Contents

[Software 6](#_Toc25139771)

[Systems Software 6](#_Toc25139772)

[Applications Software 6](#_Toc25139773)

[The different types of application software include the following: 7](#_Toc25139774)

[Software Testing? 8](#_Toc25139775)

[Why is Software Testing Important? 8](#_Toc25139776)

[Who does Testing? 9](#_Toc25139777)

[When to Start Testing? 9](#_Toc25139778)

[When to Stop Testing? 9](#_Toc25139779)

[What Does Quality Assurance Mean In Software Testing? 9](#_Toc25139780)

[What Does Quality Control Mean In Software Testing? 10](#_Toc25139781)

[Verification & Validation: 10](#_Toc25139782)

[What Is Static Testing, When Does It Start, And What Does It Cover? 10](#_Toc25139783)

[What Is Dynamic Testing, When Does It Start, And What Does It Cover? 11](#_Toc25139784)

[What is SDLC? 11](#_Toc25139785)

[SDLC Phases 12](#_Toc25139786)

[Phase 1: Requirement collection and analysis: 12](#_Toc25139787)

[Phase 2: Feasibility study: 12](#_Toc25139788)

[Phase 3: Design: 13](#_Toc25139789)

[Phase 4: Coding: 13](#_Toc25139790)

[Phase 5: Testing: 14](#_Toc25139791)

[Phase 6: Installation/Deployment: 14](#_Toc25139792)

[Phase 7: Maintenance: 14](#_Toc25139793)

[Popular SDLC models 14](#_Toc25139794)

[Waterfall model 14](#_Toc25139795)

[Prototyping Model 16](#_Toc25139796)

[Spiral Model (SDM) 18](#_Toc25139797)

[Iterative and Incremental Model 20](#_Toc25139798)

[Agile Model 21](#_Toc25139799)

[Scrum 22](#_Toc25139800)

[Software Testing Life Cycle (STLC) 27](#_Toc25139801)

[What is Entry and Exit Criteria? 27](#_Toc25139802)

[Requirement Analysis 28](#_Toc25139803)

[Test Planning 28](#_Toc25139804)

[Test Case Development 29](#_Toc25139805)

[Test Environment Setup 29](#_Toc25139806)

[Test Execution 30](#_Toc25139807)

[Test Cycle Closure 30](#_Toc25139808)

[Levels Of Testing 31](#_Toc25139809)

[UNIT TESTING: 32](#_Toc25139810)

[INTEGRATION TESTING: 32](#_Toc25139811)

[SYSTEM TESTING (END TO END TESTING): 33](#_Toc25139812)

[ACCEPTANCE TESTING: 33](#_Toc25139813)

[Different Types of Software Testing 35](#_Toc25139814)

[#1) Alpha Testing 36](#_Toc25139815)

[#2) Acceptance Testing 36](#_Toc25139816)

[#3) Ad-hoc Testing 36](#_Toc25139817)

[#4) Accessibility Testing 37](#_Toc25139818)

[#5) Beta Testing 37](#_Toc25139819)

[#6) Back-end Testing 37](#_Toc25139820)

[#7) Browser Compatibility Testing 37](#_Toc25139821)

[#8) Backward Compatibility Testing 38](#_Toc25139822)

[#9) Black Box Testing 38](#_Toc25139823)

[#10) Boundary Value Testing 38](#_Toc25139824)

[#11) Branch Testing 38](#_Toc25139825)

[#12) Comparison Testing 38](#_Toc25139826)

[#13) Compatibility Testing 39](#_Toc25139827)

[#14) Component Testing 39](#_Toc25139828)

[#15) End-to-End Testing 39](#_Toc25139829)

[#16) Equivalence Partitioning 39](#_Toc25139830)

[#17) Example Testing 39](#_Toc25139831)

[#18) Exploratory Testing 39](#_Toc25139832)

[#20) Functional Testing 40](#_Toc25139833)

[#21) Graphical User Interface (GUI) Testing 40](#_Toc25139834)

[#22) Gorilla Testing 40](#_Toc25139835)

[#23) Happy Path Testing 40](#_Toc25139836)

[#24) Incremental Integration Testing 40](#_Toc25139837)

[#25) Install/Uninstall Testing 41](#_Toc25139838)

[#26) Integration Testing 41](#_Toc25139839)

[#27) Load Testing 41](#_Toc25139840)

[#28) Monkey Testing 41](#_Toc25139841)

[#29) Mutation Testing 41](#_Toc25139842)

[#30) Negative Testing 41](#_Toc25139843)

[#31) Non-Functional Testing 42](#_Toc25139844)

[#32) Performance Testing 42](#_Toc25139845)

[#33) Recovery Testing 42](#_Toc25139846)

[#34) Regression Testing 42](#_Toc25139847)

[#35) Risk-Based Testing (RBT) 42](#_Toc25139848)

[#36) Sanity Testing 43](#_Toc25139849)

[#37) Security Testing 43](#_Toc25139850)

[#38) Smoke Testing 43](#_Toc25139851)

[#39) Static Testing 44](#_Toc25139852)

[#40) Stress Testing 44](#_Toc25139853)

[#41) System Testing 44](#_Toc25139854)

[#42) Unit Testing 44](#_Toc25139855)

[#43) Usability Testing 45](#_Toc25139856)

[#44) Vulnerability Testing 45](#_Toc25139857)

[#45) Volume Testing 45](#_Toc25139858)

[#46) White Box Testing 45](#_Toc25139859)

[Test Scenario & Test Case 45](#_Toc25139860)

[Explain Positive Testing Approach? 45](#_Toc25139861)

[Explain Negative Testing Approach? 45](#_Toc25139862)

[What is the Test Case? 46](#_Toc25139863)

[What is a Test Scenario? 46](#_Toc25139864)

[Example of Test Scenario 46](#_Toc25139865)

[Example of Test Cases 46](#_Toc25139866)

[Why do we write Test Cases? 47](#_Toc25139867)

[Why do we write Test Scenario? 47](#_Toc25139868)

[Best practices of Creating Test cases 48](#_Toc25139869)

[Best practices of creating a Test Scenario 49](#_Toc25139870)

[What Is A Test Suite? 50](#_Toc25139871)

[What Is Meant By Test Bed? 50](#_Toc25139872)

[What Is Meant By Test Environment? 50](#_Toc25139873)

[What Is Meant By Test Data? 50](#_Toc25139874)

[What Is Meant By Test Harness? 51](#_Toc25139875)

[What Is Meant By Test Coverage? 51](#_Toc25139876)

[Test case design techniques 51](#_Toc25139877)

[Specification-Based or Black-Box techniques 51](#_Toc25139878)

[Structure-Based or White-Box techniques 52](#_Toc25139879)

[Experience-Based techniques 53](#_Toc25139880)

[Requirement Traceability Matrix (RTM) 53](#_Toc25139881)

[Advantages of Requirement Traceability Matrix (RTM): 53](#_Toc25139882)

[How to prepare Requirement Traceability Matrix (RTM): 54](#_Toc25139883)

[TEST STRATEGY AND TEST PLAN: 54](#_Toc25139884)

[Components of the Test Strategy document 55](#_Toc25139885)

[Components of the Test Plan document 55](#_Toc25139886)

[Agile Test Strategy 56](#_Toc25139887)

[Web Application Testing 57](#_Toc25139888)

[Web Application Testing - Techniques: 57](#_Toc25139889)

[Functionality Testing 57](#_Toc25139890)

[Database Testing 58](#_Toc25139891)

[Usability Testing 59](#_Toc25139892)

[GUI Testing 60](#_Toc25139893)

[Interface Testing 63](#_Toc25139894)

[Compatibility Testing 64](#_Toc25139895)

[Performance Testing 64](#_Toc25139896)

[Security Testing 65](#_Toc25139897)

[Accessibility testing 65](#_Toc25139898)

[Seven Principles of Software Testing: 68](#_Toc25139899)

[1. Testing Shows Presence of Defects: 68](#_Toc25139900)

[2. Exhaustive Testing is Impossible: 68](#_Toc25139901)

[3. Early Testing: 68](#_Toc25139902)

[4. Defect clustering 68](#_Toc25139903)

[5. The pesticide paradox 69](#_Toc25139904)

[6. Testing is context dependent 69](#_Toc25139905)

[7. Absence of errors fallacy 69](#_Toc25139906)

[Software Testing Metrics & KPIs 70](#_Toc25139907)

[Software Testing Metrics 70](#_Toc25139908)

[1. Derivative Metrics: 70](#_Toc25139909)

[2. Defect Density: 70](#_Toc25139910)

[3. Defect Leakage: 70](#_Toc25139911)

[4. Defect Removal Efficiency: 71](#_Toc25139912)

[5. Defect Category: 71](#_Toc25139913)

[6. Defect Severity Index: 71](#_Toc25139914)

[7. Review Efficiency: 71](#_Toc25139915)

[8. Test Case Effectiveness: 71](#_Toc25139916)

[9. Test Case Productivity 71](#_Toc25139917)

[10. Test Coverage: 72](#_Toc25139918)

[11. Test Design Coverage 72](#_Toc25139919)

[12. Test Execution Coverage 72](#_Toc25139920)

[13. Test Tracking & Efficiency: 72](#_Toc25139921)

[14. Test Effort Percentage: 73](#_Toc25139922)

[15. Test Effectiveness: 74](#_Toc25139923)

[16. Test Economic Metrics: 74](#_Toc25139924)

[17. Test Team Metrics: 75](#_Toc25139925)

[Software Testing Key Performance Indicators 75](#_Toc25139926)

[1. Active Defects: 75](#_Toc25139927)

[2. Automated Tests: 75](#_Toc25139928)

[3. Covered Requirements: 75](#_Toc25139929)

[4. Authored Tests: 76](#_Toc25139930)

[5. Passed Tests: 76](#_Toc25139931)

[6. Test Instances Executed: 76](#_Toc25139932)

[7. Test Executed: 76](#_Toc25139933)

[8. Defects Fixed Per Day: 76](#_Toc25139934)

[9. Direct Coverage: 76](#_Toc25139935)

[10. Percentage of Critical & Escaped Defects: 76](#_Toc25139936)

[11. Time to Test: 76](#_Toc25139937)

[12. Defect Resolution Time: 77](#_Toc25139938)

[13. Successful Sprint Count Ratio: 77](#_Toc25139939)

[14. Quality Ratio: 77](#_Toc25139940)

[15. Test Case Quality: 77](#_Toc25139941)

[16. Defect Resolution Success Ratio: 78](#_Toc25139942)

[17. Process Adherence & Improvement: 78](#_Toc25139943)

# Software

'**Software**' refers to the set of electronic program instructions or data a computer processor reads in order to perform a task or operation.

There are two main types of software: **systems software** and **application software**.

## Systems Software

**Systems software** includes the programs that are dedicated to managing the computer itself, such as the **operating system**, file management utilities, and disk operating system (or DOS). The operating system manages the computer hardware resources in addition to applications and data. Without systems software installed in our computers we would have to type the instructions for everything we wanted the computer to do

## Applications Software

**Application software**, or simply **applications**, are often called productivity programs or end-user programs because they enable the user to complete tasks, such as creating documents, spreadsheets, databases and publications, doing online research, sending email, designing graphics, running businesses, and even playing games! Application software is specific to the task it is designed for and can be as simple as a calculator application or as complex as a word processing application.

### The different types of application software include the following:

| **Application Software Type** | **Examples** |
| --- | --- |
| Word processing software | MS Word, WordPad and Notepad |
| Database software | Oracle, MS Access etc |
| Spreadsheet software | Apple Numbers, Microsoft Excel |
| Multimedia software | Real Player, Media Player |
| Presentation Software | Microsoft Power Point, Keynotes |
| Enterprise Software | Customer relationship management system |
| Information Worker Software | Documentation tools, resource management tools |
| Educational Software | Dictionaries: Encarta, BritannicaMathematical: MATLABOthers: Google Earth, NASA World Wind |
| Simulation Software | Flight and scientific simulators |
| Content Access Software | Accessing content through media players, web browsers |
| Application Suites | OpenOffice, Microsoft Office |
| Software for Engineering and Product Development | **IDE** or Integrated Development Environments |

# Software Testing?

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is [Defect](https://www.guru99.com/the-unconventional-guide-to-defect-management.html)free.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. Either it can be done manually or using automated tools.

## Why is Software Testing Important?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss

* In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
* Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
* Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point store served coffee for free as they unable to process the transaction.
* Some of the Amazon’s third party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.

### Who does Testing?

* Software Tester
* Software Developer
* End User

### When to Start Testing?

Testing is done in different forms at every phase of SDLC −

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

### When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process −

* Testing Deadlines
* Completion of test case execution
* Completion of functional and code coverage to a certain point
* Bug rate falls below a certain level and no high-priority bugs are identified
* Management decision

### What Does Quality Assurance Mean In Software Testing?

Quality assurance is a process-oriented approach to certify a software development ([SDLC](https://www.techbeamers.com/software-development-life-cycle-sdlc/)) method that it is correct and follows the standard procedures. It may bring changes in the process and cause to replace the weak practices if it identifies any. It includes review activities such as the inspection of documents, test cases, source code, and automation, etc.

