Chap 01 Testing Methodology

☸Software Testing Methodology

- Software Testing Methodology is defined as strategies and testing types used to certify that the Application Under Test meets client expectations.
- Test Methodologies include functional and non-functional testing to validate the AUT.
- Examples of Testing Methodologies are Unit Testing, Integration Testing, System Testing, Performance Testing etc

Failure: When a system is tested, failure is the first term that is used. It means the inability of the system or a component of the system to perform a required function according to its specification. In other words, when results or behavior of the system under test are different as compared to specified expectations, failure exists.

Fault/Defect/Bug: Failure is a term used to describe the problem's in a system on the output side, as shown in Fig.1. Fault is a condition that in actual causes a system to produce failure. Fault is synonymous with the words defect or bug. Therefore, fault is the reason embedded in any phase of SDLC, which results in failures. It can be said that failures are manifestation of bugs. One failure may be due to one or more bugs and one bug may cause one or more failures. When a bug is executed, failures are generated. However, this is not always true. Some bugs are hidden in the sense that these are not executed, as they do not get the required conditions in the system. So, hidden bugs may not always produce failures. They may execute only in certain rare conditions.

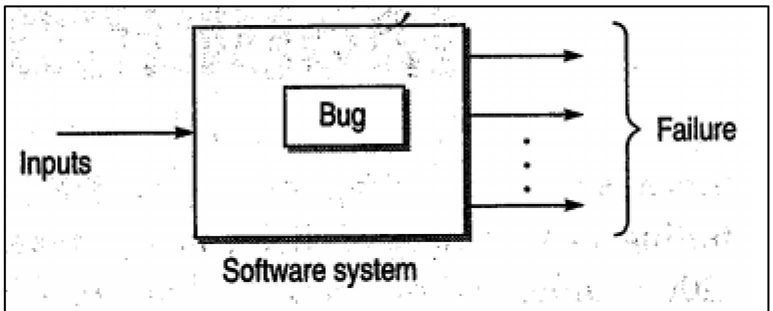


Figure 1: Testing Terminology

Error: Whenever a development team member makes a mistake in any phase of SDLC, errors are produced. It might be a typographical error, a misleading specification, a misunderstanding of what a subroutine does, etc. Error is a very general term used for human mistakes. Thus, an error causes a bug and the bug in turn causes failures, as shown in Fig. 2.

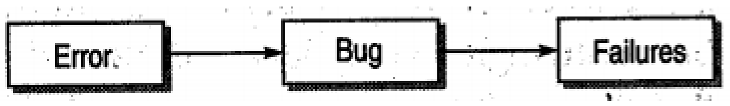


Figure 2: Flow of Faults

Test case: Test case is a well-documented procedure designed to test the functionality of a feature in the system. A test case has an identity and is associated with a program behavior. The primary purpose of designing a test case is to find errors in the system. A test case needs to specify a set of inputs and the corresponding expected outputs. The sample for a test case is shown in Fig.3.

- *Test case ID:* It is the identification number given to each test case.
- *Purpose:* It defines why the case is being designed.
- *Preconditions:* It can be defined, for running the inputs in a system if required in a test case.
- *Inputs:* It should not be hypothetical. Actual inputs must be provided, instead of general inputs. For example, if two integer numbers have to be provided as input, then specifically mention them as 23 and 56.
- *Expected outputs:* They are the outputs, which should be produced when there is no failure.
- Test cases are designed based on the chosen testing techniques. They provide inputs when the system is executed. After execution, observed results are compared with expected outputs mentioned in the test case.

Testware: The documents created during testing, activities are known as testware. It may include test plans, test specifications, test case test reports, etc. Testware documents 'should also be managed and updated like a software product.

Incident: When a failure occurs, it may, or may not be readily apparent to the user. An incident is the symptom(s) associated with a failure that alerts the user about the occurrence of a failure.

Test oracle: An oracle is the means to judge the success or failure of a test, that is, to judge the correctness of the system for some test. The simplest oracle is comparing actual results with expected results by hand. This can be very time-consuming, so automated oracles are sought.

⚛Goals of Software Testing:

The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free.

Important Goals of Software Testing:

- Detecting bugs as soon as feasible in any situation.
- Avoiding errors in a projects and product's final versions.
- Inspect to see whether the customer requirements criterion has been satisfied.
- Last but not least, the primary purpose of testing is to scale the project and product level of quality.

The goals of software testing may be classified into three major categories as follows:

- Immediate Goals
- Long-term Goals
- Post-Implementation Goals

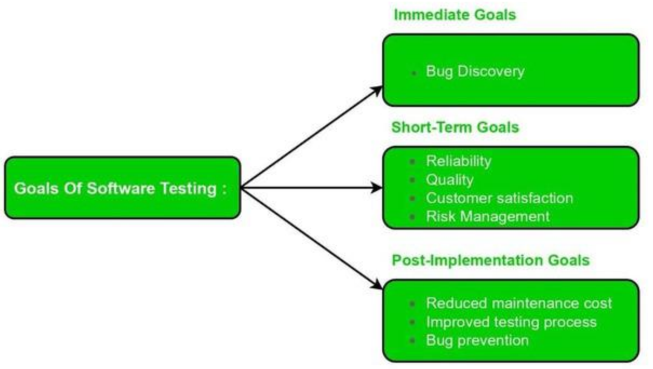


Fig : Software Testing Goals

1. Immediate Goals: These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

Bug Discovery: This is the immediate goal of software testing to find errors at any stage of software development. The number of bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process.

