

MANUAL TESTING DETAILED NOTES
BY SAJARAS K

(1) What is a software ?

Software is a set of instructions, data or programs used to operate computers and execute specific tasks.

(2) What is software testing

Software Testing encompasses all activities that are aimed at discovering errors in a software product or its components. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

(3) Typical Objectives of Testing

Delivering quality products is the ultimate objective of testing. Also, whenever someone invests time or money for an application, keeps expectations as well. Hence, testing refines the software and makes it as per the user's expectations. Let's have look over the various objectives of testing:

(a) Identification of Bugs, and Errors:

To prevent defects by evaluate work products such as requirements, user stories, design, and Code. it reduce the level of risk of inadequate software quality.

(b) Verification of requirements

To verify whether all specified requirements have been fulfilled

(c) Offers Confidence:

To build confidence in the level of quality of the test object.

(d) Quality Product

The main aim of testing is to maintain the quality of the product. Also, testing has its own cycle and in each phase, all focus revolves around quality only.

(e) Enhances Growth

A quality delivery increases the potential of a business. And we all know quality comes through testing only.

(4) Why is Testing Necessary / Important ?

Software testing is very important because of the following reasons:

(a) pointing out defects and errors

Software testing is really required to point out the defects and errors that were made during the development phases. It can help reduce the risk of failures occurring during operation.

Eg: Programmers may make a mistake during the implementation of the software. There could be many reasons for this like lack of experience of the programmer, lack of knowledge of the programming language, insufficient experience in the domain, incorrect implementation of the algorithm due to complex logic or simply human error.

(b) increase reliability and customer satisfactions.

It's essential since it makes sure that the customer finds the organization reliable and their satisfaction in the application is maintained.

(b) prevent monetary losses for the testing organisation.

If the customer does not find the testing organization reliable or is not satisfied with the quality of the deliverable, then they may switch to a competitor organization. Sometimes contracts may also include monetary penalties with respect to the timeline and quality of the product. In such cases, if proper software testing may also prevent monetary losses.

(c) Ensure the quality of the product.

It is very important to ensure the quality of the product. quality product delivered to the customers helps in gaining their confidence.

(d) Builds the customers confidence.

As explained in the previous point, delivering good quality product on time builds the customers confidence in the team and the organization.

(e) Reduces maintenance cost.

High quality product typically has fewer defects and requires lesser maintenance effort, which in turn means reduced costs.

(f) Helps to improve the performance of application.

Testing is required for an effective performance of software application or product.

(f) prevents production/future failures.

It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.

(g) Helps to identify issues early.

Proper testing ensures that bugs and issues are detected early in the life cycle of the product or application.

(h) Reduce development cost and saves time.

If defects related to requirements or design are detected late in the life cycle, it can be very expensive to fix them since this might require redesign, re-implementation and retesting of the application.

(i) Required to increase user satisfaction.

Users are not inclined to use software that has bugs. They may not adopt a software if they are not happy with the stability of the application.

(j) Prevents business risks.

In case of a product organization or startup which has only one product, poor quality of software may result in lack of adoption of the product and this may result in losses which the business may not recover from.

(5) Testing Principles

(a) Testing shows the presence of defects, not their absence.

It's important to note that not finding defects does not make your software bug-free. Because no software is without its defects, testers iterate on their tests to find as many defects as possible.

(b) Exhaustive testing is impossible.

Testing all possible combinations of inputs and preconditions is not typically possible. Rather, teams and managers should prioritize testing based on risk analysis to the product and business.

(c) Early testing saves time and money.

Testers don't need to wait until the software is deployed to test it. Bugs discovered earlier in a product's lifecycle are significantly cheaper and easier to address than if it were a customer-discovered bug.

(d) Defects cluster together.

Most software issues follow the Pareto Principle—80% of the issues stem from the same 20% of its modules. While there may be outliers to this, it's a helpful rule for focusing testing.

(e) Beware the Pesticide Paradox.

This borrows from the idea in agriculture that using the same pesticide over and over again will lead to a decline in its efficacy. In the software world, this means that the usual test cases will eventually stop finding new defects. Review and revise tests regularly.

(f) Testing is context-dependent.

A rinse-and-repeat testing model won't work for all scenarios. For instance, a high-traffic ecommerce website must have different test cases than an inventory app used by warehouse staff.

(g) Absence of errors is a fallacy.

Software without known issues doesn't equal error-free software. Finding and fixing some defects won't guarantee the software's overall success.

(5) Why software have bugs ?

(a) Miscommunication or No Communication.

The success of any software application depends on the communication between stakeholders, development, and testing teams. Unclear requirements and misinterpretation of requirements are the two major factors that cause defects in software. Defects are introduced if exact requirements are not communicated properly.

(b) Software Complexity

The complexity of the current software applications can be difficult for anyone with no experience in modern-day software development. Windows-type interfaces, Client-Server, and Distributed Applications, Data Communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity. Using object-oriented techniques can complicate, instead of simplifying, a project unless it is well-engineered.

(c) Programming Errors

Programmers, like anyone else, can make common programming mistakes. Not all developers are domain experts. Inexperienced programmers or programmers without proper domain knowledge can introduce simple mistakes while coding.

Lack of simple coding practices, unit testing, debugging are some of the common reasons for these issues to get introduced at the development stage.

(d) Changing Requirements

The customer may not understand the effects of changes or may understand and anyway request them to redesign, rescheduling of engineers, effects on the other projects, and the work already completed may have to be redone or thrown out, hardware requirements that may be affected, etc.

If there are any minor changes or major changes, known and unknown dependencies, then the parts of the project are likely to interact and cause problems, and the complexity of keeping a track of changes may result in errors. The enthusiasm of engineering staff may be affected.

In some fast-changing business environments, continuously changed requirements may be a fact of life.

In these cases, the management must understand the resulting risks, and QA & test engineers must adapt and plan for continuous extensive testing to keep the inevitable bugs from running out of control.

(e)Time Pressures

Scheduling software projects is difficult, often requiring a lot of guesswork. When deadlines loom and the crunch comes, mistakes will be made still.

Unrealistic schedules, though not common, the major concern in small-scale projects/companies results in software bugs. If there is not enough time for proper design, coding, and testing, then it's quite obvious for defects to be introduced.

(f) Egotistical or Overconfident People

People prefer to say things like:

'no problem'

'piece of cake'

'I can whip that out in a few hours'

'It should be easy to update that old code'

Instead of:

'That adds a lot of complexity and we could end up making a lot of mistakes'.

'We do not know if we can do that; we'll wing it'.

'I can't estimate how long it will take until I take a closer look at it'.

'We can't figure out what that old spaghetti code did in the first place'.

If there are too many unrealistic 'no problem's', then it results in software bugs.

(g) Poorly Documented code

It's tough to maintain and modify the code that is badly written or poorly documented; the result is Software Bugs. In many organizations, management provides no incentive for programmers to document their code or write clear, understandable code.

In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it. Any new programmer working on this code may get confused and need to spend more time because of the complexity of the project and the poorly documented code.

(h) Software Development Tools

Visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, resulting in added bugs.

Continuously changing software tools are used by software programmers. Keeping pace with the different versions and their compatibility is a major ongoing issue.

(i) Obsolete Automation Scripts

Writing automation scripts takes a lot of time, especially for complex scenarios. If automation team's record/write any test script but forget to update it over a period, then that test could become obsolete.

If the automation test is not validating the results properly, then it won't be able to catch the defects.

(j) Lack of Skilled Testers

Having skilled testers with domain knowledge is extremely important for the success of any project. But appointing all experienced testers is not possible for all companies.

Domain knowledge and the tester's ability to find defects can produce high-quality software. Compromise on any of this can result in buggy software.

(6) Testing Terminologies .

(a) What is a bug?

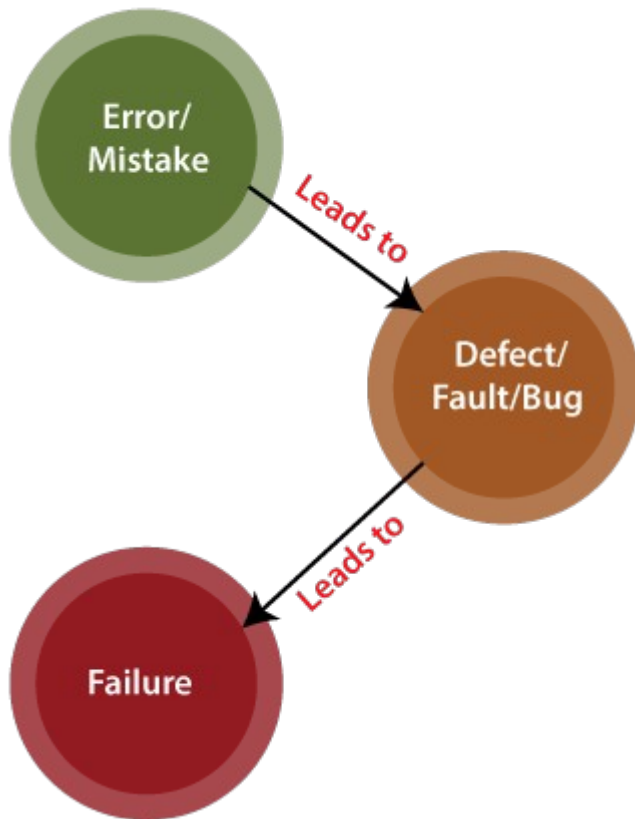
In software testing, a bug is the informal name of defects, which means that software or application is not working as per the requirement. When we have some coding error, it leads a program to its breakdown, which is known as a bug. The test engineers use the terminology Bug.

(b) What is a Defect?

When the application is not working as per the requirement is known as defects. It is specified as the difference from the actual and expected result.

(c) What is Error?

The Problem in code leads to errors, which means that a mistake can occur due to the developer's coding error as the developer misunderstood the requirement or the requirement was not defined correctly. The developers use the term error.



(d) What is Fault?

The fault may occur in software because it has not added the code for fault tolerance, making an application act up.

A fault may happen in a program because of the following reasons:

- Lack of resources.
- An invalid step.
- Inappropriate data definition.

(e) What is Failure?

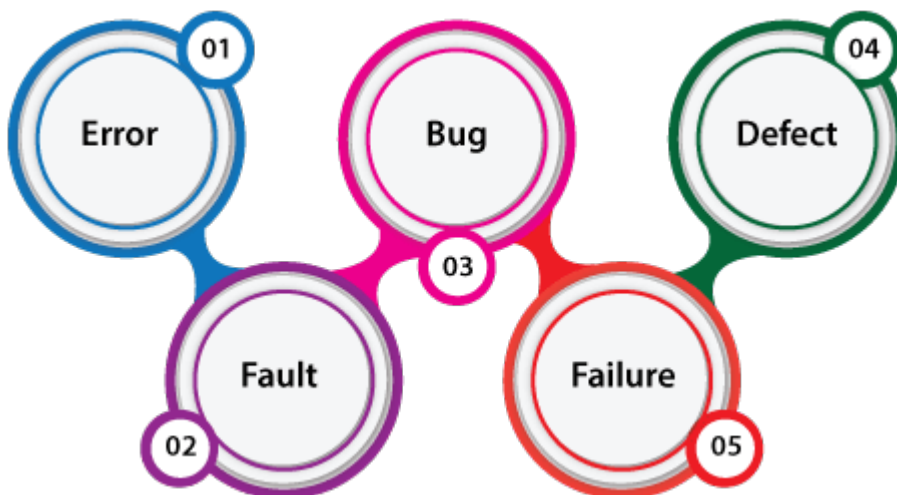
Many defects lead to the software's failure, which means that a loss specifies a fatal issue in software/ application or in its module, which makes the system unresponsive or broken.

In other words, we can say that if an end-user detects an issue in the product, then that particular issue is called a failure.

Possibilities are there one defect that might lead to one failure or several failures.

For example, in a bank application if the Amount Transfer module is not working for end-users when the end-user tries to transfer money, submit button is not working. Hence, this is a failure.