### What Does Quality Control Mean In Software Testing?

Quality control is a product-oriented approach to qualify that the product under development meets the original software specifications. It also results in changes to the product if there are bugs in the system or some deviation observed in the implementation. It includes different [type of testing](https://www.techbeamers.com/testing-types/) to perform which are functional (unit, usability, integration, etc.) and non-functional (compatibility, security, performance, etc.).

### Verification & Validation:

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Verification** | **Validation** |
| 1 | Verification addresses the concern: "Are you building it right?" | Validation addresses the concern: "Are you building the right thing?" |
| 2 | Ensures that the software system meets all the functionality. | Ensures that the functionalities meet the intended behaviour. |
| 3 | Verification takes place first and includes the checking for documentation, code, etc. | Validation occurs after verification and mainly involves the checking of the overall product. |
| 4 | Done by developers. | Done by testers. |
| 5 | It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software. | It has dynamic activities, as it includes executing the software against the requirements. |
| 6 | It is an objective process and no subjective decision should be needed to verify a software. | It is a subjective process and involves subjective decisions on how well a software works. |

### What Is Static Testing, When Does It Start, And What Does It Cover?

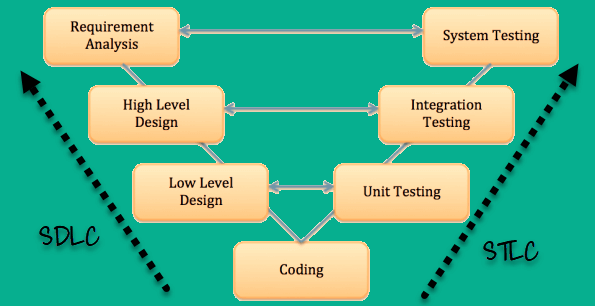
* It is a white box testing technique which directs the developers to verify their code with the help of a checklist to find errors in it.
* Developers can start it done without actually finalizing the application or program.
* Static testing is more cost-effective than Dynamic testing.
* It covers more areas than Dynamic testing in a shorter time.

### What Is Dynamic Testing, When Does It Start, And What Does It Cover?

* Dynamic testing involves the execution of an actual application with valid inputs and checking of the expected output.
* Examples of Dynamic testing are Unit Testing, Integration Testing, System Testing and Acceptance Testing.
* Dynamic testing happens after code deployment.
* It starts during the validation stage.

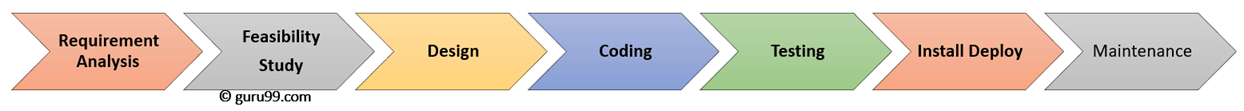
# What is SDLC?

The Software Development Lifecycle is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software which meets customer expectations.



## SDLC Phases

The entire SDLC process divided into the following stages:

[](https://www.guru99.com/images/1/080118_0641_SDLCSoftwar1.png)

* Phase 1: Requirement collection and analysis
* Phase 2: Feasibility study:
* Phase 3: Design:
* Phase 4: Coding:
* Phase 5: Testing:
* Phase 6: Installation/Deployment:
* Phase 7: Maintenance:

### Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

### Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

**There are mainly five types of feasibilities checks:**

* **Economic:**Can we complete the project within the budget or not?
* **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
* **Operation feasibility:** Can we create operations which is expected by the client?
* **Technical:** Need to check whether the current computer system can support the software
* **Schedule:** Decide that the project can be completed within the given schedule or not.

### Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

* Brief description and name of each module
* An outline about the functionality of every module
* Interface relationship and dependencies between modules
* Database tables identified along with their key elements
* Complete architecture diagrams along with technology details

Low-Level Design(LLD)

* Functional logic of the modules
* Database tables, which include type and size
* Complete detail of the interface
* Addresses all types of dependency issues
* Listing of error messages
* Complete input and outputs for every module

### Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code

### Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

### Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

### Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

* Bug fixing - bugs are reported because of some scenarios which are not tested at all
* Upgrade - Upgrading the application to the newer versions of the Software
* Enhancement - Adding some new features into the existing software

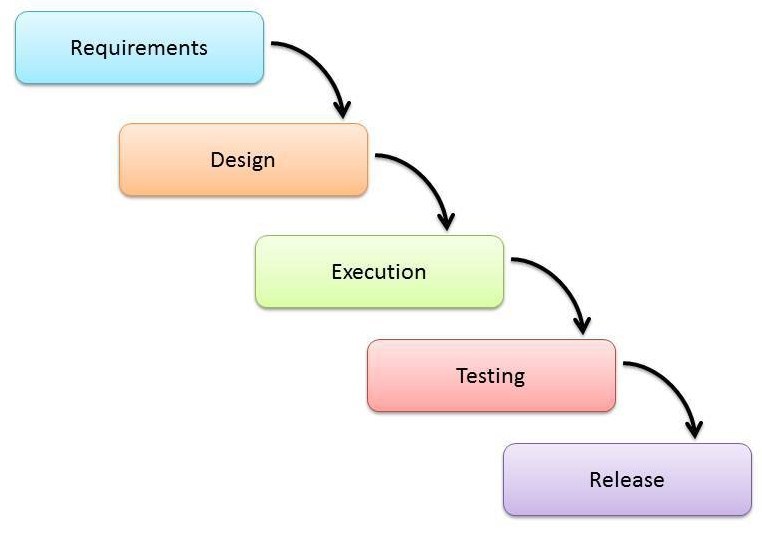
The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

## Popular SDLC models

Here, are some most important phases of SDLC life cycle:

### ****Waterfall model****

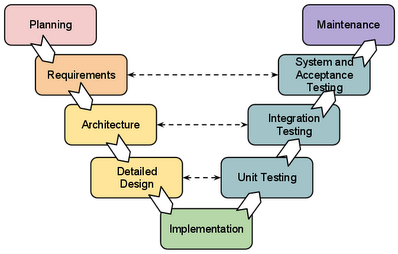
The [Waterfall Model](http://melsatar.blog/2018/02/16/the-waterfall-model-a-different-perspective/) is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.



|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Easy to explain to the users. * Structures approach. * Stages and activities are well defined. * Helps to plan and schedule the project. * Verification at each stage ensures early detection of errors/misunderstanding. * Each phase has specific deliverables. | * Assumes that the requirements of a system can be frozen. * Very difficult to go back to any stage after it finished. * A little flexibility and adjusting scope is difficult and expensive. * Costly and required more time, in addition to the detailed plan. |

**V-Model**

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.



|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Simple and easy to use * Each phase has specific deliverables. * Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle. * Works well for where requirements are easily understood. * Verification and validation of the product in the early stages of product development. | * Very inflexible, like the waterfall model. * Adjusting scope is difficult and expensive. * The software is developed during the implementation phase, so no early prototypes of the software are produced. * The model doesn’t provide a clear path for problems found during testing phases. * Costly and required more time, in addition to a detailed plan |

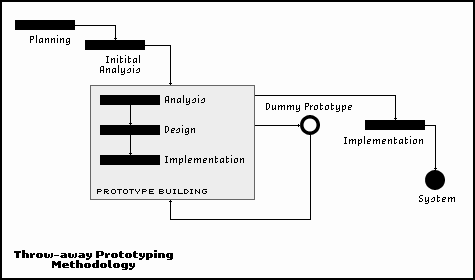
### Prototyping Model

#### Description

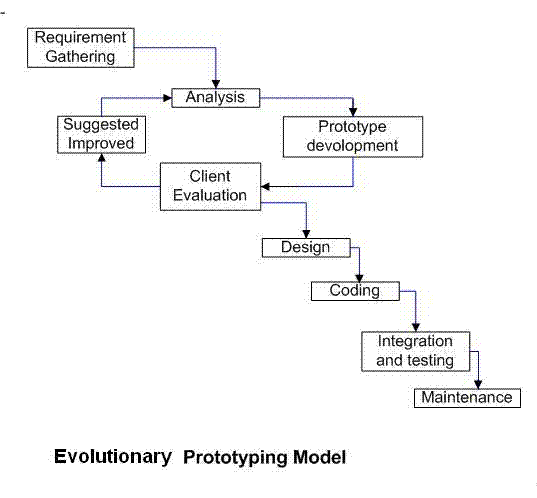
It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development and It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in the waterfall approach and hard to be implemented due to the inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

It has some types, such as:

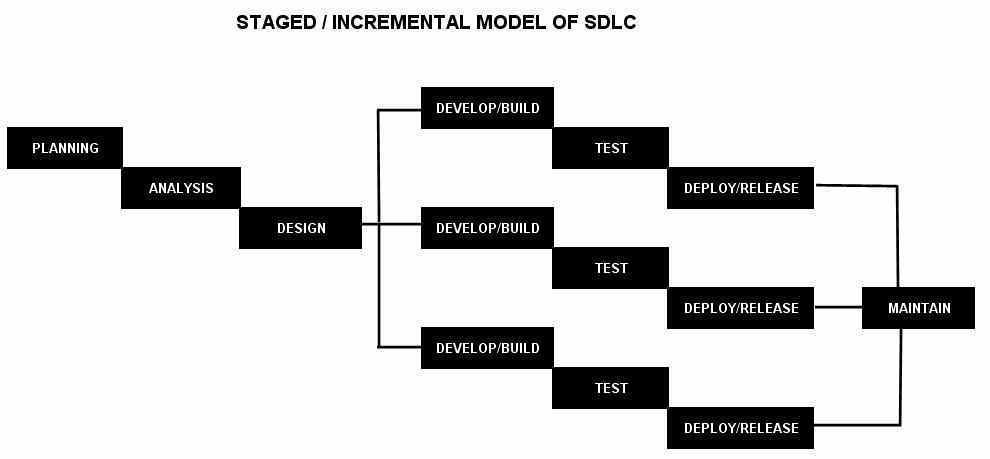
* Throwaway prototyping: Prototypes that are eventually discarded rather than becoming a part of the finally delivered software



* Evolutionary prototyping: prototypes that evolve into the final system through an iterative incorporation of user feedback.



* Incremental prototyping: The final product is built as separate prototypes. In the end, the separate prototypes are merged in an overall design.



* Extreme prototyping: used in web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented

#### The usage

* This process can be used with any software developing life cycle model. While this shall be chosen when you are developing a system has user interactions. So, if the system does not have user interactions, such as a system does some calculations shall not have prototypes.

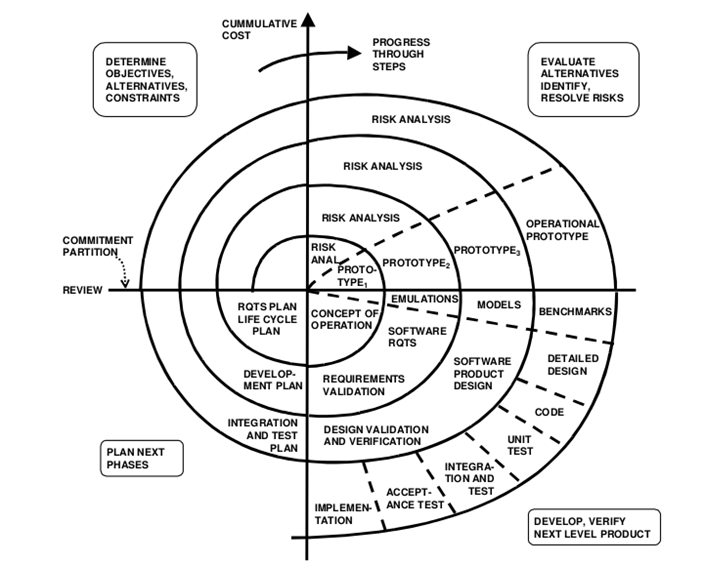
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Reduced time and costs, but this can be a disadvantage if the developer loses time in developing the prototypes. * Improved and increased user involvement. | * Insufficient analysis. User confusion of prototype and finished system. * Developer misunderstanding of user objectives. * Excessive development time of the prototype. * It is costly to implement the prototypes |

### Spiral Model (SDM)

#### Description

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.



#### The usage

It is used in the large applications and systems which built-in small phases or segments.