Bug Prevention: This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

2.Long-Term Goals: These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

Quality: This goal enhances the quality of the software product. Because software is also a product, the user's priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality.

Customer Satisfaction: This goal verifies the customer's satisfaction with a developed software product. The primary purpose of software testing, from the user's standpoint, is customer satisfaction.

Reliability: It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.

Risk Management: Risk is the probability of occurrence of uncertain events in the organization and the potential loss that could result in negative consequences.

3.Post-Implemented Goals: After the product is released, these objectives become critical. Some of these are covered in detail below:

Reduce Maintenance Cost: Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify.

Improved Software Testing Process: These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement.

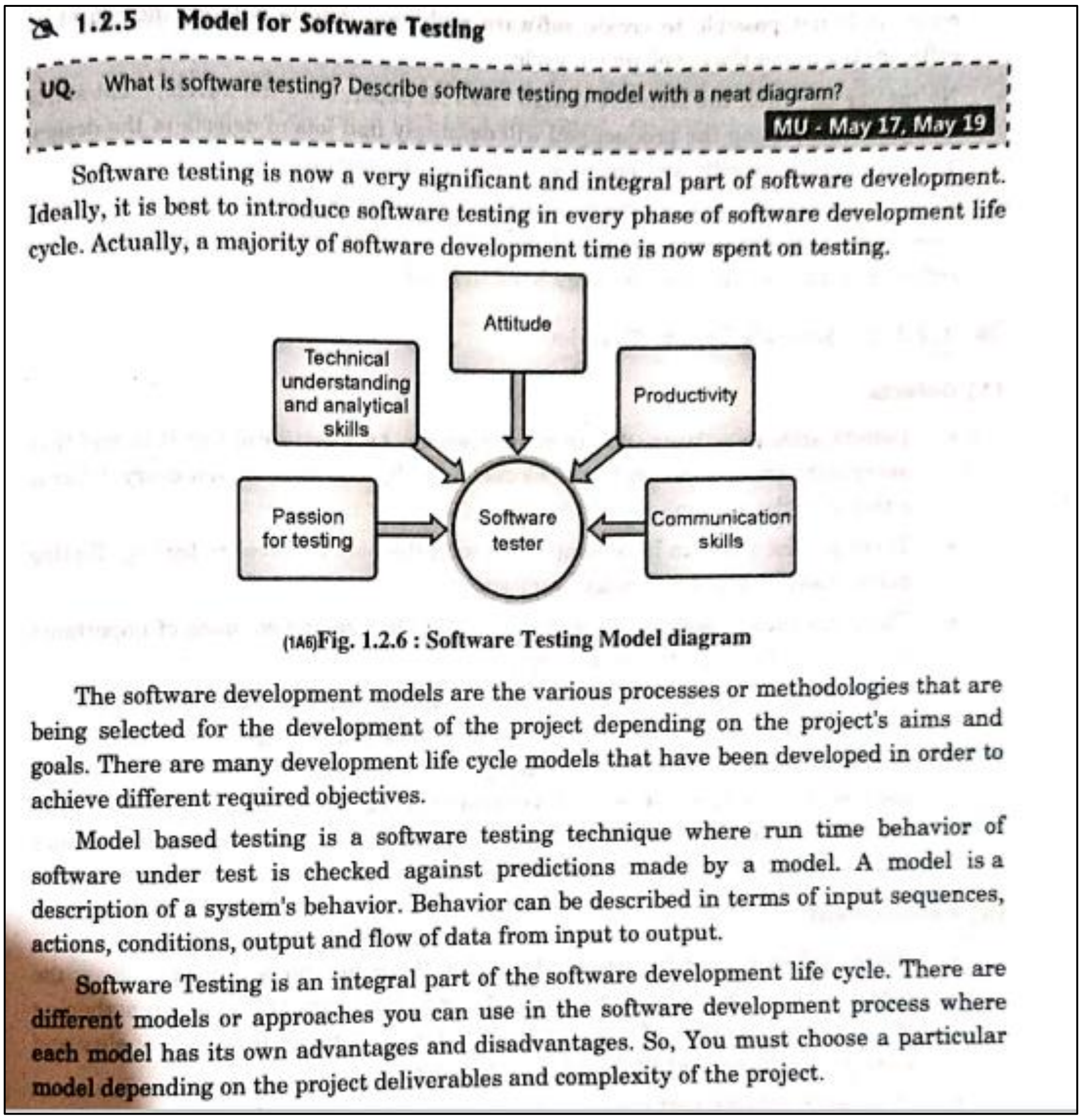
✳What is Software Testing?

Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.

The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

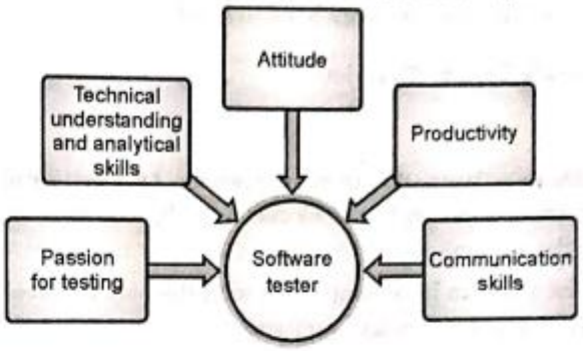
1. **Verification:** It refers to the set of tasks that ensure that the software correctly implements a specific function. It means "Are we building the product right?".
2. **Validation:** It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means "Are we building the right product?".



