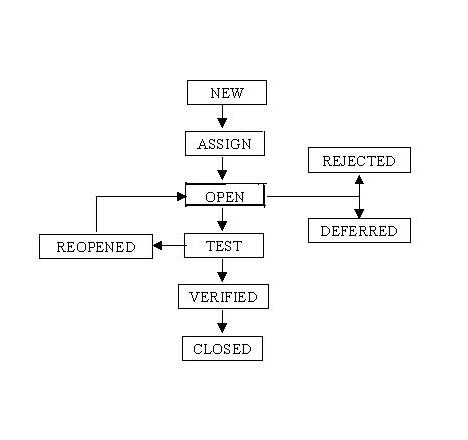
**What is Defect life cycle?**

Defect life cycle is a cycle which a defect goes through during its lifetime. It starts when defect is found and ends when a defect is closed, after ensuring it’s not reproduced. Defect life cycle is related to the bug found during testing.

The bug has different states in the Life Cycle. The Life cycle of the bug can be shown diagrammatically as follows:



**Bug or defect life cycle includes following steps or status:**

**New:** When a defect is logged and posted for the first time. Its state is given as new.

**Assigned:** After the tester has posted the bug, the lead of the tester approves that the bug is genuine and he assigns the bug to corresponding developer and the developer team. Its state given as assigned.

**Open:** At this state the developer has started analyzing and working on the defect fix.

**Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’ and the bug is passed to testing team.

**Pending retest:** After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

**Retest:** At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

**Verified:** The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “verified”.

**Reopen:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “reopened”. The bug goes through the life cycle once again.

**Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “closed”. This state means that the bug is fixed, tested and approved.

**Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “duplicate“.

**Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “rejected”.

**Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

**Not a bug:** The state given as “Not a bug” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of colour of some text then it is not a bug but just some change in the looks of the application.

**What is the difference between performance and load testing?**

**Performance Testing**

Performance Testing represents the characteristics of a system, how fast is that system, in general it refers to the tests undertaken to determine how that system behaves and performs.

Prerequisites for Performance Testing

We need to have clearly defined goals before starting any performance tests. The goals defined will help us if after getting the performance tests results there is a need for system improvement.

Performance Testing Results

After running performance tests, you need to inspect the metrics like response and latency times, hits per second, throughput per second, errors (client side metrics) and CPU, Memory, I/O (server side metrics) for the system tested.

Performance Testing Goals

Set a base line for the system under test for future releases and potential improvement of the system. It helps to identify potential bottlenecks, that can be found at different levels: system level (client and server side), DB level, network level.

Most of the times, running performance tests can be a very time consuming and hard job. First you need to set a base line for your tests, then you need to tweak the tests (number of concurrent users, duration and ramp up period between concurrent users).

Use LoadFocus to performance test your application from the cloud.

**Load Testing**

Load Testing is understanding how the system behaves under high volume of transactions. Let’s say you send an email campaign to 10000 addresses and you want to understand how your system behaves when most of those 10000 potential users will access your applications.

Load Testing helps you simulate similar scenarios in an controlled environment, and the goal of load testing is to see whether the application can sustain the increased load on the server or will it crash the servers. Use LoadFocus to cloud load test your application.

Load Testing is usually started with a low number of concurrent clients (users) and gradually increased over a given period of time until it reaches the desired load on the system. During this period, client side and server side metrics need to be inspected.

We’ve only touched the surface in terms of potential issues, performance testing and load testing tools and techniques that deserve to be mentioned in the context of performance and load testing.

**What are the different types of Non-Functional Testing types?**

Non functional testing ensures that a system/application meets the specified performance requirements. In non functional software testing, by performance we do not only mean response time, but several other factors such as security, scalability and usability of the application as well.

Let's discuss some of the most common non functional testing types below :

Types of Non Functional Testing

1. Performance Testing

2. Load Testing

3. Stress Testing

4. Volume Testing

5. Failover Testing

6. Security Testing

7. Compatibility Testing

8. Usability Testing

9. Scalability Testing

Performance Testing:

First and foremost type of non functional testing is performance testing. In order to ensure that the response time of a system is acceptable, performance testing is carried out. By setting up a considerable load and a production-sized database, the system is tested for response times of several business critical processes.

Load Testing:

Types of non functional testing in software testing also includes load testing. To check whether the system can sustain the pressure or load of many users accessing the system at one time, load testing needs to be carried out. The production load is replicated in the test environment in this case after which the application/system is tested.

Stress Testing:

This testing is done to pull the system far beyond its capabilities and see how it reacts. Contrary to load testing in which the maximum allowable load is generated, in stress testing, the load generated is more than what the system is expected to handle.

Volume Testing:

When storage requirements and capabilities of the system are to be tested, volume testing is done. When a huge database size is encountered, system’s performance and its ability to exchange data and information are tested in this case.

Failover Testing:

To test how well the redundancy mechanism works when the system encounters heavy load or unexpected failure is what failover testing is about. Also, when the specific failed system is back again, it must begin to function as per requirements – this is fail-back testing.

Security Testing:

While performing non functional testing, to test how well the system can preserve itself and the data it holds in situation of malicious attacks is called security testing. Confidentiality, integrity, availability, authentication and authorization are the main areas that are tested when security testing is considered. Also, network security, system security and application security are other areas that will be tested in this case.

Compatibility Testing:

An application is tested for its coordination with different hardware and software that it is expected to work on. Another testing that can be done is working of the application with different versions or releases of the same hardware or software.

Usability Testing:

To verify the ease of usage of an interface within an application is what usability testing is about. Learnability and memorability of the application are main factors in this case. This testing is particularly important when testing GUI.

Scalability Testing:

When an application is tested for it ability to increase and scale up on any of its non-functionality requirements such as load, number of transactions, number of servers, volume of data etc., it is known as scalability testing.

The above list is not exhaustive since there are more than 150 testing types as of today. The different types of testing, including automated testing, that needs to be done depending upon the scope of the project and maturity of the application under test.

**How do we write test cases in BDD Format?**

Given-When-Then is a style of representing tests - or as its advocates would say - specifying a system's behavior using Specification by Example. It's an approach developed by Dan North and Chris Matts as part of Behavior-Driven Development (BDD).It appears as a structuring approach for many testing frameworks such as Cucumber. You can also look at it as a reformulation of the Four-Phase Test pattern.

The essential idea is to break down writing a scenario (or test) into three sections:

The **given** part describes the state of the world before you begin the behavior you're specifying in this scenario. You can think of it as the pre-conditions to the test.

The **when** section is that behavior that you're specifying.

Finally the **then** section describes the changes you expect due to the specified behavior.

Since we're talking about using examples as specifications, it makes sense to show this with an example [2]

Feature: User trades stocks

Scenario: User requests a sell before close of trading

Given I have 100 shares of MSFT stock

And I have 150 shares of APPL stock

And the time is before close of trading

When I ask to sell 20 shares of MSFT stock

Then I should have 80 shares of MSFT stock

And I should have 150 shares of APPL stock

And a sell order for 20 shares of MSFT stock should have been executed

The above example uses Cucumber, which a popular way of writing Business Facing Tests but you can use the Given-When-Then style with any kind of tests. Some people like to put Given-When-Then as comments to mark informal blocks inside unit tests. I've also seen this convention to structure informal prose.

It's usual with this approach to see "ands" used to combine multiple expressions within each clause.