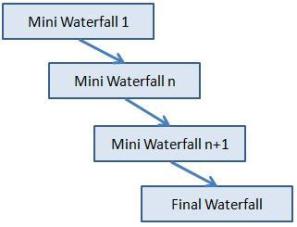
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Estimates (i.e. budget, schedule, etc.) become more realistic as work progressed because important issues are discovered earlier. * Early involvement of developers. * Manages risks and develops the system into phases. | * High cost and time to reach the final product. * Needs special skills to evaluate the risks and assumptions. * Highly customized limiting re-usability |

### Iterative and Incremental Model

#### Description

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during the development of earlier parts or versions of the system. It can consist of mini waterfalls or mini V-Shaped model



#### The usage

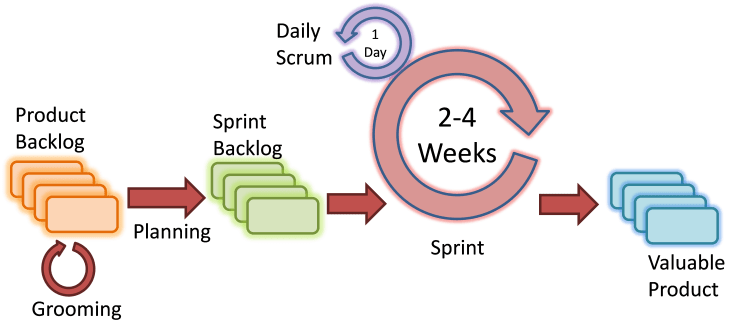
It is used in shrink-wrap application and large system which built-in small phases or segments. Also, can be used in a system has separated components, for example, ERP system. Which we can start with the budget module as a first iteration and then we can start with the inventory module and so forth.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Produces business value early in the development lifecycle. * Better use of scarce resources through proper increment definition. * Can accommodate some change requests between increments. * More focused on customer value than the linear approaches. * We can detect project issues and changes earlier. | * Requires heavy documentation. * Follows a defined set of processes. * Defines increments based on function and feature dependencies. * Requires more customer involvement than the linear approaches. * Partitioning the functions and features might be problematic. * Integration between the iterations can be an issue if it is not considered during the development and project planning. |

### Agile Model

#### Description

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.



Scrum Agile Model

#### The usage

It can be used with any type of the project, but it needs more engagement from the customer and to be interactive. Also, we can use it when the customer needs to have some functional requirement ready in less than three weeks and the requirements are not clear enough. This will enable more valuable and workable piece for software early which also increase the customer satisfaction.

#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Decrease the time required to avail some system features. * Face to face communication and continuous inputs from customer representative leaves no space for guesswork. * The end result is the high-quality software in the least possible time duration and satisfied customer. | * Scalability. * The ability and collaboration of the customer to express user needs. * Documentation is done at later stages. * Reduce the usability of components. * Needs special skills for the team. |

### Scrum

Scrum is a part of agile frameworks that has taken many industries by storm during the last few years

The main objective of Scrum is breaking down the work in such a way that it can maximize efficiency and reduce bottlenecks while moving towards project completion and customer satisfaction. The different Scrum roles within an organization include Scrum Master, Product Owner, Scrum Team etc.

The Scrum team is defined as the set of individuals working on a project, the product owner is the person who is responsible for designing different sections of the workflows and the Scrum Master facilitates both scrum team and the product owner in implementing the established work process.

Scrum framework makes sure that everyone is working in sync with the project deliverables and completely understands the milestones to be achieved. It encourages customer involvement at every stage, sets the project timelines in form of Sprints or we can say Daily Scrums. Each Sprint is given a time period when particular tasks assigned by Product Owners should be completed. The average time span for a Sprint is 7 days to one month or may also depend on the client requirements.

A few development teams focus more on daily scrums, stand-up meetings among the team members, product owner, scrum master, customers, and management etc. It helps to evaluate tasks completed on daily basis with hindrances and potential risks.



**The Scrum Events**

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. All events are time-boxed. Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened. The remaining events may end whenever the purpose of the event is achieved, ensuring an appropriate amount of time is spent without allowing waste in the process.  The Scrum Events are:

* 1. Sprint
  2. Sprint Planning
  3. Daily Scrum
  4. Sprint Review
  5. Sprint Retrospective

**Sprint**, a time-box of one month or less during which a “Done”, useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

During the Sprint:

* No changes are made that would endanger the Sprint Goal;
* Quality goals do not decrease; and,
* Scope may be clarified and re-negotiated between the Product owner and development team as more is learned.

**Sprint Planning** is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.

Sprint Planning answers the following:

* What can be delivered in the Increment resulting from the upcoming Sprint?
* How will the work needed to deliver the Increment be achieved?

Work is selected from the Product Backlog and pulled into the Sprint Backlog.  Now remember that the work in the Sprint Backlog is not a commitment, it is a forecast.  The only container of a Sprint is its time box, not the work planned for the Sprint.

**Sprint Goal**

The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting. The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint. As the Development Team works, it does so with the Sprint Goal always in mind.

**Daily Scrum** is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours. The Daily Scrum is held every day of the Sprint. At it, the Development Team plans work for the next 24 hours. This optimizes team collaboration and performance by inspecting the work since the last Daily Scrum and forecasting upcoming Sprint work. The Daily Scrum is held at the same time and place each day to reduce complexity.

The Development Team uses the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog. The Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal. Every day, the Development Team should understand how it intends to work together as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment by the end of the Sprint. The Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or replan, the rest of the Sprint’s work.

Daily Scrums improve communications, eliminate other meetings, identify impediments to development for removal, highlight and promote quick decision-making, and improve the Development Team’s level of knowledge. This is a key inspect and adapt meeting.

The structure of the meeting is set by the Development Team and can be conducted in different ways if it focuses on progress toward the Sprint Goal. Some Development Teams will use questions, some will be more discussion based.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Daily Scrum.

#### The Daily Scrum is Not a Status Meeting

**Sprint Review** is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed

During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value. This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is at most a four-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendees understand its purpose. The Scrum Master teaches everyone involved to keep it within the time-box.

The Sprint Review includes the following elements:

* Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
* The Product Owner explains what Product Backlog items have been “Done” and what has not been “Done”;
* The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;
* The Development Team demonstrates the work that it has “Done” and answers questions about the Increment;
* The Product Owner discusses the Product Backlog as it stands. He or she projects likely target and delivery dates based on progress to date (if needed);
* The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning.
* Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
* Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality and capability of the product.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

**Sprint Retrospective** is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is at most a three-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. This is the opportunity for the Scrum Team to improve and all member should be in attendance.

During the Sprint Retrospective, the team discusses:

* What went well in the Sprint
* What could be improved
* What will we commit to improve in the next Sprint

The Scrum Master encourages the Scrum Team to improve its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by improving work processes or adapting the definition of “Done” if appropriate and not in conflict with product or organizational standards.

By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation

**Scrum Artifacts**

Scrum’s artifacts represent work or value to provide transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information so that everybody has the same understanding of the artifact. The Scrum Artifacts are:

* Product Backlog
* Sprint Backlog
* Increment

**Product Backlog** is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

**Sprint Backlog** is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal.

**Increment** is the sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints. At the end of a Sprint, the new Increment must be “Done,” which means it must be in useable condition and meet the Scrum Team’s definition of “Done.”  An increment is a body of inspectable, done work that supports empiricism at the end of the Sprint. The increment is a step toward a vision or goal.  The increment must be in useable condition regardless of whether the Product Owner decides to release it.

# Software Testing Life Cycle (STLC)

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.

**Software Test Life Cycle** has the following stages.



Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.

### What is Entry and Exit Criteria?

* **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
* **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded

You have Entry and Exit Criteria for all levels in the Software Testing Life Cycle (STLC)

## Requirement Analysis

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail.

Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability )

Automation feasibility for the given testing project is also done in this stage.

**Activities**

* Identify types of tests to be performed.
* Gather details about testing priorities and focus.
* Prepare RTM
* Identify test environment details where testing is supposed to be carried out.
* Automation feasibility analysis (if required).

**Deliverables**

* RTM
* Automation feasibility report. (if applicable)

## Test Planning

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

**Activities**

* Preparation of test plan/strategy document for various types of testing
* Test tool selection
* Test effort estimation
* Resource planning and determining roles and responsibilities.
* Training requirement

**Deliverables**

* Test plan/ strategy document.
* Effort Estimation document.

## Test Case Development

This phase involves the creation, verification and rework of test cases & test scripts. Test data is identified/created and is reviewed and then reworked as well.

**Activities**

* Create test cases, automation scripts (if applicable)
* Review and baseline test cases and scripts
* Create test data (If Test Environment is available)

**Deliverables**

* Test cases/scripts
* Test data

## Test Environment Setup

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and ***can be done in parallel with Test Case Development Stage***. ***Test team may not be involved in this activity*** if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

**Activities**

* Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
* Setup test Environment and test data
* Perform smoke test on the build

**Deliverables**

* Environment ready with test data set up
* Smoke Test Results.

## Test Execution

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

**Activities**

* Execute tests as per plan
* Document test results, and log defects for failed cases
* Map defects to test cases in RTM
* Retest the Defect fixes
* Track the defects to closure

**Deliverables**

* Completed RTM with the execution status
* Test cases updated with results
* Defect reports

## Test Cycle Closure

Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

**Activities**

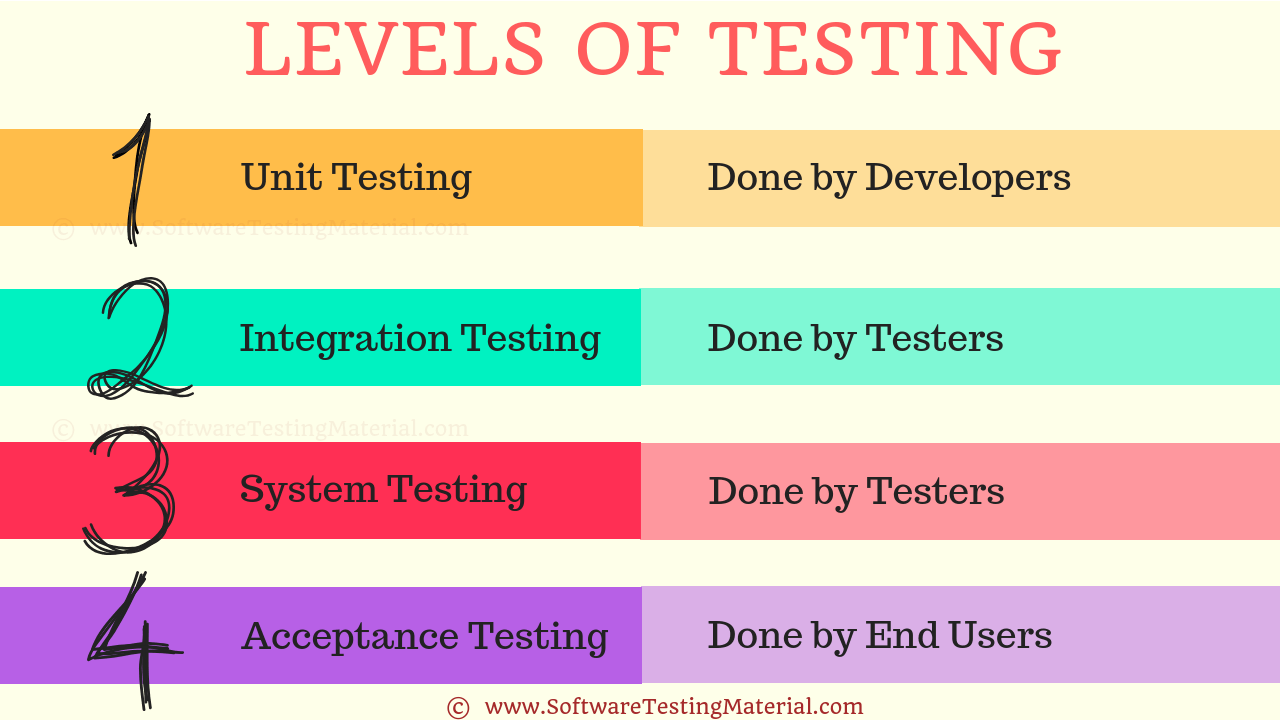
* Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
* Prepare test metrics based on the above parameters.
* Document the learning out of the project
* Prepare Test closure report
* Qualitative and quantitative reporting of quality of the work product to the customer.
* Test result analysis to find out the defect distribution by type and severity.

**Deliverables**

* Test Closure report
* Test metrics

# Levels Of Testing

|  |  |
| --- | --- |
| **Testing Category** | **Types of Testing** |
| Functional Testing |  [Unit Testing](https://www.guru99.com/unit-testing-guide.html)   [Integration Testing](https://www.guru99.com/integration-testing.html)   Smoke   UAT ( User Acceptance Testing)   Localization   Globalization   Interoperability   So on |
| Non-Functional Testing |  Performance   Endurance   Load   Volume   Scalability   Usability   So on |
| Maintenance |  Regression   Maintenance |



In Software development, both developers and testers work together to release a high-quality product. To release a high-quality product, every product goes through various testing processes. Coming to testing, testers use various levels of testing in the process of releasing a quality product. There are different levels of software testing. Each of these levels of software testing has a specific purpose. We will see each software testing level in detail.

What is 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.

Let’s see the levels of testing in detail.

### 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 the 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 a couple of unit tested modules. It is AKA I&T Testing or String Testing

It is subdivided into the 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 arouses in an interface or in a module.

#### Top-Down Integration Testing

In Top-Down Integration Testing, the high-level modules are integrated and tested first. i.e Testing from the main module to the submodule. In this type of testing, Stubs are used as a 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 the 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.