1.2.5 Model for Software Testing

UQ. What is software testing? Describe software testing model with a neat diagram? **MU - May 17, May 19**

Software testing is now a very significant and integral part of software development. Ideally, it is best to introduce software testing in every phase of software development life cycle. Actually, a majority of software development time is now spent on testing.



(1A6)Fig. 1.2.6 : Software Testing Model diagram

The software development models are the various processes or methodologies that are being selected for the development of the project depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives.

Model based testing is a software testing technique where run time behavior of software under test is checked against predictions made by a model. A model is a description of a system's behavior. Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.

Software Testing is an integral part of the software development life cycle. There are different models or approaches you can use in the software development process where each model has its own advantages and disadvantages. So, You must choose a particular model depending on the project deliverables and complexity of the project.

Aspect	Effective Testing	Exhaustive Testing
Definition	Selective testing of critical and high-impact areas	Testing all possible combinations and scenarios
Goal	Focuses on identifying major defects and risks	Aims to find all possible defects and issues
Coverage	Covers a subset of test cases	Covers all possible test cases
Efficiency	More efficient; targets key areas	Extremely resource-intensive and time-consuming
Time and Resources	Requires fewer resources and less time	Requires substantial time and resources
Real-world Usage	Suitable for most projects and time constraints	Rarely used due to impracticality
Completeness	Doesn't ensure complete coverage	Theoretically complete but often not feasible
Bug Discovery	May miss some rare defects	Likely to uncover more defects
Common Approach	Used in most software testing scenarios	Rarely applied due to time and resource limits

- Usually the program fails due to race conditions, as the possibility of preceding B in restricted condition has not been taken care, resulting in a race condition bug.
- In this way, there may be many race conditions in the system, especially in multiprocessing and interactive systems. Race conditions are among the least tested.

1.4 SOFTWARE FAILURE CASE STUDIES

- Software systems have become the backbone of almost all the organizations worldwide and how much ever you try to avoid them you end up using software systems in your daily life as a person and as an organization.
- With over 2.9 billion* of world population on Internet and rapid modernization of countries across the world, it has become inevitable to avoid the software footprint in your everyday life. Adoption of smart phones has made access to software applications more of a convenience and necessity.
- With mission critical and high-risk applications which have human lives and resources on risk depending on software applications, testing not only for expected but aiming for zero defects is required. We have listed the **Top 10 Mega software failures** which resulted in severe disruption and loss of resources in the current year which could have been avoided.

1.4.1 Top 8 Mega Software Failures

(1) Amazon Christmas Glitch	(2) Air traffic control centre NATS
(3) Microsoft Azure Crashes	(4) Bitcoin Exchange Collapse
(5) Screwfix £34.99 Price Glitch	(6) HMRC's Big Tax Blunder
(7) UK border and immigration system	(8) Sony Pictures Entertainment

(1) Amazon Christmas Glitch

- It was quite a surprise for vendors to see their products on sale for just One penny in Amazon marketplace.
- It was a festive bonanza for shoppers and many picked up items as expensive as mobile phones for just 1 penny.
- This glitch was attributed to a bug in Amazon price comparison software and resulted in $100,000 for the vendors.

(2) Air traffic control centre NATS

Another Christmas incident which affected the travel plans of more than 10000 passengers and leading to heavy delays was attributed to one line of software code which was unaltered since late 1960 and written in defunct language Jovial.

▶ (3) Microsoft Azure Crashes

- Microsoft's office 365, Xbox live gaming and Websites using the Azure platform crashed due a bug in the famous cloud computing platform.
- Many customers were unable to access the service due to this glitch which was following a performance update.
- On the Azure blog, Microsoft stated : *"The configuration change for the Blob [Binary Large Object] front-ends exposed a bug in the Blob front-ends, which had been previously performing as expected."* Service was down for more than 11 hours.

▶ (4) Bitcoin Exchange Collapse

Not implementing a feature which keeps track of the transactions proved costly for Mt. Gox, a Bitcoin exchange when lost $500 million of its virtual currency when someone hacked the exchange.

▶ (5) Screwfix £34.99 Price Glitch

- A price glitch at screwfix's website was the reason for customer's joy as all the items went on Sale for just £34.99. From garden tractors to expensive power tools, all the items priced to the delight of the shoppers.
- All this was attributed to a Data validation error and requires a redo at the testing and validation solutions used by the business. Automation testing could have avoided this glitch as the script would have flagged the glitch immediately.

▶ (6) HMRC's Big Tax Blunder

- Her Majesty's Revenue and Customs (HMRC) which is responsible for collection of taxes in UK was hit by a bug in its PAYE (Pay As You Earn) system which has affected more than 5.7 million people.
- Error in PAYE resulted in wrong tax code allotted to tax payers due to which many paid more than the actual tax and many ended up paying less tax for the last 2 financial years.

▶ (7) UK border and immigration system

- One of the costliest software failure which is estimated to cost up to £1 billion. The system was incapable of dealing with backlog cases and resulted in 29000 applications backlog.
- The department also failed to locate 50000 people when was asked to find about them. What they needed was a comprehensive system wide IT strategy with skilled staff to avoid such issues.

