL)
* Fibonacci Sequence (1, 2, 3, 5, 8, 13, 21, 34, etc.)
* Dog Breeds (Chihuahua,………,Great Dane)

**Product Backlog:**

Product Backlog is a repository where the list of Product Backlog Items stored and maintained by the Product Owner. The list of Product Backlog Items are prioritized by the Product Owner as high and low and also could re-prioritize the product backlog constantly.

**Sprint Backlog:**

Group of user stories which scrum development team agreed to do during the current sprint (Committed Product Backlog items)

**Product Burn down Chart:**

A graph which shows how many Product Backlog Items (User Stories) implemented/not implemented.

**Sprint Burn down Chart:**

A graph which shows how many Sprints implemented/not implemented by Scrum Team.

**Release Burn down Chart:**

A graph which shows List of releases still pending, which Scrum Team have planned.

**Defect Burn down Chart:**

A graph which shows how many defects identified and fixed.

**Note:** Burn Down Charts provide proof that the project is on track or not.

The sprint tracking is usually done using Burn-Down Chart. Burn-Down Chart shows

the remaining effort in day-wise number of hours. For example, let us consider a 2-

week sprint:

Sprint Duration: 2 Weeks

No. of Days per Week: 5

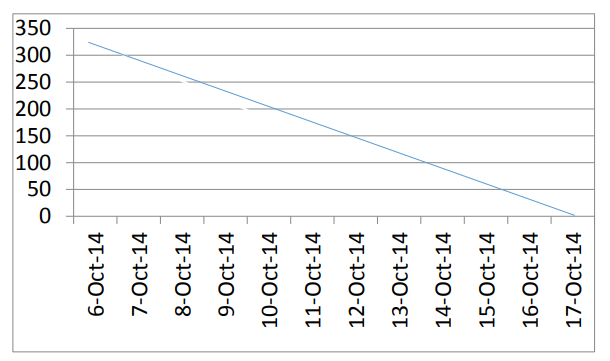
No. of Hrs. per Day: 6

No. of Resources: 6

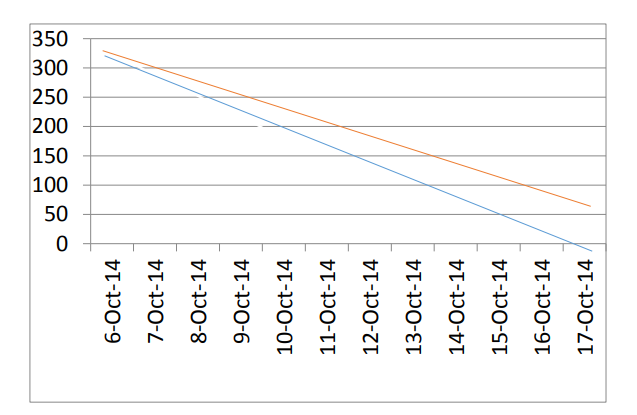
Hence, total remaining effort at the beginning of sprint is 2\*5\*6\*6 = 360 hrs.

Therefore, in an ideal scenario, 36 hours of work gets reduced in the remaining work

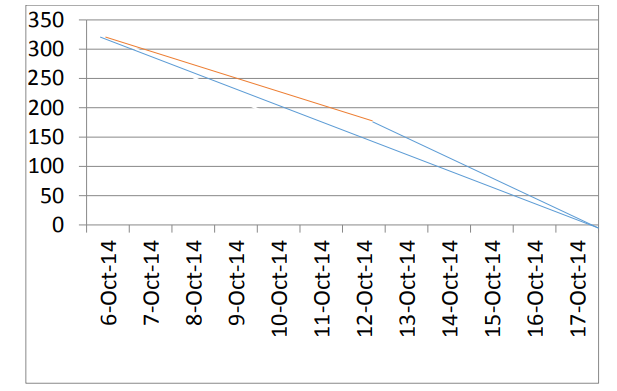
and the burn-down chart looks as follows:



If the sprint work is done as planned daily, the scrum progress is almost aligned to the ideal bar. If the sprint work gets delayed and time commitment is not met, the burn-down chart looks as follows:



But, as the burn-down chart is drawn daily, and the slippage is known early, corrective actions can be taken to meet the sprint time line. Suppose, the team stretches to meet the timeline, the burn-down chart looks as follows:



Thus, at any point in time in a Sprint, the total work remaining in the Sprint can be visualized and possibility of meeting sprint timeline can be improved.

**MEETINGS IN AGILE SCRUM METHODOLOGY:**

**Sprint Planning Meeting:**

The first step of Scrum is Sprint Planning Meeting where the entire Scrum Team attends. Here the Product Owner selects the Product Backlog Items (User Stories) from the Product Backlog.  
Most important User Stories at the top of the list and least important User Stories at the bottom. Scrum Development Team decides and provides the effort estimation.

**Daily Scrum Meeting:** (Daily Stand-up)

Daily Scrum is also known as Daily Stand-up meeting. Here each team member reports to the peer team member on what he/she did yesterday, what he/she going to do today and what obstacles are impeding in their progress. Reporting will be between peers not to Scrum Master or Product Owner. Daily Scrum will be approximately 15 mins.

**Sprint Review Meeting:**

In the Sprint Review Meeting, Scrum Development Team presents a demonstration of a potentially shippable product. Product Owner declares which items are completed and not completed. Product Owner adds the additional items to the product backlog based on the stakeholders feedback.