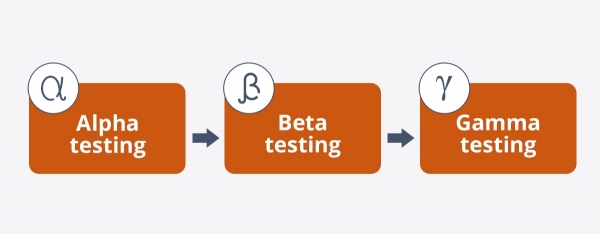
### SYSTEM TESTING (END TO END TESTING):

It’s a black box testing. Testing the fully integrated application this is also called as an 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’ experiences with the application.

### ACCEPTANCE TESTING:

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

There are three phases of software testing - alpha, beta, and gamma. They are performed one after another, and together ensures a release of high-quality software.



#### Alpha Testing

**Alpha testing** is an internal checking done by in-house development or QA team, rarely, by the customer himself. Its main purpose is to discover software bugs that were not found before. At the stage of alpha testing, software behavior is verified under real-life conditions by imitating the end users’ actions. It enables to get the fast approval from the customer before proceeding to product delivery.

The alpha phase includes the following testing types: smoke, sanity, integration, systems, usability, UI (user interface), acceptance, regression, and [functional testing](https://qatestlab.com/services/manual-testing/functional-testing/#functional%20testing). If an error is detected, then it is immediately addressed to the development team. Alpha testing helps to discover issues missed at the stage of requirement gathering. Alpha release is the software version that has passed alpha testing. The next stage is beta testing.

#### Beta Testing

**Beta testing** can be called pre-release testing. It can be conducted by a limited number of end users called [beta testers](https://qatestlab.com/resources/knowledge-center/characteristics-of-beta-tester-profession/) before the official product delivery. The main purpose of beta testing is to verify software compatibility with different software and hardware configurations, types of network connection, and to get the users’ feedback on software usability and functionality.

There are two types of beta testing:

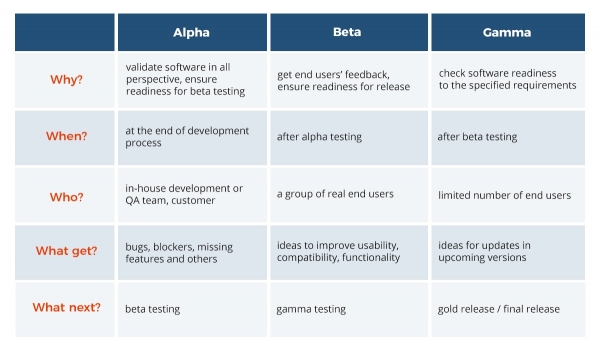
* **open beta** is available for a large group of end users or to everyone interested
* **closed beta** is available only to a limited number of users that are selected especially for beta testing.

During beta testing, end users detect and report bugs they have found. All the testing activities are performed outside the organization that has developed the product. Beta checking helps to identify the gaps between the stage of requirements gathering and their implementation. The product version that has passed beta testing is called beta release. After beta phase comes gamma testing.

#### Gamma Testing

**Gamma testing** is the final stage of the testing process conducted before software release. It makes sure that the product is ready for market release according to all the specified requirements. Gamma testing focuses on software security and functionality. But it does not include any in-house QA activities. During gamma testing, software does not undergo any modifications unless the detected bug is of a high priority and severity.

Only a limited number of users perform gamma testing, and testers do not participate. The checking includes the verification of certain specification, not the whole product. Feedback received after gamma testing are considered as updates for upcoming software versions. But, because of a limited development cycle, gamma testing is usually skipped.



# Different Types of Software Testing

Functional testing **types include:**

* Unit testing
* Integration testing
* System testing
* Sanity testing
* Smoke testing
* Interface testing
* Regression testing
* Beta/Acceptance testing

Non-functional testing **types include:**

* Performance Testing
* Load testing
* Stress testing
* Volume testing
* Security testing
* Compatibility testing
* Install testing
* Recovery testing
* Reliability testing
* Usability testing
* Compliance testing
* Localization testing

### #1) Alpha Testing

It is the most common type of testing used in the Software industry. The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user.

Alpha testing is carried out at the end of the software development phase but before the Beta Testing. Still, minor design changes may be made as a result of such testing. [Alpha testing](https://www.softwaretestinghelp.com/what-is-alpha-testing-beta-testing/) is conducted at the developer’s site. In-house virtual user environment can be created for this type of testing.

### #2) Acceptance Testing

An [acceptance test](https://www.softwaretestinghelp.com/what-is-acceptance-testing/) is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end user. Client accepts the software only when all the features and functionalities work as expected.

It is the last phase of the testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

### #3) Ad-hoc Testing

The name itself suggests that this testing is performed on [an ad-hoc](https://www.softwaretestinghelp.com/ad-hoc-testing/) basis i.e. with no reference to the test case and also without any plan or documentation in place for such type of testing. The objective of this testing is to find the defects and break the application by executing any flow of the application or any random functionality.

Ad-hoc testing is an informal way of finding defects and can be performed by anyone in the project. It is difficult to identify defects without a test case but sometimes it is possible that defects found during ad-hoc testing might not have been identified using existing test cases.

### #4) Accessibility Testing

The aim of [accessibility testing](https://www.softwaretestinghelp.com/what-is-web-accessibility-testing/) is to determine whether the software or application is accessible for disabled people or not. Here disability means deaf, color blind, mentally disabled, blind, old age and other disabled groups. Various checks are performed such as font size for visually disabled, color and contrast for color blindness etc.

### #5) Beta Testing

[Beta Testing](https://www.softwaretestinghelp.com/beta-testing/) is a formal type of software testing which is carried out by the customer. It is performed in **the Real Environment**before releasing the product to the market for the actual end users.

Beta testing is carried out to ensure that there are no major failures in the software or product and it satisfies the business requirements from an end-user perspective. Beta testing is successful when the customer accepts the software.

Usually, this testing is typically done by end-users or others. It is the final testing done before releasing an application for commercial purpose. Usually, the Beta version of the software or product released is limited to a certain number of users in a specific area.

So end user actually uses the software and shares the feedback to the company. Company then takes necessary action before releasing the software to the worldwide.

### #6) Back-end Testing

Whenever an input or data is entered on front-end application, it stores in the database and the testing of such database is known as Database Testing or Backend testing. There are different databases like SQL Server, MySQL, and Oracle etc. Database testing involves testing of table structure, schema, stored procedure, data structure and so on.

In back-end testing GUI is not involved, testers are directly connected to the database with proper access and testers can easily verify data by running a few queries on the database. There can be issues identified like data loss, deadlock, data corruption etc during this back-end testing and these issues are critical to fixing before the system goes live into the production environment

### #7) Browser Compatibility Testing

It is a subtype of Compatibility Testing (which is explained below) and is performed by the testing team.

[Browser Compatibility Testing](https://www.softwaretestinghelp.com/how-is-cross-browser-testing-performed/) is performed for web applications and it ensures that the software can run with the combination of different browser and operating system. This type of testing also validates whether web application runs on all versions of all browsers or not.

### #8) Backward Compatibility Testing

It is a type of testing which validates whether the newly developed software or updated software works well with older version of the environment or not.

Backward Compatibility Testing checks whether the new version of the software works properly with file format created by older version of the software; it also works well with data tables, data files, data structure created by older version of that software. If any of the software is updated then it should work well on top of the previous version of that software.

### #9) Black Box Testing

Internal system design is not considered in this type of testing. Tests are based on the requirements and functionality.

Detailed information about the advantages, disadvantages, and [types of Black box testing](https://www.softwaretestinghelp.com/black-box-testing/) can be seen here.

### #10) Boundary Value Testing

This type of testing checks the behavior of the application at the boundary level.

[Boundary value Testing](https://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/) is performed for checking if defects exist at boundary values. Boundary value testing is used for testing a different range of numbers. There is an upper and lower boundary for each range and testing is performed on these boundary values.

If testing requires a test range of numbers from 1 to 500 then Boundary Value Testing is performed on values at 0, 1, 2, 499, 500 and 501.

### #11) Branch Testing

It is a type of white box testing and is carried out during unit testing. Branch Testing, the name itself suggests that the code is tested thoroughly by traversing at every branch.

### #12) Comparison Testing

Comparison of a product's strength and weaknesses with its previous versions or other similar products is termed as Comparison Testing.

### #13) Compatibility Testing

It is a testing type in which it validates how software behaves and runs in a different environment, web servers, hardware, and network environment. [Compatibility testing](https://www.softwaretestinghelp.com/software-compatibility-testing/) ensures that software can run on a different configuration, different database, different browsers, and their versions. Compatibility testing is performed by the testing team.

### #14) Component Testing

It is mostly performed by developers after the completion of unit testing. [Component Testing](https://www.softwaretestinghelp.com/what-is-component-testing-or-module-testing/) involves testing of multiple functionalities as a single code and its objective is to identify if any defect exists after connecting those multiple functionalities with each other.

### #15) End-to-End Testing

Similar to system testing, [End-to-end testing](https://www.softwaretestinghelp.com/what-is-end-to-end-testing/) involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

### #16) Equivalence Partitioning

It is a testing technique and a type of Black Box Testing. During this [equivalence partitioning](https://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/), a set of group is selected and a few values or numbers are picked up for testing. It is understood that all values from that group generate the same output.

The aim of this testing is to remove redundant test cases within a specific group which generates the same output but not any defect.

Suppose, application accepts values between -10 to +10 so using equivalence partitioning the values picked up for testing are zero, one positive value, one negative value. So the Equivalence Partitioning for this testing is: -10 to -1, 0, and 1 to 10.

### #17) Example Testing

It means real-time testing. Example testing includes the real-time scenario, it also involves the scenarios based on the experience of the testers.

### #18) Exploratory Testing

Exploratory Testing is informal testing performed by the testing team. The objective of this testing is to explore the application and looking for defects that exist in the application. Sometimes it may happen that during this testing major defect discovered can even cause system failure.

During exploratory testing, it is advisable to keep a track of what flow you have tested and what activity you did before the start of the specific flow.

[An exploratory testing technique](https://www.softwaretestinghelp.com/what-is-exploratory-testing/) is performed without documentation and test cases.

### #20) Functional Testing

This type of testing ignores the internal parts and focuses only on the output to check if it is as per the requirement or not. It is a Black-box type testing geared to the functional requirements of an application. For detailed information about Functional Testing click [here](https://www.softwaretestinghelp.com/guide-to-functional-testing/).

### #21) Graphical User Interface (GUI) Testing

The objective of this GUI testing is to validate the GUI as per the business requirement. The expected GUI of the application is mentioned in the Detailed Design Document and GUI mockup screens.

The GUI testing includes the size of the buttons and input field present on the screen, alignment of all text, tables and content in the tables.

It also validates the menu of the application, after selecting different menu and menu items, it validates that the page does not fluctuate and the alignment remains same after hovering the mouse on the menu or sub-menu.

### #22) Gorilla Testing

Gorilla Testing is a testing type performed by a tester and sometimes by developer the as well. In Gorilla Testing, one module or the functionality in the module is tested thoroughly and heavily. The objective of this testing is to check the robustness of the application.

### #23) Happy Path Testing

The objective of Happy Path Testing is to test an application successfully on a positive flow. It does not look for negative or error conditions. The focus is only on the valid and positive inputs through which application generates the expected output.

### #24) Incremental Integration Testing

[Incremental Integration Testing](https://www.softwaretestinghelp.com/incremental-testing/) is a Bottom-up approach for testing i.e continuous testing of an application when a new functionality is added. Application functionality and modules should be independent enough to test separately. This is done by programmers or by testers.

### #25) Install/Uninstall Testing

[Installation and uninstallation testing](https://www.softwaretestinghelp.com/software-installationuninstallation-testing/) is done on full, partial, or upgrade install/uninstall processes on different operating systems under different hardware or software environment.

### #26) Integration Testing

Testing of all integrated modules to verify the combined functionality after integration is [termed as Integration Testing](https://www.softwaretestinghelp.com/what-is-integration-testing/). Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

### #27) Load Testing

It is a type of non-functional testing and the objective of Load testing is to check how much of load or maximum workload a system can handle without any performance degradation.

[Load testing helps](https://www.softwaretestinghelp.com/introduction-to-performance-testing-loadrunner-training-tutorial-part-1/) to find the maximum capacity of the system under specific load and any issues that cause the software performance degradation. Load testing is performed using tools like[JMeter](https://www.softwaretestinghelp.com/jmeter-tutorials/), LoadRunner, WebLoad, Silk performer etc.

### #28) Monkey Testing

[Monkey testing](https://www.softwaretestinghelp.com/what-is-monkey-testing-in-software-testing/) is carried out by a tester assuming that if the monkey uses the application then how random input, values will be entered by the Monkey without any knowledge or understanding of the application.

The objective of Monkey Testing is to check if an application or system gets crashed by providing random input values/data. Monkey Testing is performed randomly and no test cases are scripted and it is not necessary to

Monkey Testing is performed randomly and no test cases are scripted and it is not necessary to be aware of the full functionality of the system.