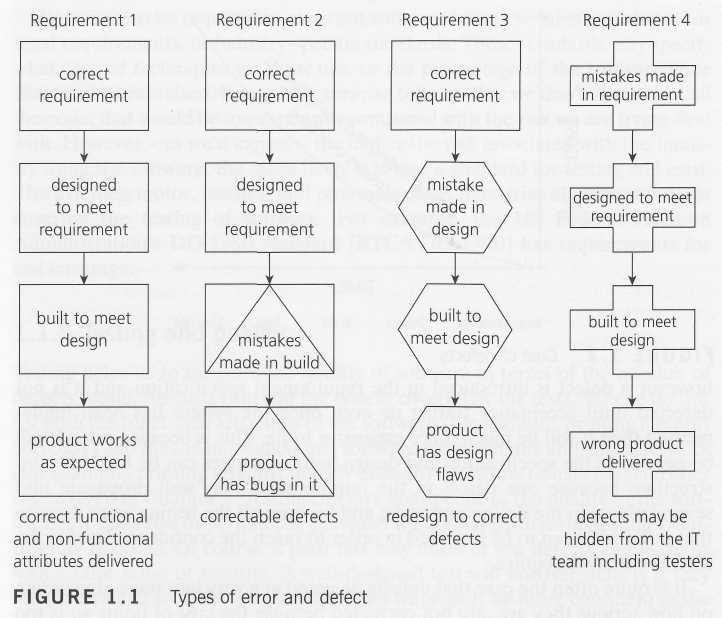
I've characterized the given as a description of the pre-condition state because that's the way I prefer to think of it. A testing framework, however, interprets the givens as a set of commands to bring the system-under-test into the correct state before executing the when command. (Which is why other naming conventions often call this "setup".) Testing frameworks provide various query methods for the then commands - these should be free of side-effects.

Although Given-When-Then style is symptomatic to BDD, the basic idea is pretty common when writing tests or specification by example. Meszaros describes the pattern as Four-Phase Test. His four phases are Setup (Given), Exercise (When), Verify (Then) and Teardown.

**How to arise a defect and what we specify while logging defect?**

To know when defects in software testing arise, let us take a small example with a diagram as given below.

We can see that Requirement 1 is implemented correctly – we understood the customer’s requirement, designed correctly to meet that requirement, built correctly to meet the design, and so deliver that requirement with the right attributes: functionally, it does what it is supposed to do and it also has the right non-functional attributes, so it is fast enough, easy to understand and so on.



Types of errors and defects

With the other requirements, errors have been made at different stages. Requirement 2 is fine until the software is coded, when we make some mistakes and introduce defects. Probably, these are easily spotted and corrected during testing, because we can see the product does not meet its design specification.

The defects introduced in Requirement 3 are harder to deal with; we built exactly what we were told to but unfortunately the designer made some mistakes so there are defects in the design. Unless we check against the requirements definition, we will not spot those defects during testing. When we do notice them they will be hard to fix because design changes will be required.

The defects in Requirement 4 were introduced during the definition of the requirements; the product has been designed and built to meet that flawed requirements definition. If we test the product meets its requirements and design, it will pass its tests but may be rejected by the user or customer. Defects reported by the customer in acceptance test or live use can be very costly. Unfortunately, requirements and design defects are not rare; assessments of thousands of projects have shown that defects introduced during requirements and design make up close to half of the total number of defects.

Defect logging, a process of finding defects in the application under test or product by testing or recording feedback from customers and making new versions of the product that fix the defects or the clients feedback.

Defect tracking is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects gets multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Examples - Hp Quality Center, IBM Rational Quality Manager.

**Defect Tracking Parameters**

Defects are tracked based on various parameters such as:

Defect Id

Priority

Severity

Created by

Created Date

Assigned to

Resolved Date

Resolved By

Status

**How to deal the production defects?**

Defect handling under Waterfall and Agile are two separate beasts. Working under the Waterfall methodology, development and testing are performed in stages and software is not handed off to testers until it is ‘done’. At this point, software defects consist of any behavior in the software that does not align with predefined specifications.

A majority of software teams working under the Waterfall method will have some sort of defect tracking system in place. This can range from a custom-built application to a simple Excel sheet. In either case, it is the responsibility of the software tester to report and rate software defects. They also provide enough background information for developers to recreate them. Through this reporting system, testing and defects are usually handled in multiple rounds until the software is ready to go into production.

In Agile, software teams work much closer together. Instead of detailed functional specifications that outline the full scope of a project, teams work on user stories. These two fundamental differences from Waterfall completely change the way software defects are reported and handled.

A user story includes three basic elements – role, feature, and business value or benefit. Together with acceptance criteria and perhaps a few visual aids, developers and testers work with one another to make the user story a reality. Because Agile is a collaborative methodology, testers begin their work as software is being developed. Since a user story is not yet complete at this stage, any defects found are not technically considered defects. There is no formal defect reporting system in place. Instead, it is up to testers and developers to work together on potential issues and complete the user story together.

It’s not until a user story is complete and software goes live that an issue is considered an actual defect. At this stage, defects are entered into the development backlog, are prioritized, and develop into their own user stories in order to be addressed.

**How to estimate test cases?**

* Work Breakdown Structure
* 3-Point Software Testing Estimation Technique
* Wideband Delphi technique
* Function Point/Testing Point Analysis
* Use – Case Point Method
* Percentage distribution
* Ad-hoc method

Work Breakdown Structure:

It is created by breaking down the test project into small pieces. Modules are divided into sub-modules. Sub modules are further divided into functionalities and functionalities are divided in sub-functionalities.

Review all the requirements from Requirement Document to make sure they are added in WBS. Now you figure out the number of tasks your team needs to complete. Estimate the duration of each task.

3-Point Software Testing Estimation Technique

3-Point Software Testing Estimation Technique is based on statistical methods in which each testing task is broken down into sub tasks and then three types on estimation are done on each tasks.

The formula used by this technique is:

Test Estimate = P + (4\*N) + E / 6

Whereas P = Positive Scenarios or Optimistic Estimate (Best case scenario in which nothing goes wrong and all conditions are optimal.)

N = Negative Scenarios or Most Likely Estimate (most likely duration and there may be some problem but most of the things will go right.)

E = Exceptional Scenarios or Pessimistic Estimate (worst case scenario which everything goes wrong.)

Standard deviation for the technique is calculated as,

Standard Deviation (SD) = (N – E)/6

Use – Case Point Method:

Use-Case Point Method is based on the use cases where we calculate the un-adjusted actor weights and un-adjusted use case weights to determine the software testing estimation.

Use case is a document which well specifies different users, systems or other stakeholders interacting with the concerned application. They are named as ‘Actors’. The interactions accomplish some defined goals protecting the interest of all stakeholders through different behavior or flow termed as scenarios.

The formula used for this technique is:

Un adjusted actor weights = total no. of actors (positive, negative and exceptional)

Un adjusted use case weight = total no. of use cases.

Un adjusted use case point = Un adjusted actor weights + Un adjusted use case weight

Determine the technical/environmental factor (TEF) ( if not available take as 0.50)

Adjusted use case point = Unadjusted use case point \* [0.65+ (0.01 \* TEF]

Total Effort = Adjusted use case point \* 2

Wideband Delphi technique:

Same as above WBS, In Wideband Delphi Method, work breakdown structure is decomposed for each task and is distributed to a team comprising of 3-7 members for re-estimating the task. The final estimate is the result of the summarized estimates based on the team consensus. This method speaks more on experience rather than any statistical formula. This method was popularized by Barry Boehm to emphasize on the group iteration to reach to a consensus where the team visualized on the different aspects of the problems while estimating the test effort.

Function Point/Testing Point Analysis:

The FP technique is a direct indicator of the functionality of software application from the user's perspective. This is the most accepted technique used to estimate the size of a software project.

This technique is a part of TMap. Base of this technique is function point technique. Here we convert function points into test points. In Test Point analysis, we usually carry out the following:

Dynamic Test Points

Static Test Points

Environmental Factor

Productivity Factor

Primary Test Hours

Control Factor

Total Test Hours

In Testing, This estimation is based on requirement specification document, or a previously created prototype of the application. To calculate FP for a project, some major components are required.