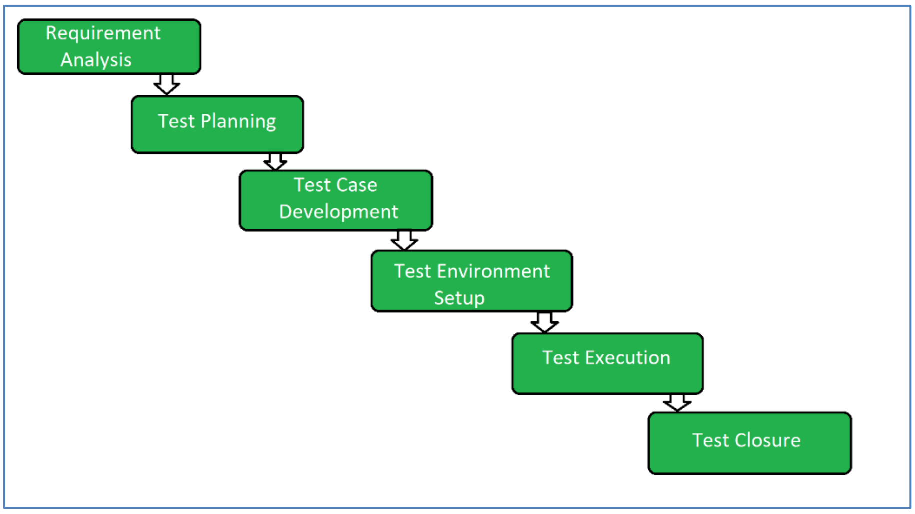
Software Testing Life Cycle (STLC):

The Software Testing Life Cycle (STLC) is a systematic approach to testing a software application to ensure that it meets the requirements and is free of defects.

- It is a process that follows a series of steps or phases, and each phase has specific objectives and deliverables.
- The STLC is used to ensure that the software is of high quality, reliable, and meets the needs of the end-users.
- The main goal of the STLC is to identify and document any defects or issues in the software application as early as possible in the development process.
- This allows for issues to be addressed and resolved before the software is released to the public.

Characteristics of STLC

- STLC is a fundamental part of the Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.
- STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.
- STLC yields a step-by-step process to ensure quality software.



1. Requirement Analysis: Requirement Analysis is the first step of the Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

The activities that take place during the Requirement Analysis stage include:

- Reviewing the software requirements document (SRD) and other related documents
- Interviewing stakeholders to gather additional information
- Identifying any ambiguities or inconsistencies in the requirements
- Identifying any missing or incomplete requirements
- Identifying any potential risks or issues that may impact the testing process

2. Test Planning: Test Planning is the most efficient phase of the software testing life cycle where all testing plans are defined. In this phase manager of the testing, team calculates the estimated effort and cost for the testing work. This phase gets started once the requirement-gathering phase is completed.

The activities that take place during the Test Planning stage include:

- Identifying the testing objectives and scope
- Developing a test strategy: selecting the testing methods and techniques that will be used
- Identifying the testing environment and resources needed
- Identifying the test cases that will be executed and the test data that will be used
- Estimating the time and cost required for testing
- Identifying the test deliverables and milestones
- Assigning roles and responsibilities to the testing team
- Reviewing and approving the test plan

3. Test Case Development: The test case development phase gets started once the test planning phase is completed. In this phase testing team notes down the detailed test cases. The testing team also prepares the required test data for the testing. When the test cases are prepared then they are reviewed by the quality assurance team.

The activities that take place during the Test Case Development stage include:

- Identifying the test cases that will be developed
- Writing test cases that are clear, concise, and easy to understand
- Creating test data and test scenarios that will be used in the test cases
- Identifying the expected results for each test case
- Reviewing and validating the test cases

4. Test Environment Setup: Test environment setup is a vital part of the STLC. Basically, the test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved. either the developer or the customer creates the testing environment.

5. Test Execution: After the test case development and test environment setup test execution phase gets started. In this phase testing team starts executing test cases based on prepared test cases in the earlier step**.**

6. Test Closure: Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main objective of the test closure stage is to ensure that all testing-related activities have been completed and that the software is ready for release.

The main activities that take place during the test closure stage include:

- **Test summary report:** A report is created that summarizes the overall testing process, including the number of test cases executed, the number of defects found, and the overall pass/fail rate.

- **Defect tracking:** All defects that were identified during testing are tracked and managed until they are resolved.

- **Test environment clean-up:** The test environment is cleaned up, and all test data and test artifacts are archived.

- **Test closure report:** A report is created that documents all the testing-related activities that took place, including the testing objectives, scope, schedule, and resources used.

1.7 CLASSIFICATION OF SOFTWARE BUGS

UQ. Classify different types of bugs based on Software development lifecycle. **MU - May 17, Dec. 19**

UQ. Classify bugs based in SDLC ? **MU - May 19**

While the classifiers for the latter two are present in bug tracking systems by default, I recommend setting up a classifier for the division of bugs by their nature as well since it helps streamline the assignment of bug fixing tasks to the responsible teams.

(1) Software Functional defects	(2) Performance defects	(3) Usability defects
(4) Compatibility defects	(5) Security defects	

(1) Software Functional defects

- Functional defects are the errors identified in case the behavior of software is not compliant with the functional requirements. Such types of defects are discovered via functional testing.
- For example, in one of our recent testing projects, a functional defect was found in an ecommerce website's search engine. It didn't return any results when a user typed in a product ID, while it was stated in the requirements that the search could be conducted by both a product's name and ID.

(2) Performance defects

- Performances defects are those bound to software's speed, stability, response time, and resource consumption, and are discovered during performance testing.
- An example of a performance defect is a system's response time being X times longer than that stated in the requirements.

(3) Usability defects

- Usability defects make an application inconvenient to use and, thus, hamper a user's experience with software.
- A content layout that is difficult to scan or navigate and an overly complex signup procedure are the examples of usability defects.
- To identify such defects, Science Soft's test engineers and business analysts (or UX designers) validate software against usability requirements and Web Content Accessibility Guidelines (WCAG) during usability testing.