### #29) Mutation Testing

[Mutation Testing](https://www.softwaretestinghelp.com/what-is-mutation-testing/) is a type of white box testing in which the source code of one of the program is changed and verifies whether the existing test cases can identify these defects in the system. The change in the program source code is very minimal so that it does not impact the entire application, only the specific area having the impact and the related test cases should able to identify those errors in the system.

### #30) Negative Testing

Testers having the mindset of “attitude to break” and using negative testing they validate that if system or application breaks. [A negative testing technique](https://www.softwaretestinghelp.com/what-is-negative-testing/) is performed using incorrect data, invalid data or input. It validates that if the system throws an error of invalid input and behaves as expected.

### #31) Non-Functional Testing

It is a type of testing for which every organization having a separate team which usually called as Non-Functional Test (NFT) team or Performance team.

[Non-functional testing involves](https://www.softwaretestinghelp.com/what-is-non-functional-testing/) testing of non-functional requirements such as Load Testing, Stress Testing, Security, Volume, Recovery Testing etc. The objective of NFT testing is to ensure whether the response time of software or application is quick enough as per the business requirement.

It should not take much time to load any page or system and should sustain during peak load.

### #32) Performance Testing

This term is often used interchangeably with ‘stress' and ‘load' testing. [Performance Testing](https://www.softwaretestinghelp.com/introduction-to-performance-testing-loadrunner-training-tutorial-part-1/) is done to check whether the system meets the performance requirements. Different performance and load tools are used to do this testing.

### #33) Recovery Testing

It is a type of testing which validates that how well the application or system recovers from crashes or disasters.

Recovery testing determines if the system is able to continue the operation after a disaster. Assume that application is receiving data through the network cable and suddenly that network cable has been unplugged.

Sometime later, plug the network cable; then the system should start receiving data from where it lost the connection due to network cable unplugged.

### #34) Regression Testing

Testing an application as a whole for the modification in any module or functionality is termed as Regression Testing. It is difficult to cover all the system in [Regression Testing](https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/), so typically [automation testing tools](https://www.softwaretestinghelp.com/automation-testing-tutorial-1/) are used for these types of testing.

### #35) Risk-Based Testing (RBT)

In [Risk Based Testing](https://www.softwaretestinghelp.com/risk-management-during-test-planning-risk-based-testing/), the functionalities or requirements are tested based on their priority. Risk-based testing includes testing of highly critical functionality, which has the highest impact on business and in which the probability of failure is very high.

The priority decision is based on the business need, so once priority is set for all functionalities then high priority functionality or test cases are executed first followed by medium and then low priority functionalities.

The low priority functionality may be tested or not tested based on the available time.

The Risk-based testing is carried out if there is insufficient time available to test entire software and software needs to be implemented on time without any delay. This approach is followed only by the discussion and approval of the client and senior management of the organization.

### #36) Sanity Testing

[Sanity Testing](https://www.softwaretestinghelp.com/smoke-testing-and-sanity-testing-difference/) is done to determine if a new software version is performing well enough to accept it for a major testing effort or not. If an application is crashing for the initial use then the system is not stable enough for further testing. Hence a build or an application is assigned to fix it.

### #37) Security Testing

It is a type of testing performed by a special team of testers. A system can be penetrated by any hacking way.

[Security Testing](https://www.softwaretestinghelp.com/how-to-test-application-security-web-and-desktop-application-security-testing-techniques/) is done to check how the software or application or website is secure from internal and external threats. This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are.

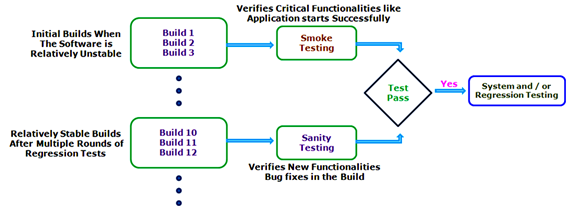
It also checks how software behaves for any hackers attack and malicious programs and how software is maintained for data security after such a hacker attack.

### #38) Smoke Testing

Whenever a new build is provided by the development team then the software testing team validates the build and ensures that no major issue exists.

The testing team ensures that the build is stable and a detailed level of testing is carried out further. [Smoke Testing](https://www.softwaretestinghelp.com/smoke-testing-and-sanity-testing-difference/) checks that no show stopper defect exists in the build which will prevent the testing team to test the application in detail.

If testers find that the major critical functionality is broken down at the initial stage itself then testing team can reject the build and inform accordingly to the development team. Smoke Testing is carried out to a detailed level of any functional or regression testing.



### #39) Static Testing

Static Testing is a type of testing which is executed without any code. The execution is performed on the documentation during the testing phase. It involves reviews, walkthrough, and inspection of the deliverables of the project. Static testing does not execute the code instead of the code syntax, naming conventions are checked.

The [static testing](https://www.softwaretestinghelp.com/static-testing-and-dynamic-testing-difference/) is also applicable for test cases, test plan, design document. It is necessary to perform static testing by the testing team as the defects identified during this type of testing are cost-effective from the project perspective.

### #40) Stress Testing

This testing is done when a system is stressed beyond its specifications in order to check how and when it fails. This is performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to the system or database load.

### #41) System Testing

Under [System Testing technique](https://www.softwaretestinghelp.com/system-testing/), the entire system is tested as per the requirements. It is a Black-box type testing that is based on overall requirement specifications and covers all the combined parts of a system.

### #42) Unit Testing

Testing of an individual software component or module is termed as [Unit Testing](https://www.softwaretestinghelp.com/unit-testing/). It is typically done by the programmer and not by testers, as it requires a detailed knowledge of the internal program design and code. It may also require developing test driver modules or test harnesses.

### #43) Usability Testing

Under [Usability Testing](https://www.softwaretestinghelp.com/usability-testing-guide/), User-friendliness check is done. Application flow is tested to know if a new user can understand the application easily or not, Proper help documented if a user gets stuck at any point. Basically, system navigation is checked in this testing.

### #44) Vulnerability Testing

The testing which involves identifying of weakness in the software, hardware and the network is known as Vulnerability Testing. Malicious programs, the hacker can take control of the system, if it is vulnerable to such kind of attacks, viruses, and worms.

So it is necessary to check if those systems undergo Vulnerability Testing before production. It may identify critical defects, flaws in the security.

### #45) Volume Testing

[Volume testing](https://www.softwaretestinghelp.com/what-is-volume-testing/) is a type of non-functional testing performed by the performance testing team.

The software or application undergoes a huge amount of data and Volume Testing checks the system behavior and response time of the application when the system came across such a high volume of data. This high volume of data may impact the system’s performance and speed of the processing time.

### #46) White Box Testing

[White Box testing](https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/) is based on the knowledge about the internal logic of an application's code.

It is also known as Glass box Testing. Internal software and code working should be known for performing this type of testing. Under these tests are based on the coverage of code statements, branches, paths, conditions etc.

# Test Scenario & Test Case

## Explain Positive Testing Approach?

The purpose of this testing is to ensure whether the system is confirming to the requirements or not.

## Explain Negative Testing Approach?

The purpose of this testing is to identify what the system should not do. It helps uncover potential flaws in the software.

## What is the Test Case?

A Test Case is a set of actions executed to verify a particular feature or functionality of your software application. The Test Case has a set test data, precondition, certain expected and actual results developed for specific test scenario to verify any requirement.

A test case includes specific variables or conditions, using which a test engineer can determine as to whether a software product is functioning as per the requirements of the client or the customer.

## What is a Test Scenario?

A Test Scenario is defined as any functionality that can be tested. It is a collective set of test cases which helps the testing team to determine the positive and negative characteristics of the project.

Test Scenario gives a high-level idea of what we need to test.

## Example of Test Scenario

For an eCommerce Application, a few test scenarios would be

**Test Scenario 1:**Check the Search Functionality

**Test Scenario 2:**Check the Payments Functionality

**Test Scenario 3:**Check the Login Functionality

## Example of Test Cases

Test cases for the **Test Scenario:** "Check the Login Functionality" would be

1. Check system behavior when valid email id and password is entered.
2. Check system behavior when invalid email id and valid password is entered.
3. Check system behavior when valid email id and invalid password is entered.
4. Check system behavior when invalid email id and invalid password is entered.
5. Check system behavior when email id and password are left blank and Sign in entered.
6. Check Forgot your password is working as expected
7. Check system behavior when valid/invalid phone number and password is entered.
8. Check system behavior when "Keep me signed" is checked

## Why do we write Test Cases?

Here, are some important reasons to create a Test Case-

* Test cases help to verify conformance to applicable standards, guidelines and customer requirements
* Helps you to validate expectations and customer requirements
* Increased control, logic, and data flow coverage
* You can simulate 'real' end user scenarios
* Exposes errors or defects
* When test cases are written for test execution, the test engineer's work will be organized better and simplified

## Why do we write Test Scenario?

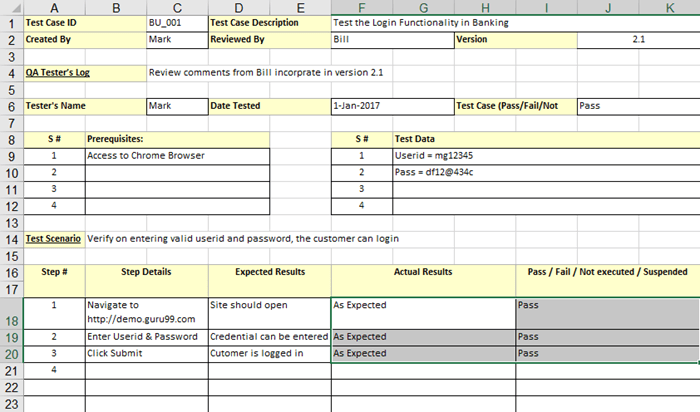
Here, are important reasons to create a Test Scenario:

* The main reason to write a test scenario is to verify the complete functionality of the software application
* It also helps you to ensure that the business processes and flows are as per the functional requirements
* Test Scenarios can be approved by various stakeholders like Business Analyst, Developers, Customers to ensure the Application Under Test is thoroughly tested. It ensures that the software is working for the most common use cases.
* They serve as a quick tool to determine the testing work effort and accordingly create a proposal for the client or organize the workforce.
* They help determine the most critical end-to-end transactions or the real use of the software applications.
* Once these Test Scenarios are finalized, test cases can be easily derived from the Test Cases.

Here, are significant differences between Test scenario and a Test Case

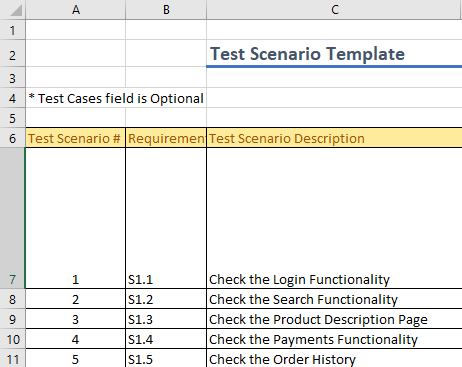
|  |  |
| --- | --- |
| **Test Scenario** | **Test Case** |
| A test scenario contains high-level documentation which describes an end to end functionality to be tested. | Test cases contain definite test steps, data, expected results for testing all the features of an application. |
| It focuses on more "what to test" **than** "how to test". | A complete emphasis on "what to test" **and**"how to test.". |
| Test scenarios are a one-liner. So, there is always the possibility of ambiguity during the testing. | Test cases have defined a step, pre-requisites, expected result, etc. Therefore, there is no ambiguity in this process. |
| Test scenarios are derived from test artifacts like BRS, SRS, etc. | Test case is mostly derived from test scenarios. Multiple Test case can be derived from a single Test Scenario |
| It helps in an agile way of testing the end to end functionality | It helps in exhaustive testing of an application |
| Test scenarios are high-level actions. | Test cases are low-level actions. |
| Comparatively less time and resources are required for creating & testing using scenarios. | More resources are needed for documentation and execution of test cases. |

## Best practices of Creating Test cases

[](https://www.guru99.com/images/1/011819_0751_TestCasevsT3.png)Test Case Example

* Test Cases should be transparent and straightforward
* Create Test Case by keeping the end user in the mind
* Avoid test case repetition
* You need to make sure that you will write test cases to check all software requirements mentioned in the specification document
* Never assume functionality and features of your software application while preparing a test case
* Test Cases must be readily identifiable

## Best practices of creating a Test Scenario

[](https://www.guru99.com/images/1/011819_0751_TestCasevsT4.png)Test Scenario Example

* Test scenarios are mostly single line statement that tells what should be tested
* Scenario description should be simple and easy to understand
* A careful assessment of the stated requirements should be done
* The required tools and resources for testing need to be accumulated before the beginning of the testing process

## What Is A Test Suite?

A test suite is a group of test cases. Each test case intends to test the application functionality.

## What Is Meant By Test Bed?

It refers to a test set up which includes necessary H/W, S/W, N/W, AUT, and any dependency software.

## What Is Meant By Test Environment?

Test Environment mostly refers to the essential hardware and software required to perform testing.

## What Is Meant By Test Data?

Test data is a set of input values required to execute the test cases. Testers define the test data according to the test requirements. They can do it manually or use generation tools.