The major components are:

Unadjusted Data Function Points:

Internal Files

External Interfaces

Unadjusted Transaction Function Points:

User Inputs

User Outputs &

User Inquiries

Capers Jones basic formula:

Number of Test cases = [Number of Function Points] x 1.2

Total Actual Effort, TAE = (Number of Test cases) \* (Percentage of development effort /100)

This method is done in a case when a detailed low level design document or requirement document is available (i.e measure of function point is available) & Previous data for development and testing is available.

**????????????**

**What are different defect metrics and measurements we prepare in testing?**

A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

Metrics can be defined as “STANDARDS OF MEASUREMENT”.

Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement.

Suppose, in general, “Kilogram” is a metric for measuring the attribute “Weight”. Similarly, in software, “How many issues are found in thousand lines of code?”, here No. of issues is one measurement & No. of lines of code is another measurement. Metric is defined from these two measurements.

Test metrics example:

How many defects are existed within the module?

How many test cases are executed per person?

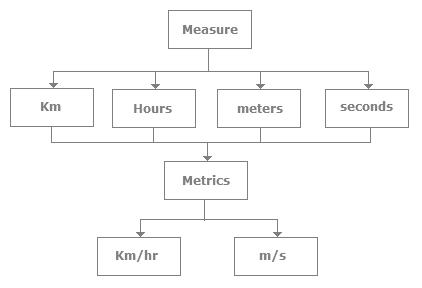
What is the Test coverage %?

What is Software Test Measurement?

Measurement is the quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.

Test measurement example: Total number of defects.

Please refer below diagram for clear understanding of the difference between Measurement & Metrics.

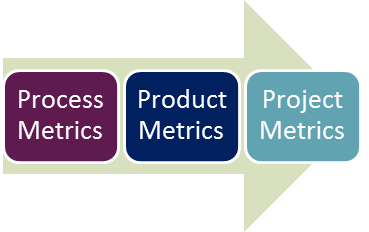


**Types of Metrics:**

Process Metrics: It can be used to improve the process efficiency of the SDLC ( Software Development Life Cycle)

Product Metrics: It deals with the quality of the software product

Project Metrics: It can be used to measure the efficiency of a project team or any tools being used by the team members



Identification of correct testing metrics is very important. Few things need to be considered before identifying the test metrics

Fix the target audience for the metric preparation

Define the goal for metrics

Introduce all the relevant metrics based on project needs

Analyze the cost benefits aspect of each metrics and the project lifestyle phase in which it results into the maximum output

Now, how can we measure the quality of the software by using Metrics?

Suppose, if a project does not have any metrics, then how the quality of the work done by a Test analyst will be measured?

For Example: A Test Analyst has to,

* Design the test cases for 5 requirements
* Execute the designed test cases
* Log the defects & need to fail the related test cases
* After the defect is resolved, need to re-test the defect & re-execute the corresponding failed test case.

In above scenario, if metrics are not followed, then the work completed by the test analyst will be subjective i.e. the test report will not have the proper information to know the status of his work/project.

**What are test design techniques?**

A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of defect and relatively poor at finding other types.

For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs.

Similarly, testing performed at different stages in the software development life cycle will find different types of defects; component testing is more likely to find coding logic defects than system design defects.

Each testing technique falls into one of a number of different categories. Broadly speaking there are two main categories:

* Static technique
* Dynamic technique

**Static testing** is the testing of the software work products manually, or with a set of tools, but they are not executed.

It starts early in the Life cycle and so it is done during the verification process.

It does not need computer as the testing of program is done without executing the program. For example: reviewing, walk through, inspection, etc.

Most static testing techniques can be used to ‘test’ any form of document including source code, design documents and models, functional specifications and requirement specifications.

**Dynamic testing:**

* This testing technique needs computer for testing.
* It is done during Validation process.
* The software is tested by executing it on computer.
* Example of this Dynamic Testing Technique: Unit testing, integration testing, system testing.

Dynamic techniques are subdivided into three more categories: specification-based (black-box, also known as behavioral techniques), structure-based (white-box or structural techniques) and experience- based. Specification-based techniques include both functional and nonfunctional techniques (i.e. quality characteristics).

**What are the tools to manage defects/stories?**

The Most Popular Bug Tracking Software Tools are:

**1. Stryka:** Stryka is a cutting-edge enterprise test management tool, built from the ground up using the latest web and mobile technologies. Supports the end-to-end test lifecycle from linking requirements to test cases to defects.

Stryka testing technology supports agile testing practices and has the only Quality Tester leaderboard available on the market.

**2. Bugzilla:** Bugzilla defect tracking has been a leading bug tracking tools widely used by many organizations for quite some time now. It is very simple to use, web based interface. It has all the features of essence, convenience and assurance. It is completely open sourced and is free to use.

**3. Lean Testing**

Lean Testing logo

Lean Testing is a free bug tracking and test case management software designed by testers. It has a browser extension to report bugs on websites quickly and easily as well as in-app reporting tools to allow users to report bugs directly from within mobile apps.

The system has everything you’d expect from a bug tracker and test case manager, but great emphasis has been placed on making sure that everything be intuitive and easy to use. Lean Testing is web-based and requires no installation.

**4. JIRA:**

JIRA defect tracking

Details: Atlassian JIRA, primarily an incident management tool is also commonly used for bug-tracking. It provides the complete set of recording, reporting, workflow and other convenience related features. It is a tool that integrates directly with the code development environments thus making it a perfect fit for developers as well. Also, due to its capability to track any and all kinds of issues, it is not necessarily concentrated to only software development industry and renders itself quite efficiently to help desks, leave management systems etc. It supports agile projects also. It is a commercial licensed product with many add-ins that support extensibility.

**5. Mantis:**

Mantis Issue tracking tool

Details: I have one thing to say about this tool – do not be deceived by its simple exterior. I mean, in terms of simplicity and ease of use, this tool wins the crown. It has every feature you can hope for and then some. To catch up with the changing times, Mantis not only comes as a web application, but also has its own mobile version. It is implemented in PHP and is free for use. If you would like it to be hosted, they do charge a price, but quite affordable.

**6. Trac:**

Trac bug tracking system

Details: Trac also is not necessarily a specialized bug tracking system and is an issue tracking system. It is written using Python and is web based. When you integrate Trac with a SCM system, you can use it to browse through the code, view changes, view history etc. The issues/incidents in Trac are referred to as ‘tickets’ and the ticket management system can be used for defect management as well, if you wish to do so.

**7. Redmine:**

Redmine issue tracking

Details: Redmine is an open source issue tracking system that integrates with SCM (source code management systems) too. Even though it is not a ‘bug tracking’ tool it involves working with issues, where issues can be features, tasks, bugs/defects etc. It is a web application that works across many platforms, but will need Ruby to be available.

**8. HP ALM/Quality Center:**

HP ALM defect tracking

Details: Well, no list of bug tracking tools will be complete without the HP QC, would it? HP ALM is an end-to-end test management solution with a robust integrated bug tracking mechanism within it. HP ALM’s bug tracking mechanism is easy, efficient and everything you can ask for. It supports Agile projects too. It is one of the pricey tools available in the market, which continues to be a prime source of criticism along with the fact that it is not very ‘friendly’ with all the web browsers.

**9. FogBugz:**

FogBugz bug management

Details: FogBugz is also a web based bug tracking system that refers to defects as ‘cases’. It allows you to create, list, assign and work on cases created. Also the project information can be created in terms of milestones so that the progress of the cases can be evaluated against the milestones. Very simple to use and has all the features of essence for sure. Additionally, with FogBugz you can create wikis to be made available for the general public. It is a commercial product but very reasonably priced.

**10. IBM Rational ClearQuest:**

IBM Rational ClearQuest logo

Details: Clear Quest is a client-server based web application that supports defect management process. It provides integration with various automation tools which can be considered an additional feature. Other than that, it has an end-to-end, customizable defect tracking systems. It is a commercial product and can seem a little costly.

**11. Lighthouse:**

Lighthouse defect tracking logo

Details: Light house is an issue tracker that is web based and is also compatible with your mobile devices. It is simple and organized. All the issues are referred to as tickets in here too. There is an activity stream, mile stones etc. Another nice feature is that, lighthouse lets you store project document online in its interface itself.

**12. Zoho bug tracker:**

Zoho Bug Tracker logo

Details: Zoho Bug Tracker is one of the modules in the task management software Zoho Project. It is an online tool that will let you create Projects, milestone, tasks, bugs, reports, documents and so on. The bug tracker module by itself has all the features of essence that you generally look for. The product is commercial but not very expensive

**13. The Bug Genie:**

The Bug Genie logo

Details: Though the name sounds like it must be a bug-tracking tool – that is not all Bug Genie is. It is a complete Project management and issue tracking tool – which involves defect management to be one of its aspects along with integration with many SCM systems, Project creation and handling features, issue tracking mechanism, integrated wiki and easy to use web interface. Supports Agile projects also.

**14. BugHost:**

BugHost logo

Details: A web-based defect tracking system that is very simple and has all the features that you will need to manage issues for your project effectively. It also has a nifty little service WebHost that you can use for the users (the end customers) to create an issue directly into your project. Though commercial, it is very affordable.

**15. DevTrack:**

Devtrack logo

Details: Devtrack cannot be categorized as your average defect tracker although it does function well if that is what you have in mind. It can be obtained as a stand-alone component or it comes along with Agile Studio, DevTest studio or the DevSuite. As the name implies it is a comprehensive solution to implementation tracking. Supports both agile and waterfall projects. It is a commercial product. A free trial is available too.

**16. BugNET:**

BugNET logo

Details: BugNET belongs to the “issue management” group of tools – quite a good one at that. The issues could be features, tasks or defects. It has all the features of creating projects, managing them, creating issues against them and tracking them to completion, search, reports, wiki pages, etc. There is a pro version for this tool that is licensed and commercial, but the regular version is free to use.

**17. eTraxis:**

eTraxis logo

Details: eTraxis is also another tracking tool, that can be used to track bugs but again, that is not all. You can choose to track basically anything. So, the target audience is not confined to software systems. The best feature of this tool is the flexibility it provides with regards to the creation of custom workflows- in other words, you can choose to define the rules that need to be followed in the process of tracking and progress-ing a certain aspect through its lifecycle stages. These custom workflows are referred to as templates and they can be very handy.

**What are typical environments we have in projects?**

**1) Testing the complete application:** There are millions of test combinations. It’s not possible to test each and every combination both in manual as well as in automation testing. If you try all these combinations you will never ship the product

**2) Misunderstanding of company processes:** Sometimes you just don’t pay proper attention what the company-defined processes are and these are for what purposes. There are some myths in testers that they should only go with company processes even these processes are not applicable for their current testing scenario. This results in incomplete and inappropriate application testing.

**3) Relationship with developers:** Requires very skilled tester to handle this relation positively and even by completing the work in testers way. There are simply hundreds of excuses developers or testers can make when they are not agree with some points. For this tester also requires good communication, troubleshooting and analyzing skill.

**4) Regression testing:** When project goes on expanding the regression testing work simply becomes uncontrolled. Pressure to handle the current functionality changes, previous working functionality checks and bug tracking.

**5) Lack of skilled testers:** I will call this as ‘wrong management decision’ while selecting or training testers for their project task in hand. These unskilled fellows may add more chaos than simplifying the testing work. This results into incomplete, insufficient and ad-hoc testing throughout the testing life cycle.

**6) Testing always under time constraint:** We want to ship this product by this weekend, are you ready for completion? When this order comes from boss, tester simply focuses on task completion and not on the test coverage and quality of work. There is huge list of tasks that you need to complete within specified time. This includes writing, executing, automating and reviewing the test cases.

**7) Which tests to execute first?** If you are facing the challenge stated in point no 6, then how will you take decision which test cases should be executed and with what priority? Which tests are important over others? This requires good experience to work under pressure.

**8) Understanding the requirements:** Sometimes testers are responsible for communicating with customers for understanding the requirements. What if tester fails to understand the requirements? Will he be able to test the application properly? Definitely No! Testers require good listening and understanding capabilities.

**9) Automation testing:** Many sub challenges – Should automate the testing work? Till what level automation should be done? Do you have sufficient and skilled resources for automation? Is time permissible for automating the test cases? Decision of automation or manual testing will need to address the pros and cons of each process.

**10) Decision to stop the testing:** When to stop testing? Very difficult decision. Requires core judgment of testing processes and importance of each process. Also requires ‘on the fly’ decision ability.

**11) One test team under multiple projects:** Challenging to keep track of each task. Communication challenges. Many times results in failure of one or both the projects.

**12) Reuse of Test scripts:** Application development methods are changing rapidly, making it difficult to manage the test tools and test scripts. Test script migration or reuse is very essential but difficult task.

**13) Testers focusing on finding easy bugs:** If organization is rewarding testers based on number of bugs (very bad approach to judge testers performance) then some testers only concentrate on finding easy bugs those don’t require deep understanding and testing. A hard or subtle bug remains unnoticed in such testing approach.

**14) To cope with attrition:** Increasing salaries and benefits making many employees leave the company at very short career intervals. Managements are facing hard problems to cope with attrition rate. Challenges – New testers require project training from the beginning, complex projects are difficult to understand, delay in shipping date.

**What is Agile method?**

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto. The Manifesto was developed by a group fourteen leading figures in the software industry, and reflects their experience of what approaches do and do not work for software development.

A software testing practice that follows the principles of agile software development is called Agile Testing. Agile is an iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.

**Advantages of Agile Testing**

Agile Testing Saves Time and Money

Less Documentation

Regular feedback from the end user

Daily meetings can help to determine the issues well in advance

**Principles of Agile Testing**

Testing is NOT a Phase: Agile team tests continuously and continuous testing is the only way to ensure continuous progress.

Testing Moves the project Forward: When following conventional methods, testing is considered as quality gate but agile testing provide feedback on an ongoing basis and the product meets the business demands.

Everyone Tests: In conventional SDLC, only test team tests while in agile including developers and BA's test the application.

Shortening Feedback Response Time: In conventional SDLC, only during the acceptance testing, the Business team will get to know the product development, while in agile for each and every iteration, they are involved and continuous feedback shortens the feedback response time and cost involved in fixing is also less.

Clean Code: Raised defects are fixed within the same iteration and thereby keeping the code clean.

Reduce Test Documentation: Instead of very lengthy documentation, agile testers use reusable checklist, focus on the essence of the test rather than the incidental details.

Test Driven: In conventional methods, testing is performed after implementation while in agile testing, testing is done while implementation.

**Best Practices in Agile Testing**

1. Automated Unit Tests

2. Test Driven Development

3. Automated Regression Tests

4. Exploratory Testing

**What is Alpha and Beta Testing?**

Alpha and beta testing are important testing phases for success of any software release. Both these testing techniques have saved thousands of dollars to large scale software releases for companies like Apple, Google and Microsoft.

The purpose of this article is to educate you on these testing terms and how you can use these techniques to earn some extra income or even to get a full time testing job. Freshers can put beta testing experience in resume to make it stand out from the crowd.

**What is Alpha Testing?**

This is a form of internal acceptance testing performed mainly by in-house software QA and testing teams. Alpha testing is the last testing done by test teams at development site after the acceptance testing and before releasing the software for beta test. Alpha testing can also be done by potential users or customers of the application. But still this is a form of in-house acceptance testing.

**What is Beta Testing?**

This is a testing stage followed by internal full alpha test cycle. This is the final testing phase where companies release the software for few external user groups outside the company test teams or employees. This initial software version is called as beta version. Most companies gather user feedback in this release.

The simple definition of beta testing – testing carried out by real users in real environment.

Though companies do rigorous in-house quality assurance from dedicated test teams, it’s practically impossible to test application for each and every combination of the test environment. Beta releases make it easier to test application on thousands of test machines and fix the issues before releasing the application publicly. The selection of beta test groups can be done based on company needs. Company can either invite few users to test the preview version of the application or they can release it openly to try by every user.

Fixing the issues in beta release can significantly reduce the development cost as most of the minor glitches get fixed before the final release. Till now many big companies successfully used beta versions of their most anticipated applications. E.g. Recently Microsoft corporation released Windows 7 beta and based on feedback from thousands of users they managed to release a stable OS version. In past Apple also releases OS X beta in public and fixed many minor issues and improved the OS based on user feedback.

**Related Useful Terms:**

Beta Software – Preview version of the software released to the public before final release.

Beta Version – Software version releases in public that include almost all of the features but not development complete yet and may still have some errors.

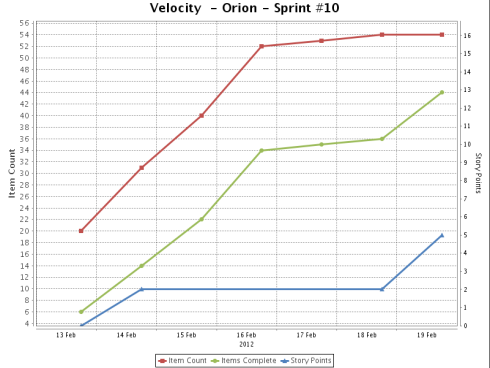
Beta Testers – Testers who work on testing beta version of the software release.

**What is burndown chart and velocity?**

Velocity and Burndown reports are used to track the progress of an agile project. The reports can be configured for a release, a sprint or team.

**Velocity Report**

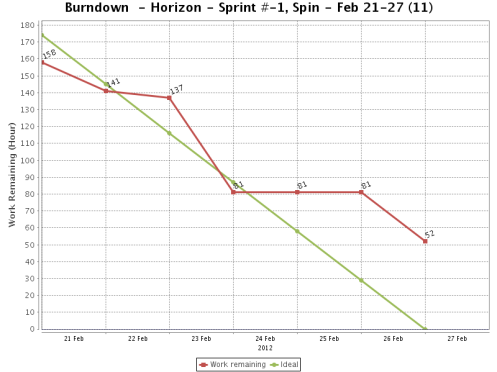
The velocity report tracks the story points completed in a specified timeframe. It enables team members to determine how well they are keeping to schedule and to measure their performance over time.



The red line tracks the number of items in the backlog; the green line tracks the number of items completed; and the blue line tracks the completed story points. Each point on the lines indicates the particular count at that point. A user can see the items completed on the left axis and the story points completed on the right axis. Ideally all three lines should meet when all of the work items are complete.

**Burndown Report**

The Burndown chart compares the remaining hours the team(s) need to complete against the ideal burndown. The ideal burndown line is green and the actual burndown line is red. Each data point on the actual burndown line represents a day in the sprint. The burndown reflects the number of hours remaining for that day and any changes to the hourly estimation.



Burndown calculation is based on the estimated resource hours (Resource Demand field) for the tasks and requirements in the sprint. The burndown line is calculated from the Remaining Work field of the tasks and requirements.

If a requirement has tasks, then the ideal line counts the total resource demand for its task in that sprint.

If a requirement with no tasks is assigned to the sprints, then its resource demand value is used.

If a requirement's tasks are spread across multiple sprints, the burndown only counts the estimates from the tasks assigned to the sprint.

The ideal line reflects the new estimate when a task's or requirement's estimate changes, even if the sprint start date has passed.

**To generate Agile Velocity and Burndown reports**

* Navigate to the Reporting module.
* Go to the Templates tab.
* Expand the Engineering Reports folder and select the Agile Velocity, Burndown item in the templates hierarchy.
* The template configuration details are displayed in the right hand pane.
* Do one of the following:
* Click the Release ellipsis (...) to specify the release and, optionally, sprint to report on.
* Click the Team/Sprint ellipsis (...) to specify the sprint to report on.
* If desired, explicitly set the dates of interest for the report to cover. By default, if a sprint is selected, the report automatically uses the sprint's start and finish dates as the report's date parameters.
* Note: If the release does not have a Start or End Date the Date field displays NA and the report displays that the date(s) were not specified. You can update the report configuration with dates and rerun the report to view the velocity chart.
* Select the interval for the X-axis of the report (day, week, sprint).
* Specify any of the following parameters:
* Exclude defects from the report - only count requirements and tasks
* Count requirements in place of tasks - only count their parent requirements
* Click Run Report to test the report.
* The system creates the report and opens it in a new browser window.
* If you are happy with the report settings, save the report by clicking Save as Report Instance. After you save the report, you can access it in the Reports tab of the Reporting module, and you can also send a link to the saved report to colleagues and stakeholders.

**What is defect?**

**Defect:** A defect is an error or a bug, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects.

When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it’s called differently like bug, issue, incidents or problem.

When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect. These defects or bugs occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results.

Additional Information about Defects / Bugs:

While testing a software application or product if large number of defects are found then it’s called Buggy.

When a tester finds a bug or defect it’s required to convey the same to the developers. Thus they report bugs with the detail steps and are called as Bug Reports, issue report, problem report, etc.

This Defect report or Bug report consists of the following information:

* **Defect ID** – Every bug or defect has it’s unique identification number
* **Defect Description** – This includes the abstract of the issue.
* **Product Version** – This includes the product version of the application in which the defect is found.
* **Detail Steps** – This includes the detailed steps of the issue with the screenshots attached so that developers can recreate it.
* **Date Raised** – This includes the Date when the bug is reported
* **Reported By** – This includes the details of the tester who reported the bug like Name and ID
* **Status** – This field includes the Status of the defect like New, Assigned, Open, Retest, Verification, Closed, Failed, Deferred, etc.
* **Fixed by** – This field includes the details of the developer who fixed it like Name and ID
* **Date Closed** – This includes the Date when the bug is closed
* **Severity** – Based on the severity (Critical, Major or Minor) it tells us about impact of the defect or bug in the software application
* **Priority** – Based on the Priority set (High/Medium/Low) the order of fixing the defect can be made. (Know more about Severity and Priority)

**What is development environment?**

A development environment is a collection of procedures and tools for developing, testing and debugging an application or program.

The development environment normally has three server tiers, called development, staging and production. All three tiers together are usually referred to as the DSP.

* Development Server: Here is where the developer tests code and checks whether the application runs successfully with that code. Once the application has been tested and the developer feels that the code is working fine, the application then moves to the staging server.
* Staging Server: This environment is made to look exactly like the production server environment. The application is tested on the staging server to check for reliability and to make sure it does not fail on the actual production server. This type of testing on the staging server is the final step before the application could be deployed on a production server. The application needs to be approved in order to deploy it on the production server.
* Production Server: Once the approval is done, the application then becomes a part of this server.

**What is Exit and Entry criteria?**

Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.

Exit criterion should be part of test plan and decided in the planning stage.

**Examples of Exit Criteria:**

* Verify if All tests planned have been run.
* Verify if the level of requirement coverage has been met.
* Verify if there are NO Critical or high severity defects that are left outstanding.
* Verify if all high risk areas are completely tested.
* Verify if software development activities are completed within the projected cost.
* Verify if software development activities are completed within the projected timelines.

**Entry criterion** is used to determine when a given test activity should start. It also includes the beginning of a level of testing, when test design or when test execution is ready to start.

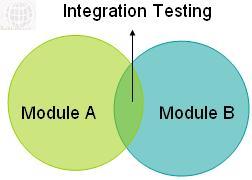
**Examples for Entry Criterion:**

* Verify if the Test environment is available and ready for use.
* Verify if test tools installed in the environment are ready for use.
* Verify if Testable code is available.
* Verify if Test Data is available and validated for correctness of Data.

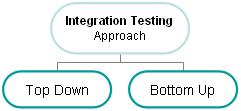
**What is integration testing?**

Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.

* Also after integrating two different components together we do the integration testing. As displayed in the image below when two different modules ‘Module A’ and ‘Module B’ are integrated then the integration testing is done.



* Integration testing is done by a specific integration tester or test team.
* Integration testing follows two approach known as ‘Top Down’ approach and ‘Bottom Up’ approach as shown in the image below:



**What is priority and severity in defect?**

**Severity** is defined as the degree of impact a defect has on the development or operation of a component application being tested.

Higher effect on the system functionality will lead to the assignment of higher severity to the bug. Quality Assurance engineer usually determines the severity level of defect

**Priority** is defined as the order in which a defect should be fixed. Higher the priority the sooner the defect should be resolved.

Defects that leave the software system unusable are given higher priority over defects that cause a small functionality of the software to fail.

Defect severity can be categorized into four class

**Critical:** This defect indicates complete shut-down of the process, nothing can proceed further

**Major:** It is a highly severe defect and collapse the system. However, certain parts of the system remain functional

**Medium:** It cause some undesirable behavior, but the system is still functional

**Low:** It won't cause any major break-down of the system

Defect priority can be categorized into three class

**Low:** The defect is an irritant but repair can be done once the more serious defect have been fixed

**Medium:** During the normal course of the development activities defect should be resolved. It can wait until a new version is created

**High:** The defect must be resolved as soon as possible as it affects the system severely and cannot be used until it is fixed

**What is product back log items?**

When your product owner defines a product backlog item, he or she should focus on its value to your customer and avoid descriptions of how your team develops a feature. The product owner can prioritize your product backlog based on each item’s business value, effort, and relative dependency on other backlog items. The product backlog will evolve quickly if the business requirements of your project and other conditions of your team change constantly. To minimize redundant work, your team can specify details only for the highest priority items.

**What is production environment?**

Production environment is a term used mostly by developers to describe the setting where software and other products are actually put into operation for their intended uses by end users. A production environment can be thought of as a real-time setting where programs are run and hardware setups are installed and relied on for organization or commercial daily operations.

**What is QA environment?**

A QA environment is where you test your upgrade procedure against data, hardware, and software that closely simulate the Production environment and where you allow intended users to test the resulting Waveset application.

**What is V-model?**

The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

**V- Model design**

Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the .V. and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

The below figure illustrates the different phases in V-Model of SDLC.

SDLC V-Model



Verification Phases

Following are the Verification phases in V-Model:

· Business Requirement Analysis: This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

· System Design: Once you have the clear and detailed product requirements, it.s time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.

· Architectural Design: Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

· Module Design:In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

Following are the Validation phases in V-Model:

· Unit Testing: Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

· Integration Testing: Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

· System Testing: System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.

· Acceptance Testing: Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non functional issues such as load and performance defects in the actual user environment.

**What is waterfall method?**

The waterfall model is a sequential (non-iterative) design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance. Despite the development of new software development process models, the waterfall method is still the dominant process model with over a third of software developers still using it.

**Waterfall Model design**

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.

SDLC Waterfall Model



The sequential phases in Waterfall model are:

· Requirement Gathering and analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

· System Design: The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

· Implementation: With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

· Integration and Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

· Deployment of system: Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.

· Maintenance: There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

**What tester will do in each phase of SDLC?**

Ans: The Role of a Tester in SDLC

1. Tester prepares the Test cases, Test Scenarios from the SRS

2. Using the script the tester performs different kinds of testing (Regression, Function)

3. Tester Notes the results (pass/Fail)

4. If Result=Fail then the scenario is raised in the Test director

5. Once it’s fixed by the developer the tester performs a regression testing

Roles and Responsibilities of a Tester are as follows. In the test planning and preparation phases of the testing, testers should review and contribute to test plans, as well as analyzing, reviewing and assessing requirements and design specifications. They may be involved in or even be the primary people identifying test conditions and creating test designs, test cases, test procedure specifications and test data, and may automate or help to automate the tests.

· They often set up the test environments or assist system administration and network management staff in doing so.

· As test execution begins, the number of testers often increases, starting with the work required to implement tests in the test environment.

· Testers execute and log the tests, evaluate the results and document problems found.

· They monitor the testing and the test environment, often using tools for this task, and often gather performance metrics.

· Throughout the software testing life cycle, they review each other’s work, including test specifications, defect reports and test results.

**What types of application we test?**

Ans: Technologies that mainly we use in testing are

· Html and xhtml

· Xml and the document object model.

· Xml-rpc,soap,wsdl,uddi

· SNMP

· SMTP

· SQLnet

· WAP standards(WML,WMLScript,WTP,wdp,etc.)

· HTTP & SSL/TLS

· C, C++, Visual Basic, Java, J2EE, Java Beans, JavaScript, and EcmaScript.

Testing is conducted on mainly on 2 applications

1. Window based applications

2. Web based applications

GUI application testing for windows and UNIX

CD-ROM testing

XML-based application testing

Java application testing

N-tier client-server application testing

WAP application testing

**What we will do if come across any critical severity issue before release day?**

This is a simple, if uncomfortable, situation to be in. Unfortunately, it does happen from time to time and you need to be ready for it.

The fact that the defect has been found close to the deadline is, in the short term, irrelevant. Your team has found a high severity defect, so you report it. Given the short timescales, you ensure that everyone who needs to know about it knows about it, so they have the information they need to determine -their- best course of action as soon as possible.

You must -absolutely- not ever hold off from reporting an issue, at least to your local management structure. That would, at the very least, ruin the reputation of your team and could potentially have much more serious consequences.

The next thing to do is to determine the answer to the obvious question: "Why was this found so late?". There are many reasons why this situation could arise - your test preparation could've been too light, you could've mis-prioritised some work, there may simply have been too much to do. As a member of the test team, you need to know what caused the issue and therefore how you can reduce the risk of it happening again.

Obviously, we don't live in a perfect world and it's possible that no action may be taken to resolve the defect before release. It's entirely possible that it makes more sense to go live with a bug and then release a quick fix, than to decide not to release at all. That's a different discussion, though.

**What we will do if we don’t have a time to test all execute test cases??**

Important to prioritize the tests from the beginning of project onwards. Especially in Agile world, Customers are expecting some executable piece in regular basis. Waiting till the end to prioritize the tests would lead to missing of key test cases.

We used to categorize the tests as P1, P2, P3 and P4. All the test cases were automated. But P1 and P2 are the tests were scheduled to execute on daily basis and P3 and P4 were added at the time of sprint closure. Those which cannot automate will be marked as Manual.

What happens if there is not enough time to test before delivery:

As all the test cases were automated and P1 and P2 will be executed on daily basis on the latest build, so there wont be much problems though we need to deliver the products within short span of duration.

Manual QA will focus on non-automated test case execution along with ad-hoc testing where as Automation will take care about P1 and P2 test cases execution. And Automation testing doesn’t involve any resource time and its purely machine time, it wont impact the delivery.

**When do we use automation testing?**

Test Automation tools help teams test faster, allows them to test substantially more code, improves test accuracy, and frees up QA engineers so they can focus on tests that require manual attention and their unique human skills.

Use these top tips to ensure that your software testing is successful and you get the maximum return on investment (ROI):

1. Decide what Test Cases to Automate

2. Test Early and Test Often

3. Select the Right Automated Testing Tool

4. Divide your Automated Testing Efforts

5. Create Good, Quality Test Data

6. Create Automated Tests that are Resistant to Changes in the UI

It is impossible to automate all testing, so it is important to determine what test cases should be automated first.

The benefit of automated testing is linked to how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are ones that are run frequently and require large amounts of data to perform the same action.

You can get the most benefit out of your automated testing efforts by automating:

· Repetitive tests that run for multiple builds.

· Tests that tend to cause human error.

· Tests that require multiple data sets.

· Frequently used functionality that introduces high risk conditions.

· Tests that are impossible to perform manually.

· Tests that run on several different hardware or software platforms and configurations.

· Tests that take a lot of effort and time when manual testing.