The flow of the above terminologies are shown in the following image:



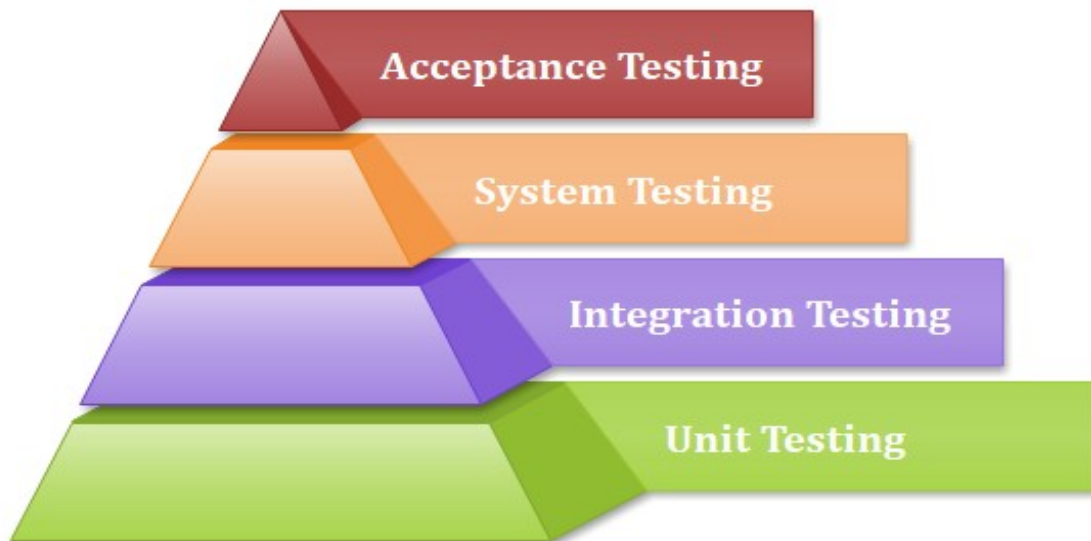
(f) Verification

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means Are we building the product right?

(g) Validation

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing. Validation means Are we building the right product?

(7) Levels of software Testing



(1) Unit testing

Unit testing involves the testing of each unit or an individual component of the software application. The aim behind unit testing is to validate unit components with its performance. A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

(2) Integration Testing.

To test whether multiple software components function well together as a group, you will need integration testing.

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

(3) System testing:

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

(4) Acceptance testing:

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

(8) Types of Testing

(a) Functional Testing

Functional testing verifies that the operational execution of a program or mobile app happens according to the technical and business requirements. Only if every feature of a software system works correctly, it can pass a functional test.

Functional testing is usually conducted before non-functional testing and is done manually. The tester provides specific inputs to the program and compares the result with the expected output. Functional testers are not concerned about the source code but focus on checking the functionality.

You can perform functional testing either by following the manual or automation testing approaches.

Eg: If you test whether a user able to login into a system or not, after registration, you are doing functional testing

(a1) Functional testing types

(1) Smoke testing.

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not. Smoke testing is a confirmation for QA team to proceed with further software testing. It

consists of a minimal set of tests run on each build to test software functionalities. Smoke testing is also known as “Build Verification Testing” or “Confidence Testing.”

For Example, a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive ... etc.

(2) Sanity Testing

Sanity Testing is a subset of regression testing. This is done prior to a release as final inspection and after applying minor patches. It normally includes a set of core tests of basic GUI functionality to demonstrate connectivity to the database, application servers, printers etc

Sanity testing helps in quickly identify defects in the core functionality. It can be carried out in lesser time as no documentation is required for sanity testing. If the defects are found during sanity testing, project is rejected that is helpful in saving time for execution of regression tests.

Example of Sanity Testing:

In an e-commerce project, main modules are login page, home page, user profile page, user registration etc.

There is a defect in the login page when the password field accepts less than four alpha numeric characters and the requirement mentions that this password field should not be below eight characters.

Hence, the defect is reported by the testing team to the development team to resolve it.

Then the development team fixes the reported defect and sends it to the testing team for

clearance. Then the testing team checks whether the changes done are working fine or not. It is also determined if it does have an impact on other related functionalities. Now there is a functionality to update the password in the user profile page. As part of the sanity testing, login page is validated as well as the profile page to ensure that the checks are working fine at both the places.

Features of Sanity Testing:

-Subset of Regression Testing:

Sanity testing is a subset of regression testing and focuses on the smaller section of the application.

-Unscripted:

Most of the times sanity testing is not scripted.

Not documented:

Usually sanity testing is undocumented.

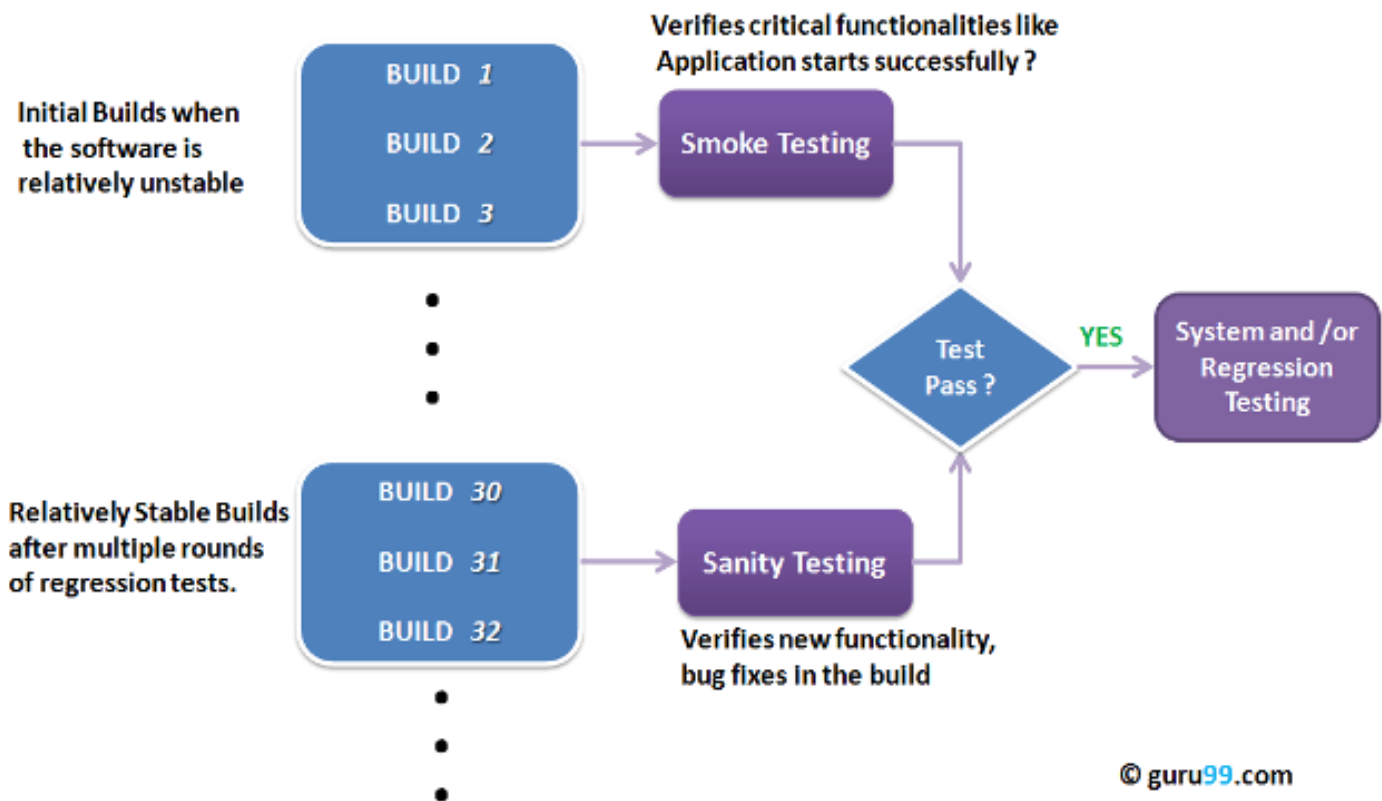
Narrow and deep:

Sanity testing is narrow and deep approach of testing where limited functionalities are covered deeply.

Performed by testers:

Sanity testing is normally performed by testers.

Advantages of Sanity Testing:



(3) Regression Testing

Regression Testing is a type of software testing executed to check whether a code change has not unfavorably disturbed current features & functions of an Application. In other words, regression testing means re-executing test cases that have been cleared in the past against the new version to ensure that the app's functionalities are working correctly. Moreover, regression testing is a series of tests and not a single test performed whenever you add a new code.

Regression Testing Example

Let's understand why you need regression testing with the help of a hypothetical example.

Suppose you're a software development firm, and the client has assigned you the project to develop an application for image and video editing. With the primary requirements, you create the first build with core features. Once done, you conduct regression testing with 1000 test cases to ensure that the app works correctly. After the app passes the regression test, you send the copy to the client for feedback. The client is happy with the first build but wants some extra features. So, you develop those features and add them to the existing app. But, with the addition of new codes, you will be required to conduct regression testing again. Hence, you write 100 new test cases and verify the functionality of the app. But along with that, you will have to run the 1000 old test cases already conducted to ensure essential functions haven't been affected. This is how a typical regression testing scenario works.

(5) Retesting

Retesting is a process to check specific failed test cases that are found with bug/s in the final execution.

Generally, testers find these bugs while testing the software application and assign it to the developers to fix it.

Then the developers fix the bug/s and assign it back to the testers for verification. This continuous process is called Retesting.

Where retesting differs from regression testing is that, instead of being designed to search through all the previous updates and features of the software to find unforeseen defects and bugs, retesting is designed to test specific defects that you've already detected (typically during your regression testing).

(6) Mutation Testing

Mutation Testing is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to find errors in source code.

The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source coding/Error Seeding.

(7) Adhoc Testing :

This testing requires no documentation or any specific procedure to be followed. Since this testing targets at discovering defects via a random approach, except any documentation, defects will no longer be mapped to check cases. You can use this testing to randomly test any part of the application.

(8) Exploratory Testing :

It is a kind of software testing that targets to optimize and enhance the software program in every viable way. In this kind of testing the tester is free to choose any feasible methodology to check the software. It is an unscripted strategy for software program testing. In exploratory testing, software program developers use their private learning, knowledge, abilities, and competencies to check the software program developed by means of themselves.

(9)Gorilla Testing

This is used to test the user interface of the software package. One can apply this technique to test the GUI, navigation sequences, message displays etc. If the software is given to a gorilla, it will play around the key board resulting in keys getting pressed randomly. This is the approach used in this technique.

(b) Non-Functional Testing

Apart from functional requirements that define the users activities, these requirements specify aspects like performance, response time, peak load, usability, security etc.

(1) Performance Testing

it is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application.

The first step in performance testing is determining what factors matter most to the customer, like

Response-time of critical transactions

System throughput

Peak Loads

System performance is generally assessed in terms of response times and throughput rates under differing processing and configuration conditions.

(2) Volume Testing

it is a type of Software Testing, where the software is subjected to a huge volume of data. It is also referred to as flood testing. Volume testing is done to analyze the system performance by increasing the volume of data in the database.

With the help of Volume testing, the impact on response time and system behavior can be studied when exposed to a high volume of data.

For example, testing the music site behavior when there are millions of user to download the song.

(3)Capacity Testing

Capacity Testing ensures that the application and environment can smoothly handle the maximum number of users or transactions according to the performance requirements defined in your Service-Level Agreement (SLA). Capacity Testing is aimed at testing the maximum capacity of your system in terms of traffic, while still being able to deliver optimal user experience.

(4)Scalability Testing

is a non functional testing method that measures performance of a system or network when the number of user requests are scaled up or down. The purpose of Scalability testing is to ensure that the system can handle projected increase in user traffic, data volume, transaction counts frequency, etc. It tests system ability to meet the growing needs.

(5)Security Testing

Security Testing is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information, revenue, reputation at the hands of the employees or outsiders of the Organization.

Examples

- A password should be in encrypted format
- Application or System should not allow invalid users
- Check cookies and session time for application
- For financial sites, the Browser back button should not work.

(6)Compatibility Testing

it is performed on an application to check its compatibility (running capability) on different platform/environments. This testing is done only when the application becomes stable. Means simply this compatibility test aims to check the developed software application functionality on various software, hardware platforms, network and browser etc. This compatibility testing is very important in product production and implementation point of view as it is performed to avoid future issues regarding compatibility.

(7)Usability Testing

Usability Testing also known as User Experience(UX) Testing, is a testing method for measuring how easy and user-friendly a software application is. A small set of target end-users, use software application to expose usability defects. Usability testing mainly focuses on user's ease of using application, flexibility of application to handle controls and ability of application to meet its objectives. This testing is recommended during the initial design phase of SDLC, which gives more visibility on the expectations of the users.

(8)Recovery Testing

Recovery Testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures etc. The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss. Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point.

(9) Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is the acronym of Software Development Life Cycle. It is also called as Software Development Process.

SDLC is a framework defining tasks performed at each step in the software development process.



(9a)The stages of SDLC are as follows:

(1)Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs(Small and mid-size enterprises) in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

(2)Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

(3)Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

(4)Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

(5)Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

(6)Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

(7)Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

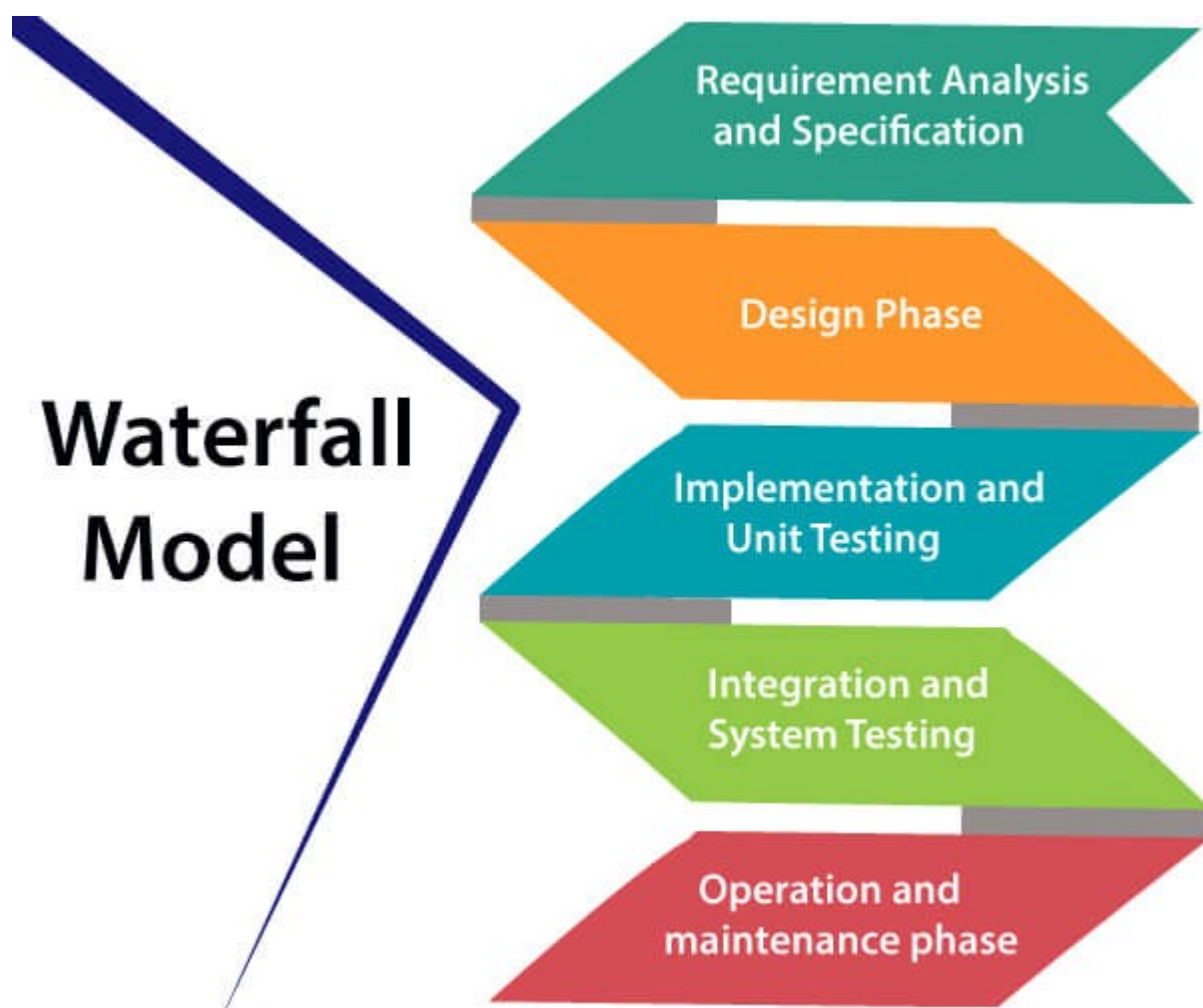
(9b) SDLC Models

(1) Sequential development models

A sequential development model describes the software development process as a linear, sequential flow of activities. This means that any phase in the development process should begin when the previous phase is complete. In theory, there is no overlap of phases, but in practice, it is beneficial to have early feedback from the following phase. following models use sequential development models.

(a) Waterfall Model

In the Waterfall model, the development activities (e.g., requirements analysis, design, coding, testing) are completed one after another. In this model, test activities only occur after all other development activities have been completed. The Waterfall model is the earliest SDLC approach that was used for software development.



When to use SDLC Waterfall Model?

Some Circumstances where the use of the Waterfall model is most suited are:

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing
- When resources are well prepared and are available to use.

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

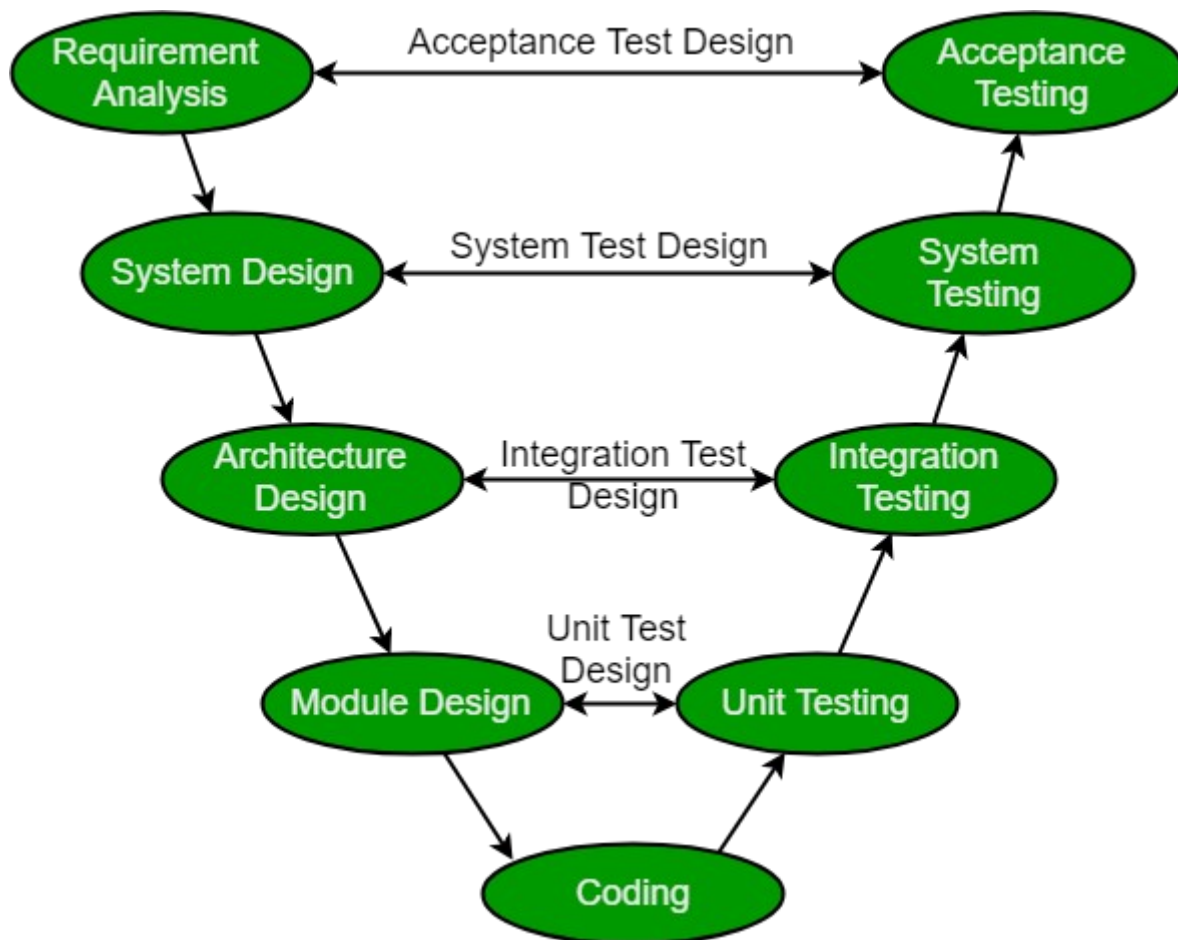
In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects. This model cannot accept the changes in requirements during development.

It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.

Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

(b)V-model

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it.



When to use?

Where requirements are clearly defined and fixed.
The V-Model is used when ample technical resources are available with technical expertise.

Advantages:

This is a highly disciplined model and Phases are completed one at a time.

V-Model is used for small projects where project requirements are clear.

Simple and easy to understand and use.

This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.

It enables project management to track progress accurately.

Disadvantages:

High risk and uncertainty.

It is not a good for complex and object-oriented projects.

It is not suitable for projects where requirements are not clear and contains high risk of changing.

This model does not support iteration of phases.

It does not easily handle concurrent events.

(2) Iterative and incremental development models.

Iterative and incremental development is a process that combines the iterative design method with the incremental build model. It is used by software developers to help manage projects.

To fully understand the incremental and iterative development process, you must first split it into its two parts:

-incremental: An incremental approach breaks the software development process down into small, manageable portions known as increments. Each increment builds on the previous version so that improvements are made step by step.

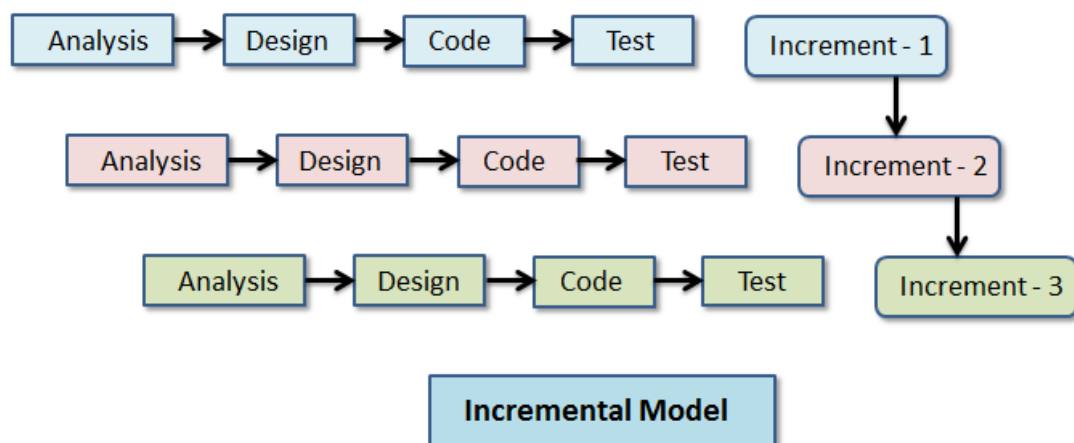
in simple words

Incremental development is a development approach that slices the product into fully working slices that are called increments. Each new increment builds on top of the existing released functionality.

Example: Ecommerce website

Consider a team building an ecommerce website using incremental development. The final target product has search, product information, a shopping basket, checkout, favourites, and customer reviews.

For the first released increment, the team builds the basic functionality to buy a product. It includes search, product information, adding products to a shopping basket and checkout. This first slice would only be released once it's complete. The second released increment builds on that basic functionality, and would add another capability such as favourites. The would be released when the favourites functionality is complete. The third released increment adds customer reviews once that is complete, and so on.



Iterative: An iterative model means software development activities are systematically repeated in cycles known as iterations. A new version of the software is produced after each iteration until the optimal product is achieved.

Iterative development is when teams gradually build up the features and functions but don't wait until each of these is complete before releasing. They release a basic version of each feature and then add to that feature in subsequent iterative releases, usually based on feedback from the basic version released.

Example: Ecommerce website

Assume a team building the same ecommerce website using an iterative process. The first release has a really stripped back version of all the required functionality; namely search, product information, a shopping basket, checkout, favourites, and customer reviews. For the second iterative release, the team would improve some of the existing basic functionality, taking into account feedback from stakeholders or customer, or other inputs such as analytics.

Iterative and Incremental Development in Agile

The incremental and iterative development process is closely associated with Agile project management, most notably the Scrum methodology. This is because it aligns with one of the key pillars of Agile: responding to change over following a set plan.

Rather than adhering to a linear Waterfall method, software developers will react quickly to changes as their product evolves. They will build on previous versions to improve their product and repeat this process until the desired deliverables are achieved.

An example of iterative and incremental development in Agile could be the creation of a new e-commerce website. The project would be broken down into smaller increments, such as building a wireframe, uploading products, and creating advertising copy. As these steps are unfolding, the software development team would repeat the cycles of prototyping and testing to make improvements to the website with each iteration.

Why is iterative and incremental development important?

The incremental and iterative development process is integral to the field of Agile software development as it enables project managers to reap the benefits of both incremental and iterative approaches.

Incremental development ensures that developers can make changes early on in the process rather than waiting until the end when the allotted time has run out and the money has been spent.

Iterative development means improvements are made on an ongoing basis, so the end result is likely to be delivered on time and be of higher quality. This belief is echoed by CIO.com, which notes that short, iterative sprints can help teams to “deliver a better product, in a faster manner.”

What is Agile project management?

Agile project management is an iterative approach to project management that focuses on breaking down large projects into more manageable tasks, which are completed in short iterations throughout the project life cycle. Teams that adopt the Agile methodology are able to complete work faster, adapt to changing project requirements, and optimize their workflow.

As the name suggests, the Agile allows teams to be better equipped to quickly change direction and focus. Software companies and marketing agencies are especially aware of the tendency for changes from project stakeholders to happen from week-to-week. The Agile methodology allows teams to re-evaluate the work they are doing and adjust in given increments to make sure that as the work and customer landscape changes, the focus also changes for the team.

The 4 Core Values of Agile are:

1. Individuals and interactions over processes and tools.

As sophisticated as technology gets, the human element will always serve as an important role in any kind of project management. Relying too heavily on processes and tools results in an inability to adapt to changing circumstances.

2. Working software over comprehensive documentation.

As important as documentation is, working software is more important. This value is all about giving the developers exactly what they need to get the job done, without overloading them.

3. Customer collaboration over contract negotiation

Your customers are one of your most powerful assets. Whether internal or external customers, involving them throughout the process can help to ensure that the end product meets their needs more effectively.

4. Responding to change over following a plan.

This value is one of the biggest departures from traditional project management. Historically, change was seen as an expense, and one to be avoided. Agile allows for continuous change throughout the life of any given project. Each sprint provides an opportunity for review and course correction.

few methods that follows Iterative and Incremental Development Models.

Rational Unified Process:

Each iteration tends to be relatively long (e.g., two to three months), and the feature increments are correspondingly large, such as two or three groups of related features.

Scrum:

Scrum is a project management framework that is applicable to any project with aggressive deadlines, complex requirements and a degree of uniqueness. In Scrum, projects move forward via a series of iterations called sprints. Each sprint is typically two to four weeks long.

Each iteration tends to be relatively short (e.g., hours, days, or a few weeks), and the feature increments are correspondingly small, such as a few enhancements and/or two or three new features.

Kanban: Implemented with or without fixed-length iterations, which can deliver either a single enhancement or feature upon completion, or can group features together to release at once.

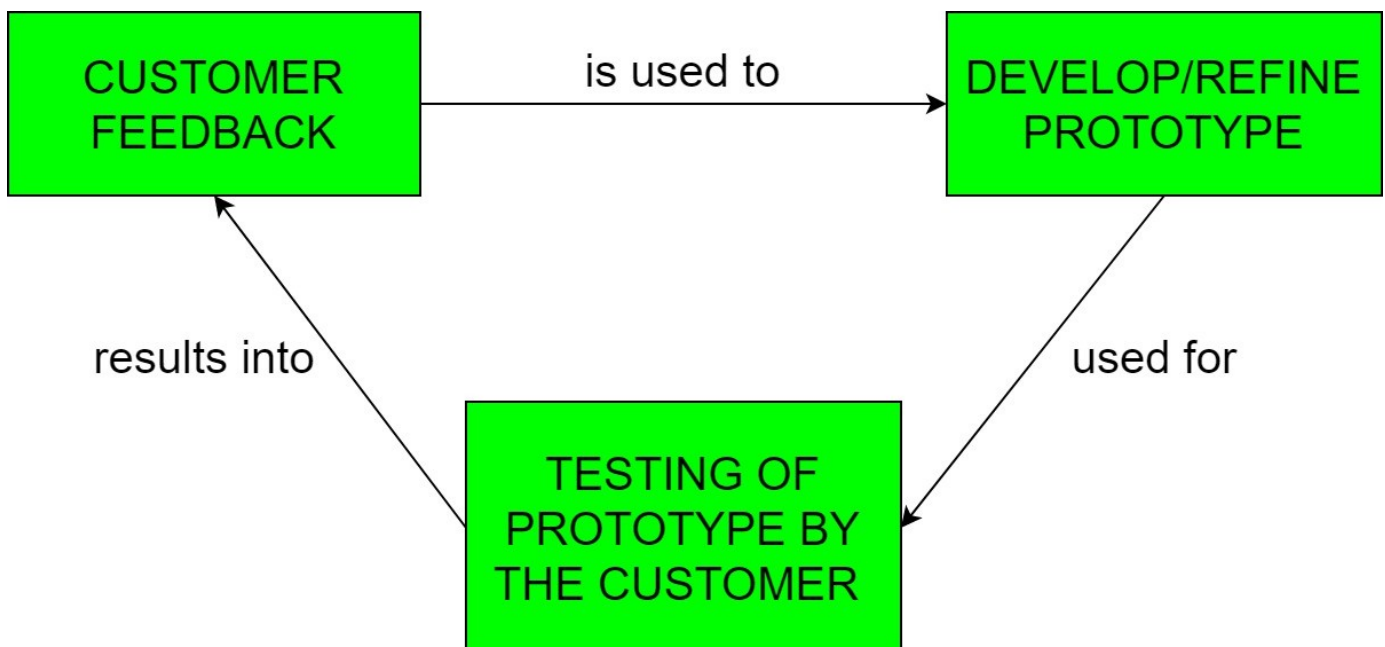
Spiral:

Involves creating experimental increments, some of which may be heavily re-worked or even abandoned in subsequent development work.

(3)The Prototyping Model

Prototyping Model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory



(11) Manual testing process

What is Manual Testing

Manual Testing is a process of finding out the defects or bugs in a software program. In this method the tester plays an important role of end user and verifies that all the features of the application are working correctly.

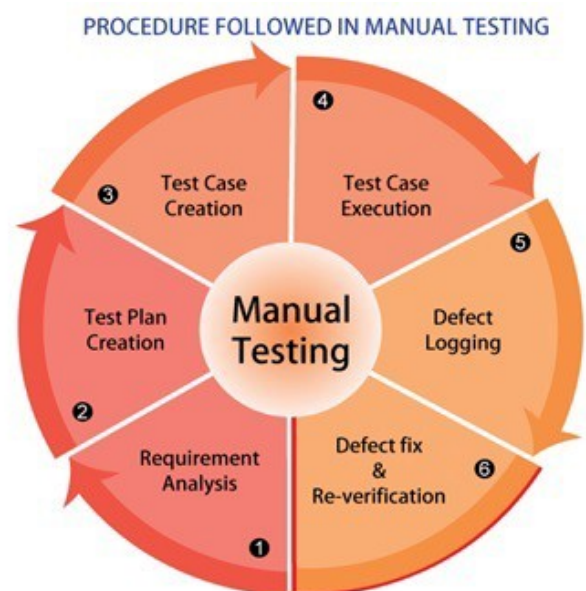
The tester manually executes test cases without using any automation tools.

The tester prepares a test plan document which describes the detailed and systematic approach to testing of software applications. Test cases are planned to cover almost 100% of the software application. As manual testing involves complete test cases it is a time consuming test.

The differences between actual and desired results are treated as defects. The defects are then fixed by the developer of software application. The tester retests the defects to ensure that defects are fixed. The goal of Manual testing is to ensure that application is defect & error free and is working fine to provide good quality work to customers.

Procedure of Manual Testing

- Requirement Analysis
- Test Plan Creation
- Test case Creation
- Test case Execution
- Defect Logging
- Defect Fix & Re-Verification



(1)Requirement Analysis

Some major task of requirement analysis:

To review the test basis. The test basis is the information on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces

To identify test conditions

To design the tests

To design the test environment set-up and identify the required infrastructure and tools

(2)Test Plan Creation

Test planning involves producing a document that describes an overall approach and test objectives. It involves reviewing the test basis, identifying the test conditions based on analysis of test items, writing test cases and Designing the test environment. Completion or exit criteria must be specified so that we know when testing (at any stage) is complete.

To determine the scope and risks and identify the objectives of testing.

To determine the required test resources like people, test environments etc.

To schedule test analysis and design tasks, test implementation, execution and evaluation.

(3)Test case Creation

To develop and prioritize test cases by using techniques and create test data for those tests.

(4)Test case Execution

Test execution involves actually running the specified test on a computer system either manually or by using an automated test tool.It is a Fundamental Test Process in which actual work is done.

Test implementation has the following major task:

To create test suites from the test cases for efficient test execution.Test suite is a collection of test cases that are used to test a software program

To re-execute the tests that previously failed in order to confirm a fix.

To log the outcome of the test execution. A test log is the status of the test case (pass/fail).

How to do Manual Testing

Here's how to perform manual testing step by step:

- Analyze requirements from the software requirement specification document
- Create a clear test plan
- Write test cases that cover all the requirements defined in the document
- Get test cases reviewed by the QA lead
- Execute test cases and detect any bugs
- Report bugs, if any, and once fixed, run the failed tests again to re-verify the fixes

Manual testing can never be avoided entirely as it is a continuous process that requires human verification at regular intervals throughout the software development lifecycle. As a result, it's essential for teams to find the right balance between manual and automated tests.

Test Plan

Test plan can be defined as a document for a software project which defines the approach, scope, and intensity on the effort of software testing.

Test Strategy

The test strategy is a set of instructions or protocols which explain the test design and determine how the test should be performed.

The objective of the Test Strategy is to provide a systematic approach to the software testing process in order to ensure the quality, traceability, reliability and better planning.

Test Plan has the primary goal of how to test, when to test and who will verify whereas Test Strategy has the primary goal of what technique to follow and which module to check

Requirements Traceability Matrix (RTM)

Requirements Traceability Matrix (RTM) is used to trace the requirements to the tests that are needed to verify whether the requirements are fulfilled.

The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

Advantage of Requirements Traceability Matrix (RTM):

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

How to prepare Requirement Traceability Matrix (RTM):

- Collect all the available requirement documents.
- Allot an unique Requirement ID for each and every Requirement
- Create Test Cases for each and every requirement and link Test Case IDs to the respective Requirement ID.

Test Report

Test Execution Report or Test Summary Report is a document which contains a summary of all test activities and final test results of a testing project.

Test report is an assessment of how well the Testing is performed. Based on the test report, stakeholders can evaluate the quality of the tested product and make a decision on the software release.

Sample RTM

Business Requirement ID	Business Requirement	Test Case ID	Test Case Name
BR 01	System should allow user to login using a valid username and password	TC 01	Login functionality with valid user name and password
BR 02	System should accept only username with minimum 6 characters and maximum 10 characters	TC 02	Login functionality with username of 5 characters
		TC 03	Login functionality with username of 11 characters
		TC 04	Login functionality with username of 7 characters , but also a special character
BR 03	System should accept only username with minimum 8 characters and maximum 10 characters, with atleast one number	TC 05	Login functionality with password of 7 characters
		TC 06	Login functionality with password of 11 characters
		TC 07	Login functionality with password of 9 characters, but without a number

What does a test report contain?

Project Information	Test Objective	Test Summary	Defect
<ul style="list-style-type: none">• Project Name• Description	<ul style="list-style-type: none">• Test Type• Purpose	<ul style="list-style-type: none">• Test Passed• Test Failed• Test Blocked	<ul style="list-style-type: none">• Description• Priority• Status

Test Report

Test Cycle **System Test**

EXECUTED	PASSED			130
	FAILED			0
	(Total) TESTS EXECUTED (PASSED + FAILED)			130
PENDING				0
IN PROGRESS				0
BLOCKED				0
(Sub-Total) TEST PLANNED				130
(PENDING + IN PROGRESS + BLOCKED + TEST EXECUTED)				

Functions	Description	% TCs Executed	% TCs Passed	TCs pending	Priority	Remarks
New Customer	Check new Customer is created	100%	100%	0	High	
Edit Customer	Check Customer can be edited	100%	100%	0	High	
New Account	Check New account is added	100%	100%	0	High	
Edit Account	Check Account is edit	100%	100%	0	High	
Delete Account	Verify Account is delete	100%	100%	0	High	
Delete customer	Verify Customer is Deleted	100%	100%	0	High	
Mini Statement	Verify Ministatement is generated	100%	100%	0	High	
Customized Statement	Check Customized Statement is generated	100%	100%	0	High	

Defect Report

A Defect Report in Software Testing is a detailed document about defects found in the software application.

Defect report contains each detail about Defects like description, date when defect was found, name of tester who found it, name of developer who fixed it, etc. Defect report helps to identify similar defects in future so it can be avoided. While reporting the defect to developer, your Defect Report should contain the following information (This is not a complete list, but only a sample)

- Defect_ID - Unique identification number for the defect.
- Defect Description - Detailed description of the Defect including information about the module in which Defect was found.
- Version - Version of the application in which defect was found.
- Steps - Detailed steps along with screenshots with which the developer can reproduce the defects.
- Date Raised - Date when the defect is raised
- Reference- where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect
- Detected By - Name/ID of the tester who raised the defect
- Status - Status of the defect , more on this later
- Fixed by - Name/ID of the developer who fixed it
- Date Closed - Date when the defect is closed
- Severity which describes the impact of the defect on the application
- Priority which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively

Defect Log

A defect Log is the primary means by which defects are tracked to closure/resolution before the product is released to the customer.

What is Defect Life Cycle?

Defect Life Cycle or Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life. The purpose of Defect life cycle is to easily coordinate and communicate current status of defect which changes to various assignees and make the defect fixing process systematic and efficient.

Defect Status

Defect Status or Bug Status in defect life cycle is the present state from which the defect or a bug is currently undergoing. The goal of defect status is to precisely convey the current state or progress of a defect or bug in order to better track and understand the actual progress of the defect life cycle.

The number of states that a defect goes through varies from project to project. Below lifecycle diagram, covers all possible states

New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.

Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team

Open: The developer starts analyzing and works on the defect fix

Fixed: When a developer makes a necessary code change and verifies the change, he or she can make bug status as “Fixed.”

Pending retest: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is “pending retest.”

Retest: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to “Re-test.”

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

Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to “reopened”. Once again the bug goes through the life cycle.

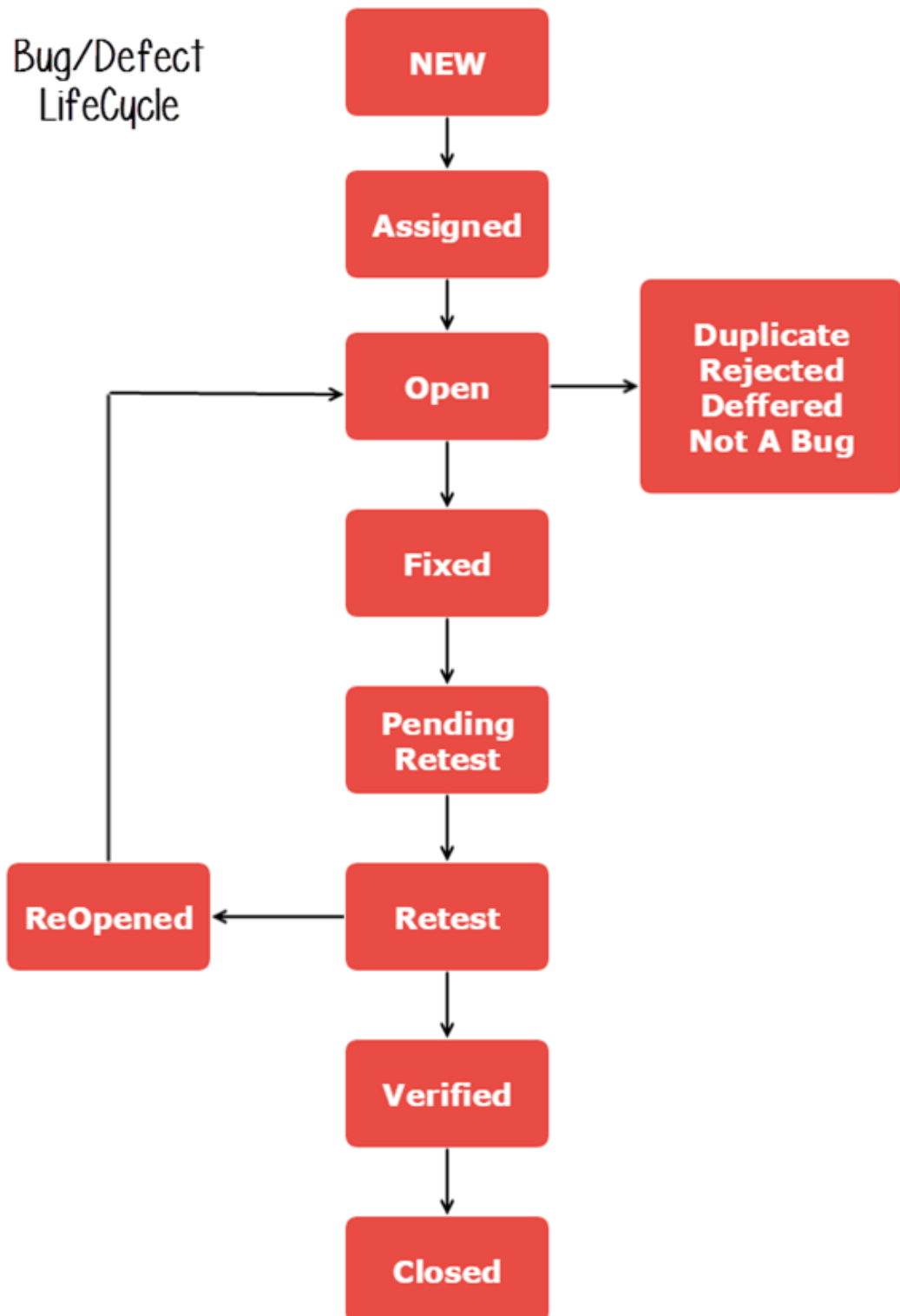
Closed: If the bug is no longer exists then tester assigns the status “Closed.”

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

Rejected: If the developer feels the defect is not a genuine defect then it changes the defect to “rejected.”

Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status “Deferred” is assigned to such bugs

Not a bug: If it does not affect the functionality of the application then the status assigned to a bug is “Not a bug”.



Other terminologies and QA metrics

What is Defect Density?

Defect Density is the number of defects confirmed in software/module during a specific period of operation or a development divided by the size of the software/module. It enables one to decide if a piece of software is ready to be released.

Defect density is counted per thousand lines of code also known as KLOC.

A formula to measure Defect Density:

Defect Density = Defect count/size of the release

Size of release can be measured in terms of a line of code (LoC).

Defect Density Example: Suppose, you have 3 modules integrated into your software product. Each module has the following number of bugs discovered-

Module 1 = 10 bugs

Module 2 = 20 bugs

Module 3 = 10 bugs

Total bugs = $10+20+10=40$

The total line of code for each module is

Module 1 = 1000 LOC

Module 2 = 1500 LOC

Module 3 = 500 LOC

Total Line of Code = $1000+1500+500 = 3000$

Defect Density is calculated as:

Defect Density = $40/3000 = 0.013333$ defects/loc = 13.333 defects/Kloc

Test Effort

Metrics measuring test effort will answer the following questions: “how many and how long?” with regard to tests. They help to set baselines, which the final test results will be compared to.

Some of these QA metrics examples are:

Number of tests in a certain time period = Number of tests run/Total time

Test design efficiency = Number of tests designed/Total time

Test review efficiency = Number of tests reviewed/Total time

Number of bugs per test = Total number of defects/Total number of tests

Test Effectiveness

Use this metric to answer the questions – “How successful are the tests?”, “Are testers running high-value test cases?” In other words, it measures the ability of a test case to detect bugs AKA the quality of the test set. This metric is represented as a percentage of the difference between the number of bugs detected by a certain test, and the total number of bugs found for that website or app.

$$(\text{Bugs detected in 1 test} / \text{Total number of bugs found in tests} + \text{after release}) \times 100$$

The higher the percentage, the better the test effectiveness. Consequently, the lower the test case maintenance effort required in the long-term.

Test Coverage

Test Coverage measures how much an application has been put through testing. Some key test coverage examples are:

Test Coverage Percentage = $(\text{Number of tests runs} / \text{Number of tests to be run}) \times 100$

Requirements Coverage = $(\text{Number of requirements coverage} / \text{Total number of requirements}) \times 100$

Test Economy

The cost of testing comprises manpower, infrastructure, and tools. Unless a testing team has infinite resources, they have to meticulously plan how much to spend and track how much they actually spend. Some of the QA performance metrics below can help with this:

Total Allocated Cost: The amount approved by QA Directors for testing activities and resources for a certain project or period of time.

Actual Cost: The actual amount used for testing. Calculate this on the basis of cost per requirement, per test case or per hour of testing.

Budget Variance: The difference between the Allocated Cost and Actual Cost

Time Variance: The difference between the actual time taken to finish testing and planned time.

Cost Per Bug Fix: The amount spent on a defect per developer.

Cost of Not Testing: Say, a set of new features that went into prod need to be reworked, then the cost of the reworking activities is basically, the cost of not testing.

Test Team

These metrics denote if work is being allocated uniformly for each team member. They can also cast light on any incidental requirements that individual team members may have.

Important Test Team metrics include:

The number of defects returned per team member

The number of open bugs to be retested by each team member

The number of test cases allocated to each team member

The number of test cases executed by each team member

Defect Distribution

Software quality assurance metrics must also be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters.

Some useful defect distribution metrics would be:

Defect distribution by cause

Defect distribution by feature/functional area

Defect distribution by Severity

Defect distribution by Priority

Defect distribution by type

Defect distribution by tester (or tester type) – Dev, QA, End-user.

Types of manual Testing

(1)White box Testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

WhiteBox Testing Example

Consider the following piece of code

```
Printme (int a, int b) {          ----- Printme is a function
    int result = a+ b;
    If (result> 0)
        Print ("Positive", result)
    Else
        Print ("Negative", result)
}                                ----- End of the source code
```

The goal of WhiteBox testing in software engineering is to verify all the decision branches, loops, statements in the code.

To exercise the statements in the above white box testing example, WhiteBox test cases would be

A = 1, B = 1

A = -1, B = -3

White Box Testing Techniques

(a) Statement Testing and Coverage

Statement testing exercises the potential executable statements in the code.

This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

- Coverage is measured as the number of statements executed by the tests divided by the total number of Executable statements in the test object, normally expressed as a percentage.
 - Code is executed in such a manner that every statement of the application is executed at least once
- 100% statement coverage doesn't guarantee 100% branch or path coverage

(b) Branch Testing and Coverage

Decision testing exercises the decisions(eg : if and else conditions) in the code and tests the code that is executed based on the decision outcomes. To do this, the test cases follow the control flows that occur from a decision point (e.g., for an IF statement, one for the true outcome and one for the false outcome; for a CASE statement, test cases would be required for all the possible outcomes, including the default outcome).

Each branch should be executed at least once for 100% branch coverage

100% branch coverage guarantees 100% statement coverage.

100% branch coverage doesn't guarantee 100% path coverage

(c) Path Testing and Coverage

Each path has to be executed at least once for 100% path coverage. Execute all possible control flow paths through the program. 100% path coverage guarantees 100% statement and branch coverages.

(2) Black Box Testing

Black-box testing is a test approach in which the QA doesn't have any knowledge about the underlying code or structure of the application. The QA interacts with the software application just like an end-user to test its functional and non-functional behavior. This helps to discover some bugs typically overlooked in the earlier stages.

Black Box Testing Techniques

(a) Equivalence Partitioning

Equivalence partitioning divides data into partitions (also known as equivalence classes) in such a way that all the members of a given partition are expected to be processed in the same way. There are equivalence partitions for both valid and invalid values.

Valid values are values that should be accepted by the component or system. An equivalence partition containing valid values is called a "valid equivalence partition."

Invalid values are values that should be rejected by the component or system. An equivalence partition containing invalid values is called an "invalid equivalence partition."

When invalid equivalence partitions are used in test cases, they should be tested individually, i.e., not combined with other invalid equivalence partitions, to ensure that failures are not masked. Failures can be masked when several failures occur at the same time but only one is visible, causing the other failures to be undetected.

To achieve 100% coverage with this technique, test cases must cover all identified partitions (including invalid partitions) by using a minimum of one value from each partition.

If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.

If a specific value is given as input, then one valid and two invalid equivalence classes are defined.

If a member of set is given as an input, then one valid and one invalid equivalence class is defined.

If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.

Example

Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage.

Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page.

If percentage entered by user is less than 50 % or more than 90 %, that equivalence partitioning method will show an invalid percentage. If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage.