For example, if a tester is validating a graphics tool, then he would need to procure relevant data for graph generation.

## What Is Meant By Test Harness?

A test harness is a set of scripts and demo data which tests an application under variable conditions and observes its behavior and outputs.

It emphasizes on running the test cases randomly rather than in a sequence.

## What Is Meant By Test Coverage?

Test coverage is a quality metric to represent the amount (in %) of testing completed for a software product.

It is applicable for both functional and non-functional testing activities. This metric helps testers to add missing test cases.

## Test case design techniques

The main purpose of test case design techniques is to test the functionalities and features of the software with the help of effective test cases. The test case design techniques are broadly classified into three major categories.

1.    Specification-Based techniques

2.    Structure-Based techniques

3.    Experience-Based techniques

### Specification-Based or Black-Box techniques

This technique leverages the external description of the software such as technical specifications, design, and client’s requirements to design test cases. The technique enables testers to develop test cases that provide full test coverage. The Specification-based or [black box test](https://reqtest.com/testing-blog/test-design-techniques-explained-1-black-box-vs-white-box-testing/) case design techniques are divided further into 5 categories. These categories are as follows:

#### Boundary Value Analysis (BVA)

This technique is applied to explore errors at the boundary of the input domain. It catches any input errors that might interrupt with the proper functioning of the program.

#### Equivalence Partitioning (EP)

In this technique, the test input data is partitioned into a number of classes having an equivalent number of data. The test cases are then designed for each class or partition.  This helps to reduce the number of test cases.

#### Decision Table Testing

In this technique, test cases are designed on the basis of the decision tables that are formulated using different combinations of inputs and their corresponding outputs based on various conditions and scenarios adhering to different business rules.

#### State Transition Diagrams

In this technique, the software under test is perceived as a system having a finite number of states of different types. The transition from one state to another is guided by a set of rules. The rules define the response to different inputs. This technique can be implemented on the systems which have certain workflows within them.

#### Use Case Testing

A use case is a description of a particular use of the software by a user. In this technique, the test cases are designed to execute different business scenarios and end-user functionalities.  Use case testing helps to identify test cases that cover the entire system.

### Structure-Based or White-Box techniques

The structure-based or white-box technique design test cases based on the internal structure of the software.  This technique exhaustively tests the developed code. Developers who have complete information of the software code, its internal structure, and design help to design the test cases. This technique is further divided into five categories.

#### Statement Testing & Coverage

This technique involves execution of all the executable statements in the source code at least once.  The percentage of the executable statements is calculated as per the given requirement. This is the least preferred metric for checking test coverage.

#### Decision Testing Coverage

This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed.  This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

#### Condition Testing

Condition testing also is known as Predicate coverage testing, each Boolean expression is predicted as TRUE or FALSE.  All the testing outcomes are at least tested once.  This type of testing involves 100% coverage of the code.  The test cases are designed as such that the condition outcomes are easily executed.

#### Multiple Condition Testing

The purpose of Multiple condition testing is to test the different combination of conditions to get 100% coverage.  To ensure complete coverage, two or more test scripts are required which requires more efforts.

#### All Path Testing

In this technique, the source code of a program is leveraged to find every executable path. This helps to determine all the faults within a particular code.

### Experience-Based techniques

These techniques are highly dependent on tester’s experience to understand the most important areas of the software.  The outcomes of these techniques are based on the skills, knowledge, and expertise of the people involved. The types of experience-based techniques are as follows:

#### Error Guessing

In this technique, the testers anticipate errors based on their experience, availability of data and their knowledge of product failure.  Error guessing is dependent on the skills, intuition, and experience of the testers.

#### Exploratory Testing

This technique is used to test the application without any formal documentation.  There is minimum time available for testing and maximum for test execution.  In this, the test design and test execution are performed concurrently.

# Requirement Traceability Matrix (RTM)

equirement Traceability Matrix (RTM) is a table (mostly a spreadsheet) which shows if each requirement has a respective Test case / cases to make sure if the requirement is covered for testing.

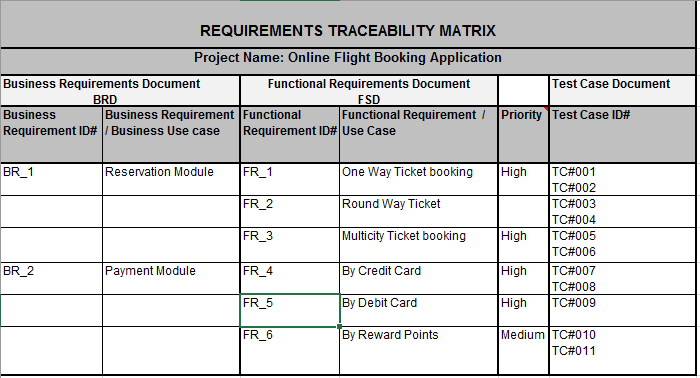
It is basically used to ensure that ALL the requirements and Change Requests are or will be tested.

### Advantages of Requirement Traceability Matrix (RTM):

1. Gives Overview of ALL the requirements
2. Shows how requirements are linked to Test Cases
3. Makes sure 100% coverage of requirements
4. Easy to prepare
5. No special tool is required

### How to prepare Requirement Traceability Matrix (RTM):

1. Get all available requirement documents. For eg. Business Requirement Document(BRD), Functional Requirement Document(FSD), Technical Requirement Document(TSD)
2. First list down All the requirements from BRD one by one with requirement ID#
3. Now go to FSD, and list all respective functional requirements for each Business Requirements
4. Open Test Scenario or Test Case document and link available TC IDs to respective Functional Requirements



# TEST STRATEGY AND TEST PLAN:

A Test Strategy document is a high level document and normally developed by project manager. This document defines “Software Testing Approach” to achieve testing objectives. The Test Strategy is normally derived from the Business Requirement Specification document.

The Test Strategy document is a static document meaning that it is not updated too often. It sets the standards for testing processes and activities and other documents such as the Test Plan draws its contents from those standards set in the Test Strategy Document.

### Components of the Test Strategy document

* Scope and Objectives
* Business issues
* Roles and responsibilities
* Communication and status reporting
* Test deliverables
* Industry standards to follow
* Test automation and tools
* Testing measurements and metrics
* Risks and mitigation
* Defect reporting and tracking
* Change and configuration management
* Training plan

The Test Plan document on the other hand, is derived from the Product Description, Software Requirement Specification SRS, or Use Case Documents.  
The Test Plan document is usually prepared by the Test Lead or Test Manager and the focus of the document is to describe what to test, how to test, when to test and who will do what test.

It is not uncommon to have one Master Test Plan which is a common document for the test phases and each test phase have their own Test Plan documents.

The test plan should be updated to reflect any deviation from the original plan. After all, Planning and Control are continuous activities in the formal test process.

### Components of the Test Plan document

* Test Plan id
* Introduction
* Test items
* Features to be tested
* Features not to be tested
* Test techniques
* Testing tasks
* Suspension criteria
* Features pass or fail criteria
* Test environment (Entry criteria, Exit criteria)
* Test deliverables
* Staff and training needs
* Responsibilities
* Schedule

This is a standard approach to prepare test plan and test strategy documents, but things can vary company-to-company.

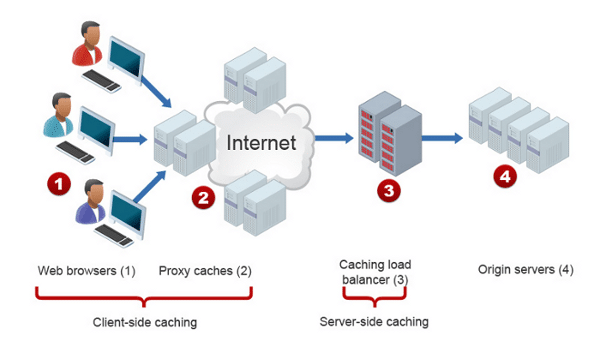
## Agile Test Strategy

In an agile environment, where we work in short sprints or iterations, each sprint is focused on only a few requirements or user stories, so it is natural that documentation may not be as extensive, in terms of both number and content.

Previously we concluded that we may not need to have an [extensive test plan in agile projects](https://www.testingexcellence.com/agile-test-plan-do-we-really-need-one/) for each sprint due to time constraints, but we do require a high-level agile test strategy as a guideline for the agile teams.

The purpose of the agile test strategy document is to list best practices and some form of structure that the teams can follow. Remember, agile does not mean unstructured.

Here, we take a look at a sample Agile Test Strategy and what to include in the document.

Web Application Testing

Web application testing, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested.

## Web Application Testing - Techniques:

### Functionality Testing

The first step of web testing ensures that the functions of a system are tested. Functional testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (unlike white-box testing).

**Check all the links:**

* Test the outgoing links from all the pages to the specific domain under test.
* Test all internal links.
* Test links jumping on the same pages.
* Test links used to send email to admin or other users from web pages.
* Test to check if there are any orphan pages.
* Finally, link checking includes, check for broken links in all above-mentioned links.

**Test forms on all pages:**  
Forms are an integral part of any website. Forms are used for receiving information from users and to interact with them. So what should be checked in these forms?

* First, check all the validations on each field.
* Check default values are being populated
* Wrong inputs in the forms to the fields in the forms.
* if a user does not fill a mandatory field in a form an error message is shown.
* Once submitted, the data in the forms is submitted to a live database

**Test Cookies** are working as expected. Cookies are small files used by websites to primarily remember active user sessions so you do not need to log in every time you visit a website. Cookie Testing will include

* Testing cookies (sessions) are deleted either when cache is cleared or when they reach their expiry.
* Delete cookies (sessions) and test that login credentials are asked for when you next visit the site.
* Test if the cookies are encrypted before writing to the user machine

**Validate your HTML/CSS:**

* If you are optimizing your site for Search engines then HTML/CSS validation is the most important one. Mainly validate the site for HTML syntax errors. Check if the site is crawlable to different search engines.
* Standard Compliance. Ensure standards such W3C, OASIS, IETF, ISO, ECMA, or WS-I are followed

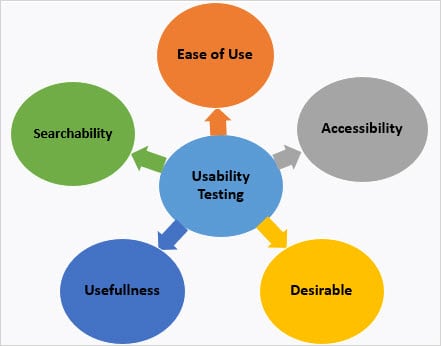
### Database Testing

* Check all the functionality which is happening on every action performed in the application. Actions can include deletion, addition or save options (Data Integrity Testing).
* Check whether the added record is added in the DB with the exact value.
* Check the deleted record gets removed from the database.
* Test data retrieved from your database is shown accurately in your web application
* Test if any errors are shown while executing queries
* Check response time of queries and fine tune them if necessary.

### Usability Testing

Usability testing is a way to see how easy to use something is by testing it with real users. Users are asked to complete tasks, typically while they are being observed by a researcher, to see where they encounter problems and experience confusion. If more people encounter similar problems, recommendations will be made to overcome these usability issues.

* Website content should be informative, structured and linked logically so that user can understand easily
* Web page controls should be easy for users to navigate
* The website should have Help & Instruction documents uploaded
* The website should have the Search feature for end-user convenience
* Access to/from the Main menu to all pages should be there
* Website content should be verified for any spelling mistakes
* The website should follow defined guidelines in the context of background color, pattern, style, fonts, image placements, frames, borders etc.
* The website should be accustomed to the translation feature considering the fact that it can be accessed by users from different nations with different languages, currencies etc



In many companies there dedicated departments for usability testing, separate from QA testing

### GUI Testing

Graphical User Interface (GUI) testing is defined as the process of testing the system's Graphical User Interface of the Application Under Test. GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc.

* Check all the GUI elements for size, position, width, length, and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
* Check you can execute the intended functionality of the application using the GUI
* Check Error Messages are displayed correctly
* Check for Clear demarcation of different sections on screen
* Check Font used in an application is readable
* Check the alignment of the text is proper
* Check the Color of the font and warning messages is aesthetically pleasing
* Check that the images have good clarity
* Check that the images are properly aligned
* Check the positioning of GUI elements for different screen resolution. Testing of the screen in different resolutions with the help of zooming in and zooming out like 640 x 480, 600x800, etc.
* Testing the different sections of the screen.
* Testing of the font whether it is readable or not.
* Testing the alignment of the texts and other elements like icons, buttons, etc. are in proper place or not.
* Testing the colors of the fonts.
* Testing the colors of the error messages, warning messages.
* Testing of the spelling.
* Testing whether the interface is attractive or not.
* Testing of the scrollbars according to the size of the page if any.
* Testing of the disabled fields if any.
* Testing of the size of the images.
* Testing of the headings whether it is properly aligned or not.
* Testing of the color of the hyperlink.