Success in test automation requires careful planning and design work. Start out by creating an automation plan. This allows you to identify the initial set of tests to automate, and serve as a guide for future tests. First, you should define your goal for automated testing and determine which types of tests to automate. There are a few different types of testing, and each has its place in the testing process. For instance, unit testing is used to test a small part of the intended application. To test a certain piece of the application’s UI, you would use functional or GUI testing.

After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Don’t just create test steps that test various aspects of the application’s behavior at one time. Large, complex automated tests are difficult to edit and debug. It is best to divide your tests into several logical, smaller tests. It makes your test environment more coherent and manageable and allows you to share test code, test data and processes. You will get more opportunities to update your automated tests just by adding small tests that address new functionality. Test the functionality of your application as you add it, rather than waiting until the whole feature is implemented.

When creating tests, try to keep them small and focused on one objective. For example, separate tests for read-only versus read/write tests. This allows you to use these individual tests repeatedly without including them in every automated test.

Once you create several simple automated tests, you can group your tests into one, larger automated test. You can organize automated tests by the application’s functional area, major/minor division in the application, common functions or a base set of test data. If an automated test refers to other tests, you may need to create a test tree, where you can run tests in a specific order.

**Test Early and Test Often**

To get the most out of your automated testing, testing should be started as early as possible and ran as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find. Automated unit testing can be implemented on day one and then you can gradually build your automated test suite. Bugs detected early are a lot cheaper to fix than those discovered later in production or deployment.

**When do we use integration testing?**

Integration testing is a software testing methodology used to test individual software components or units of code to verify interaction between various software components and detect interface defects. Components are tested as a single group or organized in an iterative manner. After the integration testing has been performed on the components, they are readily available for system testing.

Integration is a key software development life cycle (SDLC) strategy. Generally, small software systems are integrated and tested in a single phase, whereas larger systems involve several integration phases to build a complete system, such as integrating modules into low-level subsystems for integration with larger subsystems. Integration testing encompasses all aspects of a software system's performance, functionality and reliability.

Most unit-tested software systems are comprised of integrated components that are tested for error isolation due to grouping. Module details are presumed accurate, but prior to integration testing, each module is separately tested via partial component implementation, also known as a stub.

The three main integration testing strategies are as follows:

* Big Bang: Involves integrating the modules to build a complete software system. This is considered a high-risk approach because it requires proper documentation to prevent failure.
* Bottom-Up: Involves low-level component testing, followed by high-level components. Testing continues until all hierarchical components are tested. Bottom-up testing facilitates efficient error detection.
* Top-Down: Involves testing the top integrated modules first. Subsystems are tested individually. Top-down testing facilitates detection of lost module branch links.

**When do we use regression testing?**

Whenever developers change or modify their software, even a small tweak can have unexpected consequences. Regression testing is testing existing software applications to make sure that a change or addition hasn’t broken any existing functionality. Its purpose is to catch bugs that may have been accidentally introduced into a new build or release candidate, and to ensure that previously eradicated bugs continue to stay dead. By re-running testing scenarios that were originally scripted when known problems were first fixed, you can make sure that any new changes to an application haven’t resulted in a regression, or caused components that formerly worked to fail. Such tests can be performed manually on small projects, but in most cases repeating a suite of tests each time an update is made is too time-consuming and complicated to consider, so an automated testing tool is typically required.

Understanding Regression Tests

Some software development teams try to get by without performing regular regression tests, opting to test essential functions just once to make sure they work and, if they check out, proceeding with the hopeful assumption that those functions will still work unless they’re directly modified again. In a way, this makes sense: it’s natural to want to simply make a change, test it, and move on. Performing functional tests or highly specific unit tests to determine that a new software component works as it should has been called “non-regression testing” by Doug Hoffman and others. But it can be relatively easy to find a specific problem when you’re looking for it; what’s harder is catching all the ones you don’t expect.

Again, it’s important for developers and testers to always bear in mind that even small, seemingly insignificant alterations to an application’s source code can ripple outward in surprising ways, breaking functions that seem completely unrelated to the new modification. When you run regression tests, you’re checking to make sure that your modification not only behaves as you want it to, but that it also hasn’t inadvertently caused problems in functions that had otherwise worked correctly when previously tested.

Fortunately for the would-be regression tester, on any given project your regression test libraries can be built from the existing test cases developed from day one. Functional tests, unit tests, integration tests, and build verification tests—anything that has successfully verified, throughout the development process, that various components work as intended—can all be incorporated into a regression testing suite, and “regression tests,” per se, don’t necessarily need to be written. Each time you modify your source code, you can simply re-run the potentially relevant tests to ensure that they continue to pass. Naturally, over the course of a complex development project, those test cases—and the various functions and processes that they attempt to check—can number in the thousands, making the use of automated testing software mandatory for full-scale regression tests. TestComplete can help you reuse your previous automated tests to easily create continuous regression tests.

**When do we use smoke testing and sanitary testing?**

Smoke testing is a more generalized, high-level approach to testing, while sanity testing is more particular and focused on fine-grain details.

Smoke testing

Basics of smoke testing: Name and function

The name is definitely unusual, but it makes sense:

In fact, the term originates with hardware testing.

Test engineers who turn on a PC, server or storage appliance check for literal smoke coming from the components once the power is running. If no smoke is detected, the test is passed; if not, all other project-related work has to be put on hold until it passes.

As we can see, the idea is to verify that the most basic functionality - the ability to be powered on without catching on fire in the case of hardware, successful startup and connection to various necessary libraries, services, etc. for software - is operating properly before additional testing is undertaken.

Smoke testing usually takes place at the beginning of the software testing lifecycle. It verifies the quality of a build - i.e., a collection of files that make up (or "build") a program - and checks to see if basic tasks can be properly executed. The idea is to ensure that the initial build is stable; if it can't pass a smoke test, then it has to be redone before the actual testing phase can be pursued. At some organizations, smoke testing is called build verification testing.

"In smoke testing, the test cases chosen cover the most important functionality or component of the system," explained a guide from Guru99. "The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system [are] working fine. For example a typical smoke test would be – verify that the application launches successfully, check that the GUI is responsive, etc."

Performing smoke tests

A smoke test might be done manually or be automated. So you could create manual test cases or come up with a script that would automatically check to see if the software could be installed and launched without incident. An enterprise test management suite can be used to help with your smoke tests.

**Sanity testing**

Sanity testing is sometimes called a sanity check. Like a literal sanity check, it is not meant to be exhaustive and instead supposed to verify that recent changes are not causing any major problems. The "sanity" in the name just refers to the idea of making sure that developers were rational and sane when updating an application.

Basics of sanity testing: Differences from smoke testing

It is common to categorize sanity testing as a subset of acceptance testing, which is a much more thorough process. Sanity testing:

Is usually done near the end of a test cycle, to see if bugs have been fixed and if any minor changes to the code are being well tolerated.

Is typically executed after receiving a new build, to see if the most recent fixes break any part of the application.

Is often unscripted and may take a "narrow and deep" approach as opposed to the "wide and shallow" route of smoke testing.

Helps determine that an app can do the basics. So a sanity check could be used to see if a calculator app can give a correct result for 2 + 2; if it can't, then there's no point yet in doing further tests on its ability to handle more advanced things like trigonometric functions.

Can be done manually or with the help of automated tools.

As we can see, there is some overlap between smoke testing and sanity testing, especially when it comes to the fact that neither is really designed to be a thorough process. However, there are also obvious and important differences.

Developers and testers rely on smoke and sanity testing to move through application development and deployment with as few delays and technical errors as possible. Overall, we can look at smoke testing and sanity testing as similar processes at the opposite ends of a test cycle. Smoke testing ensures that the fundamentals of the software are sound so that more in-depth testing can be conducted, while sanity testing looks back to see if the changes made after additional development and testing broke anything.

**When do we use white box testing and black box testing?**

Two common types of testing are black-box and white-box testing. Both can drive or be driven by development.

Black-box testing: Black-box testing (also known as functional testing) treats software under test as a black-box without knowing its internals. Tests are using software interfaces and trying to ensure that they work as expected. As long as functionality of interfaces remains unchanged, tests should pass even if internals are changed. Tester is aware of what the program should do but does not have the knowledge of how it does it. Black-box testing is most commonly used type of testing in traditional organizations that have testers as a separate department, especially when they are not proficient in coding and have difficulties to understand the code. It provides external perspective of the software under test.

Some of the advantages of black-box testing are:

1. Efficient for large segments of code

2. Code access is not required

3. Separation between user’s and developer’s perspectives

Some of the disadvantages of black-box testing are:

1. Limited coverage since only a fraction of test scenarios is performed

2. Inefficient testing due to tester’s luck of knowledge about software internals

3. Blind coverage since tester has limited knowledge about the application

If tests are driving the development, they are often done in the form of acceptance criteria that is later used as definition of what should be developed. In that case black-box testing relies on some form of automation like Behavior Driven Development.

White-box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) looks inside the software that is being tested and uses that knowledge as part of the testing process. If, for example, exception is thrown under certain conditions, test might want to reproduce those conditions. White-box testing requires internal knowledge of the system and programming skills. It provides internal perspective of the software under test.

Some of the advantages of white-box testing are:

1. Efficient in finding errors and problems

2. Required knowledge of internals of the software under test is beneficial for thorough testing

3. Allows finding hidden errors

4. Programmers introspection

5. Helps optimizing the code

6. Due to required internal knowledge of the software, maximum coverage is obtained

Some of the disadvantages of white-box testing are:

1. Might not find unimplemented or missing features

2. Requires high level knowledge of internals of the software under test

3. Requires code access

White-box testing is almost always automated and in most cases has the form of unit tests. If done before the development, it takes the form of Test Driven Development (TDD).

**Who will assign the work?**

Projects of different sizes have different needs for how the people are organized. In a small project, little organization structure is needed. There might be a primary sponsor, project manager and a project team. However, for large projects, there are more and more people involved, and it is important that people understand what they are expected to do, and what role people are expected to fill. This section identifies some of the common (and not so common) project roles that may be required for your project.

Analyst

The Analyst is responsible for ensuring that the requirements of the business clients are captured and documented correctly before a solution is developed and implemented. In some companies, this person might be called a Business Analyst, Business Systems Analyst, Systems Analyst or Requirements Analyst. For more information on this role see 408.2 The Role of an Analyst.

Change Control Board

The Change Control Board is usually made up of a group of decision makers authorized to accept changes to the projects requirements, budget, and timelines. This organization would be helpful if the project directly impacted a number of functional areas and the sponsor wanted to share the scope change authority with this broader group. The details of the Change Control Board and the processes they follow are defined in the project management processes.

Client

This is the people (or groups) that are the direct beneficiaries of a project or service. They are the people for whom the project is being undertaken. (Indirect beneficiaries are probably stakeholders.) These might also be called "customers", but if they are internal to the company, LifecycleStep refers to them generically as clients. If they are outside your company, they would be referred to as "customers".

Client Project Manager

If the project is large enough, the business client may have a primary contact that is designated as a comparable project manager for work on the client side. The IT project manager would have overall responsibility for the IT solution. However, there may be projects on the client side that are also needed to support the initiative, and the client project manager would be responsible for those. The IT project manager and the client project manager would be peers who work together to build and implement the complete solution.

Database Administrator

A Database Administrator is a specialist that models, designs and creates the databases and tables used by a software solution. This role combines Data Administrator (logical) and DBA (physical). For more information on this role, see 408.8 The Role of the Database Administrator.

Designer

The designer is responsible for understanding the business requirements and designing a solution that will meet the business needs. There are many potential solutions that will meet the client's needs. The designer determines the best approach. A designer typically needs to understand how technology can be used to create this optimum solution for the client. The designer determines the overall model and framework for the solution, down to the level of designing screens, reports, programs and other components. They also determine the data needs. The work of the designer is then handed off to the programmers and other people who will construct the solution based on the design specifications. For more information on this role, see 408.5 The Role of a Designer.

Developer

The Developer is responsible for the actual building of the solution. For more information on this role, see 408.6 The Role of the Developer.

Project Manager

This is the person with authority to manage a project. This includes leading the planning and the development of all project deliverables. The project manager is responsible for managing the budget and schedule and all project management procedures (scope management, issues management, risk management, etc.). For more information on this role see 408.1 The Role of a Project Manager.

Project Team

The project team consists of the full-time and part-time resources assigned to work on the deliverables of the project. This includes the analysts, designers, programmers, etc. They are responsible for:

Understanding the work to be completed

Planning the assigned activities in more detail if needed

Completing assigned work within the budget, timeline and quality expectations

Informing the project manager of issues, scope changes, risk and quality concerns

Proactively communicating status and managing expectations

The project team can consist of staff within one functional organization, or it can consist of members from many different functional organizations. A cross-functional team has members from multiple organizations. Having a cross-functional team is usually a sign that your organization is utilizing matrix management.

Quality Manager

On a large project, quality management could take up a large amount of project management time. In this case, it could be worthwhile to appoint someone as quality manager. For more information on this role, see 408.9 The Role of a Quality Manager.

Sponsor (Executive Sponsor and Project Sponsor)

This is the person who has ultimate authority over the project. The Executive Sponsor provides project funding, resolves issues and scope changes, approves major deliverables and provides high-level direction. They also champion the project within their organization. Depending on the project and the organizational level of the Executive Sponsor, they may delegate day-to-day tactical management to a Project Sponsor. If assigned, the Project Sponsor represents the Executive Sponsor on a day-to-day basis and makes most of the decisions requiring sponsor approval. If the decision is large enough, the Project Sponsor will take it to the Executive Sponsor for resolution. For more information on this role, see 408.3 The Role of the Project Sponsor.

Stakeholder

These are the specific people or groups who have a stake, or an interest, in the outcome of the project. Normally stakeholders are from within the company, and could include internal clients, management, employees, administrators, etc. A project may also have external stakeholders, including suppliers, investors, community groups and government organizations.

Steering Committee

A Steering Committee is a group of high-level stakeholders who are responsible for providing guidance on overall strategic direction. They do not take the place of a Sponsor, but help to spread the strategic input and buy-in to a larger portion of the organization. The Steering Committee is usually made up of organizational peers and is a combination of direct clients and indirect stakeholders. Some members on the Steering Committee may also sit on the Change Control Board.

Subject Matter Expert

A Subject Matter Expert (SME) has superior (expert) knowledge of a discipline, technology, product, business process or entire business area. For more information on this role, see 408.4 The Role of the SME.

Suppliers / Vendors

Suppliers and vendors are third party companies or specific people that work for third parties. They may be subcontractors who are working under your direction, or they may be supplying material, equipment, hardware, software or supplies to your project. Depending on their role, they may need to be identified on your organization chart. For instance, if you are partnering with a supplier to develop your requirements, you probably want them on your organization chart. On the other hand, if the vendor is supplying a common piece of hardware, you probably would not consider them a part of the team.

Tester

The Tester ensures that the solution meets the business requirements and that it is free of errors and defects.

Users

These are the people who will actually use the deliverables of the project. These people may also be involved heavily in the project in activities such as defining business requirements. In other cases, they may not get involved until the testing process. Sometimes you want to specifically identify the user organization or the specific users of the solution and assign a formal set of responsibilities to them, like developing use cases or user scenarios based on the needs of the business requirements.

**Why software testing is required?**

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Software testing is very important because of the following reasons:

* Software testing is really required to point out the defects and errors that were made during the development phases.
* It’s essential since it makes sure of the Customer’s reliability and their satisfaction in the application.
* It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence. (Know more about Software Quality)
* Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
* Testing is required for an effective performance of software application or product.
* 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.
* It’s required to stay in the business.