Percentage

* Accepts Percentage value between 50 to 90

Equivalence Partitioning		
Invalid	Valid	Invalid
≤ 50	50-90	≥ 90

(b) Boundary Value Analysis

Boundary value analysis (BVA) is an extension of equivalence partitioning, but can only be used when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values (or first and last values) of a partition are its boundary values. Behavior at the boundaries of equivalence partitions is more likely to be incorrect than behavior within the partitions.

BVA is also called Range Checking, in this technique, you test boundaries between equivalence partitions.

Boundary value analysis can be applied at all test levels. This technique is generally used to test requirements that call for a range of numbers (including dates and times)

2 way BVA – Boundary Value and value that is just over the boundary by the smallest possible increment are used

3 way BVA – Values before, on and over the boundary are used.

(c) Cause Effect Graph or Decision Table Testing

Cause Effect Graphing based technique is a technique in which a graph is used to represent the situations of combinations of input conditions. The graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions. But since there may be some critical behaviour to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

Steps used in deriving test cases using this technique

(1) Division of specification:

Since it is difficult to work with cause-effect graphs of large specifications as they are complex, the specifications are divided into small workable pieces and then converted into cause-effect graphs separately.

(2) Identification of cause and effects:

This involves identifying the causes(distinct input conditions) and effects(output conditions) in the specification.

Transforming the specifications into a cause-effect graph:

The causes and effects are linked together using Boolean expressions to obtain a cause-effect graph. Constraints are also added between causes and effects if possible.

(3) Conversion into decision table:

The cause-effect graph is then converted into a limited entry decision table. If you're not aware of the concept of decision tables, check out this link.

(4) Deriving test cases:

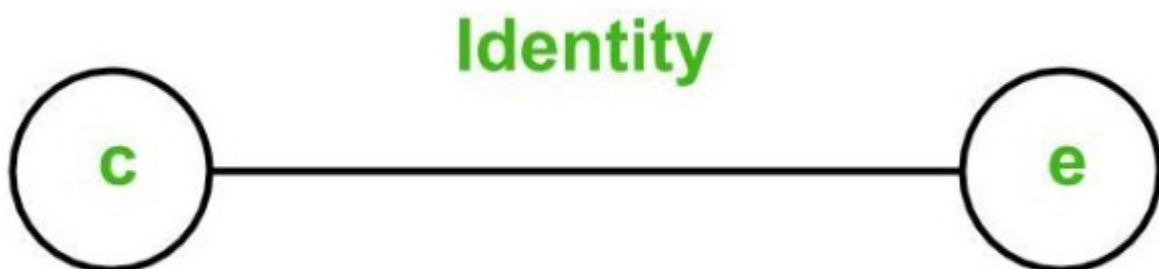
Each column of the decision-table is converted into a test case.

Basic Notations used in Cause-effect graph:

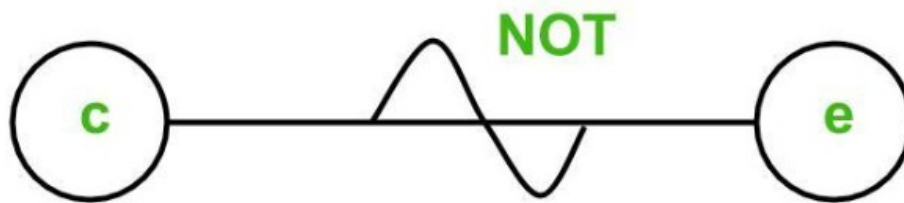
Here c represents cause and e represents effect.

The following notations are always used between a cause and an effect:

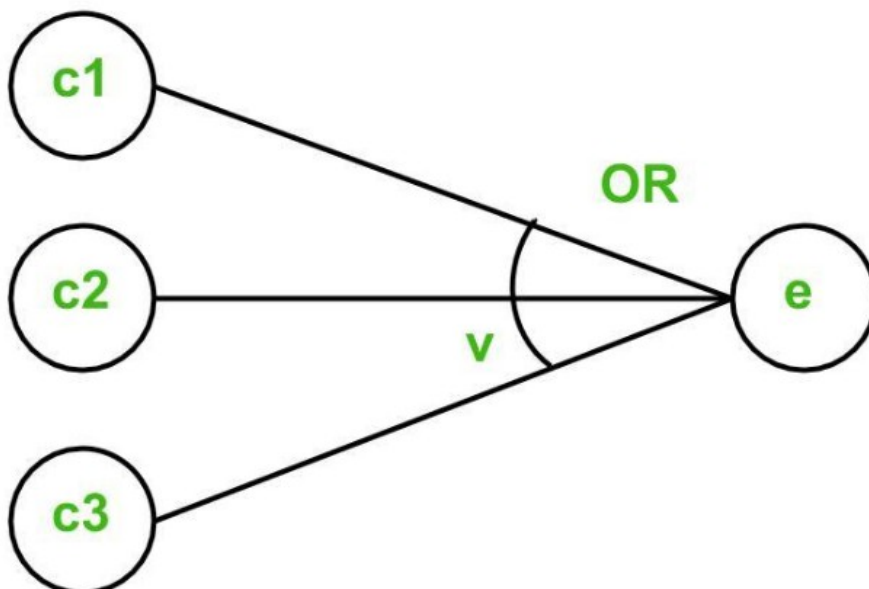
Identity Function: if c is 1, then e is 1. Else e is 0.



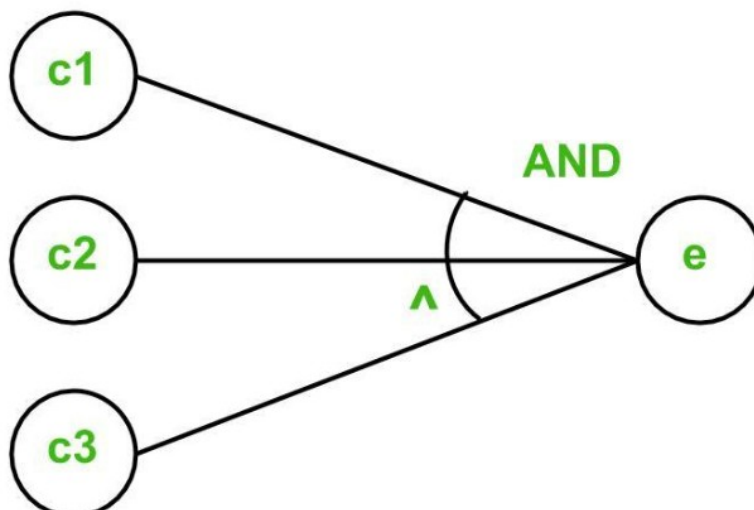
NOT Function: if c is 1, then e is 0. Else e is 1.



OR Function: if c1 or c2 or c3 is 1, then e is 1. Else e is 0

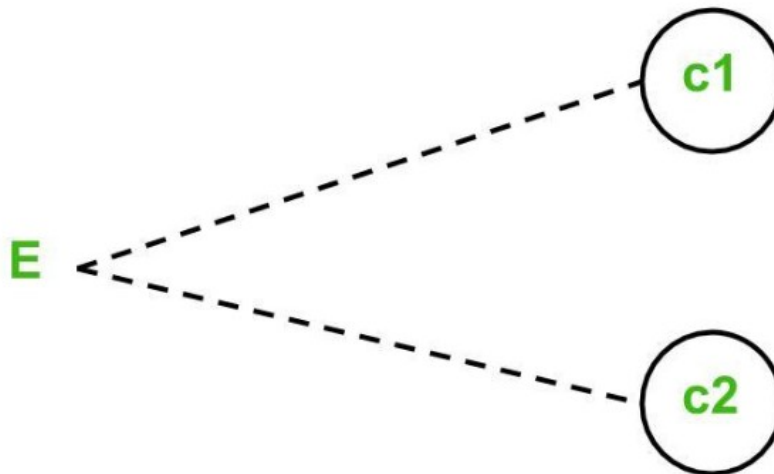


AND Function: if both c1 and c2 and c3 is 1, then e is 1. Else e is 0

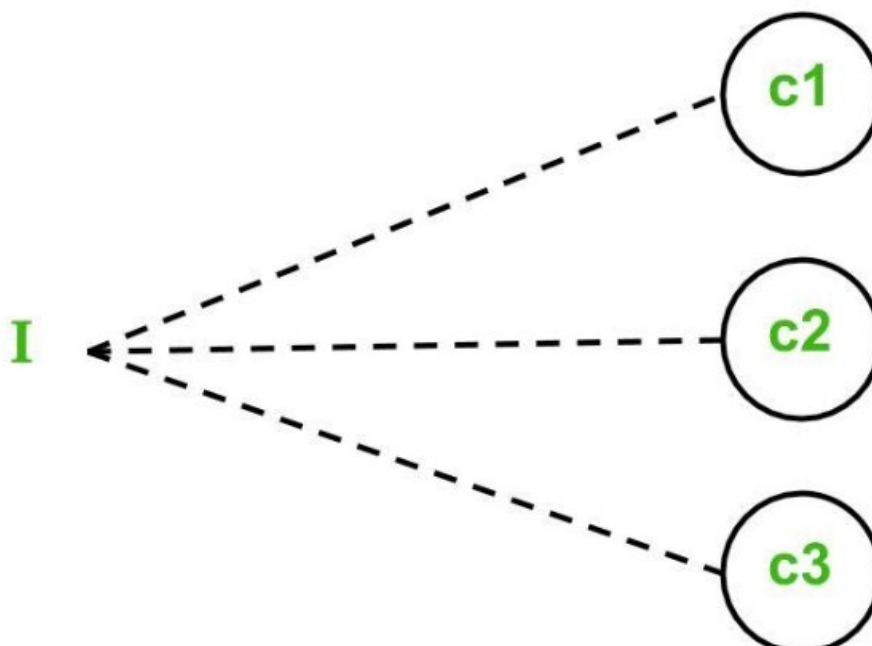


To represent some impossible combinations of causes or impossible combinations of effects, constraints are used. The following constraints are used in cause-effect graphs

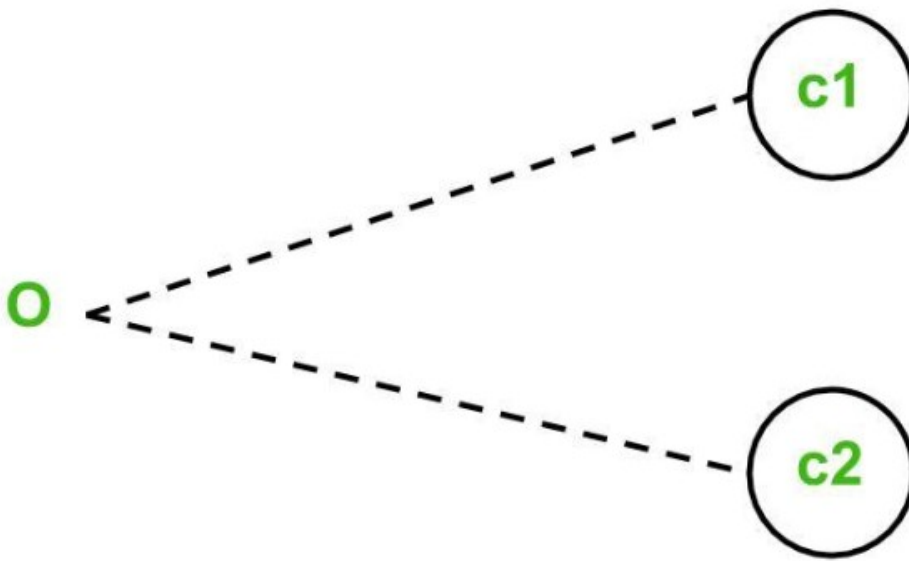
Exclusive constraint or E-constraint: This constraint exists between causes. It states that either c1 or c2 can be 1, i.e., c1 and c2 cannot be 1 simultaneously.



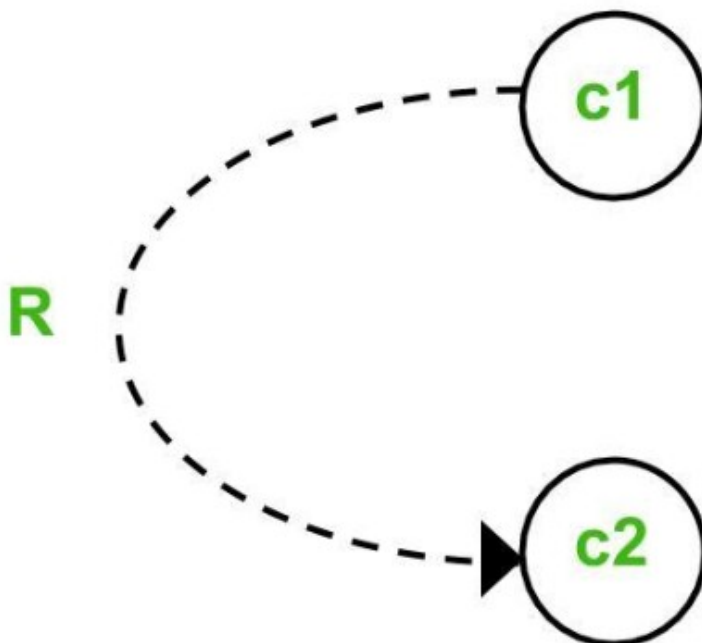
Inclusive constraint or I-constraint: This constraint exists between causes. It states that at least one of c1, c2 and c3 must always be 1, i.e., c1, c2 and c3 cannot be 0 simultaneously.



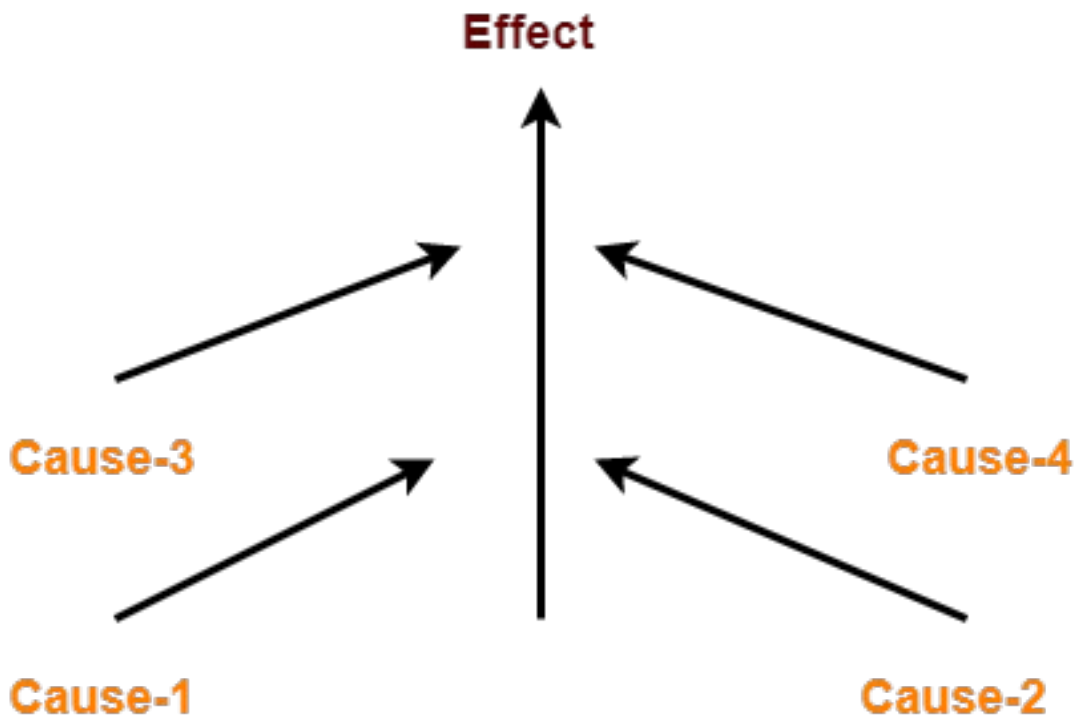
One and Only One constraint or O-constraint: This constraint exists between causes. It states that one and only one of c1 and c2 must be 1.



Requires constraint or R-constraint: This constraint exists between causes. It states that for c1 to be 1, c2 must be 1. It is impossible for c1 to be 1 and c2 to be 0.



Cause Effect Graph is a popular black box testing technique. It illustrates the relationship between a given outcome and all the factors that influence the outcome graphically.



Cause-Effect Flow Diagram

A “Cause” stands for a distinct input condition that fetches about an internal change in the system.

An “Effect” represents an output condition, a system state that results from a combination of causes.

Applications-

For analyzing the existing problem so that corrective actions can be taken at the earliest.

For relating the interactions of the system with the factors affecting a particular process.

For identifying the possible root causes, reasons for a particular effect, problem or outcome.

Advantages

It helps to determine the root causes of a problem or quality.

It indicates possible causes of variation in a process.

It identifies those areas where data should be collected for further study.

It utilizes the team knowledge of the process by encouraging team participation.

Steps For Drawing Cause Effect Diagram-

The following steps are followed-

Identify and describe the input conditions (causes) and actions (effect).

Build up a cause-effect graph.

Convert cause-effect graph into a decision table.

Convert decision table rules to test cases where each column of the decision table represents a test case.

PRACTICE PROBLEMS BASED ON CAUSE-EFFECT GRAPH TECHNIQUE.

Design test cases for the following problem

If the character of the first column is 'A' or 'B' and the second column is a number, then the file is considered updated. If the first character is erroneous, then message x should be printed. If the second column is not a number, then message y should be printed.

Step-01:

Identify and describe the input conditions (causes) and actions (effect).

The causes represented by letter "C" are as follows-

C1 : The character in column 1 is 'A'

C2 : The character in column 1 is 'B'

C3 : The character in column 2 is a number

The effects represented by letter "e" are as follows-

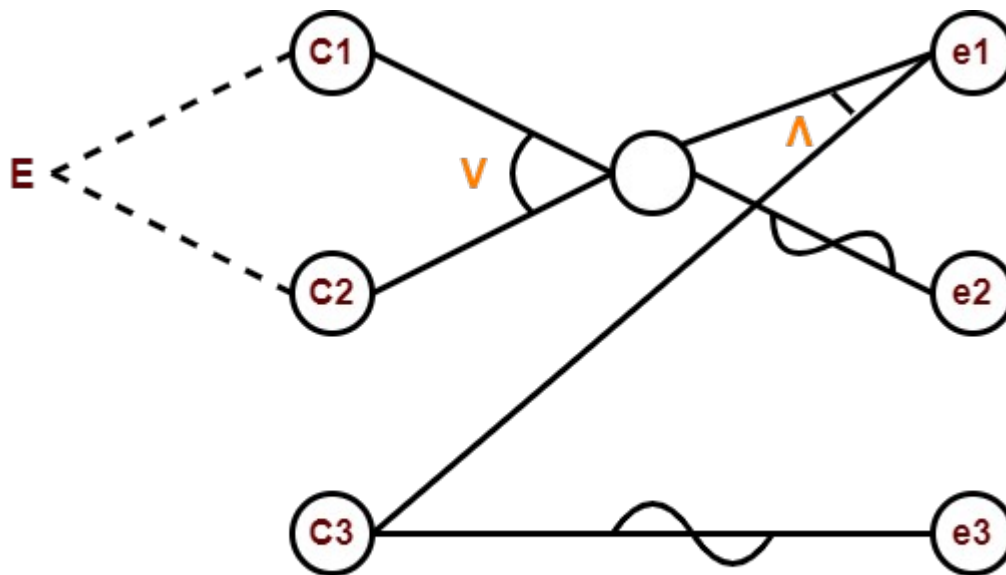
e1 : File update is made

e2 : Message x is printed

e3 : Message y is printed

Step-02:

Build up a cause-effect graph-



Cause Effect Graph

Step-03:

Convert cause-effect graph into a decision table-

Test data	Causes			Effect		
	A1	A2	A3	M1	M2	M3
1	0	0	0	0	1	1
2	0	0	1	0	1	0
3	0	1	0	0	0	1
4	0	1	1	1	0	0
5	1	0	0	0	0	1
6	1	0	1	1	0	0

Why Cause Effect Graphing Technique is Better Than Any Other Black Box Testing Technique ?

Boundary value analysis and equivalence partitioning do not explore combinations of input circumstances.

These only consider the single input conditions.

However, combinations of inputs may result in interesting situations.

These situations should be tested.

By considering all the valid combinations of equivalence classes, there will be large number of test cases.

Many of these test cases will not be useful for revealing any new errors. On the other hand, Cause Effect Graph is a technique that helps in selecting a high-yield set of test cases in a systematic way. It has a beneficial effect in pointing out incompleteness and ambiguities in the specifications.

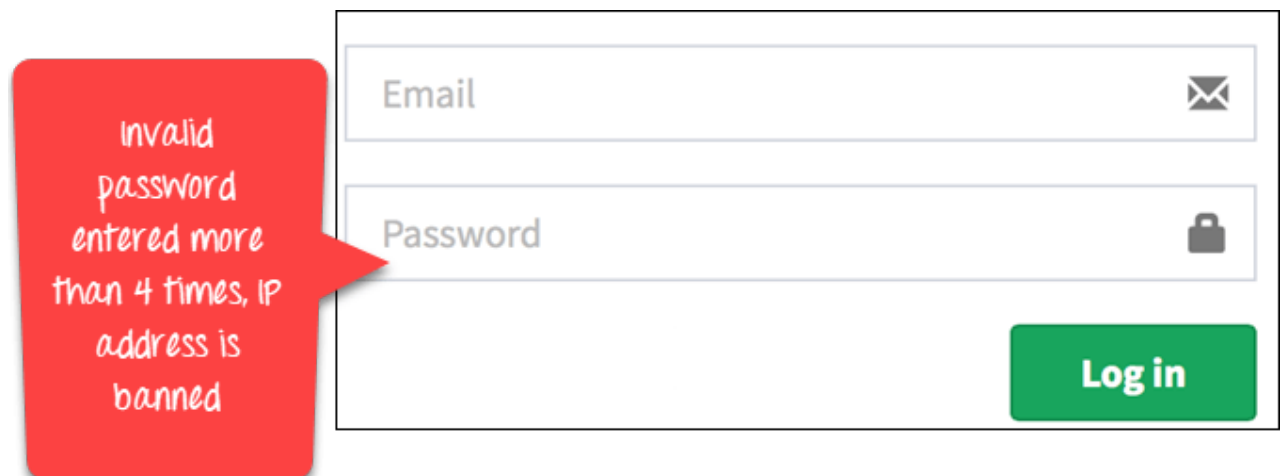
(d) State Transition Testing

- State transition testing is used for menu-based applications and is widely used within the embedded software industry.
- The technique is also suitable for modeling a business scenario having specific states or for testing screen navigation.
- A state transition table shows all valid transitions and potentially invalid transitions between states, as well
 - as the events, and resulting actions for valid transitions.
- State transition diagrams normally show only the valid transitions and exclude the invalid transitions.
- Tests can be designed to cover a typical sequence of states, to exercise all states, to exercise every transition, to exercise specific sequences of transitions, or to test invalid transitions .

(e) Use Case Testing

Use Case Testing is a software testing technique that helps to identify test cases that cover entire system on a transaction by transaction basis from start to end. Test cases are the interactions between users and software application. Use case testing helps to identify gaps in software application that might not be found by testing individual software components.

Eg: We create Use for a login functionality of a Web Application as shown below



The image shows a login form with two input fields: 'Email' and 'Password'. The 'Email' field has an envelope icon on the right, and the 'Password' field has a lock icon on the right. Below the fields is a green 'Log in' button. A red callout box with a speech bubble points to the 'Password' field, containing the text: 'Invalid password entered more than 4 times, IP address is banned'. This illustrates a security use case where the system should ban an IP address after multiple failed login attempts.