**Sprint Retrospective Meeting:**

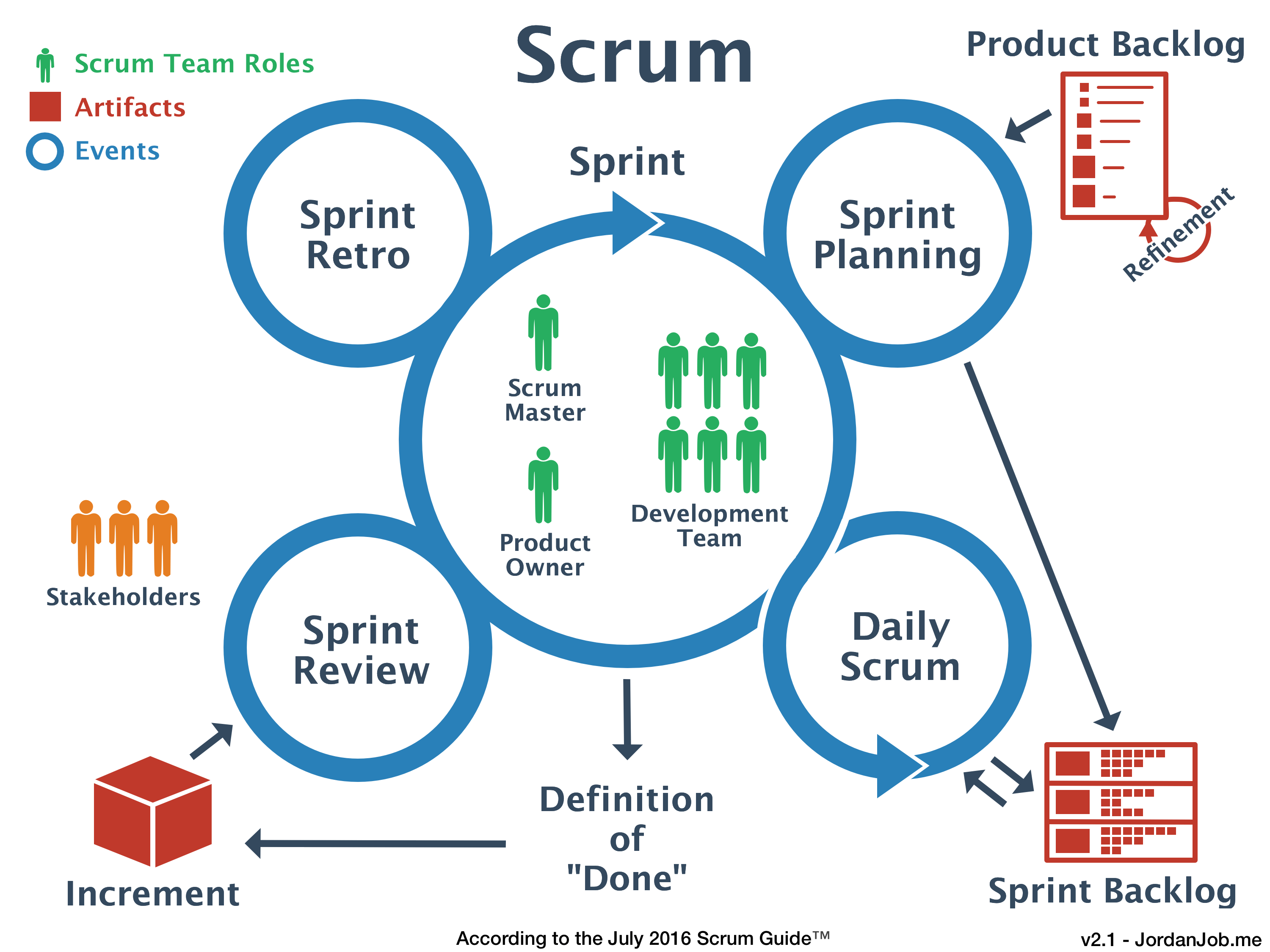
Scrum Team meets again after the Sprint Review Meeting and documents the lessons learnt in the earlier sprint such as “What went well”, “What could be improved”. It helps the Scrum Team to avoid the mistakes in the next Sprints.

When do we use Agile Scrum Methodology?

The client is not so clear on requirements  
The client expects quick releases  
The client doesn’t give all the requirements at a time

Conclusion:

In an Agile Scrum Methodology, all the members in a Scrum Team gathers and finalize the Product Backlog Items (User Stories) for a particular Sprint and commits time line to release the product. Based on the Daily Scrum meetings, Scrum Development Team develops and tests the product and presents to the Product Owner on Sprint Review Meeting. If the Product Owner accepts all the developed User Stories then the Sprint is completed and the Scrum Team goes for the next Sprint in a same manner.



**12 principles Agile Software Development:**

1. Highest priority is to satisfy the customer through early and continuous delivery of business valuable software  
2. Welcome changing requirements, even late in development  
3. Deliver working software frequently  
4. Business people and developers must work together daily with transparency throughout the project  
5. Build projects around motivated individuals  
6. The best form of communication is to do face-to-face conversation  
7. Working software is the primary measure of progress  
8. Able to maintain a constant pace  
9. Continuous attention to technical excellence  
10. Simplicity – the art of maximizing the amount of work not done – is essential  
11. Self-organizing teams  
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly