Software Testing (Unit-IV)

Software testing can be stated as the process of verifying and validating that software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

Verification: it refers to the set of tasks that ensure that software correctly implements a specific function.
Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: "Are we building the product right?"
Validation: "Are we building the right product?"

What are different types of software testing?

Software Testing can be broadly classified into two types:

1. Manual Testing: Manual testing includes testing software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behaviour or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing. Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.
2. Automation Testing: Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly. Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

Why Software Testing?

Software that does not work correctly can lead to many problems such as:
Delay / Loss of time
Futility / Loss of effort
Wastage / Loss of money
Shame / Loss of business reputation

Injury or death
Testing helps in ensuring that software work correctly and reduces the risk of software failure, thereby avoiding the problems mentioned above.

Software Testing Goals

The three main goals of Software Testing are:
Defect Detection: Find defects / bugs in the software during all stages of its development (earlier, the better).
Defect Prevention: As a consequence of defect detection, help anticipate and prevent defects from occurring at later stages of development or from recurring in the future.
User Satisfaction: Ensure customers / users are satisfied that their requirements (explicit or implicit) are met.

The Psychology of Testing

Software development, including software testing, involves human beings. Therefore, human psychology has important effect on software testing. In software testing, psychology plays an extremely important role. It is one of those factors that stay behind the scene, but has a great impact on the end result. Categorized into three sections, the psychology of testing enables smooth testing as well as makes the process hassle-free. It is mainly dependent on the mindset of the developers and testers, as well as the quality of communication between them. Moreover, the psychology of testing improves mutual understanding among team members and helps them work towards a common goal.

The three sections of the psychology of testing are:

The mindset of Developers and Testers
The software development life cycle is a combination of various activities, which are performed by different individuals using their expertise and knowledge. It is not an unknown fact that to accomplish the success, development of software, people with different skills and mindset are required. Developers synthesize code. They build up things, putting pieces together and figuring out fun and unique ways of combining those distinct little bits to do wonderful and amazing things. But Testers are all about analysis. Once it has all been put together, the tester likes to take it apart again, piece by piece, this time looking for those little corners, edges, and absurdities that hide in those weird and strange interactions that come from those new and amazing ways of putting pieces together. Testing and Reviewing the applications are different from analyzing and developing it. While testing or reviewing a product, testers mainly look for defects or failures in the product. If we are building or developing applications, we have to work positively to solve the problems during the development process and to make the product according to the user specification. As an example,
Identifying defects during static testing such as requirements review or user story refinement session or identifying failures during dynamic test execution, may be perceived as Criticism of the product and of its author.
So the developer or the analyst may have problems with you as a tester because he thinks that you are criticizing them. There's an element of human psychology called Confirmation bias, which means that most of people find it difficult to accept information that disagrees with

currently held believes.
For example, since developers expect their code to be correct, they have a confirmation bias that makes it difficult to accept that the code is incorrect.
In addition to confirmation bias, other cognitive biases may make it difficult for people to understand or accept information produced by testing.
Further, it is a common human trait to blame the bearer of bad news and information produced by testing often contains bad news.
So as a result of these psychological factors, some people may perceive testing as a destructive activity, even though it contributes greatly to project progress and product quality.
To try to reduce these perceptions, information about defects and failures should be communicated in a Constructive Way.
This way, tensions between the testers and the analysts, product owners, designers, and developers can be reduced. This applies during both static and dynamic testing in which Static Testing is a software testing technique where the software is tested without executing the code and Dynamic Testing will use the dynamic behaviour of the code by opening the application and run it.

Communication in a Constructive Manner

Testers and test managers need to have good interpersonal skills to be able to communicate effectively about the defects, failures, test results, test progress and risks and to build positive relationships among colleagues. Communication, if done in a polite and respectful manner can help build a strong and reliable relationship between the team members and help them avoid any misunderstanding. Similarly during the process of testing also, the requirement of communication in a constructive manner is extremely high. As testers are responsible for finding bugs and reporting them, it becomes vitally important for them to communicate it in a respectful, polite, and suitable way to avoid hurting and disappointing someone.
Finding and reporting defects can be an achievement for the tester but is merely an inconvenience for programmers, designers, developers, and other stakeholders of the project. As testing can be a destructive activity, it is important for software testers to report defects and failures as objectively and politely as possible to avoid hurting and upsetting other members related to the project.

So as testers, there are many ways to communicate collaboratively with developers. Start with collaboration rather than battels. Remind everyone about their common goal of having better quality systems. Emphasize the benefits of testing.
For example, for the authors, defect information can help them to improve the work products and their skills. For the organization, defects found and fixed during testing will save money, time and reduce overall risk to product quality. Communicate test results and other findings in a neutral, fact focused way without criticizing the person who created the defected item. Write objectives, defect reports and review findings. Try to understand how the other person feels and the reasons they may react negatively to the information. Confirm that the other person has understood what has been said. Clearly define the right set of test objectives has important psychological implications. Most people tend to align their plans and behaviors with the objectives set by the team, management, and stakeholders. So it’s also important that testers adhere to these objectives with minimum personnel bias.

Self-testing and Independent testing

Comparison of the mindsets of a tester and a programmer does not mean that a tester cannot be a programmer, or that a programmer cannot be the tester, although they often are separate roles. In fact, programmers are the testers. They always test the component which they built. While testing their own code they find many problems so the programmers, architect, and developers always test their own code before giving it to anyone. So, programmers, architects, business analysts depend on others to help test their work. This other person might be some other developer from the same team or the Testing specialists or professional testers. Giving applications to the testing specialists or professional testers allows an independent test of the system.

As this type of testing is mainly performed by individuals, who are not related to the project directly or are from a different organization, they are hired mainly to test the quality as well as the effectiveness of the developed product. Test independence, therefore, helps developers and other stakeholders get more accurate test results, which helps them build a better software product, with innovative and unique features and functionality.

There are several levels of independence in software testing which is listed here from the lowest level of independence to the highest:

i. Tests by the person who wrote the item.

ii. Tests by another person within the same team, like another programmer.

iii. Tests by the person from some different groups such as an independent test team.

iv. Tests by a person from a different organization or company, such as outsourced testing or certification by an external body.

To achieve successful testing it's important for the software engineers to consider the psychological aspects of testing, as they can interrupt the result of testing and can impact the performance of the end product. The importance of these technological aspects is similar to any other tool or technique adopted for the process of testing.

The mindset of the team members, the communication between them, and most importantly the test independence while performing the testing is crucial and needs great consideration. Therefore, if you want your software testing to be free of any hurdles, do consider the psychological aspects stated above.

7 Principles of Software Testing

It is important that you achieve optimum test results while conducting software testing without deviating from the goal. But how you determine that you are following the right strategy for testing? For that, you need to stick to some basic testing principles. Here are the common seven testing principles that are widely practiced in the software industry.

Here are the 7 Principles:

1) Exhaustive testing is not possible

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application. And the million dollar question is, how do you determine this risk? To answer this let's do an exercise, In your opinion, Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time. So, if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle Defect Clustering.

2) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects. To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let's see this video of the public launch of Windows 98. You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

4) Testing shows a presence of defects

Hence, testing principle states that – Testing talks about the presence of defects and don't talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

This leads us to our next principle, which states that- Absence of Error.

5) Absence of Error – fallacy

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.

To solve this problem, the next principle of testing states that Early Testing.

6) Early Testing

Early Testing – Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined. More on this principle in a later training tutorial.

7) Testing is context dependent

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software's are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.
Myth: "Principles are just for reference. I will not use them in practice"
This is so very untrue. Test Principles will help you create an effective Test Strategy and draft error catching test cases.
But learning testing principles is just like learning to drive for the first time. Initially, while you learn to drive, you pay attention to each and everything like gear shifts, speed, clutch handling, etc. But with experience, you just focus on driving the rest comes naturally. Such that you even hold conversations with other passengers in the car.
Same is true for testing principles. Experienced testers have internalized these principles to a level that they apply them even without thinking. Hence the myth that the principles are not used in practice is simply not true.

Walkthrough:
Walkthrough is a method of conducting informal group/individual review. In a walkthrough, author describes and explain work product in an informal meeting to his peers or supervisor to get feedback. Here, validity of the proposed solution for work product is checked.
It is cheaper to make changes when design is on the paper rather than at time of conversion. Walkthrough is a static method of quality assurance. Walkthrough are informal meetings but with purpose.
Inspection:
An inspection is defined as formal, rigorous, in depth group review designed to identify problems as close to their point of origin as possible. Inspections improve reliability, availability, and maintainability of software product. Anything readable that is produced during the software development can be inspected. Inspections can be combined with structured, systematic testing to provide a powerful tool for creating defect-free programs. Inspection activity follows a specified process and participants play well-defined roles. An inspection team consists of three to eight members who plays roles of moderator, author, reader, recorder and inspector.
For example, designer can acts as inspector during code inspections while a quality assurance representative can act as standard enforcer.

Stages in the inspections process :

Planning : Inspection is planned by moderator.
Overview meeting : Author describes background of work product.
Preparation : Each inspector examines work product to identify possible defects.
Inspection meeting : During this meeting, reader reads through work product, part by part and inspectors points out the defects for every part.
Rework : Author makes changes to work product according to action plans from the inspection meeting.
Follow-up : Changes made by author are checked to make sure that everything is correct.

Difference between Inspection and Walkthrough :

S.N.	Inspection	Walkthrough
1.	It is formal.	It is informal.
2.	Initiated by project team.	Initiated by author.
3.	A group of relevant persons from different departments participate in the inspection.	Usually team members of the same project take participation in the walkthrough. Author himself acts walkthrough leader.
4.	Checklist is used to find faults.	No checklist is used in the walkthrough.
5.	Inspection processes includes overview, preparation, inspection, and rework and follow up.	Walkthrough process includes overview, little or no preparation, little or no preparation examination (actual walkthrough meeting), and rework and follow up.
6.	Formalized procedure in each step.	No formalized procedure in the steps.
7.	Inspection takes longer time as list of items in checklist is tracked to completion.	Shorter time is spent on walkthrough as there is no formal checklist used to evaluate program.
8.	Planned meeting with the fixed roles assigned to all the members involved.	Unplanned
9.	Reader reads product code. Everyone inspects it and comes up with detects.	Author reads product code and his teammate comes up with the defects or suggestions.
10.	Recorder records the defects.	Author make a note of defects and suggestions offered by teammate.
11.	Moderator has a role that moderator making sure that the discussions proceed on the productive lines.	Informal, so there is no moderator.

Desk Checking:

A third human error-detection process is the older practice of desk checking. A desk check can be viewed as a one-person inspection or walkthrough: A person reads a program, checks it with respect to an error list, and/or walks test data through it.
For most people, desk checking is relatively unproductive. One reason is that it is a completely undisciplined process. A second, and more important, reason is that it runs counter to a testing principle of Chapter 2—the principal that people are generally ineffective in test- ing their own programs. For this reason, you could deduce that desk checking is best performed by a person other than the author of the program (e.g., two programmers might swap programs rather than desk check their own programs), but even this is less effective than the walkthrough or inspection process. The reason is the synergistic effect of the walkthrough or inspection team. The team session fos- ters a healthy environment of competition; people like to show off by finding errors. In a desk-checking process, since there is no one to whom you can show off, this apparently valuable effect is missing. In short, desk checking may be more valuable than doing nothing at all, but it is much less effective than the inspection or walkthrough.

Peer Ratings:
The last human review process is not associated with program testing (i.e., its objective is not to find errors). This process is included here, however, because it is related to the idea of code reading. Peer rating is a technique of evaluating anonymous programs in terms of their overall quality, maintainability, extensibility, usability, and clarity. The purpose of the technique is to provide programmer self-evaluation. A programmer is selected to serve as an administrator of the process. The administrator, in turn, selects approximately 6 to 20 participants (6 is the minimum to preserve anonymity). The participants are expected to have similar backgrounds (you shouldn't group Java application programmers with assembly language system programmers, for example). Each participant is asked to select two of his or her own programs to be reviewed. One program should be representative of what the participant considers to be his or her finest work; the other should be a program that the programmer considers to be poorer in quality.
Once the programs have been collected, they are randomly dis- tributed to the participants. Each participant is given four programs to review. Two of the programs are the "finest" programs and two are "poorer" programs, but the reviewer is not told which is which. Each participant spends 30 minutes with each program and then completes an evaluation form after reviewing the program. After reviewing all four programs, each participant rates the relative quality of the four programs. The evaluation form asks the reviewer to answer, on a scale from 1 to 7 (1 meaning definitely "yes," 7 meaning definitely "no"), such questions as these:

- Was the program easy to understand?
- Was the high-level design visible and reasonable?
- Was the low-level design visible and reasonable?
- Would it be easy for you to modify this program?
- Would you be proud to have written this program?

The reviewer also is asked for general comments and suggested improvements.
After the review, the participants are given the anonymous evaluation forms for their two contributed programs. The participants also are given a statistical summary showing the overall and detailed ranking of their original programs across the entire set of programs, as well as an analysis of how their ratings of other programs compared with those ratings of other reviewers

of the same program. The purpose of the process is to allow programmers to self-assess their programming skills. As such, the process appears to be useful in both industrial and classroom environments.

What is a Software Bug?
A Software Bug is a failure or flaw in a program that produces undesired or incorrect results. It's an error that prevents the application from functioning as it should.

Why Does Software Have Bugs?
There are many reasons for the occurrence of Software Bugs. The most common reason is human mistakes in software design and coding. Once you get to know the causes for Software Defects, then it will be easier for you to take corrective actions to minimize these defects. For a tester, the failure to report a bug/defect isn't a good. Yet, it may not be the tester who's entirely to blame since several other factors can lead to defects leaking into production.

Here are the top ten reasons.

Communication problems: Miscommunication or lack of communication during various stages of the software development process (right from requirement gathering to its interpretation/documentation or translation stage) can make defects go unnoticed and/or unreported.

Human errors: Humans are prone to errors and quite naturally, expecting the products they develop to be flawless and without errors/defects would be foolish. That explains why bugs/defects may creep up in software. But despite the possibility of human-induced errors, humans still continue to develop a better product than any other non-human agent. So, unless we find one that can do the job better than humans, we need to rely on human intelligence and thus risk the chance of errors.

Impractical development timeframe: Insufficient or limited resources, unrealistic release schedules, and project deadlines – software developers often have to face some or all of them. As a result, they probably have to make certain compromises (such as not investing adequate time for designing, not testing their code giving it to the testing team etc), which can increase the chances of bugs/errors. Design/feature changes at a much later stage of the SDLC too can boost the chances of errors.

Faulty design logic: As software applications are getting increasingly complicated, some level of brainstorming as well as R&D is needed to arrive at a reliable solution. Yet, the strong desire to get the job completed at the earliest, lack of patience, faulty application of technology (products, components, techniques etc), improper understanding of the technical viability before designing the architecture, the need/temptation to take the fastest and easiest route to implement a solution etc – all can make errors/bugs to creep in.

Bad coding: This involves bad coding practices such as unhandled errors, missing exceptions, and faulty validations of inputs together with the use of bad tools (such as like faulty compilers, debuggers, validators etc. that some programmers are using) – all of which can invite errors in

the code, which may be too tricky to debug. At times, it could just be the case of bad coding that causes errors to slip into the code.

Absence of version control: In case you continue to face lots of regression bugs that keep occurring at regular intervals, you should check the version control system (if there's any). With concurrent version systems, you can keep track of all changes in a set of code base. If there's a complete lack of a version control system, you are likely to encounter lots of regression errors. However, errors may still occur in the final builds despite having a version control system in case the programmers fail to ensure that the most current version of every module is linked when they create a new version to be tested.

Third-party tools containing bugs: Third-party tools such as debuggers, HTML editors, shared DLLs, add-ons/plug-ins to save time (map navigation API, shopping cart plug-in etc) are often needed during software development but may have bugs in them that pass onto your software.

Lack of skilled testing: In several companies, poor testing is often the norm, which may include scarcity of skilled testers, shortcomings in the testing process that's followed, testing being taken lightly, and the process getting conducted without giving it much importance etc. All of these can cause bugs/errors to creep into the software.

Excessive dependability on automation testing: Since automation testing lacks human intuition and experience, being excessively dependent on it may make a manual tester miss a bug/defect. It's important to remember that for successful automation testing, you need experienced testers along with good management.

Last-minute changes: Just before a product release, changes made to the infrastructure, requirement, tools, platform etc can be dangerous. Actions such as testing the compatibility of your software across a variety of browsers and operating systems, database migration etc are complex things and if done in a hurry (due to a last-minute change), they may introduce bugs/ errors in the software/application.

Cost of bugs

Software bugs are the monster under the bed for any tech-first business. Coders, developers, designers and QA professionals are desperate to avoid them at any cost, and when a product ships with critical bugs, it can be a hard pill to swallow. But software bugs are not just an annoyance, as many of us know, they come at a cost to fix too. Considering the cost of a software bug and the amount of effort that goes behind the whole process of testing in order to find bugs, many companies tend to avoid the whole process. But this only leads to the formation of a flawed product, which ultimately will not be serving the purpose.

Why Emphasize On Software Testing So Much?

Software testing is a detailed procedure in which flaws are basically searched for on the website or software. Multiple tests are written, keeping in mind certain situations and criteria which will help in making the app go through different environments, and checking will be done on how it performs under different areas. Accordingly, flaws or bugs will be detected in the

system. If any are present, proper measures will be taken further to eliminate them. Not emphasizing on what is the cost of a software bug, let's see what not performing software testing can cost you:

A flawed system is comprising of disputes. The software or website will not work effectively under different situations and will not function properly on different devices as well.

Degraded user experience. Your targeted audience will find it difficult to navigate through your website and hence will not gravitate towards your website on a regular basis.

Loss in product sale. Since user experience is damaged in the first place, it is obvious that your products will also not get the deserving attention from your targeted audience and will not be sold either.

The cost for development and support increases as the bug creates complications in the system. The whole development process is delayed due to this, and also the requests from the customer support system keep building up. This in turn jams the entire system.

There will be a delay in the process of adding new features to the system because time will be required to fix the previously detected issues. That will ultimately delay the whole process in general.

Customer's credibility is decreased to a huge extent which is one of the main reasons behind the degraded reputation of the company.

There will be bad reviews generated about the company and the products. Due to hampered user experience, the feedback that will be posted on the website will be not positive, obviously.

Cost Of A Software Bug Depending On The Different Stages:

Now let's see what is the cost of a software bug, depending on the different stages where it is detected.

Early Development

Finding the flaw at the stage of the early development of the website or software.

If this is done, then the cost of the software bug will be greatly reduced, going as far as next to zero. Because flaws are found at the initial stage; hence, the complication is not such a huge factor here, and the defect can be easily fixed almost immediately. If the defect is not detected immediately, but by the developer or some other employee at a later stage. In this case, the cost of a software bug will increase because the extra effort is put behind it, and it went out of the developer's grip while it was during development. The moment when a tester enters the process of testing the cost of a software bug will be counted into it.

Detecting The Cause Of The Bug

This is the place that clearly controls the cost of a software bug, and it keeps elevating as the complication in the system also increases.

Fixing The Bug After The Software Or Website Is Launched

As more aspects are added to the system, the amount of complication keeps elevating. And as the complication increases in the system, finding the primary cause for the bug also increases. Hence a lot of effort goes behind first finding the bug (and how to reproduce it), then searching for the cause or the roots of the bug, and lastly fixing it is some other level of work. Due to all of this, the cost of a software bug is also increasing.

Fixing the bug will obviously charge you a good fortune, but other than the cost of a software bug, if it remains undetected in the system, it will keep causing a loss in the sale and hamper your user experience as well. Hence, other than the cost of a software bug, you will be losing a lot of your profits as well.

So if estimated, the cost of a software bug is not something which is light on the pockets, unless it is detected in the initial stage of development. Otherwise, as the stages get complicated and features are added as well, the price keeps elevating.

Direct and Indirect Cost Of Software Bug

Other than the generic cost of a software bug, there are two types of main costs, namely direct and indirect costs.

Direct Costs

This includes the money that the management invests as a whole in the development of the project, including the salary of the employees and developers as well as the quality assurance work.

Indirect Costs

This can be considered an extra cost which you need to avoid as much as possible. If testing is conducted, bugs will obviously remain in the system, the complications it will cause due to its presence will be falling under the indirect costs. To fix all the problems caused by the bug, the amount of money you will spend will be considered as nothing but indirect costs. To avoid it, you need to conduct testing procedures for your project. If the bug is detected in the initial stage or the development stage, then you can easily save a lot of money and also create a project devoid of any minor or major problems in it.

Software Testing Roles and Responsibilities

In case of software testing every company defines its own level of hierarchy, roles and responsibilities.

Test lead/manager: A test lead is responsible for:

Defining the testing activities for subordinates – testers or test engineers.
All responsibilities of test planning.
To check if the team has all the necessary resources to execute the testing activities.
To check if testing is going hand in hand with the software development in all phases.
Prepare the status report of testing activities.
Required Interactions with customers.
Updating project manager regularly about the progress of testing activities.

Test engineers/QA testers/QC testers are responsible for:

To read all the documents and understand what needs to be tested.
Based on the information procured in the above step decide how it is to be tested.
Inform the test lead about what all resources will be required for software testing.
Develop test cases and prioritize testing activities.
Execute all the test case and report defects, define severity and priority for each defect.
Carry out regression testing every time when changes are made to the code to fix defects.

Overview Of Software Engineering Team

How a software application shapes up during the development process entirely depends on the how the software engineering team organizes work and implements various methodologies. For an application to develop properly, it is important that all processes incorporated during the software development are stable and sustainable. Many times developers come under pressure as the delivery date approaches closer this often affects the quality of the software. Rushing through the processes to finish the project on time will only produce a software

application which has no or minimal use for the customers. Hence, work organization and planning is important and sticking to the plan is very important. The project manager should ensure that there are no obstacles in the development process and if at all there is an issue it must be resolved with immediate attention.

Overview Of Software Testing Team

How soon and how well you can achieve your testing goals depends solely on the capabilities of the testing team. Within the testing team itself it is important to have the correct blend of testers who can efficiently work together to achieve the common testing goals. While forming a team for testing, it is important to ensure that the members of the team jointly have a combination of all the relevant domain knowledge that is required to test the software under development.

It is very important to ensure that the software testing team has a proper structure. The hierarchy and roles should be clearly defined and responsibilities too should be well defined and properly distributed amongst the team members. When the team is well organized the work can be handled well. If every team member knows what duties he or she has to perform then they will be able to finish their duties as required well within the time limit. It is important to keep track of the testers' performance. It is very important to check what kind of defects the tester is able to uncover and what kind of detects he tends to miss. This will give you a fair idea about how serious your team is about the work.

All the team members should work together to prepare a document that clearly defines the roles and responsibilities of all the team members. Once the document is prepared the role of each member should be communicated clearly to everyone. Once the team members are clear about who is going to handle which area of the project, then in case of any issue it will be easy to determine who needs to be contacted.

Each member of the team should be provided with the necessary documents that provide information on how the task would be organized, what approach will be followed, how things are scheduled, how many hours have been allocated to each member and all details related to applicable standards and quality processes.

Software Tester Role

A Software tester (software test engineer) should be capable of designing test suites and should have the ability to understand usability issues. Such a tester is expected to have sound knowledge of software test design and test execution methodologies. It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently.

The roles and responsibilities for a usability software tester are as follows:

A Software Tester is responsible for designing testing scenarios for usability testing.

He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.

He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.

Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.

A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing. He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects. In order to create test cases it is important that the software tester is aware of various testing techniques and which approach is best for a particular system. He should know what are various phases of software testing and how testing should be carried out in each phase. The responsibilities of the software tester include:
Creation of test designs, test processes, test cases and test data.
Carry out testing as per the defined procedures.
Participate in walkthroughs of testing procedures.
Prepare all reports related to software testing carried out.
Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Test Manager Role

Managing or leading a test team is not an easy job. The company expects the test manager to know testing methodologies in detail. A test manager has to take very important decisions regarding the testing environment that is required, how information flow would be managed and how testing procedure would go hand in hand with development. He should have sound knowledge about both manual as well as automated testing so that he can decide how both the methodologies can be put together to test the software. A test manager should have sound knowledge about the business area and the client's requirement, based on that he should be able to design a test strategy, test goal and objectives. He should be good at project planning, task and people coordination, and he should be familiar with various types of testing tools. Many people get confused between the roles and responsibilities of a test manager and test lead.

Software Test Automator Role

Software test automator or an automated test engineer should have very good understanding of what he needs to test- GUI designs, load or stress testing. He should be proficient in automation of software testing, and he should be able to design test suites accordingly. A software test automator should be comfortable using various kinds of automation tools and should be capable of upgrading their skills with changing trends. He should also have programming skills so that he is able to write test scripts without any issues. The responsibilities of a tester at this position are as follows:
He should be able to understand the requirement and design test procedures and test cases for automated software testing.
Design automated test scripts that are reusable.
Ensure that all automated testing related activities are carried out as per the standards defined by the company.

Interactions between Software Test Team And Business Teams

If at all a customer has any issues related to testing activities and operational matters of the project then it is the software testing manager who is responsible for communicating the details to the client regarding how things are being managed. The software testing manager not only answers the queries of the customers but also ensures that the project is completed on time as per the requirement of the customer.

Interactions between Software Test Team And Development Teams

In order to produce good software applications, it is important that software testing and software development teams work together with good understanding. For this it is important that the testers and developers are comfortable with each other's role and understand well that they have a common goal and it is wise to listen each other. A good communication skill is very important both for testers and developers.

Before getting started with testing work it is important to discuss the basic guidelines and expectations so that there is no confusion in later stages. Criticism should be taken in a positive sense. It is important to understand that developers and testers have a common goal of producing high quality software. A tester is not discovering bugs to show someone down, the idea is to learn from mistakes and avoid repeating them in future. A culture of constructive criticism can be of great help.

Interactions between Software Test Team And Release Management Teams

The release management teams are responsible for moving the software from development into production. This team is responsible for planning the releases for hardware, software and testing. It is also responsible for development of software development procedures and for coordinating interactions and training of releases. Software testing is considered to be a very important aspect of software engineering life cycle but it does not get over with development. Testing and verification is a very important part of release management exercise.

Interactions between Software Test Manager And Software Project Manager

The job of a software test manager is not an easy one. He has to recruit testing team and take responsibility for getting them trained. A software manager has to perform ongoing analysis of various testing processes and ensure that the testing team is carrying out all the processes correctly. This job is of great responsibility as the software testing manager is the one who selects, introduces and implement various tools for testing. A software test manager is responsible for finalizing templates for testing documents, test reports and other procedures.

Since a software tester manager has to deal with all the details of various testing activities, it is very important for him to be in constant touch with the project manager and provide necessary support in project planning and scheduling so that the project can be successfully completed in time within the specified financial budget limits.

Being A Tester

Be Proud to be a tester. You are the person who reduces the work in many fields, you are the ones who often release the projects without any defects, and you are the person whom people and the company believe more.

Finally, you are the person who ensures peace of mind for the end-users. Without you, it's almost difficult to complete the project successfully**.**

As a tester, you should continuously strive hard to become better and better.

16 Characteristics of a Great Software Tester

To be a great Software Tester, you need to develop the following 16 characteristics within you:

#1) Be Skeptical

Don’t believe that the build given by the developers is Bug-free or quality outcome. Question everything. Accept the build only if you test it and find it defect free.
Don’t believe anyone whatever is the designation they hold, should just apply your knowledge and try to find the errors. You need to follow this until the last phase of the testing cycle.

#2) Don’t Compromise On Quality
Don’t compromise after certain testing stages. There is no limit for testing until you produce a quality product. Quality is a word made by Software testers to achieve more effective testing. Compromising at any level leads to a defective product, so don’t do that at any point.

#3) Ensure End-User Satisfaction
Always think about what can make an end-user happy. How can they use the product with ease? Don’t stop by testing the standard requirements alone. The end-user can be happy only when you provide an error-free product.

#4) Think from the Users Perspective
Every product is developed for the customers. Customers may or may not be technical persons. If you don’t consider the scenarios from their perspective, you will miss many important bugs. So put yourself in their shoes. Know your end-users first. Their age, education and even location can matter most while using the product.
Make sure to prepare your test scenarios and test the data accordingly. After all, the project is said to be successful only if the end-user is able to use the application successfully.

#5) Prioritize Tests
First, identify the important tests and then prioritize the execution based on test importance. Never ever execute test cases sequentially without deciding the priority. This will ensure that all your important test cases get executed early and you won’t cut down on these at the last stage of the release cycle due to time pressure.
Also, consider the defect history while estimating test efforts. In most cases, the defect count at the beginning is more and keeps on reducing at the end of the test cycle.

#6) Never Promise 100% Coverage
Saying 100% coverage on paper is easy but practically it is impossible. So never promise to anyone, including your clients, about total Test coverage. In business, there is a philosophy – *“Under promise and over-deliver.”* So don’t focus on the goal for 100% coverage but focus on the quality of your tests.

#7) Be Open to Suggestions
Listen to everyone even though you are an authority on the project having in-depth project knowledge.
There is always scope for improvements and getting suggestions from fellow software testers is a good idea. Everyone’s feedback to improve the quality of the project would certainly help you to release bug-free software.

#8) Start Early

Don't wait until you get your first build for testing. Start analyzing the requirements, preparing Test cases, Test plan and Test strategy documents in the early design phase. Starting early for testing helps to visualize the complete project scope and hence planning can be done accordingly.

Most of the defects can be detected in the early design and analysis phase, thereby saving huge time and money. Early requirement analysis will also help you to question design decisions.

#9) Identify and Manage Risks

Risks are associated with every project. Risk management is a three-step process i.e. Risk identification, analysis, and mitigation. Incorporate risk driven testing process. Priorities in software testing are based on risk evaluation.

#10) Do Market Research

Don't think that your responsibility is just to validate software against the set of requirements. Be proactive, do your product market research and provide suggestions to improve it. This research will also help you to understand your product and its market.

#11) Develop Good Analyzing Skill

This is a must for requirement analysis but even further this could be helpful for understanding customer feedback while defining the Test strategy. Question everything around you. This will trigger the analysis process and it will help you to resolve many complex problems.

#12) Focus on the Negative Side as well

Testers should have the test to break attitude. Concentrating only on the positive side will almost certainly create many security issues in your application. You should be the hacker of your project to keep other hackers away from it. Negative testing is equally important. So cover a good chunk of your test cases based on the negative scenarios.

#13) Be a Good Judge of Your Product

A Judge usually thinks if something is right or wrong. Judges will listen to both sides. The same is applicable for testing as well. As a Software Tester, if you think something is right, try to prove why it is not wrong and later accept it. You must have a valid reason for all your decisions.

#14) Learn to Negotiate

Testers must negotiate with everyone at all stages of a project lifecycle. Negotiating with the developers is especially important. Developers can do anything to prove that their code is correct and the defect logged by the testers is not valid. It requires great skills to convince the developers about the defect and get it resolved.

Though some software testers think that this is not our task, explaining the true impact of any issue is very helpful for the developers to quickly understand the overall scenario and its implications. This requires years of practice but once you learn to negotiate you will gain more respect.

#15) Stop the Blame Game

It’s common to blame others for any defects which are not caught in testing. This is even more common when the tester’s responsibilities are not defined concretely. But in any situation never blame anyone. If an error occurs, first try to resolve it rather than finding someone to blame.
As a human everybody makes mistakes, so try to avoid blaming others. Work as a team to build team spirit.

#16) Finally, Be a Good Observer

Observe things happening around you. Keep track of all major and minor things in your project. Observe the way of developing the code, types of testing and its objectives. Observe and understand the test progress and make the necessary changes if it is off track in terms of schedule or testing activities.
This skill will essentially help you to keep yourself updated and get ready for the course of action for any situation.

Testing Axioms

This article lists the axioms, or truisms, related to software testing. Think of them as the "rules of the road" or the "facts of life" for software testing and software development.

It's Impossible to Test a Program Completely

As a new tester, you might believe that you can approach a piece of software, fully test it, find all the bugs, and assure that the software is perfect. Unfortunately, this isn't possible, even with the simplest programs, due to four key reasons:
The number of possible inputs is very large.
The number of possible outputs is very large.
The number of paths through the software is very large.
The software specification is subjective. You might say that a bug is in the eye of the beholder.

Software Testing Is a Risk-Based Exercise

If you decide not to test every possible test scenario, you've chosen to take on risk. In the calculator example, what if you choose not to test that 1024+1024=2048? It's possible the programmer accidentally left in a bug for that situation. If you don't test it, a customer will still use it, and he or she will discover the bug. It'll be a costly bug, too, since it wasn't found until the software was in the customer's hands.
This may all sound pretty scary. You can't test everything, and if you don't, you will likely miss bugs. The product has to be released, so you will need to stop testing, but if you stop too soon, there will still be areas untested. What do you do?
One key concept that software testers need to learn is how to reduce the huge domain of possible tests into a manageable set, and how to make wise risk-based decisions on what's important to test and what's not.

Testing Can't Show That Bugs Don't Exist

Think about this for a moment. You're an exterminator charged with examining a house for bugs. You inspect the house and find evidence of bugs—maybe live bugs, dead bugs, or nests. You can safely say that the house has bugs.
You visit another house. This time you find no evidence of bugs. You look in all the obvious places and see no signs of an infestation. Maybe you find a few dead bugs or old nests but see

nothing that tells you that live bugs exist. Can you absolutely, positively state that the house is bug free? Nope. All you can conclude is that in your search you didn't find any live bugs. Unless you completely dismantled the house down to the foundation, you can't be sure that you didn't simply just miss them.
Software testing works exactly as the exterminator does. It can show that bugs exist, but it can't show that bugs don't exist. You can perform your tests, find and report bugs, but at no point can you guarantee that there are no longer any bugs to find. You can only continue your testing and possibly find more.

The More Bugs You Find, the More Bugs There Are
There are even more similarities between real bugs and software bugs. Both types tend to come in groups. If you see one, odds are there will be more nearby.
Frequently, a tester will go for long spells without finding a bug. He'll then find one bug, then quickly another and another. There are several reasons for this:
Programmers have bad days. Like all of us, programmers can have off days. Code written one day may be perfect; code written another may be sloppy. One bug can be a tell-tale sign that there are more nearby.
Programmers often make the same mistake. Everyone has habits. A programmer who is prone to a certain error will often repeat it.
Some bugs are really just the tip of the iceberg. Very often the software's design or architecture has a fundamental problem. A tester will find several bugs that at first may seem unrelated but eventually are discovered to have one primary serious cause.
It's important to note that the inverse of this "bugs follow bugs" idea is true, as well. If you fail to find bugs no matter how hard you try, it may very well be that the software was cleanly written and that there are indeed few if any bugs to be found.
Not All the Bugs You Find Will Be Fixed
One sad reality of software testing is that even after all your hard work, not every bug you find will be fixed. Now, don't be disappointed—this doesn't mean that you've failed in achieving your goal as a software tester, nor does it mean that you or your team will release a poor quality product. It does mean, however, that you'll need to rely on a couple of traits of a software tester—exercising good judgment and knowing when perfection isn't reasonably attainable. You and your team will need to make trade-offs, risk-based decisions for each and every bug, deciding which ones will be fixed and which ones won't.

There are several reasons why you might choose not to fix a bug:
There's not enough time. In every project there are always too many software features, too few people to code and test them, and not enough room left in the schedule to finish. If you're working on a tax preparation program, April 15 isn't going to move—you must have your software ready in time.
It's really not a bug. Maybe you've heard the phrase, "It's not a bug, it's a feature!" It's not uncommon for misunderstandings, test errors, or spec changes to result in would-be bugs being dismissed as features.
It's too risky to fix. Unfortunately, this is all too often true. Software is fragile, intertwined, and sometimes like spaghetti. You might make a bug fix that causes other bugs to appear. Under the pressure to release a product under a tight schedule, it might be too risky to change the software. It may be better to leave in the known bug to avoid the risk of creating new, unknown ones.

It's just not worth it. This may sound harsh, but it's reality. Bugs that would occur infrequently or appear in little-used features may be dismissed. Bugs that have work-arounds, ways that a user can prevent or avoid the bug, are often not fixed. It all comes down to a business decision based on risk.
The decision-making process usually involves the software testers, the project managers, and the programmers. Each carries a unique perspective on the bugs and has his own information and opinions as to why they should or shouldn't be fixed.

Difference between Bug, Defect, Error, Fault & Failure

In this section, we are going to discuss the difference between the Bug, Defect, Error, Fault & Failure as we understood that all the terms are used whenever the system or an application act abnormally. Sometimes we call it an error and sometimes a bug or a defect and so on. In software testing, many of the new test engineers have confusion in using these terminologies.

In other words, we can say that in the era of software testing, the terms bugs, defects, error, fault, and failure come across every second of the day.

But for a beginner or the inexperienced in this field, all these terminologies may seem synonyms. It became essential to understand each of these terms independently if the software doesn't work as expected.

What is a bug?

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

What is a Defect?

When the application is not working as per the requirement is knows as defects. It is specified as the aberration from the actual and expected result of the application or software.

In other words, we can say that the bug announced by the programmer and inside the code is called a defect.

What is Error?

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

What is Fault?

The fault may occur in software because it has not added the code for fault tolerance, making an application act up.A fault may happen in a program because of the following reasons:

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

What is Failure?

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

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

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

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

Unit 4

Introduction to Testing

A strategy for software testing integrates software test case design methods into a well planned series of steps that result in the successful construction of software. Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:
1. **Verification:** it refers to the set of tasks that ensure that the software correctly implements a specific function.

2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

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

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

Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.

Verification	Validation
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.
Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.

Why Software Testing is Important?

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Test Cases- A **TEST CASE** is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement.

Test Strategy

1. **For Conventional Software Architectures**

    1) Unit testing
    2) Integration testing
    3) Validation testing
    4) System testing

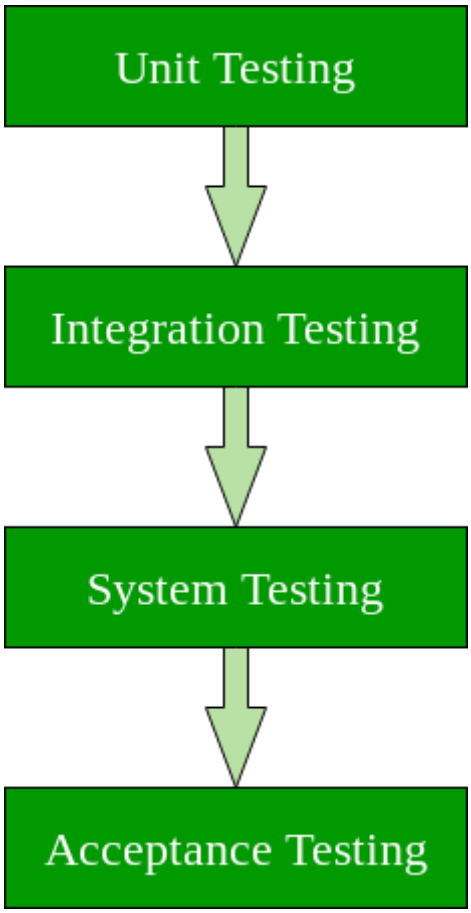
Software level testing can be majorly classified into 4 levels:

1. **Unit Testing:** A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.

2. **Integration Testing:** A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

3. **System Testing:** A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

4. **Acceptance Testing:** A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.



Unit Testing- Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements. Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules. Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer. In SDLC or V Model, Unit testing is the first level of testing done before integration testing

Why Unit Testing/ Objective of Unit Testing

The objective of Unit Testing is:

1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help with code reuse.

Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1. **Black Box Testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Advantages of Unit Testing:

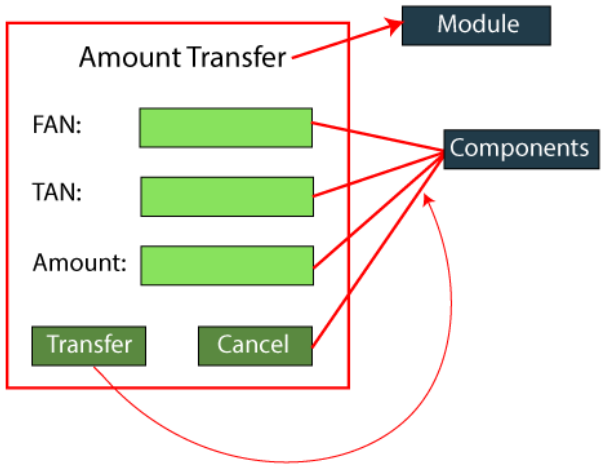
1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.
5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.
7. Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.
8. Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.
9. Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.
10. Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
6. Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.
7. Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.
8. Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.

Start performing the unit testing on the different components such as

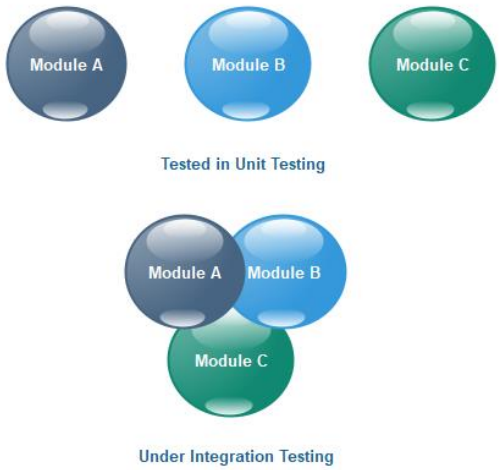
- **From account number(FAN)**
- **To account number(TAN)**
- **Amount**
- **Transfer**
- **Cancel**



Integration Testing- INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated. **Integration testing** is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The

purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application. The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other.



Why do Integration Testing?

Example of Integration Test Case

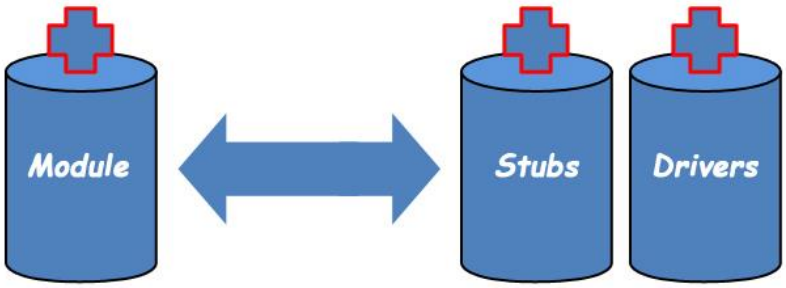
Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Stubs and Drivers

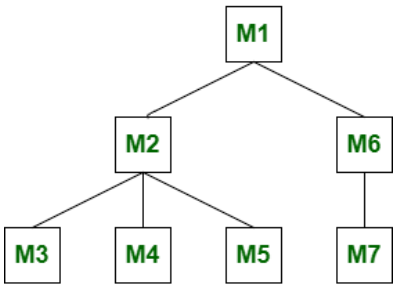
In the field of **software testing**, the term **stubs and drivers** refers to the replica of the modules, which acts as a substitute to the undeveloped or missing module. The stubs and drives are specifically developed to meet the necessary requirements of the unavailable modules and are immensely useful in getting expected results.



Top Down Integration

Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. In these, high-level modules are tested first, and then low-level modules are tested. Then, finally, integration is done to ensure that system is working properly. Stubs and drivers are used to carry out this project. This technique is used to increase or stimulate behavior of Modules that are not integrated into a lower level.

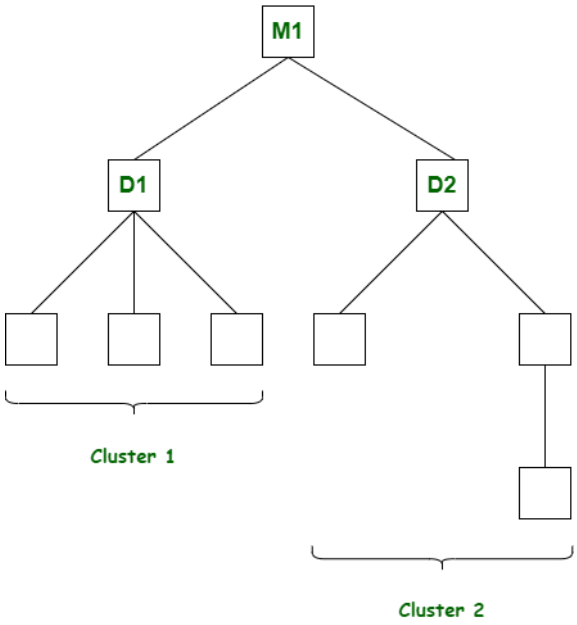
In the top-down integration testing, we will start integration from module M1. Then we will integrate M2, then M3, M4, M5, M6, and at last M7.



Program Structure

Bottom-Up Integration Testing –

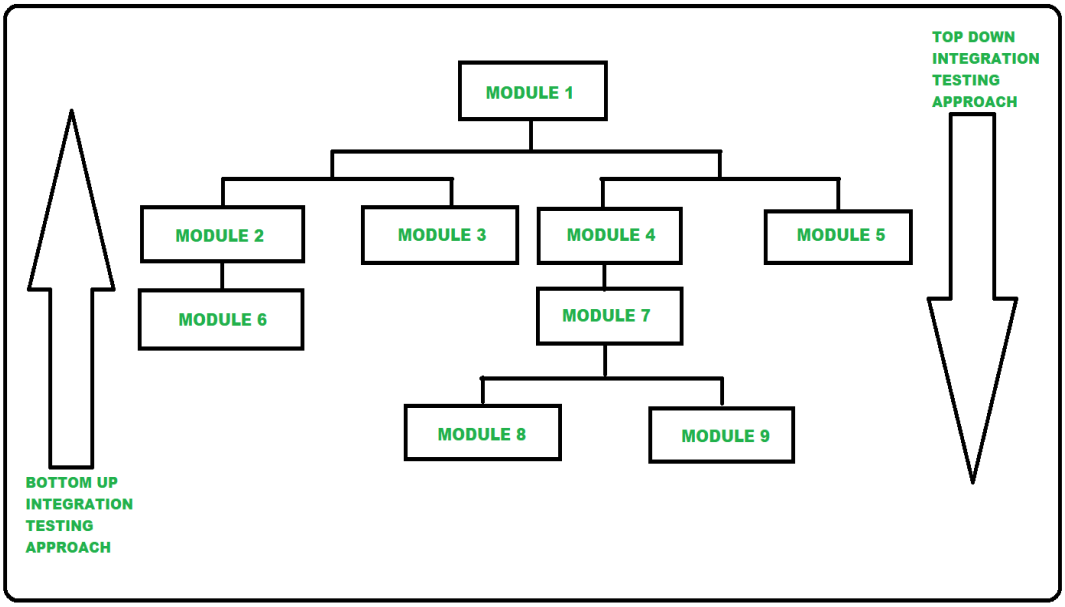
In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem.



Bottom Up Integration Testing

Advantages :

- It is easy and simple to create and develop test conditions.
- It is also easy to observe test results.
- It is not necessary to know about the details of the structural design.



Difference between Top Down Integration Testing and Bottom Up Integration Testing :

S.No.	TOP DOWN INTEGRATION TESTING	BOTTOM UP INTEGRATION TESTING
01.	Top Down Integration testing is one of the approach of Integration testing in which integration testing takes place from top to bottom means system integration begins with top level modules.	Bottom Up Integration testing is one of the approach of Integration testing in which integration testing takes place from bottom to top means system integration begins with lowest level modules.
02.	In this testing the higher level modules are tested first then the lower level modules are tested and then the modules are integrated accordingly.	In this testing the lower level modules are tested first then the higher level modules are tested and then the modules are integrated accordingly.
03.	In this testing stubs are used for simulate the submodule if the invoked submodule is not developed means Stub works as a momentary replacement.	In this testing drivers are used for simulate the main module if the main module is not developed means Driver works as a momentary replacement.
04.	Top Down Integration testing approach is beneficial if the significant defect occurs toward the top of the program.	Bottom Up Integration testing approach is beneficial if the crucial flaws encounters towards the bottom of the program.
05.	In Top Down Integration testing approach the main module is designed at first then the submodules/subroutines are called from it.	In Bottom Up Integration testing approach different modules are created first then these modules are integrated with the main function.
06.	It is implemented on Structure/procedure-oriented programming languages.	It is implemented on Object-oriented programming languages.
07.	The complexity of this testing is simple.	The complexity of this testing is complex and highly data intensive.
08.	It works on big to small components.	It works on small to big components.

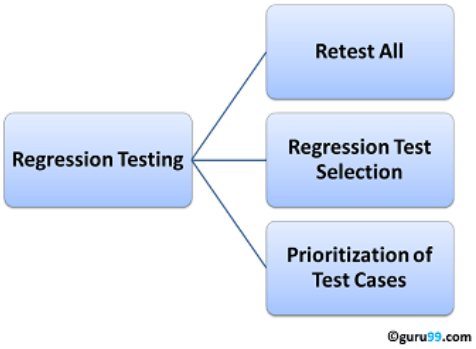
Regression Testing

It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.

Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.

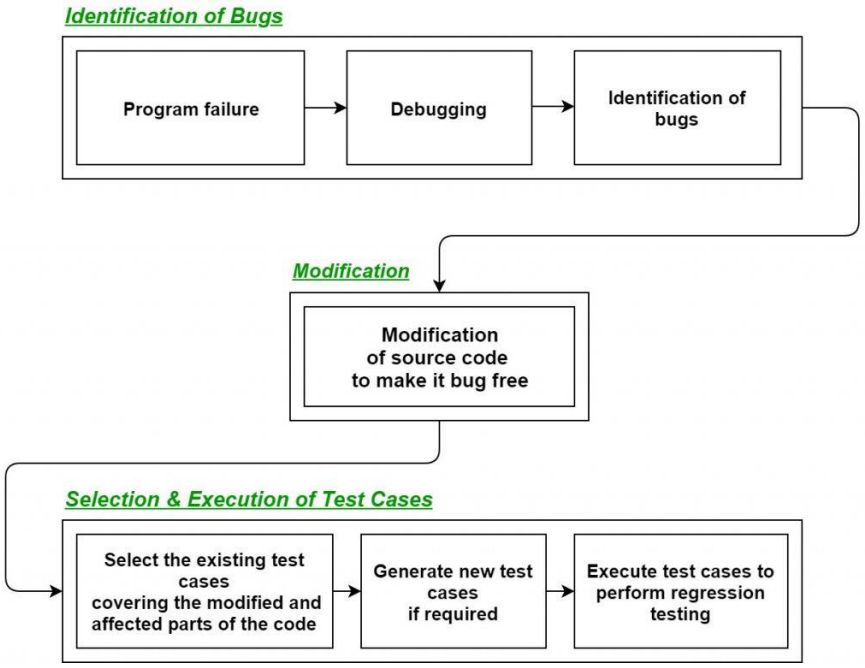
When can we perform Regression Testing?

- When new functionality added to the application.
- When there is a Change Requirement.
- When the defect fixed
- When there is an environment change



Process of Regression testing:

Firstly, whenever we make some changes to the source code for any reasons like adding new functionality, optimization, etc. then our program when executed fails in the previously designed test suite for obvious reasons. After the failure, the source code is debugged in order to identify the bugs in the program. After identification of the bugs in the source code, appropriate modifications are made. Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end regression testing is performed using the selected test cases.



Advantages of Regression Testing:

- It ensures that no new bugs has been introduced after adding new functionalities to the system.
- As most of the test cases used in Regression Testing are selected from the existing test suite and we already know their expected outputs. Hence, it can be easily automated by the automated tools.
- It helps to maintain the quality of the source code.

Disadvantages of Regression Testing:

- It can be time and resource consuming if automated tools are not used.
- It is required even after very small changes in the code.

Smoke Testing

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

Smoke Testing comes into the picture at the time of receiving build software from the development team. The purpose of smoke testing is to determine whether the build software is testable or not. It is done at the time of "building software." This process is also known as "Day 0".

It is a time-saving process. It reduces testing time because testing is done only when the key features of the application are not working or if the key bugs are not fixed. The focus of Smoke Testing is on the workflow of the core and primary functions of the application.

Testing the basic & critical feature of an application before doing one round of deep, rigorous testing (before checking all possible positive and negative values) is known as smoke testing.

In the smoke testing, we only focus on the positive flow of the application and enter only valid data, not the invalid data. In smoke testing, we verify every build is testable or not; hence it is also known as **Build Verification Testing.**

Why we do smoke testing?

- We will do the smoke testing to ensure that the product is testable.
- We will perform smoke testing in the beginning and detect the bugs in the basic features and send it to the development team so that the development team will have enough time to fix the bugs.
- We do smoke testing to make sure that the application is installed correctly.

Who will do Smoke Testing

After releasing the build to QA environment, Smoke Testing is performed by QA engineers/QA lead. Whenever there is a new build, QA team determines the major

functionality in the application to perform smoke testing. QA team checks for showstoppers in the application that is under testing.

Validation Testing

Validation testing is the process of ensuring if the tested and developed software satisfies the client /user needs. The business requirement logic or scenarios have to be tested in detail. All the critical functionalities of an application must be tested here. Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.

Validation testing is the process of assessing a new software product to ensure that its performance matches consumer needs. Product development teams might perform validation testing to learn about the integrity of the product itself and its performance in different environments.

Developers can perform validation testing themselves, or collaborate with quality assurance professionals, external validation testing professionals or clients to identify elements of the code to improve. Developers can also combine this type of testing with other useful techniques like product verification, debugging and certification to help ensure the product is ready for the market.

Why Validation is important?

Software validation can help product development teams ensure their product can satisfy customer needs and expectations. Validation testing can also help software developers identify and fix coding bugs or address other areas of improvement before launching the product.

A product that satisfies customer needs from its initial introduction to the market may be more likely to gain positive reviews, perform well and improve company reputation. Ultimately, this can improve sales and increase company revenue.

Validation Test Criteria

Configuration Review

Alpha and Beta Testing- Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.

Beta Testing is performed by real users of the software application in a real environment. Beta testing is one of the type of **User Acceptance Testing**.

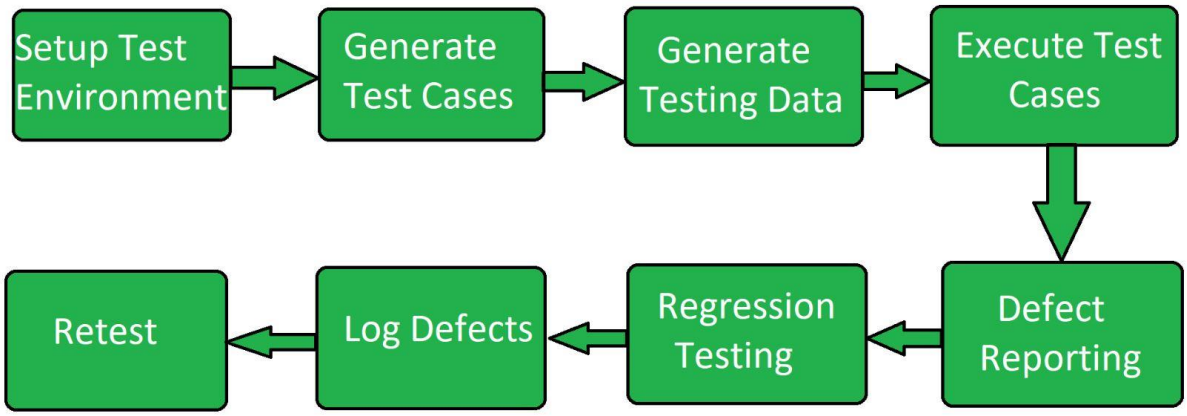
Alpha Testing	Beta Testing
It is done by internal testers of the organization.	It is done by real users.
It is an internal test, performed within the organization.	It is an external test, carried out in the user's environment.
Alpha Testing uses both black box and white box testing techniques	Beta Testing only uses the black box testing technique.
Identifies possible errors.	Checks the quality of the product.
Developers start fixing bugs as soon as they are identified.	Errors are found by users and feedback is necessary.
Long execution cycles.	It only takes a few weeks.
It can be easily implemented as it is done before the near end of development.	It will be implemented in the future version of the product.
It is performed before Beta Testing.	It is the final test before launching the product on the market.

It answers the question: Does the product work?	It answers the question: Do customers like the product?
Functionality and usability are tested.	Usability, functionality, security and reliability are tested with the same depth.

System Testing- SYSTEM TESTING is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications.

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.



System Testing Process: System Testing is performed in the following steps:

- **Test Environment Setup:** Create testing environment for the better quality testing.
- **Create Test Case:** Generate test case for the testing process.
- **Create Test Data:** Generate the data that is to be tested.
- **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.

- **Defect Reporting:** Defects in the system are detected.
- **Regression Testing:** It is carried out to test the side effects of the testing process.
- **Log Defects:** Defects are fixed in this step.
- **Retest:** If the test is not successful then again test is performed.

It is **end-to-end testing** where the testing environment is similar to the production environment.

Types of System Testing

1. **Recovery Testing- Recovery Testing** is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures etc.
2. **Security Testing- SECURITY TESTING** is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information by employees or outsiders of organization.

   Principles of Security Testing
   Confidentiality
   Integrity
   Authentication
   Authorization

3. **Stress Testing- Stress Testing** is a type of software testing that verifies stability & reliability of software application. The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations.
4. **Performance Testing- Performance Testing** is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload.
   Common Performance Problems
   Long load time
   Poor response time
   Poor scalability
   Bottleneck

Advantages of System Testing :

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Test Monitoring and Test Control

Test Monitoring in test execution is a process in which the testing activities and testing efforts are evaluated in order to track current progress of testing activity, finding and tracking test metrics, estimating the future actions based on the test metrics and providing feedback to the concerned team as well as stakeholders about current testing process.

Test Control in test execution is a process of taking actions based on results of the test monitoring process. In the test control phase, test activities are prioritized, test schedule is revised, test environment is reorganized and other changes related to testing activities are made in order to improve the quality and efficiency of future testing process.

Test Monitoring activity includes:

- Providing feedback to the team and the other concerned stakeholders about the progress of the testing efforts.
- Broadcasting the results of testing performed, to the associated members.
- Finding and tracking the Test Metrics.
- Planning and Estimation, for deciding the future course of action, based on the metrics calculated.

Test Control examples include:

- Prioritizing the Testing efforts
- Revisiting the Test schedules and Dates
- Reorganizing the Test environment
- Re prioritizing the Test cases/Conditions

Best Practices in Test Monitoring and Test Control

- Establish standards
- Prioritize Documentation
- Be Proactive and Prepare for Change

Test Strategy and Planning

Test Strategy in software testing is defined as a set of guiding principles that determines the test design & regulates how the software testing process will be done. The objective of the Test Strategy is to provide a systematic approach to the software testing process in order to ensure the quality, traceability, reliability and better planning.

A Test Plan is defined as a document which outlines the scope, objective, method and weight on a software testing task.

Master Test Plan

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test.

Details provided by Master Test Plan

- List of tests to be performed
- Testing levels to be covered
- Relationship among different test levels and related coding activity
- Test implementation strategy
- Explain testing effort which is a component of the project
- Master test plan should align with test policy and test strategy.

Content of Master Test Plan

- List of things to be tested
- List of items that will not be tested
- List of quality characteristics that will be tested
- List of product quality characteristics that will not be tested
- Test execution cycles
- Test budgets according to project or overall operational budget
- Testing schedule
- Correlation between testing cycles and release plan
- Interrelation between other departments and testing team
- Scope of each test item
- Predefined entry, continuation and exit conditions for each test level
- Risks associated with test project
- Management and control of testing
- Team members responsible for each test level
- Inputs and outputs for each test level

Test Case Management

Test case management involves managing different versions of test cases, keeping track of changes in them, keeping a separate repository of test cases based on type of tests, as well as creating and managing automation scripts.

Understanding the Idea of a Test Case

Requirements tell you what the software needs to do.

Test scenarios define a means for confirming the requirement.

Test cases describe how, the component pieces of confirming that requirement.

Test scripts (whether automated or executed manually) tell you exactly how to execute that component piece.

Typically, you'll have a number of test cases per test scenario, covering various permutations of inputs and behaviors. You then have one or possibly more scripts per test case.

Test management, process of managing the tests. A test management is also performed using tools to manage both types of tests, automated and manual, that have been previously specified by a test procedure.

ANATOMY – TEST CASE

- An ID: A unique way of identifying the test case.
- Title or brief description: A quick means of understanding the software activity in question.
- Related requirement and/or test: What broader scenario and requirement does this test case roll up to?
- Remarks/Notes: Free form comments about the test case.
- Script: Exact steps for executing the test.
- Pass/Fail Status: Is the test case currently passing or failing?
- History and Audit Trail: You should be able to see its pass/fail history as well as general changes and who has run/modified the test case.

Test Management Process is a procedure of managing the software testing activities from start to the end.

There are two main Parts of Test Management Process: -

- Planning
    1. Risk Analysis
    2. Test Estimation
    3. Test Planning
    4. Test Organization
- Execution
    1. Test Monitoring and Control

2. Issue Management
3. Test Report and Evaluation

Test Reporting

During the execution of a test project, many initial and final reports are made.

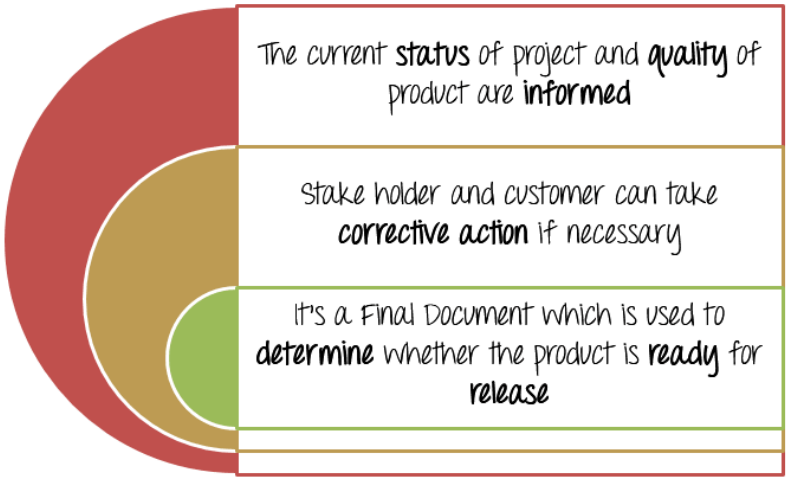
Test Report is a document which contains

A summary of test activities and final test results An assessment of how well the Testing is performed.

Based on the test report, the stakeholders can Evaluate the quality of the tested

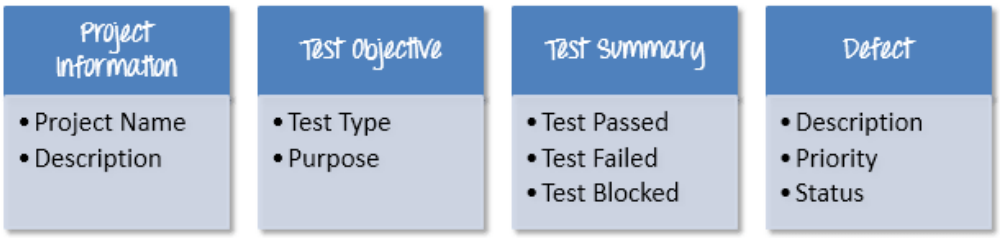
Product. Make a decision on the software release.

The typical benefits of a test report include:



What does a test report contain?

`



Project Information

All information of the project such as the project name, product name, and version should be described in the test report.

Test Objective

As mentioned in Test Planning tutorial, Test Report should include the objective of each round of testing, such as Unit Test, Performance Test, System Test …Etc.

Test Summary

This section includes the summary of testing activity in general. Information detailed here includes

- The number of test cases executed
- The numbers of test cases pass
- The numbers of test cases fail
- Pass percentage
- Fail percentage
- Comments

Defect

One of the most important information in Test Report is defect. The report should contain following information

- Total number of bugs
- Status of bugs (open, closed, responding)
- Number of bugs open, resolved, closed

Black Box Testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.

Black Box techniques

1. Equivalence Class Testing

2. Boundary Value testing
3. Decision Table Testing

White Box Testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security.

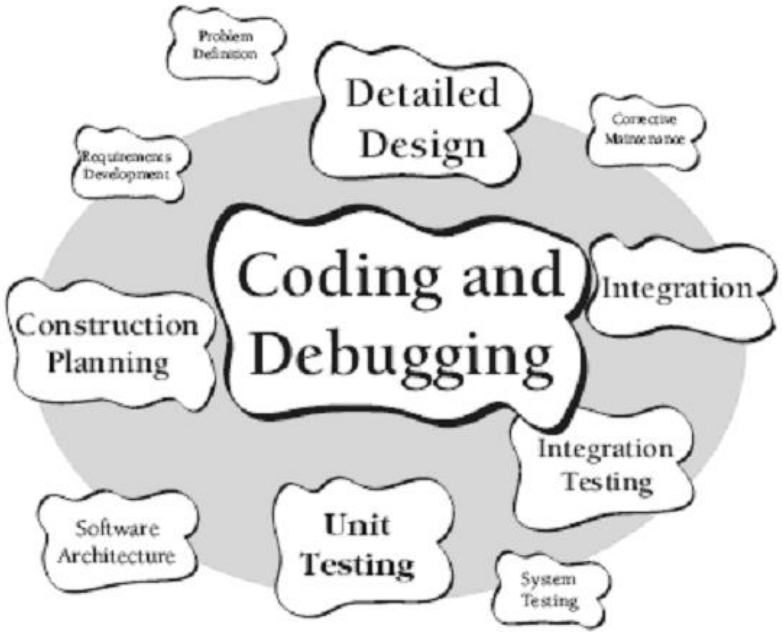
Differences between Black Box Testing vs White Box Testing:

Black Box Testing	White Box Testing
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
It is a functional test of the software.	It is a structural test of the software.
This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
Black-box test design techniques- • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	White-box test design techniques- • Control flow testing • Data flow testing • Branch testing
Types of Black Box Testing: • Functional Testing • Non-functional testing • Regression Testing	Types of White Box Testing: • Path Testing • Loop Testing • Condition testing

UNIT-3

Software Construction

The term software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.



The diagram here shows several component activities that should be familiar to you from our earlier discussion of process models. The size of each cloud is intended to suggest the relative amount of time and effort spent in that activity. The position of the cloud, relative to the yellow oval that defines "software construction", indicates whether that activity is central or peripheral to software construction, and whether it is entirely a software construction activity (e.g., coding and debugging), partially software construction (e.g., detailed design), or not part of software construction at all (e.g., problem definition).

Software Construction Fundamentals

The fundamentals of software construction include:

Minimizing complexity-A major factor in how people convey intent to computers is the severely limited ability of people to hold complex structures and information in their working memories, especially over long periods of time. This leads to one of the strongest drivers in software construction: minimizing complexity. The need to reduce complexity applies to essentially every aspect of software construction, and is particularly critical to the process of verification and testing of software constructions.

In software construction, reduced complexity is achieved through emphasizing the creation of code that is simple and readable rather than clever.

Anticipating change- Most software will change over time, and the anticipation of change drives many aspects of software construction. Software is unavoidably part of changing external environments, and changes in those outside environments affect software in diverse ways.

Anticipating change is supported by many specific techniques:

- Communication methods (for example, standards for document formats and contents)
- Programming languages (for example, language standards for languages like Java and C++)
- Platforms (for example, programmer interface standards for operating system calls)
- Tools (for example, diagrammatic standards for notations like UML (Unified Modeling Language))

Constructing for verification-Constructing for verification means building software in such a way that faults can be ferreted out readily by the software engineers writing the software, as well as during independent testing and operational activities. Specific techniques that support constructing for verification include following coding standards to support code reviews, unit testing, organizing code to support automated testing, and restricted use of complex or hard-to-understand language structures, among others.

Standards in construction-Standards that directly affect construction issues include Use of external standards. Construction depends on the use of external standards for construction languages, construction tools, technical interfaces, and interactions between Software Construction and other software engineering. Standards come from numerous sources, including hardware and software interface specifications such as the Object Management Group (OMG) and international organizations such as the IEEE or ISO.

Use of internal standards- Standards may also be created on an organizational basis at the corporate level or for use on specific projects. These standards support coordination of group activities, minimizing complexity, anticipating change, and constructing for verification.

Key Construction Decisions

Choice of Programming Language- The language that the system will be implemented with should be one that you are familiar with.

Programmers are more productive using a familiar language than an unfamiliar one.

Language choice affects productivity and code equality.

Construction Practices

- Coding
- Teamwork
- Quality Assurance
- Tools

Managing Construction

Construction Models- Numerous models have been created to develop software, some of which emphasize construction more than others.

Some models are more linear from the construction point of view, such as the waterfall and staged-delivery life cycle models. These models treat construction as an activity which occurs only after significant prerequisite work has been completed - including detailed requirements work, extensive design work, and detailed planning. The more linear approaches tend to emphasize the activities that precede construction (requirements and design), and tend to create more distinct separations between the activities.

Other models are more iterative, such as evolutionary prototyping, Extreme Programming, and Scrum. These approaches tend to treat construction as an activity that occurs concurrently with other software development activities, including requirements, design, and planning, or overlaps them. These approaches tend to mix design, coding, and testing activities, and they often treat the combination of activities as construction.

Construction Planning- The choice of construction method is a key aspect of the construction planning activity. The choice of construction method affects the extent to which construction prerequisites are performed, the order in which they are performed, and the degree to which they are expected to be completed before construction work begins.

The approach to construction affects the project's ability to reduce complexity, anticipate change, and construct for verification. Each of these objectives may also be addressed at the process, requirements, and design levels - but they will also be influenced by the choice of construction method.

Construction planning also defines the order in which components are created and integrated, the software quality management processes, the allocation of task assignments to specific software engineers, and the other tasks, according to the chosen method.

Construction Measurement- Numerous construction activities and artifacts can be measured, including code developed, code modified, code reused, code destroyed, code complexity, code inspection statistics, fault-fix and fault-find rates, effort, and scheduling. These measurements

can be useful for purposes of managing construction, ensuring quality during construction, improving the construction process, as well as for other reasons.

Coding Standards

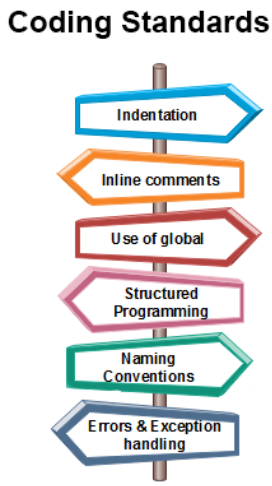
Different modules specified in the design document are coded in the Coding phase according to the module specification. Good software development organizations want their programmers to maintain to some well-defined and standard style of coding called coding standards.

The main goal of the coding phase is to code from the design document prepared after the design phase through a high-level language and then to unit test this code. Good software development organizations want their programmers to maintain to some well-defined and standard style of coding called coding standards. They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop. It is very important for the programmers to maintain the coding standards otherwise the code will be rejected during code review.

Purpose of Having Coding Standards:

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

Coding Standards



1. **Limited use of globals:**
   These rules tell about which types of data that can be declared global and the data that can't be.

2. **Standard headers for different modules:**
   For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:

   - Name of the module
   - Date of module creation
   - Author of the module
   - Modification history
   - Synopsis of the module about what the module does
   - Different functions supported in the module along with their input output parameters
   - Global variables accessed or modified by the module

3. **Naming conventions for local variables, global variables, constants and functions:**
   Some of the naming conventions are given below:

- Meaningful and understandable variables name helps anyone to understand the reason of using it.
- Local variables should be named using camel case lettering starting with small letter (e.g. localData) whereas Global variables names should start with a capital letter (e.g. GlobalData). Constant names should be formed using capital letters only (e.g. CONSDATA).
- It is better to avoid the use of digits in variable names.
- The names of the function should be written in camel case starting with small letters.
- The name of the function must describe the reason of using the function clearly and briefly.

4. **Indentation:**
   Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:

- There must be a space after giving a comma between two function arguments.
- Each nested block should be properly indented and spaced.
- Proper Indentation should be there at the beginning and at the end of each block in the program.
- All braces should start from a new line and the code following the end of braces also start from a new line.

5. **Error return values and exception handling conventions:**
   All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.
6. **Avoid using a coding style that is too difficult to understand:**

Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.

7. **Avoid using an identifier for multiple purposes:**
   Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.
8. **Code should be well documented:**
   The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.
9. **Length of functions should not be very large:**
   Lengthy functions are very difficult to understand. That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
10. **Try not to use GOTO statement:**
    GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

Advantages of Coding Guidelines:

- Coding guidelines increase the efficiency of the software and reduces the development time.
- Coding guidelines help in detecting errors in the early phases, so it helps to reduce the extra cost incurred by the software project.
- If coding guidelines are maintained properly, then the software code increases readability and understandability thus it reduces the complexity of the code.
- It reduces the hidden cost for developing the software.

Coding Framework

For software construction, a coding framework is needed that will ensure a consistent coding production with standard code that will be easy to debug and test.

In computer **programming**, a software **framework** is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.

Reviews (From PDF File)

Structured Programming

In structured programming, we sub-divide the whole program into small modules so that the program becomes easy to understand.

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute

the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner.

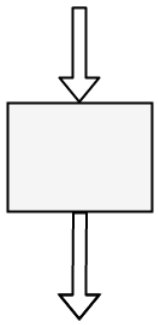
Why we use structured programming?

We use structured programming because it allows the programmer to understand the program easily. If a program consists of thousands of instructions and an error occurs then it is complicated to find that error in the whole program, but in structured programming, we can easily detect the error and then go to that location and correct it. This saves a lot of time.

Structured Rule One: Code Block

If the entry conditions are correct, but the exit conditions are wrong, the error must be in the block. This is not true if the execution is allowed to jump into a block. The error might be anywhere in the program. Debugging under these circumstances is much harder.

Rule 1 of Structured Programming: A code block is structured, as shown in the figure. In flow-charting condition, a box with a single-entry point and single exit point are structured. Structured programming is a method of making it evident that the program is correct.

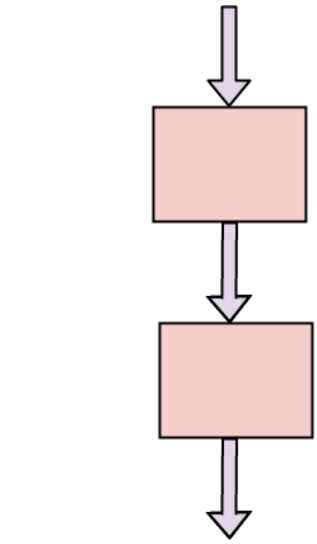


Rule1: Code block is structured

Structure Rule Two: Sequence

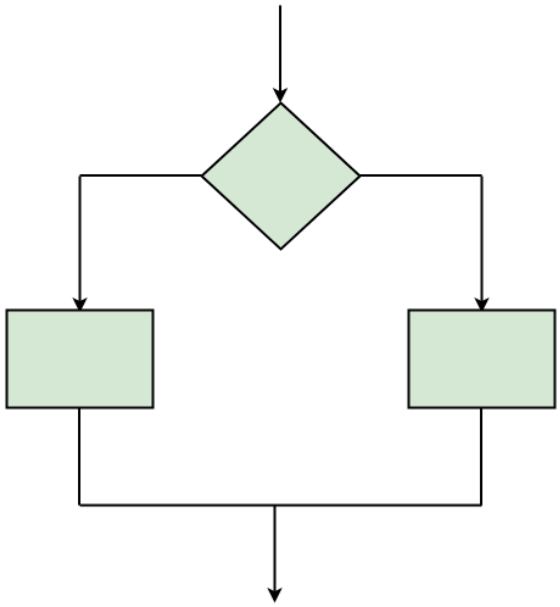
A sequence of blocks is correct if the exit conditions of each block match the entry conditions of the following block. Execution enters each block at the block's entry point and leaves through the block's exit point. The whole series can be regarded as a single block, with an entry point and an exit point.

Rule 2 of Structured Programming: Two or more code blocks in the sequence are structured, as shown in the figure.



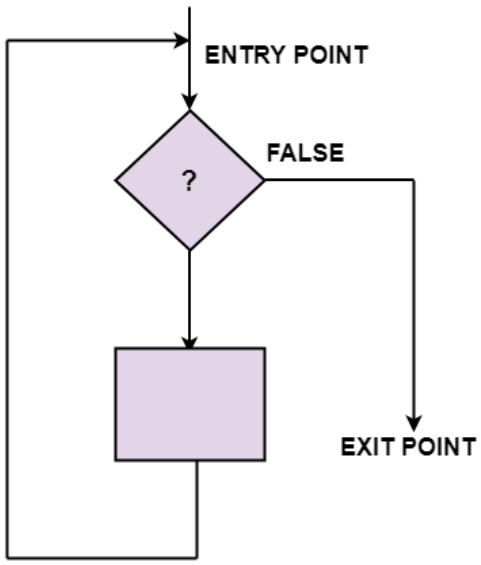
Rule2: A sequence of code blocks is structured

Structured Rule Three: Alternation If-then-else is frequently called alternation (because there are alternative options). In structured programming, each choice is a code block. If alternation is organized as in the flowchart at right, then there is one entry point (at the top) and one exit point (at the bottom). The structure should be coded so that if the entry conditions are fulfilled, then the exit conditions are satisfied (just like a code block).



Rule 3: An alternation of code blocks is structured

Structured Rule 4: Iteration- Iteration (while-loop) is organized as at right. It also has one entry point and one exit point. The entry point has conditions that must be satisfied, and the exit point has requirements that will be fulfilled. There are no jumps into the form from external points of the code.



Rule 4: Iteration of code blocks is structured

Advantages of Structured Programming Approach:

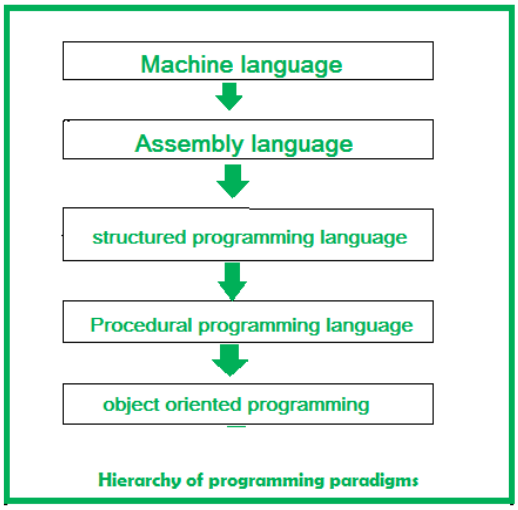
1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent. So, it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore, it needs to be updated with the need on the go.
4. Usually, the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Object Oriented Programming

Object-oriented programming (OOP) is nothing but that which allows the writing of programs with the help of certain classes and real-time objects. We can say that this approach is very close to the real-world and its applications because the state and behaviour of these classes and objects are almost the same as real-world objects.



Hierarchy of programming paradigms

Class and Object

It is an abstract and user-defined data type. It consists of several variables and functions. The primary purpose of the class is to store data and information. The members of a class define the behaviour of the class. A class is the blueprint of the object, but also, we can say the implementation of the class is the object. The class is not visible to the world, but the object is.

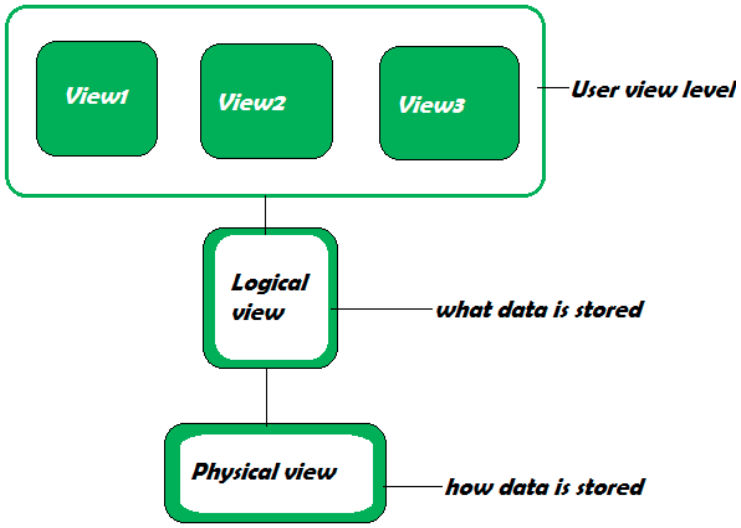
A ***class*** is a program-code-template that allows developers to create an object that has both variables (data) and behaviors (functions or methods).

```
Class car
{
   int car_id;
   char colour[4];
   float engine_no;
   double distance;

   void distance_travelled();
   float petrol_used();
   void display();
}
```

Data Abstraction –Abstraction refers to the act of representing important and special features without including the background details or explanation about that feature.

Abstraction occurs when a programmer hides any irrelevant data about an object or an instantiated class to reduce complexity and help users interact with a program more efficiently.



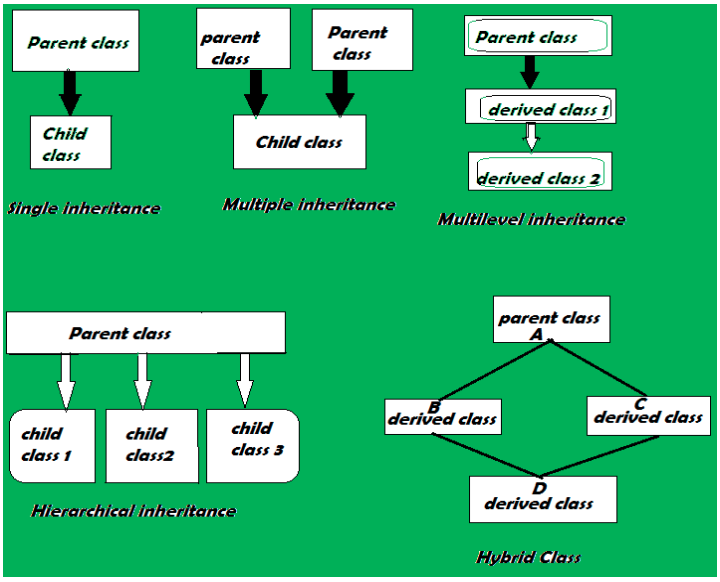
Encapsulation –

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This concept is often used to hide the internal state representation of an object from the outside.

A class is an example of encapsulation in computer science in that it consists of data and methods that have been bundled into a single unit.

Inheritance –

Inheritance is the ability of one class to inherit capabilities or properties of another class, called the parent class. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class which possesses it



Polymorphism –

Polymorphism is the ability of data to be processed in more than one form. It allows the performance of the same task in various ways. The word itself indicates the meaning as **poly** means **many** and **morphism** means **types**. Polymorphism is one of the most important concepts of object-oriented programming language.

Automatic Code Generation

Automatic code generation refers to using programs to using programs generate code that the user would otherwise have to write themselves. **(From PPT)**

Software Code Reuse

- Code reuse is the practice of using existing code for a new function or software.
- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.

Reuse based software engineering

- System Reuse
- Application Reuse
- Component Reuse
- Object and function reuse

Benefits of software reuse

- Accelerated development
- Effective use of specialists
- Increased dependability
- Lower development costs
- Reduced process risk
- Standards compliance

Pair Programming

As the name implies, pair programming is where two developers work using only one machine. Each one has a keyboard and a mouse. One programmer act as the driver who codes while the other will serve as the observer who will check the code being written, proofread and spell check it, while also figuring out where to go next. These roles can be switched at any time: the driver will then become the observer and vice versa.

Pair Programming – Advantages

- Many mistakes are detected at the time they are typed, rather than in QA Testing or in the field.
- The team solves problems faster.
- People learn to work together and talk more often together, giving better information flow and team dynamics.
- The project ends up with multiple people understanding each piece of the system.
- People learn significantly more about the system and about software development.
- It would allow programmers to get instant face-to-face instruction, which is much better than online tutorials and faster than looking for resources on the Internet.
- Because there is another programmer looking over your work, it results in better code.

Why is Programming in Pairs Better than Code Reviews?

Code reviews are a process wherein another programmer takes a look at your code to find something that needs improvement or find defects in it. It combines testing with quality control to ensure that everything in your code is good. This helps you ensure that your code is improved. However, it is challenging to find somebody to review your code because people may not want to look at another's code and understand their reasoning just for the sake of checking its quality.
With pairing, it is like having somebody review your code instantly and regularly. It is a higher form of code reviews. Two people have to be there and understand the program being written. And if one sees problems with the other's code, then it can be instantly corrected. You also have fewer chances of having bugs written into your code.

How to Effectively Pair Your Programmers?

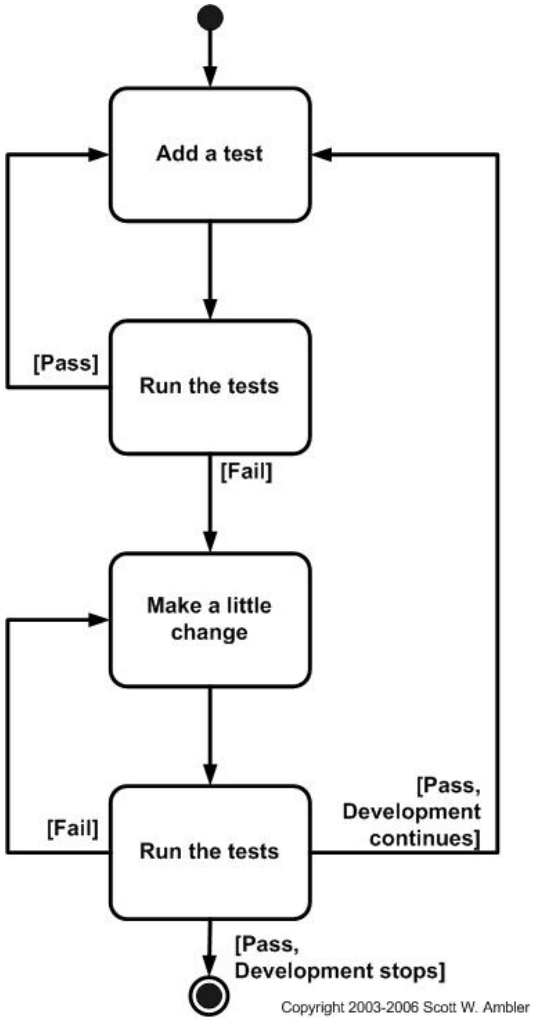
The best way to approach pairing is to partner two programmers and have them share a computer. Make them work together to architect, code and then test their codes in a genuine sense of a partnership. While the ideal setup would include two programmers who are equally skilled (expert – expert or novice – novice), you can also use pair programming for training and educational purposes (expert – novice).

The pair should be able to decide how to split the work, and it is advisable that they should switch roles often.

Test Driven Development

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.
Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The full form of TDD is Test-driven development.

Steps of TDD



TDD cycle defines

1. Write a test case
2. Make it run.
3. Change the code to make it right i.e. Refactor.
4. Repeat process.

There are two levels of TDD

Acceptance TDD (ATDD): With ATDD you write a single acceptance test. This test fulfills the requirement of the specification or satisfies the behavior of the system. After that write just enough production/functionality code to fulfill that acceptance test. Acceptance test focuses on the overall behavior of the system. ATDD also was known as Behavioral Driven Development (BDD).

Developer TDD: With Developer TDD you write single developer test i.e. unit test and then just enough production code to fulfill that test. The unit test focuses on every small functionality of the system. Developer TDD is simply called as TDD.

The main goal of ATDD and TDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis. JIT means taking only those requirements in consideration that are needed in the system. So, increase efficiency.

Advantages of TDD

- Early bug notification.
- Better Designed, cleaner and more extensible code.
- Confidence to Refactor
- Good for teamwork
- Good for Developers

Software Construction Artifacts

An artifact is one of many kinds of tangible by-products produced during the development of software. Some artifacts (e.g., use cases, class diagrams, and other Unified Modeling Language (UML) models, requirements and design documents) help describe the function, architecture, and design of software. Other artifacts are concerned with the process of development itself—such as project plans, business cases, and risk assessments.

The basic task for the SCA is to grant the software developer / designer) a simplified view of given information and functionality.