(4) Compatibility defects

- An application with compatibility errors doesn't show consistent performance on particular types of hardware, operating systems, browsers, and devices or when integrated with certain software or operating under certain network configurations.
- Compatibility testing is carried out in order to discover such issues. For instance, testing a mobile app for car insurance claim estimation, we uncovered that it failed to behave according to the requirements on Android 8.0 and 8.1. The defects were related to the changes in font size, content alignment, and scroll bar.

(5) Security defects

- Penetration Testing Consultant and Certified Ethical Hacker, says: "Security defects are the weaknesses allowing for a potential security attack.
- The most frequent security defects in projects we perform security testing for are encryption errors, susceptibility to SQL injections, XSS vulnerabilities, buffer overflows, weak authentication, and logical errors in role-based access".

✳**Defect Life Cycle or Bug Life Cycle:**

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

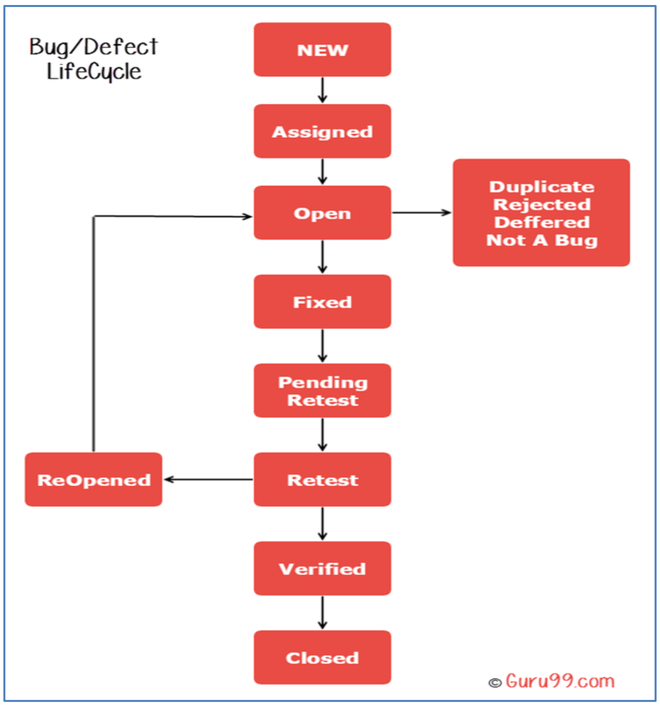
Defect Status

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

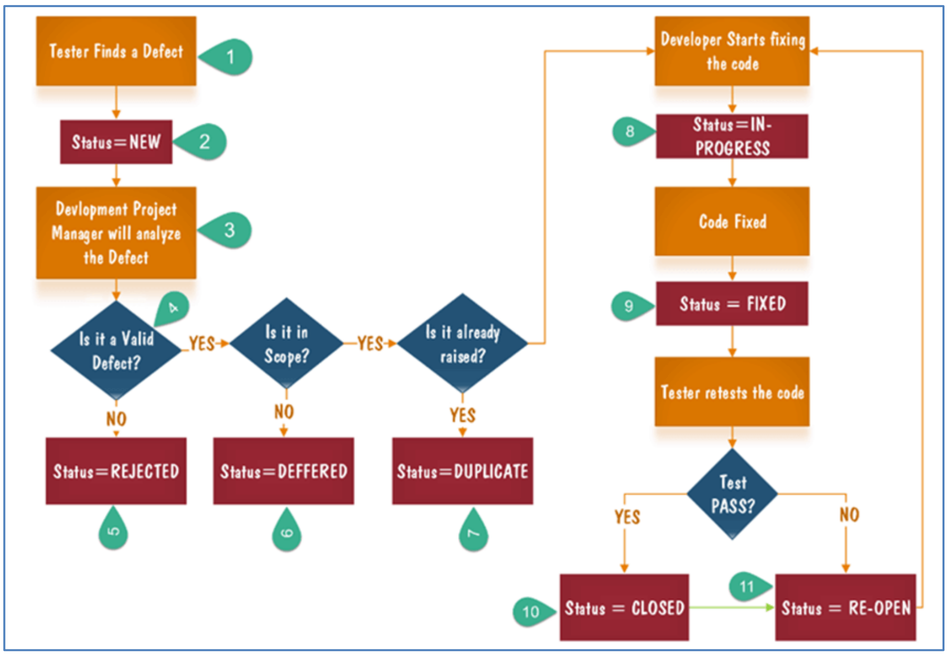
Defect States Workflow

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

- **New:** When a new defect is logged and posted for the first time. It is assigned a status as NEW.
- **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team
- **Open:** The developer starts analyzing and works on the defect fix
- **Fixed:** When a developer makes a necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Pending retest**: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is "pending retest."
- **Retest:** Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to "Re-test."
- Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."
- **Reopen**: If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.
- **Closed**: If the bug is no longer exists then tester assigns the status "Closed."
- **Duplicate**: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to "duplicate."
- **Rejected**: If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."
- **Deferred**: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs
- **Not a bug**: If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".



♻Defect/Bug Life Cycle Explained:



1. Tester finds the defect
2. Status assigned to defect- New
3. A defect is forwarded to Project Manager for analyze
4. Project Manager decides whether a defect is valid
5. Here the defect is not valid- a status is given “Rejected.”
6. So, project manager assigns a status rejected. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a postponed or deferred status.
7. Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status duplicate.

8. If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status in- progress.
9. Once the code is fixed. A defect is assigned a status fixed
10. Next, the tester will re-test the code. In case, the Test Case passes the defect is closed. If the test cases fail again, the defect is re-opened and assigned to the developer.
11. Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be re-opened.

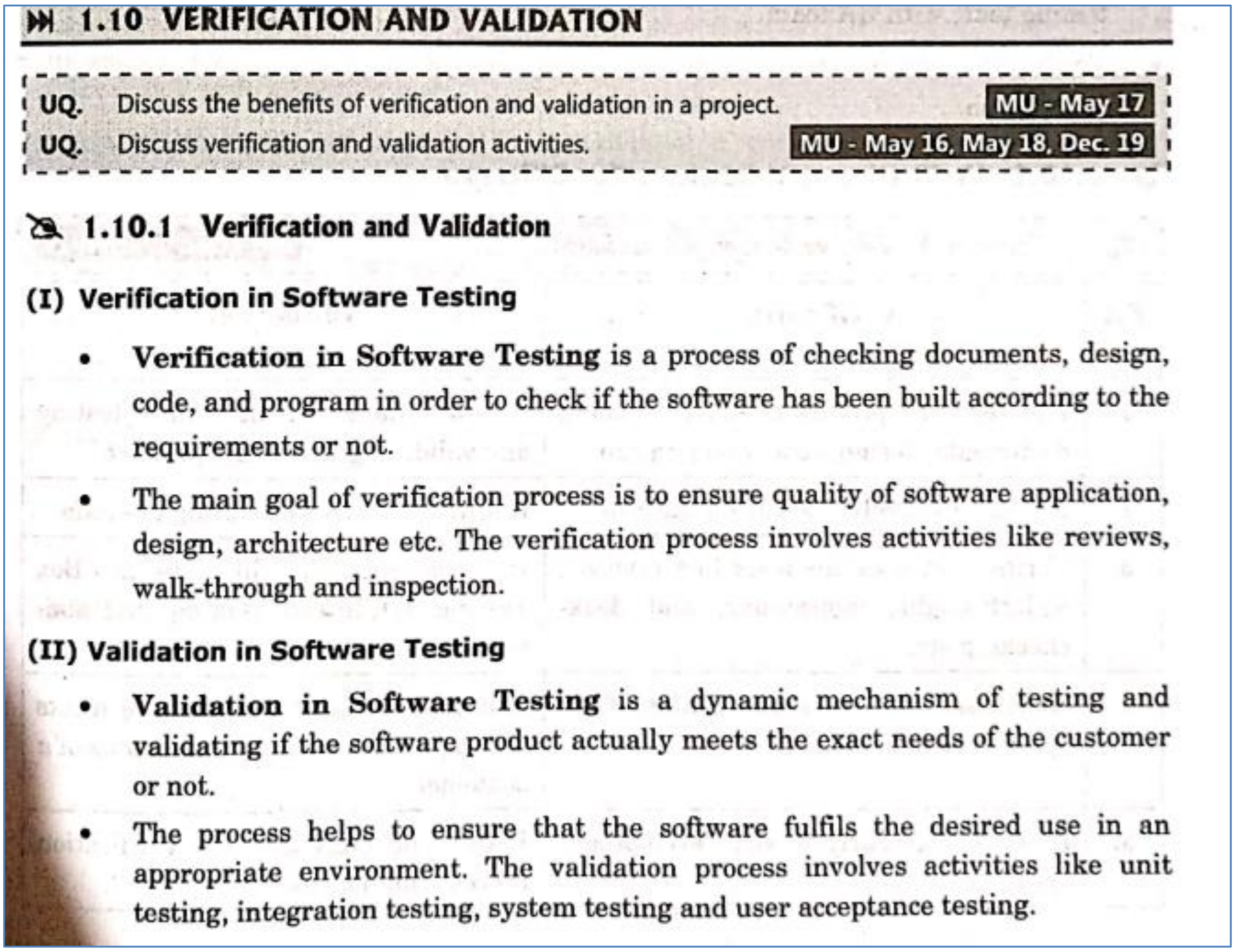
Benefits of Defect Lifecycle

- Deliver High-Quality Product
- Improve Return on Investment (ROI) by Reducing the Cost of Development
- Better Communication, Teamwork, and Connectivity
- Detect Issues Earlier and Understand Defect Trends
- Better Service and Customer Satisfaction

Limitations in Defect Lifecycle

- Variations of the Bug Life Cycle
- No Control on Test Environment

Verification and Validation:



1.10 VERIFICATION AND VALIDATION

UQ.	Discuss the benefits of verification and validation in a project.	MU - May 17
UQ.	Discuss verification and validation activities.	MU - May 16, May 18, Dec. 19

1.10.1 Verification and Validation

(I) Verification in Software Testing

- **Verification in Software Testing** is a process of checking documents, design, code, and program in order to check if the software has been built according to the requirements or not.
- The main goal of verification process is to ensure quality of software application, design, architecture etc. The verification process involves activities like reviews, walk-through and inspection.

(II) Validation in Software Testing

- **Validation in Software Testing** is a dynamic mechanism of testing and validating if the software product actually meets the exact needs of the customer or not.
- The process helps to ensure that the software fulfils the desired use in an appropriate environment. The validation process involves activities like unit testing, integration testing, system testing and user acceptance testing.