TOOLS –

**Details of various Browser Add-ons**

| **Name** | **Usage Details** | **Compatibility** |
| --- | --- | --- |
| Page Ruler | This add-on assists in testing width and height of the components. The top, left, right and bottom positions of the components can also be figured out | Chrome and Firefox |
| Web Inspector | Web inspector displays the font, text color and background color of the just by clicking on the web inspector icon and hovering it over the section which is to be tested | Chrome and Safari |
| Fire Bug | Firebug is an open source add on for monitoring the web page’s CSS, HTML, DOM, XHR and JavaScript. This is an alternative of inspect element, compatible to Firefox. | Firefox |
| ColorZilla | It is a color picker add-on used to analyze the color of the webpage | Chrome and Firefox |
| Measure it | It is used to test the width, height and alignment of elements in pixels. | Chrome, Safari and Firefox |

#### Exact **Difference between GUI Testing and Usability Testing**

|  |  |
| --- | --- |
| **GUI Testing** | **Usability testing** |
| 1. In GUI Testing tester tests the application front end design to see whether its meets the client requirements or not. | 1. In Usability Testing tester tests that whether the application is user friendly or not by checking how easily user can access the application. |
| 2. In GUI Testing we check whether the design and layout of application as per the standards and client requirements or not. | 2. In Usability Testing we check whether the design and layout of application is easy to use or not means it is user friendly or not. |
| 3. GUI Testing is more concerned with look and feel of the application means how people react and feel after look in to the application so its testing is done accordingly that. | 3. Usability Testing is more concerned with easiness and user friendliness of the application means how people react after using the application means application is easy to use or not so it’s testing is done accordingly that. |
| 4. In GUI Testing tester tests the appearance of the software. | 4. In Usability Testing tester tests the easiness to use the software. |
| 5. GUI Testing is done to ensure it meets the design specifications like links, colors, fonts, font sizes, fields etc are displayed as specified in SRS or as specified in client requirements. | 5. Usability Testing is done to ensure that the GUI is well designed and easy to use like links and buttons are easily clickable and leaving any of the mandatory field blank gives the proper message that please enter the xyz in mandatory field. |
| 6. GUI Testing is done by keeping in mind the look and feel of application means how application looks. | 6. Usability Testing is done by keeping the end user in mind. |
| 7. It stands for Graphical User Interface. It is nothing its only confirm the design specifications with the application. | 7. It is done to ensure that the GUI is well designed and easy to use. |
| 8. It is done on different platforms to verify the [Look and Feel Testing](http://testingbasicinterviewquestions.blogspot.in/2012/01/what-is-look-and-feel-testing.html). (Look and Feel of the application). | 8. It is done to verify how much the application is user friendly to an end user. |
| 9. In GUI Testing, tester test whether the front end design of the system is meeting with project standards or not. | 9. In Usability Testing, tester tests whether the control flow of the system is convenient for end user or not. |
| 10. In this testing we just test the appearance of the application. | 10. In this testing we test the interaction of functionality with the user is effective or not. |
| 11. Example: Example includes colors, fonts, font sizes, buttons, links, icons, placement of data labels and fields etc. are displayed as specified or not. | 11. Example: Example includes firstly displayed all mandatory fields, cursor positioning for enter the data into the right field, tab button should work easily etc. |
| 12. In GUI Testing we only focus on the interface of the application. | 12. Quality of product is depending on Usability Testing. |
| 13. In this testing we test only the front end of the application. | 13. In this Testing we test the overall working of application according to a non-technical user’s point of view. |

### Interface Testing

Interface Testing is defined as a software testing type which verifies whether the communication between two different software systems is done correctly.

A connection that integrates two components is called interface. This interface in a computer world could be anything like API's, web services, etc. Testing of these connecting services or interface is referred to as Interface Testing.

Interface Testing includes testing of two main segments:

1. Web server and application server interface
2. Application server and Database server interface

* Check servers are executed properly or not
* Errors are handled properly or return an error message for any query made by an application
* To check the result when a connection to the server is reset.
* To check the security aspect when the components communicate within themselves.
* To check the impact of network failure on the communication between the components.

Interface testing is basically done on the messaging layer of the system architecture. It mostly involves testing the REST API or SOAP web service with JSON or XML format.

Tools – Soap UI, Postman, REST ASSURED, JMETER, FIDDLER

### Compatibility Testing

Compatibility of your website is a very important testing aspect. See which compatibility test to be executed:

* Browser compatibility
* Operating system compatibility
* Device compatibity
* Printing options

Different browsers have different configurations and settings that your web page should be compatible with. Test web application on different browsers like Internet Explorer, Firefox, Netscape Navigator, AOL, Safari, Opera browsers with different versions

Test your web application on different operating systems like Windows, Unix, MAC, Linux, Solaris with different OS flavors.

A website can be browsed through different devices like laptops, mobiles, tablets etc.  Hence, testing should be performed on the devices too covering the below scenarios.

* Website screen size should be adjustable as per the device
* A device should be screen rotation featured
* The website should not show up any loading issues on different devices with different network speeds
* Verify the website behaviour when the device is in/out of network range
* Verify the website behaviour on low CPU and Memory to support different form factors

If you are giving page-printing options then make sure fonts, page alignment, page graphics etc., are getting printed properly. Pages should fit the paper size or as per the size mentioned in the printing option.

### Performance Testing

**Checklist**

* Website application response times at different connection speeds
* Load test your web application to determine its behavior under normal and peak loads
* Stress test your web site to determine its break point when pushed to beyond normal loads at peak time.
* Test if a crash occurs due to peak load, how does the site recover from such an event
* Make sure optimization techniques like gzip compression, browser and server side cache enabled to reduce load times

### Security Testing

**Checklist**

* The website should be accessible to only authenticated users
* Website users should be able to perform only those tasks for which they are authorized
* The website should be verified for CAPTCHA fields for user identification
* Browser security settings should be verified while moving from secure to insecure pages
* Web Server protection should be there for inaccessible web directories or files
* Ensure restricted files should not download without appropriate access
* Sessions which got inactive should automatically get killed after a certain period of time
* All invalid and unauthorized attempts by end-users or intermittent system errors/failures should get logged for analysis purpose

### Accessibility testing

Accessibility Testing is defined as a type of Software Testing performed to ensure that the application being tested is usable by people with disabilities like hearing, color blindness, old age and other disadvantaged groups. It is a subset of [Usability Testing](https://www.guru99.com/usability-testing-tutorial.html).

**Speech RecognitionSoftware -** It will convert the spoken word to text , which serves as input to the computer.

**Screen reader software** - Used to read out the text that is displayed on the screen

**Screen Magnification Software**- Used to enlarge the monitor and make reading easy for vision-impaired users.

**Special keyboard** made for the users for easy typing who have motor control difficulties

Web Accessibility Guidelines are a set of defined rules to make web contents accessible to people with disabilities. There are number of guidelines defined by different countries. Section 508 and WCAG guidelines are popular accessibility standard guidelines in use today.

1. Section 508- Section 508 is the accessibility standard defined by the US government, to make sure that all US government websites can be accessed by people with disabilities. As per section 508 guidelines all electronic and information technology should be accessible to disabled users.

2. WCAG- Web Content Accessibility Guidelines or WCAG define the standards for accessibility for individuals, organizations and governments worldwide. WCAG 2.0 has been accepted as an International Standards Organization (ISO) standard, and many countries have adopted WCAG 2.0 as their legal standard for web accessibility. Web Content Accessibility Guidelines: Includes both WCAG1.0 and WCAG2.0 specifications.

Checklist:

1. Valid HTML- Use the HTML Validator Service by W3C  to validate your code.

Open the [W3C HTML Validator](http://validator.w3.org/)( site. Enter the URL of application in Address and click the Check button.

2. Headings- In the W3C HTML Validator under More Options, select the “Show Outline” option and validate the application to check that headings are present and in the right order.

3. Large Fonts- In your browser window change the text size to ‘Largest’ to test the large fonts. For instance, to activate large fonts in Internet Explorer, open IE>navigate to the View menu>Text Size>Select “Largest”. Verify text size is increased. Also verify text size of form fields is also increased.

4. High Contrast- To set the high contrast in windows, open the Control Panel> navigate to “Ease of Access Center”> click on “Make the computer easier to see”. Under “High Contrast” section, select a High Contrast color scheme.

High Contrast can also be turned on or off using keyboard shortcut keys, press left ALT + left SHIFT + PRINT SCREEN> a pop up to turn on High Contrast is displayed>click on Yes button to enable High Contrast.

After changing the color scheme to High Contrast, verify that all contents are displayed as per the color changes and important information is not disappeared from page.

5. Alternate Text- Text associated with an image is Alternate text. Images should include Alt Text in order to make these images accessible especially for people with visual disabilities. Screen readers can read text associated with images for the people who cannot see. In Internet Explorer, hover the mouse over each image; the Alt text will display for the image as tool tip.

6. Captions and Transcripts**–** Captions and Transcripts allow the web audio and video content to be accessible to people who have hearing disabilities and who cannot access to audio or video. Check that your videos recordings have captions and transcripts accompany audio.

7. Skip Navigation**–** Skip navigation links make web pages easily accessible for people with certain mobility impairments. In order the verify that, press Ctrl+Home keys to move focuses to the top of the page. Use the Tab key to start moving through links. Make sure that “Skip to Content” link is visible near the top.

8. Tab Order & Link Text- Make sure that all the fields and links are accessible using Tab key in right order. Text associated with each link should be unique and understandable.

9. Form Labels- To make sure that each form label is accessible, put the mouse on label of each form field; flashing cursor should appear to the associated field. Check that each form field has a label. If a field does not have a label, hover the mouse over the field and confirm that it has an appropriate title attribute, which will appear as a tool tip.

10. Keyboard Operations- Make sure that all the dynamic elements (drop-down menus, tab pages, Flash interfaces, etc.) are accessible using keyboard. Use the Tab key, up/down arrows, space bar, etc. to check the accessibility of all the dynamic elements.

11. PDF Documents**–** Open PDF document and click on File> Save as Text>Save the text file. Open the saved text file and make sure that all text of pdf file is present in text file in the correct order.

12. PDF Forms**–** Make sure that each form field is accessible through keyboard. To confirm that each form field contains label, put the mouse over each form field and check that a label displayed for each field as a tool tip.

13. Disable styles and linearize tables**–** Disable the styles using WAVE or with the “Web Developer toolbar” extension for Firefox. Make sure that contents without style are displayed in order and understandable.

14. Test content scaling**–** To check the test content scaling, increase the font size in web browser to 150% and make sure that page is readable and usable. In order to enlarge the images, zoom the web page and check text in images is readable.

**TOOLS – ACHECKER, WAVE (FF Addon), TAW, Accessibility Developer Tools(chrome addon), Web accessibility toolbar(IE or Opera addon)**

# ****Seven Principles of Software Testing:****

### ****1.**** Testing Shows Presence of Defects:

Testing an application can only reveal that one or more defects exist in the application, however, testing alone cannot prove that the application is error free. Therefore, it is important to design test cases which find as many defects as possible.

### ****2.**** Exhaustive Testing is Impossible:

Unless the application under test (AUT) has a very simple logical structure and limited input, it is not possible to test all possible combinations of data and scenarios. For this reason, risk and priorities are used to concentrate on the most important aspects to test.

### ****3****. Early Testing:

The sooner we start the testing activities the better we can utilize the available time. As soon as the initial products, such the requirement or design documents are available, we can start testing. It is common for the testing phase to get squeezed at the end of the development lifecycle, i.e. when development has finished, so by starting testing early, we can prepare testing for each level of the development lifecycle.

Another important point about early testing is that when defects are found earlier in the lifecycle, they are much easier and cheaper to fix. It is much cheaper to change an incorrect requirement than having to change a functionality in a large system that is not working as requested or as designed!

### 4. **Defect clustering**

During testing, it can be observed that most of the reported defects are related to small number of modules within a system. i.e. small number of modules contain most of the defects in the system. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

### 5. The pesticide paradox

If you keep running the same set of tests over and over again, chances are no more new defects will be discovered by those test cases. Because as the system evolves, many of the previously reported defects will have been fixed and the old test cases do not apply anymore. Anytime a fault is fixed or a new functionality added, we need to do regression testing to make sure the new changed software has not broken any other part of the software. However, those regression test cases also need to change to reflect the changes made in the software to be applicable and hopefully fine new defects.

### 6. Testing is context dependent

Different methodologies, techniques and types of testing is related to the type and nature of the application. For example, a software application in a medical device needs more testing than a games software. More importantly a medical device software requires risk based testing, be compliant with medical industry regulators and possibly specific test design techniques. By the same token, a very popular website, needs to go through rigorous performance testing as well as functionality testing to make sure the performance is not affected by the load on the servers.

### 7. Absence of errors fallacy

Just because testing didn’t find any defects in the software, it doesn’t mean that the software is ready to be shipped. Were the executed tests really designed to catch the most defects? or where they designed to see if the software matched the user’s requirements? There are many other factors to be considered before making a decision to ship the software.

Other principles to note are:

* + **Testing must be done by an independent party.**

Testing should not be performed by the person or team that developed the software since they tend to defend the correctness of the program.