Scrum Testing in Detail

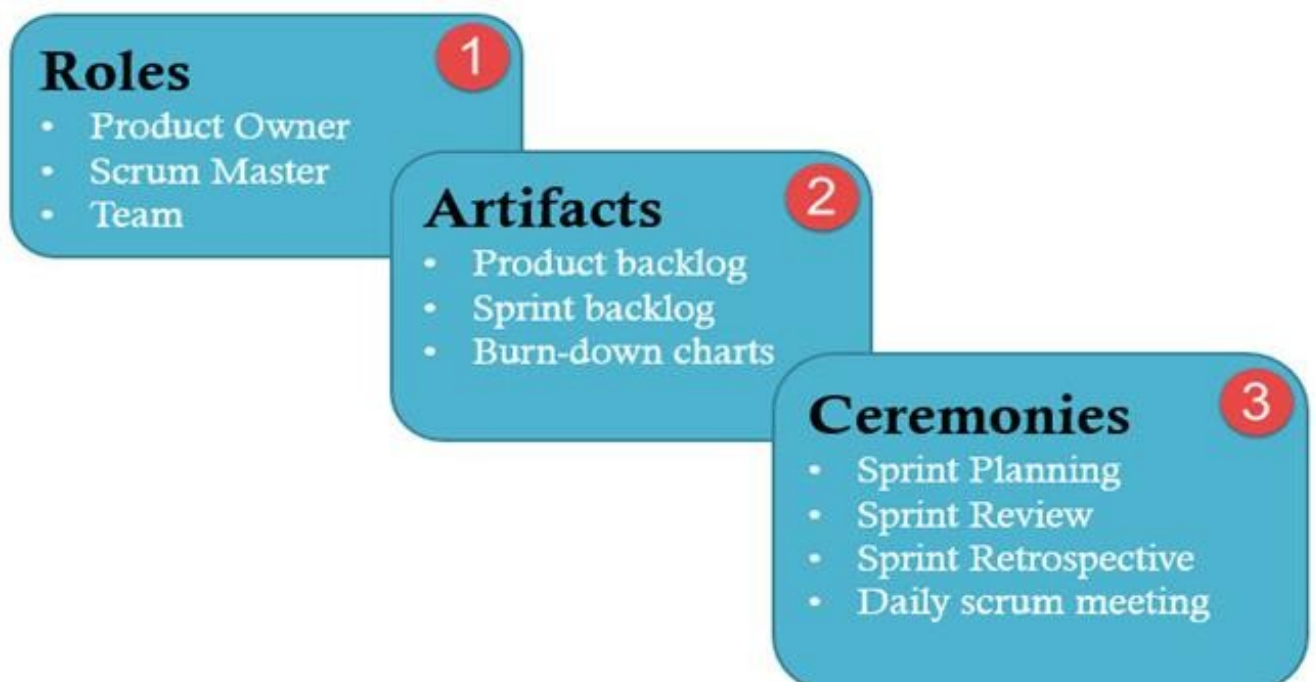
Scrum in Software Testing is a methodology for building complex software applications. It provides easy solutions for executing complicated tasks. Scrum helps the development team to focus on all aspects of the software product development like quality, performance, usability and so on. It provides with transparency, inspection and adaptation during the software development to avoid complexity.

Following are Key Features of Scrum

- Scrum has a short fixed schedule of release cycles with adjustable scope known as sprints to address rapidly changing development needs. Each release could have multiple sprints. Each Scrum Project could have multiple Release Cycles.

- A repeating sequence of meetings, events, and milestones A practice of testing and implementing new requirements, known as stories, to make sure some work is released ready after each sprint

Scrum is based on the following 3 Pillars

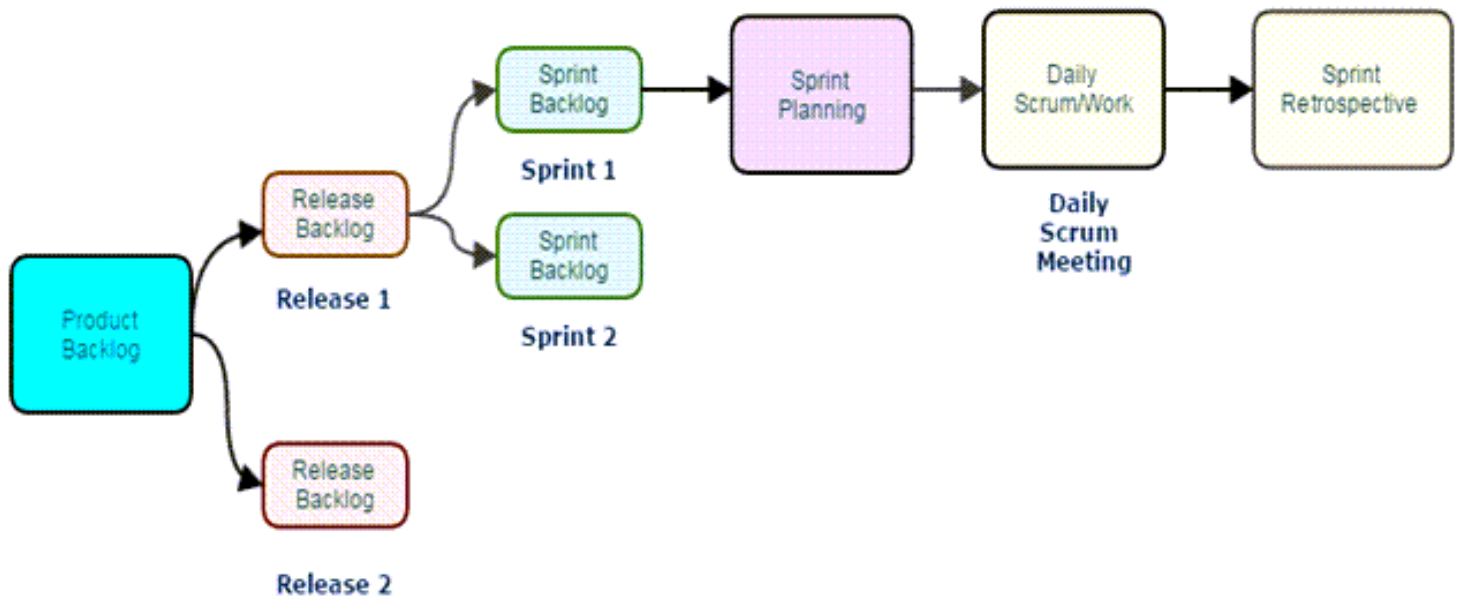


1. Roles in Scrum

There are three chief roles in Scrum Testing – Product Owner, Scrum Master and The Development Team. Let's study them in detail.

Product Owner	Scrum Master	The Team
<ul style="list-style-type: none">• He/She defines features of the product.	<ul style="list-style-type: none">• He/She manages the team and look after the team's productivity	<ul style="list-style-type: none">• The team is usually about 5-9 members
<ul style="list-style-type: none">• Product Owner decides the release date and corresponding features	<ul style="list-style-type: none">• He/She maintains the block list and removes barriers in the development	<ul style="list-style-type: none">• It includes developers, designer and sometimes testers, etc.
<ul style="list-style-type: none">• They prioritize the features according to the market value and profitability of the product	<ul style="list-style-type: none">• He/She coordinates with all roles and functions	<ul style="list-style-type: none">• The team organizes and schedule their work on their own
<ul style="list-style-type: none">• He/She is responsible for the profitability of the product	<ul style="list-style-type: none">• He/She shields team from external interferences	<ul style="list-style-type: none">• Has right to do everything within the boundaries of the project to meet the sprint goal
<ul style="list-style-type: none">• He/She can accept or reject work item result	<ul style="list-style-type: none">• Invites to the daily scrum, sprint review and planning meetings	<ul style="list-style-type: none">• Actively participate in daily ceremonies

2. Scrum Artifacts



A scrum process includes

User stories: They are a short explanation of functionalities of the system under test. Example for Insurance Provider is – “Premium can be paid using the online system.”

Product Backlog: It is a collection of user stories captured for a scrum product. The product owner prepares and maintains the product backlog. It is prioritized by the product owner, and anyone can add to it with approval from the product owner.

Release Backlog: A release is a time frame in which the number of iterations is completed. The product owner coordinates with the scrum master to decide which stories should be targeted for a release. Stories in the release backlog are targeted to be completed in a release.

Sprints: It is a set period of time to complete the user stories, decided by the product owner and developer team, usually 2-4 weeks of time.

Sprint Backlog: It's a set of user stories to be completed in a sprint. During sprint backlog, work is never assigned, and the team signs up for work on their own. It is owned and managed by the team while the estimated work remaining is updated daily. It is the list of task that has to be performed in Sprint

Block List: It is a list of blocks and unmade decisions owned by scrum master and updated daily.

Burndown chart: Burn-down chart represents overall progress of the work in progress and work completed throughout the process. It represents in a graph format the stories and features not completed

Ceremonies (Processes) in Scrum

Sprint Planning: A sprint begins with the team importing stories from the release backlog into the sprint backlog; it is hosted by scrum master. The Testers estimate effort to test the various stories in the Sprint Backlog.

Daily Scrum: It is hosted by scrum master, it last about 15 minutes. During Daily Scrum, the members will discuss the work completed the previous day, the planned work for the next day and issues faced during a sprint. During daily stand-up meeting team progress is tracked.

Sprint Review/ Retrospective: It is also hosted by scrum master, it last about 2-4 hours and discuss what the team has accomplished in the last sprint and what lessons were learned.

Role of Tester in Scrum

There is no active role of Tester in the Scrum Process.

Usually, testing is carried out by a developer with Unit Test. While product owner is also frequently involved in the testing process during each sprint. Some Scrum projects do have dedicated test teams depending on the nature & complexity of the project.

Testing Activities in Scrum

Testers do following activities during the various stages of Scrum-

Sprint Planning

In sprint planning, a tester should pick a user-story from the product backlog that should be tested.

As a tester, he/she should decide how many hours (Effort Estimation) it should take to finish testing for each of selected user stories.

As a tester, he/she must know what sprint goals are.

As a tester, contribute to the prioritizing process

Sprint

- Support developers in unit testing

- Test user-story when completed. Test execution is performed in a lab where both tester and developer work hand in hand. Defect are logged in Defect Management tool which are tracked on a daily basis. Defects can be conferred and analyzed during the scrum meeting. Defects are retested as soon as it is resolved and deployed for testing

- As a tester, he/she attends all daily standup meeting to speak up

- As a tester, he/ she can bring any backlog item that cannot be completed in the current sprint and put to the next sprint

- Tester is responsible for developing automation scripts. He schedules automation testing with Continuous Integration (CI) system. Automation receives the importance due to short delivery timelines. Test Automation can be accomplished by utilizing various open source or paid tools available in the market. This proves effective in ensuring that everything that needs to be tested was covered. Sufficient Test coverage can be achieved with a close communication with the team.

- Review CI automation results and send Reports to the stakeholders

-Executing non-functional testing for approved user stories
Coordinate with customer and product owner to define acceptance criteria for Acceptance Tests
At the end of the sprint, the tester also does acceptance testing(UAT) in some case and confirms testing completeness for the current sprint.

Sprint Retrospective

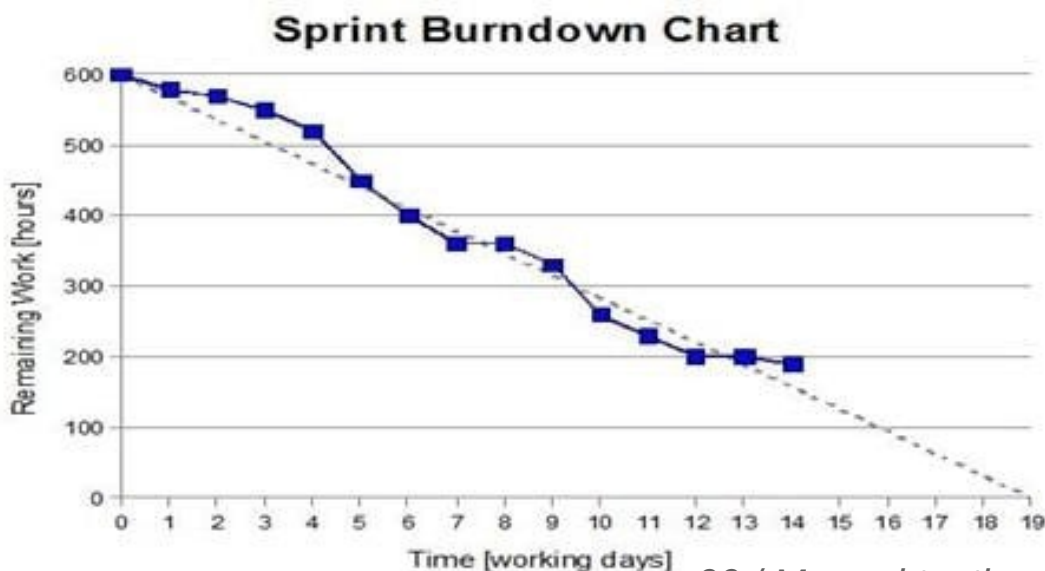
-As a tester, he will figure out what went wrong and what went right in the current sprint
-As a tester, he identifies lesson learned and best practices.

Test Reporting

Scrum Test metrics reporting provides transparency and visibility to stakeholders about the project. The metrics that are reported allow a team to analyze their progress and plan their future strategy to improve the product. There are two metrics that are frequently used to report.

Burn down chart: Each day, Scrum Master records the estimated remaining work for the sprint. This is nothing but the Burn Down Chart. It is updated daily.

A burndown chart gives a quick overview of the project progress, this chart contains information like the total amount of work in the project that must be completed, amount of work completed during each sprint and so on.



Velocity history graph: The velocity history graph predicts the velocity of the team reached in each sprint. It is a bar graph and represents how teams output has changed over time..