Verification	**Validation**
The verifying process includes checking documents, design, code, and program	It is a dynamic mechanism of testing and validating the actual product
It does not involve executing the code	It always involves executing the code
Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc.	It uses methods like Black Box Testing, White Box Testing, and non-functional testing
Whether the software conforms to specification is checked	It checks whether the software meets the requirements and expectations of a customer
It finds bugs early in the development cycle	It can find bugs that the verification process can not catch
Target is application and software architecture, specification, complete design, high level, and database design etc.	Target is an actual product
QA team does verification and make sure that the software is as per the requirement in the SRS document.	With the involvement of testing team validation is executed on software code.
It comes before validation	It comes after verification

🌐**Verification of high-level design, Verification of low-level design:**

1. **High Level Design :**

High Level Design in short HLD is the general system design means it refers to the overall system design. It describes the overall description/architecture of the application. It includes the description of system architecture, data base design, brief description on systems, services, platforms and relationship among modules. It is also known as macro level/system design. It is created by solution architect. It converts the Business/client requirement into High Level Solution. It is created first means before Low Level Design.

2. **Low Level Design :**

Low Level Design in short LLD is like detailing HLD means it refers to component-level design process. It describes detailed description of each and every module means it includes actual logic for every system component and it goes deep into each modules specification. It is also known as micro level/detailed design. It is created by designers and developers. It converts the High Level Solution into Detailed solution. It is created second means after High Level Design.

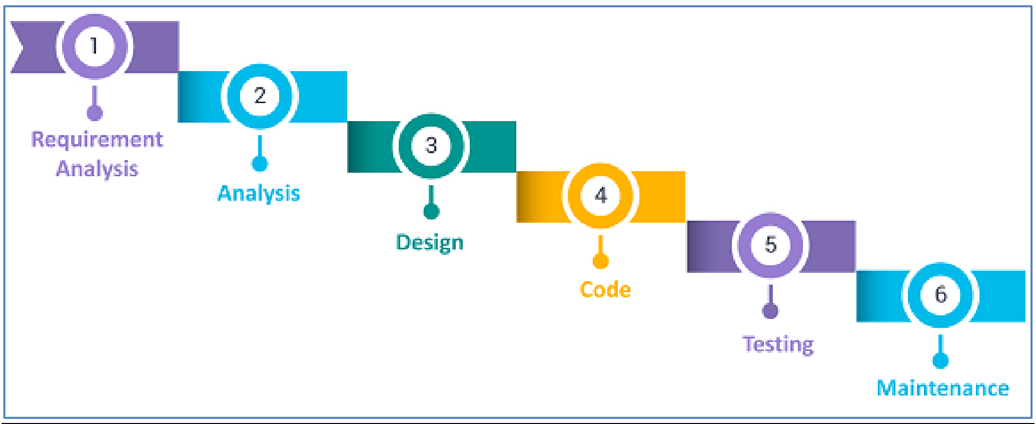
S.No.	**HIGH LEVEL DESIGN**	**LOW LEVEL DESIGN**
01.	High Level Design is the general system design means it refers to the overall system design.	Low Level Design is like detailing HLD means it refers to component-level design process.
02.	High Level Design in short called as HLD.	Low Level Design in short called as LLD.
03.	It is also known as macro level/system design.	It is also known as micro level/detailed design.
04.	It describes the overall description/architecture of the application.	It describes detailed description of each and every module.
05.	High Level Design expresses the brief functionality of each module.	Low Level Design expresses details functional logic of the module.
06.	It is created by solution architect.	It is created by designers and developers.
07.	Here in High Level Design the participants are design team, review team, and client team.	Here in Low Level Design participants are design team, Operation Teams, and Implementers.
08.	It is created first means before Low Level Design.	It is created second means after High Level Design.
09.	In HLD the input criteria is Software Requirement Specification (SRS).	In LLD the input criteria is reviewed High Level Design (HLD).
10.	High Level Solution converts the Business/client requirement into High Level Solution.	Low Level Design converts the High Level Solution into Detailed solution.
11.	In HLD the output criteria is data base design, functional design and review record.	In LLD the output criteria is program specification and unit test plan.

✅ Software Testing Models

- **Waterfall Model**
- **V Model**
- **Agile Model**
- **Spiral Model**
- **Iterative Model**

Waterfall Model

This is the most basic software development life cycle process which is followed broadly in the industry. In this model, the developers follow a sequence of processes downwards towards the ultimate goal. It is like a waterfall where there are various phases involved. the developer must complete every phase before the next phase begins. This model is named "Waterfall Model",