* + **Assign best personnel to the task.**

Because testing requires high creativity and responsibility only the best personnel must be assigned to design, implement, and analyze test cases, test data and test results.

* + **Test for invalid and unexpected input conditions as well as valid conditions.**

The program should generate correct messages when an invalid test is encountered and should generate correct results when the test is valid.

* + **Keep software static during test.**

The program must not be modified during the implementation of the set of designed test cases.

* + **Provide expected test results if possible.**

A necessary part of test documentation is the specification of expected results, even if providing such results is impractical.

# Software Testing Metrics & KPIs

The metrics and KPIs serve a crucial role and help the team determine the metrics that calculate the effectiveness of the testing teams and help them gauge the quality, efficiency, progress, and the health of the software testing.

## Software Testing Metrics

Software testing metrics are the best way of measuring and monitoring the various testing activities performed by the team of testers during the [software testing life cycle](https://www.thinksys.com/qa-testing/complete-guide-to-stlc/).

### Derivative Metrics:

Derivative metrics help identify the various areas that have issues in the software testing process and allows the team to take effective steps that increase the accuracy of testing.

### Defect Density:

The total number of defects found in a software during a specific period of time- operation or development. The results are then divided by the size of that particular moduleThe defect density of a software is counted per thousand lines of the code, which is also known as KLOC. The formula used for this is:

|  |
| --- |
| Defect Density = Defect Count/Size of the Release/Module |

### ****Defect Leakage:****

If any defects are left undetected by the team and are found by the user, it is known as defect leakage or bug leakage.

|  |
| --- |
| Defect Leakage = (Total Number of Defects Found in UAT/ Total Number of Defects Found Before UAT) x 100 |

### ****Defect Removal Efficiency:****

DRE = Number of defects resolved by the development team/ (Total number of defects at the moment of measurement)

### ****Defect Category:****

Defect category metric offers an insight into the different quality attributes of the software, such as its usability, performance, functionality, stability, reliability, and more.

|  |
| --- |
| Defect Category = Defects belonging to a particular category/ Total number of defects. |

### ****Defect Severity Index:****

Defect severity index (DSI) offers an insight into the quality of the product under test

|  |
| --- |
| Defect Severity Index (DSI) = Sum of (Defect \* Severity Level) / Total number of defects |

### ****Review Efficiency:****

Review defects can be found in documents.  It helps to decrease the probability of defect leakage in subsequent stages of testing and validates the test case effectiveness.

|  |
| --- |
| Review Efficiency (RE) = Total number of review defects / (Total number of review defects + Total number of testing defects) x 100 |

### ****Test Case Effectiveness:****

The objective of this metric is to know the efficiency of test cases that are executed by the team of testers during every testing phase. It helps in determining the quality of the test cases.

|  |
| --- |
| Test Case Effectiveness = (Number of defects detected / Number of test cases run) x 100 |

### ****Test Case Productivity****

This metric is used to measure and calculate the number of test cases prepared by the team of testers and the efforts invested by them in the process. It is used to determine the test case design productivity and is used as an input for future measurement and estimation.

|  |
| --- |
| Test Case Productivity = (Number of Test Cases / Efforts Spent for Test Case Preparation) |

### ****Test Coverage:****

Test coverage is another important metric that defines the extent to which the software product’s complete functionality is covered. It indicates the completion of testing activities and can be used as criteria for concluding testing

|  |
| --- |
| Test Coverage = Number of detected faults/number of predicted defects. |

Another important formula that is used while calculating this metric is:

|  |
| --- |
| Requirement Coverage = (Number of requirements covered / Total number of requirements) x 100 |

### ****Test Design Coverage****

measures the percentage of test cases coverage against the number of requirements

|  |
| --- |
| Test Design Coverage = (Total number of requirements mapped to test cases / Total number of requirements) x 100 |

### ****Test Execution Coverage****

 It helps us get an idea about the total number of test cases executed as well as the number of test cases left pending.

|  |
| --- |
| Test Execution Coverage = (Total number of executed test cases or scripts / Total number of test cases or scripts planned to be executed) x 100 |

### ****Test Tracking & Efficiency:****

It is a quality attribute of the testing team that is measured to ensure all testing activities are carried out in an efficient manner. The various metrics that assist in test tracking and efficiency are as follows:

**Passed Test Cases Coverage:** It measures the percentage of passed test cases.

|  |
| --- |
| (Number of passed tests / Total number of tests executed) x 100 |

**Failed Test Case Coverage:** It measures the percentage of all the failed test cases.

|  |
| --- |
| (Number of failed tests / Total number of test cases failed) x 100 |

**Test Cases Blocked:** Determines the percentage of test cases blocked, during the software testing process.

|  |
| --- |
| (Number of blocked tests / Total number of tests executed) x 100 |

**Fixed Defects Percentage:** With the assistance of this metric, the team is able to identify the percentage of defects fixed.

|  |
| --- |
| (Defect fixed / Total number of defects reported) x 100 |

**Accepted Defects Percentage:** The focus here is to define the total number of defects accepted by the development team. These are also measured in percentage.

|  |
| --- |
| (Defects accepted as valid / Total defect reported) x 100 |

**Defects Rejected Percentage:** Another important metric considered under test track and efficiency is the percentage of defects rejected by the development team.

|  |
| --- |
| (Number of defects rejected by the development team / total defects reported) x 100 |

**Defects Deferred Percentage:** It determines the percentage of defects deferred by the team for future releases.

|  |
| --- |
| (Defects deferred for future releases / Total defects reported) x 100 |

**Critical Defects Percentage:** Measures the percentage of critical defects in the software.

|  |
| --- |
| (Critical defects / Total defects reported) x 100 |

**Average Time Taken to Rectify Defects:** With the assistance of this formula, the team members are able to determine the average time taken by the development and testing team to rectify the defects.

|  |
| --- |
| (Total time taken for bug fixes / Number of bugs) |

### ****Test Effort Percentage:****

test efforts percentage offer an evaluation of what was estimated before the commencement of the testing process vs the actual efforts invested by the team of testersSimilar to test efficiency, test efforts are also evaluated with the assistance of various metrics:

**Number of Test Run Per Time Period:** Here, the team measures the number of tests executed in a particular time frame.

|  |
| --- |
| (Number of test run / Total time) |

**Test Design Efficiency:** The objective of this metric is to evaluate the design efficiency of the executed test.

|  |
| --- |
| (Number of test run / Total Time) |

**Bug Find Rate:** One of the most important metrics used during the test effort percentage is bug find rate. It measures the number of defects/bugs found by the team during the process of testing.

|  |
| --- |
| (Total number of defects / Total number of test hours) |

**Number of Bugs Per Test:** As suggested by the name, the focus here is to measure the number of defects found during every testing stage.

|  |
| --- |
| (Total number of defects / Total number of tests) |

**Average Time to Test a Bug Fix:** After evaluating the above metrics, the team finally identifies the time taken to test a bug fix

|  |
| --- |
| (Total time between defect fix & retest for all defects / Total number of defects) |

### ****Test Effectiveness:****

the test effectiveness metrics offer the percentage of the difference between the total number of defects found by the software testing and the number of defects found in the software.

|  |
| --- |
| Test Effectiveness (TEF) = (Total number of defects injected + Total number of defects found / Total number of defect escaped) x 100 |

### ****Test Economic Metrics:****

While testing the software product, various components contribute to the cost of testing, like people involved, resources, tools, and infrastructure. Hence, it is vital for the team to evaluate the estimated amount of testing, with the actual expenditure of money during the process of testing. This is achieved by evaluating the following aspects:

* Total allocated the cost of testing.
* The actual cost of testing.
* Variance from the estimated budget.
* Variance from the schedule.
* Cost per bug fix.
* The cost of not testing

### ****Test Team Metrics:****

Finally, the test team metrics are defined by the team. This metric is used to understand if the work allocated to various test team members is distributed uniformly and to verify if any team member requires more information or clarification about the test process or the project. This metric is immensely helpful as it promotes knowledge transfer among team members and allows them to share necessary details regarding the project, without pointing or blaming an individual for certain irregularities and defects. Represented in the form of graphs and charts, this is fulfilled with the assistance of the following aspects:

* Returned defects are distributed team member vise, along with other important details, like defects reported, accepted, and rejected.
* The open defects are distributed to retest per test team member.
* Test case allocated to each test team member.
* The number of test cases executed by each test team member.

## Software Testing Key Performance Indicators

Key performance indicators are the important metrics that are calculated by the software testing teams to ensure the project is moving in the right direction and is achieving the target effectively, which was defined during the planning, strategic, and/or budget sessions.

### ****Active Defects:****

A simple yet important KPI, active defects help identify the status of a defect- new, open, or fixed -and allows the team to take the necessary steps to rectify it. These are measured based on the threshold set by the team and are tagged for immediate action if they are above the threshold.

### Automated Tests:

While monitoring and analyzing the key performance indicators, it is important for the test manager to identify the automated tests. Through tricky, it allows the team to track the number of automated tests, which can help catch/detect the critical and high priority defects introduced in the software delivery stream.

### Covered Requirements:

With the assistance of this key performance indicator the team can track the percentage of requirements covered by at least one test. The test manager monitors the these this KPI every day to ensure 100% test and requirements coverage.

### Authored Tests:

Another important key performance indicator, authored tests are analyzed by the test manager, as it helps them analyze the test design activity of their business analysts and testing engineers.

### Passed Tests:

The percentage of passed tests is evaluated/measured by the team by monitoring the execution of every last configuration within a test. This helps the team in understanding how effective the test configurations are in detecting and trapping the defects during the process of testing.

### Test Instances Executed:

This key performance indicator is related to the velocity of the test execution plan and is used by the team to highlight the percentage of the total instances available in a test set. However, this KPI does not offer an insight into the quality of the build.

### Test Executed:

Once the test instances are determined the team moves ahead and monitors the different types of test execution, such as manual, automates, etc. Just like test instances executed, this is also a velocity KPI.

### Defects Fixed Per Day:

By evaluating this KPI the test manager is able to keep a track of the number of defects fixed on a daily basis as well as the efforts invested by the team to rectify these defects and issues. Moreover, it allows them to see the progress of the project as well as the testing activities.

### Direct Coverage:

This KPI helps to perform a manual or automated coverage of a feature or component and ensures that all features and their functions are completely and thoroughly tested. If a component is not tested during a particular sprint, it will be considered incomplete and will not be moved until it is tested.

### Percentage of Critical & Escaped Defects:

The percentage of critical and escaped defects is an important KPI that needs the attention of software testers. It ensures that the team and their testing efforts are focused on rectifying the critical issues and defects in the product, which in turn helps them ensure the quality of the entire testing process as well as the product.

### Time to Test:

The focus of this key performance indicator is to help the software testing team measure the time that a feature takes to move from the stage of “testing” to “done”. It offers assistance in calculating the effectiveness as well as the efficiency of the testers and understanding the complexity of the feature under test.

### Defect Resolution Time:

Defect resolution time is used to measure the time it takes for the team to find the bugs in the software and to verify and validate the fix. Apart from this, it also keeps a track of the resolution time, while measuring and qualifying the tester’s responsibility and ownership for their bugs. In short, from tracking the bugs and making sure the bugs are fixed the way they were supposed to, to closing out the issue in a reasonable time, this KPI ensures it all.

### Successful Sprint Count Ratio:

Though a software testing metric, this is also used by software testers as a KPI, once all the successful sprint statistics are collected. It helps them calculate the percentage of successful sprints, with the assistance of the following formula:

|  |
| --- |
| Successful Sprint Count Ratio: (Successful Sprint / Total Number of Sprints) x 100 |

### Quality Ratio:

Based on the passed or failed rates of all the tests executed by the software testers, the quality ratio, is used as both a software testing metrics as well as a KPI. The formula used for this is:

|  |
| --- |
| Quality Ratio: (Successful Tests Cases / Total Number of Test Cases) x 100 |

### Test Case Quality:

 A software testing metric and a KPI, test case quality, helps evaluate and score the written test cases according to the defined criteria. It ensures that all the test cases are examined either by producing quality test case scenarios or with the assistance of sampling. Moreover, to ensure the quality of the test cases, certain factors should be considered by the team, such as:

* They should be written for finding faults and defects.
* Test & requirements coverage should be fully established.
* The areas affected by the defects should be identified and mentioned clearly.
* Test data should be provided accurately and should cover all the possible situations.
* It should also cover success and failure scenarios.
* Expected results should be written in a correct and clear format.

### Defect Resolution Success Ratio:

By calculating this KPI, the team of software testers can find out the number of defects resolved and reopened. If none of the defects are reopened then 100% success is achieved in terms of resolution. Defect resolution success ratio is evaluated with the assistance of the following formula:

|  |
| --- |
| Defect Resolution Success Ratio = [ (Total Number of Resolved Defects) – (Total Number of Reopened Defects) / (Total Number of Resolved Defects) ] x 100 |

### Process Adherence & Improvement:

This KPI can be used for the software testing team to reward them and their efforts if they come up with any ideas or solutions that simplify the process of testing and make it agile as well as more accurate.

s