SQA

What is Software Testing?

Software testing is a process of executing a program or application with the intent of finding the software bugs.

- It can also be stated as the **process of validating and verifying** that a software program or application or product:
  - Meets the business and technical requirements that guided it's design and development
  - Works as expected
  - Can be implemented with the same characteristic.
  -

Software Development Life Cycle (SDLC).

- The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it's referred as **"verifying the test basis via the test design"**.
- The **test basis** includes documents such as the requirements and design specifications.

3) Static Testing: It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example: reviewing, walkthrough, inspection, etc.

4) Dynamic Testing: In dynamic testing the software code is executed to demonstrate the result of running tests. It's done during validation process. For example: unit testing, integration testing, system testing, etc.

5) Planning: We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.

6) Preparation: We need to choose what testing we will do, by selecting test conditions and designing test cases.

7) Evaluation: During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.

8) Software products and related work products: Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

Why is software testing necessary?

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

Since we assume that our work may have mistakes, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.

Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several reasons which clearly tells us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.
2. It's essential since it makes sure of the Customer's reliability and their satisfaction in the application.
3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence.
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.

7. It’s required to stay in the business.

Software Testing has different goals and objectives. The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it’s performing well and as per the specifications.

While determining the coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.

Once the delivery is made to the end users or the customers they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer’s requirements.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

What is Defect or bugs or faults in software testing?

Definition:

- A defect is an error or a bug, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects.
- When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it's called differently like bug, issue, incidents or problem.
- When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect. These defects or bugs occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results.

Additional Information about Defects / Bugs:

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

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

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

- **Defect ID** – Every bug or defect has it's unique identification number
- **Defect Description** – This includes the abstract of the issue.
- **Product Version** – This includes the product version of the application in which the defect is found.
- **Detail Steps** – This includes the detailed steps of the issue with the screenshots attached so that developers can recreate it.
- **Date Raised** – This includes the Date when the bug is reported
- **Reported By** – This includes the details of the tester who reported the bug like Name and ID
- **Status** – This field includes the Status of the defect like New, Assigned, Open, Retest, Verification, Closed, Failed, Deferred, etc.

- **Fixed by** – This field includes the details of the developer who fixed it like Name and ID
- **Date Closed** – This includes the Date when the bug is closed
- **Severity –** Based on the severity (Critical, Major or Minor) it tells us about impact of the defect or bug in the software application
- **Priority –** Based on the Priority set (High/Medium/Low) the order of fixing the defect can be made.

<u>What is a Failure in software testing?</u>

If under certain environment and situation defects in the application or product get executed then the system will produce the wrong results causing a failure.

Not all defects result in failures, some may stay inactive in the code and we may never notice them. Example: Defects in dead code will never result in failures.

It is not just defects that give rise to failure. Failures can also be caused because of the other reasons also like:

- Because of the environmental conditions as well like a radiation burst, a strong magnetic field, electronic field or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.
- <u>Failures may also arise</u> because of human error in interacting with the software, perhaps a wrong input value being entered or an output being misinterpreted.
- Finally failures may also be caused by someone deliberately trying to cause a failure in the system.

Difference between Error, Defect and Failure in software testing:

Error: The mistakes made by programmer is knowns as an ‘Error’. This could happen because of the following reasons:

– Because of some confusion in understanding the functionality of the software

– Because of some miscalculation of the values

– Because of misinterpretation of any value, etc.

Defect: The bugs introduced by programmer inside the code are known as a defect. This can happen because of some programmatically mistakes.

Failure: If under certain circumstances these defects get executed by the tester during the testing then it results into the failure which is known as software failure.

Few points that is important to know:

- When tester is executing a test he/she may observe some difference in the behavior of the feature or functionality, but this not because of the failure. This may happen because of the wrong test data entered, tester may not be aware of the feature or functionality or because of the bad environment. Because of these reasons incidents are reported. They are known as incident report. The condition or situation which requires further analysis or clarification is known as incident. To deal with the incidents the programmer need to to the analysis that whether this incident has occurred because of the failure or not.

- It's not necessary that defects or bugs introduced in the product are only by the software. To understand it further let's take an example. A bug or defect can also be introduced by a business analyst. Defects present in the specifications like requirements specification and design specifications can be detected during the reviews. When the defect or bug is caught during the review cannot result into failure because the software has not yet been executed.

- These defects or bugs are reported not to blame the developers or any people but to judge the quality of the product. The quality of product is of utmost importance. To gain the confidence of the customers it's very important to deliver the quality product on time.

What is the cost of defects in software testing?

The cost of defects can be measured by the impact of the defects and when we find them. Earlier the defect is found lesser is the cost of defect. For example if error is found in the requirement specifications then it is somewhat cheap to fix it. The correction to the requirement specification can be done and then it can be re-issued. In the same way when defect or error is found in the design then the design can be corrected and it can be re-issued. But if the error is not caught in the specifications and is not found till the user acceptance then the cost to fix those errors or defects will be way too expensive.

If the error is made and the consequent defect is detected in the **requirements phase** then it is relatively cheap to fix it.

Similarly if an error is made and the consequent defect is found in the **design phase** then the design can be corrected and reissued with relatively little expense.

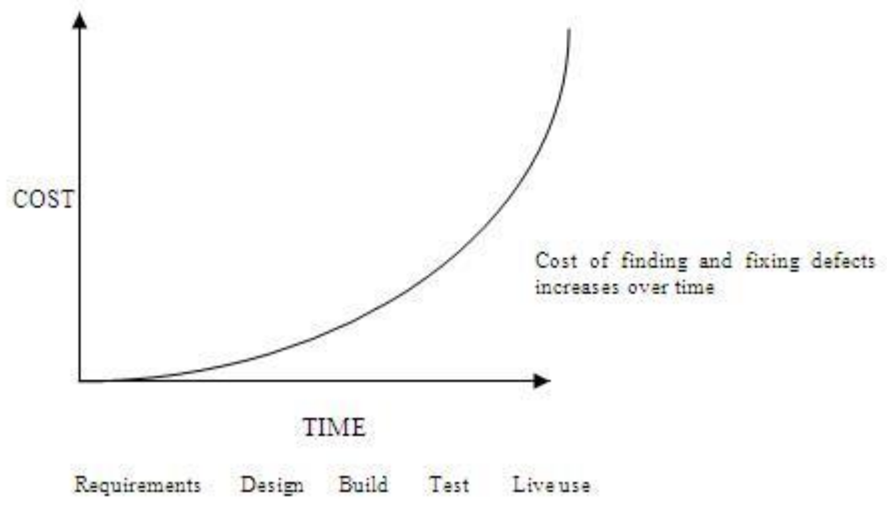


FIGURE 1.2

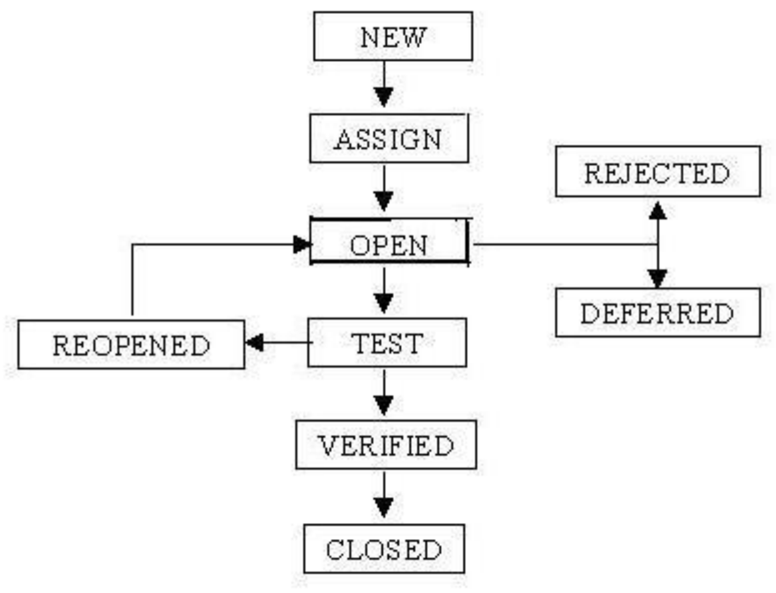
The same applies for **construction phase**. If however, a defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix. This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in the requirements may well propagate into several places in the design and code; and because all the testing work done-to that point will need to be repeated in order to reach the confidence level in the software that we require.

It is quite often the case that defects detected at a very late stage, depending on how serious they are, are not corrected because the cost of doing so is too expensive.

What is a Defect Life Cycle or a Bug lifecycle in software testing?

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

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



Bug or defect life cycle includes following steps or status:

1. **New:** When a defect is logged and posted for the first time. Its state is given as new.
2. **Assigned:** After the tester has posted the bug, the lead of the tester approves that the bug is genuine and he assigns the bug to corresponding developer and the developer team. Its state given as assigned.
3. **Open:** At this state the developer has started analyzing and working on the defect fix.
4. **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as 'Fixed' and the bug is passed to testing team.
5. **Pending retest:** After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.
6. **Retest:** At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

7. **Verified:** The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “verified”.
8. **Reopen:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “reopened”. The bug goes through the life cycle once again.
9. **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “closed”. This state means that the bug is fixed, tested and approved.
10. **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “duplicate**“.**
11. **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “rejected”.
12. **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
13. **Not a bug:** The state given as “Not a bug” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of colour of some text then it is not a bug but just some change in the looks of the application.

What is the difference between Severity and Priority?

There are two key things in defects of the software testing. They are:

1) Severity

2) Priority

What is the difference between Severity and Priority?

1) Severity:

It is the extent to which the defect can affect the software. In other words it defines the impact that a given defect has on the system. **For example:** If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by an user is rare but the impact of application crashing is severe. So the severity is high but priority is low.

Severity can be of following types:

- **Critical:** The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable and there is no acceptable alternative method to achieve the required results then the severity will be stated as critical.
- **Major:** The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable but there exists an acceptable alternative method to achieve the required results then the severity will be stated as major.
- **Moderate:** The defect that does not result in the termination, but causes the system to produce incorrect, incomplete or inconsistent results then the severity will be stated as moderate.
- **Minor:** The defect that does not result in the termination and does not damage the usability of the system and the desired results can be easily obtained by working around the defects then the severity is stated as minor.
- **Cosmetic:** The defect that is related to the enhancement of the system where the changes are related to the look and field of the application then the severity is stated as cosmetic.

2) Priority:

Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest. The priority status is set based on the customer requirements. **For example:** If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it.

Priority can be of following types:

- **Low:** The defect is an irritant which should be repaired, but repair can be deferred until after more serious defect have been fixed.
- **Medium:** The defect should be resolved in the normal course of development activities. It can wait until a new build or version is created.
- **High:** The defect must be resolved as soon as possible because the defect is affecting the application or the product severely. The system cannot be used until the repair has been done.

What are the principles of testing?

Principles of Testing

There are seven principles of testing. They are as follows:

1) Testing shows presence of defects: Testing can show the defects are present, but cannot prove that there are no defects. Even after testing the application or product thoroughly we cannot say that the product is 100% defect free. Testing always reduces the number of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

2) Exhaustive testing is impossible: Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts. For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30 517 578 125 (5^{15}) tests. This is very unlikely that the project timescales would allow for this number of tests. So, accessing and managing risk is one of the most important activities and reason for testing in any project.

3) Early testing: In the software development life cycle testing activities should start as early as possible and should be focused on defined objectives.

4) Defect clustering: A small number of modules contains most of the defects discovered during pre-release testing or shows the most operational failures.

5) Pesticide paradox: If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, it is really very important to review the test cases regularly and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

6) Testing is context depending: Testing is basically context dependent. Different kinds of sites are tested differently. For example, safety – critical software is tested differently from an e-commerce site.

7) Absence – of – errors fallacy: If the system built is unusable and does not fulfil the user’s needs and expectations then finding and fixing defects does not help.

What is fundamental test process in software testing?

Testing is a process rather than a single activity. This process starts from test planning then designing test cases, preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps:

1) Planning and Control
2) Analysis and Design
3) Implementation and Execution
4) Evaluating exit criteria and Reporting
5) Test Closure activities

1) Planning and Control:

Test planning has following major tasks:
i. To determine the scope and risks and identify the objectives of testing.
ii. To determine the test approach.
iii. To implement the test policy and/or the **test strategy**. (Test strategy is an outline that describes the testing portion of the software development cycle. It is created to inform PM, testers and developers about some key issues of the testing process. This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.).
iv. To determine the required test resources like people, test environments, PCs, etc.
v. To schedule test analysis and design tasks, test implementation, execution and evaluation.
vi. To determine the **Exit criteria** we need to set criteria such as **Coverage criteria.**(Coverage criteria are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

Test control has the following major tasks:
i. To measure and analyze the results of reviews and testing.
ii. To monitor and document progress, test coverage and exit criteria.
iii. To provide information on testing.
iv. To initiate corrective actions.
v. To make decisions.

2) Analysis and Design:

<u>Test analysis</u> and <u>Test Design</u> has the following major tasks:
i. To review the **test basis.** (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
ii. To identify test conditions.
iii. To design the tests.
iv. To evaluate testability of the requirements and system.
v. To design the test environment set-up and identify and required infrastructure and tools.

3) Implementation and Execution:
During test implementation and execution, we take the test conditions into **test cases** and procedures and other **testware** such as scripts for automation, the test environment and any other test infrastructure. (Test cases are a set of conditions under which a tester will determine whether an application is working correctly or not.)
(Testware is a term for all utilities that serve in combination for testing software like scripts, the test environment and any other test infrastructure for later reuse.)

Test implementation has the following major task:
i. To develop and prioritize our test cases by using techniques and create **test data** for those tests. (In order to test a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data.)
We also write some instructions for carrying out the tests which is known as **test procedures.**
We may also need to automate some tests using **<u>test harness</u>** and automated tests scripts. (A test harness is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)
ii. To create test suites from the test cases for efficient test execution.
(Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviors. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)
iii. To implement and verify the environment.

Test execution has the following major task:
i. To execute test suites and individual test cases following the test procedures.
ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or re-testing.**
iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial. (A test log is nothing but, what

are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
iv. To Compare actual results with expected results.
v. Where there are differences between actual and expected results, it report discrepancies as Incidents.

4) Evaluating Exit criteria and Reporting:
Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the "enough testing". These criteria vary from project to project and are known as **exit criteria**.
Exit criteria come into picture, when:
— Maximum test cases are executed with certain pass percentage.
— Bug rate falls below certain level.
— When achieved the deadlines.

Evaluating exit criteria has the following major tasks:
i. To check the test logs against the exit criteria specified in test planning.
ii. To assess if more test are needed or if the exit criteria specified should be changed.
iii. To write a test summary report for stakeholders.

5) Test Closure activities:
Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing.
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

Test closure activities have the following major tasks:
i. To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.
ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.
iii. To handover the testware to the maintenance organization. They will give support to the software.
iv To evaluate how the testing went and learn lessons for future releases and projects.

What is Software Quality?

Quality software is reasonably bug or defect free, delivered on time and within budget, meets requirements and/or expectations, and is maintainable.

ISO 8402-1986 standard defines quality as “the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs.”

Key aspects of quality for the customer include:

- Good design – looks and style
- Good functionality – it does the job well
- Reliable – acceptable level of breakdowns or failure
- Consistency
- Durable – lasts as long as it should
- Good after sales service
- Value for money

Good design – looks and style:

It is very important to have a good design. The application or product should meet all the requirement specifications and at the same time it should be user friendly. The customers are basically attracted by the good looks and style of the application. The right color combinations, font size and the styling of the texts and buttons are very important.

Good functionality – it does the job well:

Along with the good looks of the application or the product it’s very important that the functionality should be intact. All the features and their functionality should work as expected. There should not be any deviation in the actual result and the expected result.

Reliable – acceptable level of breakdowns or failure:

After we have tested for all the features and their functionalities it also very important that the application or product should be reliable. For example: There is an application of saving the students records. This application should save all the students records and should not fail after entering 100 records. This is called reliability.

Consistency:

The software should have consistency across the application or product. Single software can be multi-dimensional. It is very important that all the different dimensions should behave in a consistent manner.

Durable – lasts as long as it should:

The software should be durable. For example if software is being used for a year and the number of data has exceed 5000 records then it should not fail if number of records increases. The software product or application should continue to behave in the same way without any functional breaks.

Good after sales service:

Once the product is shipped to the customers then maintenance comes into the picture. It is very important to provide good sales services to keep the customers happy and satisfied. For example if after using the product for six months the customer realizes to make some changes to the application then those changes should be done as fast as possible and should be delivered to the customers on time with quality.

Value for money:

It's always important to deliver the product to the customers which have value for money. The product should meet the requirement specifications. It should work as expected, should be user friendly. We should provide good services to the customers. Other than the features mentioned in the requirement specifications some additional functionality could be given to the customers which they might not have thought of. These additional functionalities should make their product more users friendly and easy to use. This also adds value for money.

Verification Vs. Validation

Verification	Validation
• It makes sure that the product is designed to deliver all functionality to the customer. • **Am I building the product right?** • Verification is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications	• Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs. • **Am I building the right product?** • Validation is done at the end of the development process and takes place after verifications are completed.

What are the Software Development Life Cycle (SDLC) phases?

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

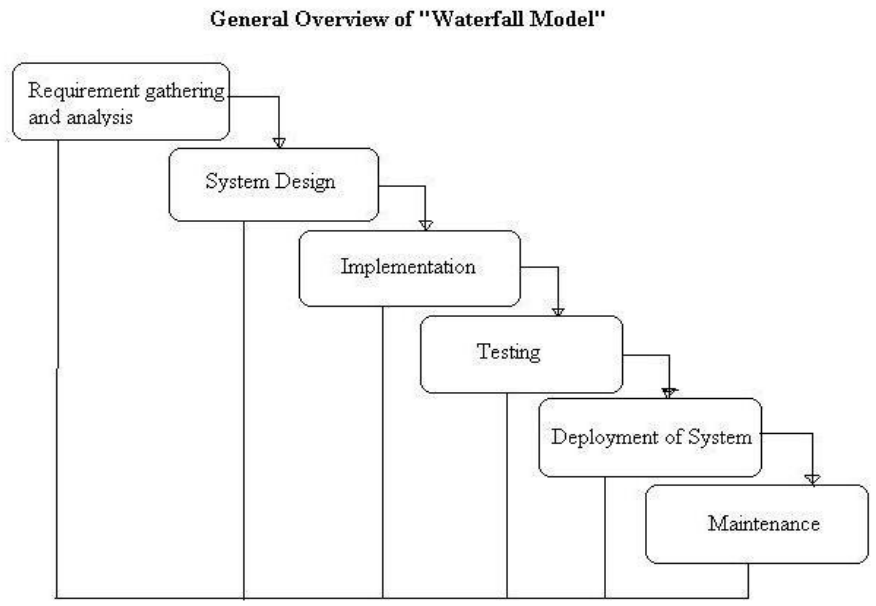
What are the Software Development Models?

1. Waterfall model
2. V model
3. Iterative model
4. Spiral model

What is Waterfall model?

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

Diagram of Waterfall-model:

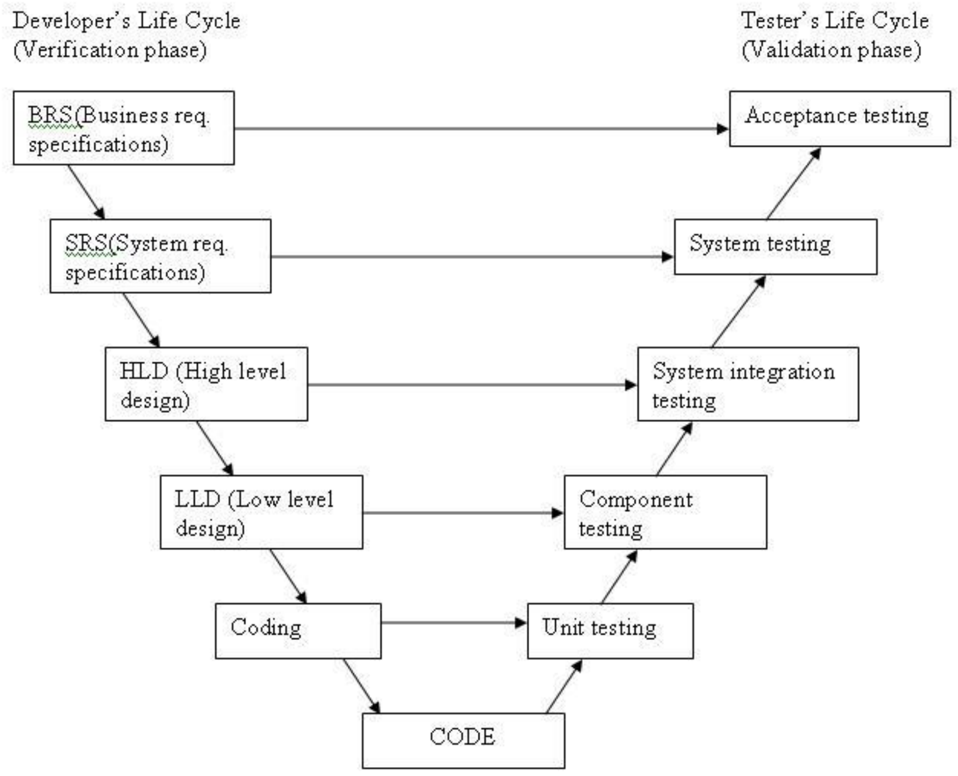

General Overview of "Waterfall Model"
Requirement gathering and analysis
System Design
Implementation
Testing
Deployment of System
Maintenance

- Waterfall model works well for smaller projects where requirements are very well understood.

What is V-model?

V- model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.

Diagram of V-model:


Developer's Life Cycle (Verification phase)
Tester's Life Cycle (Validation phase)
BRS(Business req. specifications)
SRS(System req. specifications)
HLD (High level design)
LLD (Low level design)
Coding
CODE
Unit testing
Component testing
System integration testing
System testing
Acceptance testing

The various phases of the V-model are as follows:

Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in the requirements gathering.

The high-level design (HLD) phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An integration plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system.

Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Coding: This is at the bottom of the V-Shape model. Module design is converted into code by developers.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

What is Spiral model?

The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

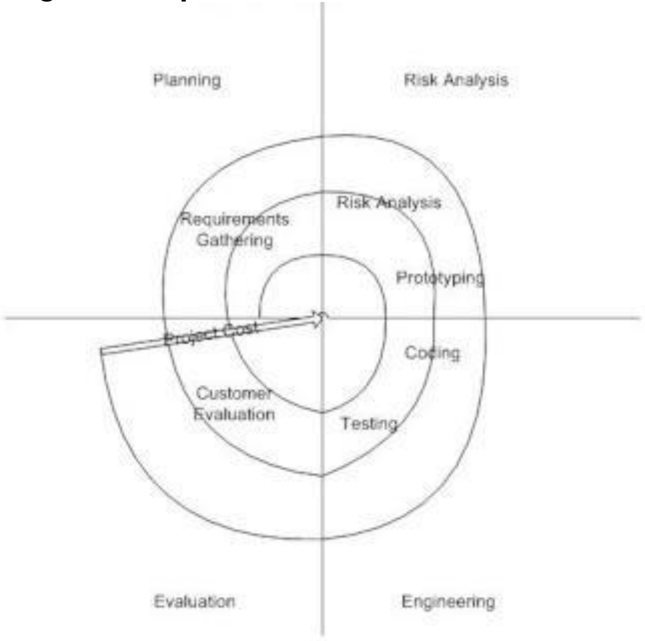
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is **developed**, along with testing at the end of the phase. Hence in this phase the development and testing is done.

E**valuation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Diagram of Spiral model:



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

What are Software Testing Levels?

Testing levels are basically to identify missing areas and prevent overlap and repetition between the development life cycle phases. In software development life cycle models there are defined phases like requirement gathering and analysis, design, coding or implementation, testing and deployment. Each phase goes through the testing. Hence there are various levels of testing. The various levels of testing are:

1. Unit testing: It is basically done by the developers to make sure that their code is working fine and meet the user specifications. They test their piece of code which they have written like classes, functions, interfaces and procedures.
   The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.

2. Component testing: It is also called as module testing. The basic difference between the unit testing and component testing is in unit testing the developers test their piece of code but in component testing the whole component is tested. For example, in a student record application there are two modules one which will save the records of the students and other module is to upload the results of the students. Both the modules are developed separately and when they are tested one by one then we call this as a component or module testing.

3. Integration testing: Integration testing is done when two modules are integrated, in order to test the behavior and functionality of both the modules after integration. Below are few types of integration testing:
   - Big bang integration testing
   - Top down
   - Bottom up
   - Functional incremental
   -
4. Component integration testing: It tests the interactions between software components and is done after component testing.
   The software components themselves may be specified at different times by different specification groups, yet the integration of all of the pieces must work together.

5. System integration testing: It tests the interactions between different systems and may be done after system testing.
   It verifies the proper execution of software components and proper interfacing between components within the solution.

6. System testing: In system testing the behavior of whole system/product is tested as defined by the scope of the development project or product.

7. Acceptance testing: After the system test has corrected all or most defects, the system will be delivered to the user or customer for acceptance testing.
   Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well.
   The goal of acceptance testing is to establish confidence in the system.

- The **User Acceptance test:** focuses mainly on the functionality thereby validating the fitness-for-use of the system by the business user. The user acceptance test is performed by the users and application managers.
- The **Operational Acceptance test:** also known as Production acceptance test validates whether the system meets the requirements for operation. In most of the organization the operational acceptance test is performed by the system administration before the system is released. The operational acceptance test may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities.

- **Contract Acceptance testing**: It is performed against the contract's acceptance criteria for producing custom developed software. Acceptance should be formally defined when the contract is agreed.
- **Compliance acceptance testing:** It is also known as regulation acceptance testing is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

8. Alpha testing: Alpha testing is done at the developer's site. It is done at the end of the development process

9. Beta testing: It is also known as field testing. It takes place at **customer's site**. It sends the system to users who install it and use it under real-world working conditions.

What are Software Test Types?

A test type is focused on a particular test objective,

1. which could be the testing of the **function** to be performed by the component or system;
2. a **non-functional** quality characteristics, such as reliability or usability;
3. the **structure** or architecture of the component or system;
4. or related to changes, i.e. confirming that defects have been fixed (**confirmation testing** or **retesting**)
5. & looking for unintended changes (**regression testing**).

There are four software test types:

- Functional testing
- Non-functional testing
- Structural testing
- Change related testing

What is Functional testing?

Functional Testing is a testing technique that is used to test the features/functionality of the system or Software, should cover all the scenarios including failure paths and boundary cases.

There are two major Functional Testing techniques:

Black box testing & white box testing

What is Non-functional testing?

Non-Functional testing is a software testing technique that verifies the attributes of the system such as memory leaks, performance or robustness of the system. Non-Functional testing is performed at all test levels.

Non-functional testing includes:

- **Functionality testing:** Functionality testing is performed to verify that a software application performs and functions correctly according to design specifications. During functionality testing we check the core application functions, text input, menu functions and installation and setup on localized machines, etc.

- **Reliability testing:** Reliability Testing is about exercising an application so that failures are discovered and removed before the system is deployed. The purpose of reliability testing is to determine product reliability, and to determine whether the software meets the customer’s reliability requirements.

- **Usability testing:** In usability testing basically the testers tests the ease with which the user interfaces can be used. It tests that whether the application or the product built is user-friendly or not. (It is Black Box Testing Technique.)

  Usability Testing tests the following features of the software.

  — How easy it is to use the software.
  — How easy it is to learn the software.
  — How convenient is the software to end user.

  Usability testing includes the following five components:

  1. **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design?

2. **Efficiency:** How fast can experienced users accomplish tasks?

3. **Memorability:** When users return to the design after a period of not using it, does the user remember enough to use it effectively the next time, or does the user have to start over again learning everything?

4. **Errors:** How many errors do users make, how severe are these errors and how easily can they recover from the errors?

5. **Satisfaction:** How much does the user like using the system?

- **Efficiency testing:** Efficiency testing test the amount of code and testing resources required by a program to perform a particular function. Software Test Efficiency is number of test cases executed divided by unit of time (generally per hour).

- **Maintainability testing:** It basically defines that how easy it is to maintain the system. This means that how easy it is to analyze, change and test the application or product.

- **Portability testing:** It refers to the process of testing the ease with which a computer software component or application can be moved from one environment to another, e.g. moving of any application from Windows 2000 to Windows XP. This is usually measured in terms of the maximum amount of effort permitted. Results are measured in terms of the time required to move the software and complete the and documentation updates.

- **Baseline testing:** It refers to the validation of documents and specifications on which test cases would be designed. The requirement specification validation is baseline testing.

- **Compliance testing:** It is related with the IT standards followed by the company and it is the testing done to find the deviations from the company prescribed standards.

- **Documentation testing:** As per the IEEE Documentation describing plans for, or results of, the testing of a system or component, Types include test case specification, test incident report, test log, test plan, test procedure, test report. Hence the testing of all the above mentioned documents is known as documentation testing.

- **Endurance testing:** Endurance testing involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. For example, in software testing, a system may behave exactly as expected when tested for 1 hour but when the same system is tested for 3 hours, problems such as memory leaks cause the system to fail or behave randomly.

- **Load testing:** A load test is usually conducted to understand the behavior of the application under a specific expected load. Load testing is performed to determine a system's behavior under both normal and at peak conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. E.g. If the number of users are in creased then how much CPU, memory will be consumed, what is the network and bandwidth response time

- **Performance testing:** Performance testing is testing that is performed, to determine how fast some aspect of a system performs under a particular workload. It can serve different purposes like it can demonstrate that the system meets performance criteria. It can compare two systems to find which performs better. Or it can measure what part of the system or workload causes the system to perform badly.

- **Compatibility testing:** Compatibility testing is basically the testing of the application or the product built with the computing environment. It tests whether the application or the software product built is compatible with the hardware, operating system, database or other system software or not.

- **Security testing:** Security testing is basically to check that whether the application or the product is secured or not. Can anyone came tomorrow and hack the system or login the application without any authorization. It is a process to determine that an information system protects data and maintains functionality as intended.

- **Scalability testing:** It is the testing of a software application for measuring its capability to scale up in terms of any of its non-functional capability like load supported, the number of transactions, the data volume etc.

- **Volume testing:** Volume testing refers to testing a software application or the product with a certain amount of data. E.g., if we want to volume test our application with a specific database size, we need to expand our database to that size and then test the application's performance on it.

- **Stress testing:** It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. It is a form of testing that is used to determine the stability of a given system. It put greater emphasis on robustness, availability, and error handling under a heavy load, rather than on what would be considered correct behavior under normal circumstances. The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

- **Recovery testing:** Recovery testing is done in order to check how fast and better the application can recover after it has gone through any type of crash or hardware failure etc. Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed. For example, when an application is receiving data from a network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection got disappeared. Restart the system while a browser has a definite number of sessions and check whether the browser is able to recover all of them or not.

- **Internationalization testing and Localization testing:** Internationalization is a process of designing a software application so that it can be adapted to various languages and regions without any changes. Whereas Localization is a process of adapting internationalized software for a specific region or language by adding local specific components and translating text.

What is Confirmation testing?

Confirmation testing or re-testing: When a test fails because of the defect then that defect is reported and a new version of the software is expected that has had the defect fixed. In this case we need to execute the test again to confirm that whether the defect got actually fixed or not. This is known as confirmation testing and also known as re-testing. It is important to ensure that the test is executed in exactly the same way it was the first time using the same inputs, data and environments.

Hence, when the change is made to the defect in order to fix it then confirmation testing or re-testing is helpful.

What is Regression testing?

During confirmation testing the defect got fixed and that part of the application started working as intended. But there might be a possibility that the fix may have introduced or uncovered a different defect elsewhere in the software. The way to detect these '**unexpected side-effects**' of fixes is to do regression testing. The purpose of a regression testing is to verify that modifications in the software or the environment have not caused any unintended adverse side effects and that the system still meets its requirements

What is Structural testing?

Structural testing, also known as glass box testing or white box testing is an approach where the tests are derived from the knowledge of the software's structure or internal implementation.

What is Maintenance Testing?

Once a system is deployed it is in service for years and decades. During this time the system and its operational environment is often corrected, changed or extended. Testing that is provided during this phase is called maintenance testing.

What is Exhaustive Testing?

Exhaustive testing is a test approach in which all possible data combinations are used for testing. Exploratory testing includes implicit data combinations present in the state of the software/data at the start of testing.

What is test design technique?

Test Design is creating a set of inputs for given software that will provide a set of expected outputs.

Test Design Techniques

1. Static Techniques
2. Dynamic Techniques

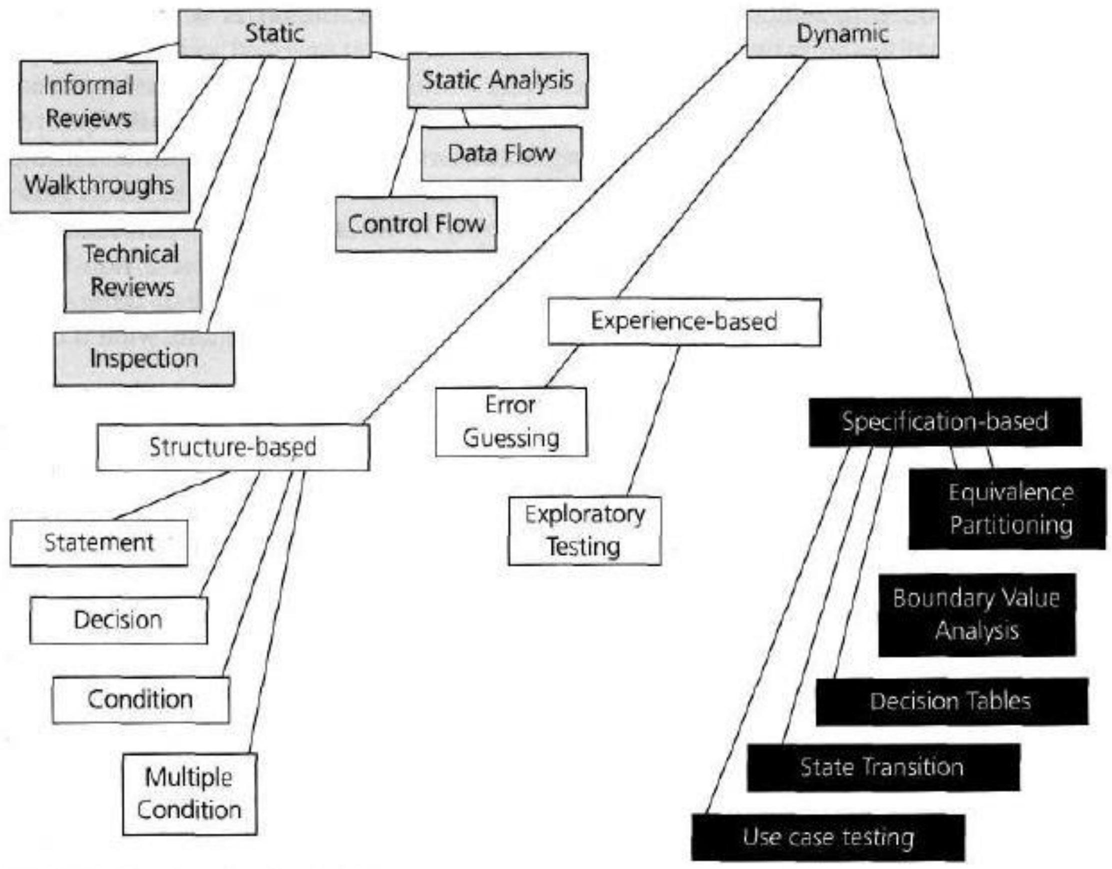


FIGURE 4.1 Testing techniques

<u>What is a static test technique?</u>

Static test techniques provide a great way to improve the quality and productivity of software development. It includes the reviews and provides the overview of how they are conducted. The primary objective of static testing is to improve the quality of software products by assisting engineers to recognize and fix their own defects early in the software development process.

What is static Testing?

- Static testing is the testing of the software work products manually, or with a set of tools, but they are **not executed**.
- It starts early in the Life cycle and so it is done during the verification process.
- It does not need computer as the testing of program is done without executing the program. For example: reviewing, walk through, inspection, etc.

Types of static Testing?

- People based: people-based techniques are generally known as reviews. (Individual –Desk checking, Data stepping & group – formal, informal, walkthrough, inspection)
- Tool based: the tool – based techniques examine source code & known as static analysis. (coding standards, code metrics, code structure)

Reviews

Static Review provides a powerful way to improve the quality and productivity of software development to recognize and fix their own defects early in the software development process.
Nowadays, all software organizations are conducting reviews in all major aspects of their work including requirements, design, implementation, testing, and maintenance.

Advantages of reviews:

- Development productivity improvement
- Reduced development timescale
- Reduced testing time & cost

Informal reviews

Informal reviews are applied many times during the early stages of the life cycle of the document. A two person team can conduct an informal review. In later stages these reviews often involve more people and a meeting. The goal is to keep the author and to improve the quality of the document. The most important thing to keep in mind about the informal reviews is that they are **not documented.**

Formal review

Formal reviews follow a formal process. It is well structured and regulated.
A formal review process consists of six main steps:

1. Planning
2. Kick-off
3. Preparation
4. Review meeting
5. Rework
6. Follow-up

Walkthrough

A walkthrough is a form of review that is typically used to educate a group of people about a technical document. Typically the author “walks” the group through the ideas to explain them so that the attendees understand the content.

Inspection

Inspection is the most formal of all the formal review techniques. Its main focus during the process is to find faults, and is the most effective review technique.

Desk checking, proof reading

Are 2 techniques that can be used by individuals to review a document such as a specification or a piece of source code. They are basically the same processes. Here the review double checks the document or source code on their own.

Data stepping

Is a slightly different process for reviewing source code: here the reviewer follows a set of data values through the source code to ensure that the values are correct at each step of the processing

What is test design technique?

A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system.

What are the categories of test design techniques?

Dynamic techniques are subdivided into three more categories:

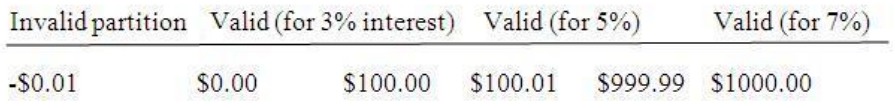
- specification-based - (black-box, also known as behavioral techniques)
- structure-based - (white-box or structural techniques)
- Experience- based.

Black-Box Testing	**Grey-Box Testing**	**White-Box Testing**
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations -	Testing is done on the basis of high-level database	Internal workings are fully known and the

Internal behavior of the application is unknown.	diagrams and data flow diagrams.	tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

<u>What is Equivalence partitioning in Software testing?</u>

- Equivalence partitioning can be applied at any level of testing and is often a good technique to use first.
- The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups
- In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.
  - For example, a savings account in a bank has a different rate of interest depending on the balance in the account. In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest. For example, 3% rate of interest is given if the balance in the account is in the range of $0 to $100, 5% rate of interest is given if the balance in the account is in the range of $100 to $1000, and 7% rate of interest is given if the balance in the account is $1000 and above, we would initially identify three valid equivalence partitions and one invalid partition as shown below.



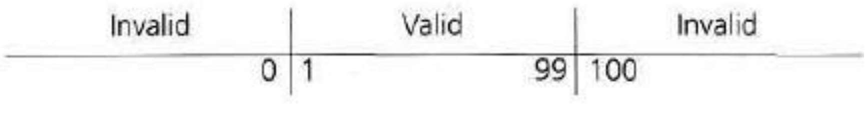
Invalid partition	Valid (for 3% interest)		Valid (for 5%)		Valid (for 7%)
-$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

What is Boundary value analysis in software testing?

- Boundary value analysis (BVA) is based on testing at the boundaries between partitions.
- As an example, consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case). In this example we would have three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests. Consider the bank system described in the previous section in equivalence partitioning.

While testing why it is important to do both equivalence partitioning and boundary value analysis?

Technically, because every boundary is in some partition, if you did only boundary value analysis you would also have tested every equivalence partition. However, this approach may cause problems if that value fails – was it only the boundary value that failed or did the whole partition fail? Also by testing only boundaries we would probably not give the users much confidence as we are using extreme values rather than normal values. The boundaries may be more difficult (and therefore more costly) to set up as well. For example, in the printer copies example described earlier we identified the following boundary values:



Suppose we test only the valid boundary values 1 and 99 and nothing in between. If both tests pass, this seems to indicate that all the values in between should also work. However, suppose that one page prints correctly, but 99 pages do not. Now we don't know whether any set of more than one page works, so the first thing we would do would be to test for say 10 pages, i.e. a value from the equivalence partition. We recommend that you test the partitions separately from boundaries – this means choosing partition values that are NOT boundary values. However, if you use the three-value boundary value approach, then you would have valid boundary values of 1, 2, 98 and 99, so having a separate equivalence value in addition to the extra two boundary values would not give much additional benefit. But notice that one equivalence value, e.g. 10, replaces both of the extra two boundary values (2 and 98). This is why equivalence partitioning with two-value boundary value analysis is more efficient than three-value boundary value analysis.

What is Decision table Testing technique?

Decision tables are one of the simplest & most powerful technique in dealing with complex logic. A decision table comprises a set of conditions & a set of actions to be performed.

……….

Cause – effect graphing testing technique?

…….

What is State transition testing?

State transition testing is used where some aspect of the system can be described in what is called a 'finite state machine'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'. This is the model on which the system and the tests are based.

…………

What is Use case testing?

Use case testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.

………….

What is Experience- based testing technique?

In **experience-based techniques,** people’s knowledge, skills and background are of prime importance to the test conditions and test cases.

………..

What is Error guessing?

The Error guessing is a technique where the experienced and good testers are encouraged to think of situations in which the software may not be able to cope. Some people seem to be naturally good at testing and others are good testers because they have a lot of experience either as a tester or working with a particular system and so are able to find out its weaknesses. This is why an error guessing approach, used after more formal techniques have been applied to some extent, can be very effective.