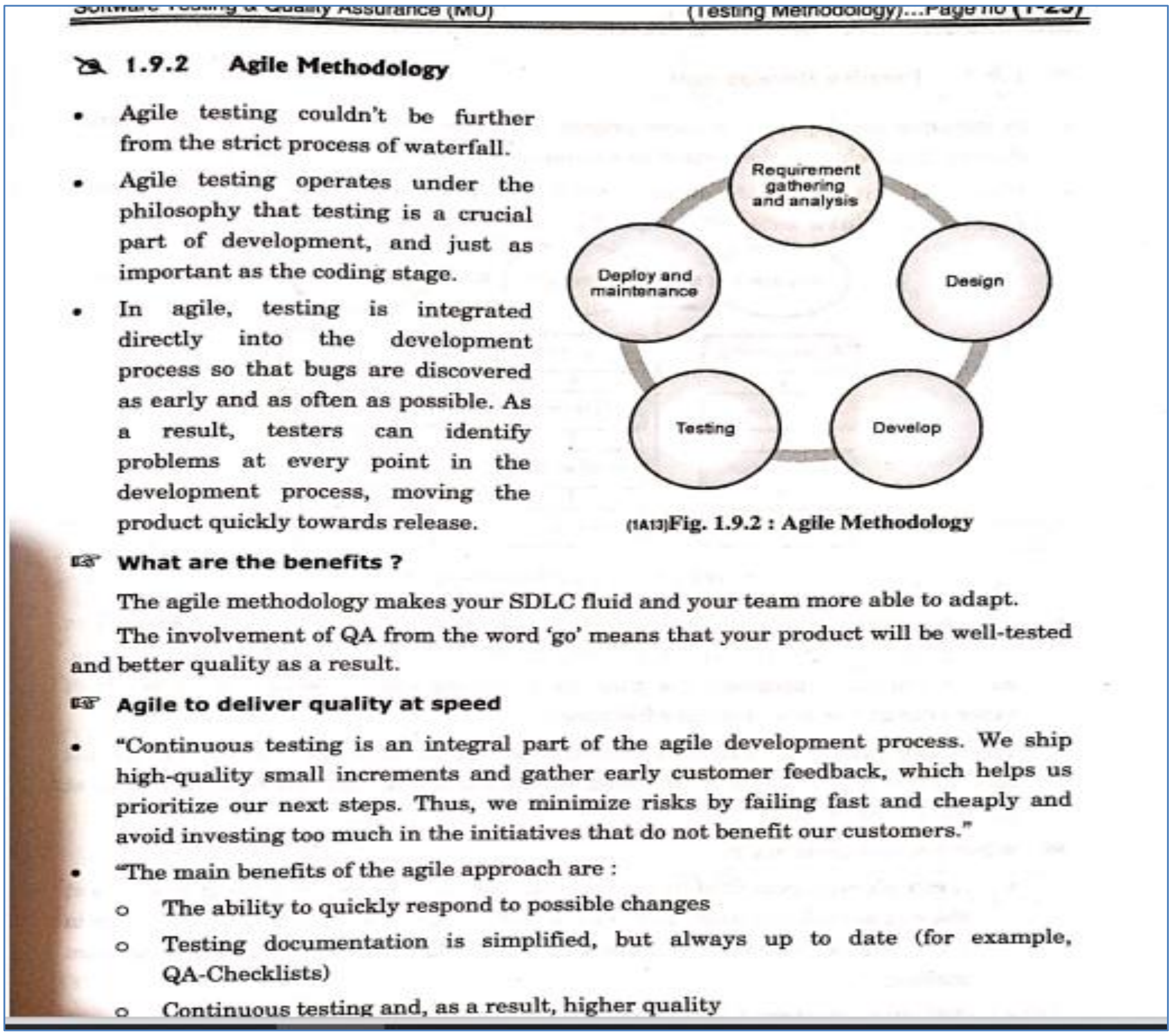
- Requirements Gathering and Analysis: The first phase involves gathering requirements from stakeholders and analyzing them to understand the scope and objectives of the project.
- Design: Once the requirements are understood, the design phase begins. This involves creating a detailed design document that outlines the software architecture, user interface, and system components.
- Implementation: The implementation phase involves coding the software based on the design specifications. This phase also includes unit testing to ensure that each component of the software is working as expected.
- Testing: In the testing phase, the software is tested as a whole to ensure that it meets the requirements and is free from defects.
- Deployment: Once the software has been tested and approved, it is deployed to the production environment.
- Maintenance: The final phase of the Waterfall Model is maintenance, which involves fixing any issues that arise after the software has been deployed and ensuring that it continues to meet the requirements over time.

Advantages

- It is easy to implement and maintain.
- The initial phase of rigorous scrutiny of requirements and systems helps in saving time later in the developmental phase.
- The requirement of resources is minimal and testing is done after each phase has been completed.

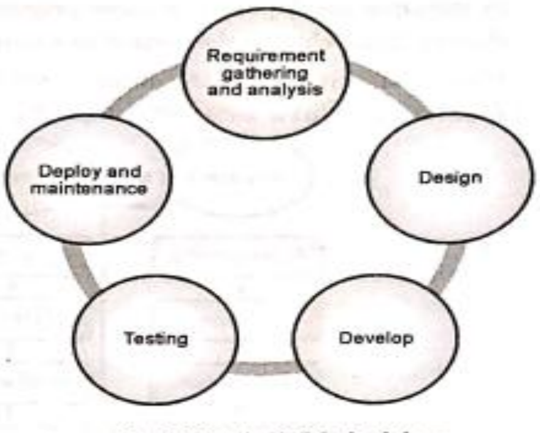
Disadvantages

- It is not possible to alter or update requirements.
- Once you move into the next phase you cannot make changes.
- You cannot start the next phase until the previous phase is completed.



1.9.2 Agile Methodology

- Agile testing couldn't be further from the strict process of waterfall.
- Agile testing operates under the philosophy that testing is a crucial part of development, and just as important as the coding stage.
- In agile, testing is integrated directly into the development process so that bugs are discovered as early and as often as possible. As a result, testers can identify problems at every point in the development process, moving the product quickly towards release.



(1A13)Fig. 1.9.2 : Agile Methodology

What are the benefits ?

The agile methodology makes your SDLC fluid and your team more able to adapt.

The involvement of QA from the word 'go' means that your product will be well-tested and better quality as a result.

Agile to deliver quality at speed

- "Continuous testing is an integral part of the agile development process. We ship high-quality small increments and gather early customer feedback, which helps us prioritize our next steps. Thus, we minimize risks by failing fast and cheaply and avoid investing too much in the initiatives that do not benefit our customers."
- "The main benefits of the agile approach are :
  - The ability to quickly respond to possible changes
  - Testing documentation is simplified, but always up to date (for example, QA-Checklists)
  - Continuous testing and, as a result, higher quality