MU

Sem 7 **INFORMATION TECHNOLOGY**
[Department Optional Course - 3] (ITDO7014)

Sem 5 **ELECTRONICS & COMPUTER SCIENCE (ECS)**
[Department Level Optional Course - I] (ECCDO5011)

New Syllabus 2022-23

Software Testing & Quality Assurance

Dr. Pankaj Agarkar | Yogesh Mali

- With Solved Latest UNIVERSITY QUESTION PAPERS.
- Multiple Choice Questions.

ISBN : 978-93-5583-029-6

TECH-NEO PUBLICATIONS
Where Authors Inspire Innovation
A Sachin Shah Venture

www.techneobooks.in
info@techneobooks.in

M7-44B

Price ₹ 395/-

Index

Module 1

➡ Chapter 1 : Testing Methodology .....1-1 to 1-37

Module 2

➡ Chapter 2 : Testing Techniques.....2-1 to 2-72

Module 3

➡ Chapter 3 : Managing the Test Process .....3-1 to 3-48

Module 4

➡ Chapter 4 : Test Automation .....4-1 to 4-28

Module 5

➡ Chapter 5 : Testing for Specialized Environment.....5-1 to 5-26

Module 6

➡ Chapter 6 : Quality Management.....6-1 to 6-33

◈ **Multiple Choice Questions (MCQ's)**

❑❑❑

Module 1

CHAPTER 1

Testing Methodology

University Prescribed Syllabus

1.1 Introduction to Software Testing: Introduction, Goals of Software Testing, Software Testing Definitions, Model for Software Testing, Effective Software Testing vs Exhaustive Software Testing, Software Failure Case Studies

1.2 Software Testing Terminology and Methodology: Software Testing Terminology, Software Testing Life Cycle (STLC), Software Testing methodology

1.3 Verification and Validation : Verification, Verification requirements, Validation.

1.1 Introduction 1-3

1.2 Goals of Software Testing 1-3

1.2.1 Short-term or Immediate Goals 1-3

1.2.2 Long-term Goals 1-4

1.2.3 Post-implementation Goals 1-5

1.2.4 Software Testing Definitions 1-6

UQ. What is software testing ? (MU - May 17, May 19) 1-6

1.2.5 Model for Software Testing 1-7

UQ. What is software testing ? Describe software testing model with a neat diagram? (MU - May 17, May 19) 1-7

1.2.5.1 Software Testing Summarization 1-7

1.2.5.2 Software Testing Overview 1-8

1.3 Effective Software Testing vs. Exhaustive Software Testing 1-9

UQ. Explain effective and exhaustive software testing. (MU - Dec. 16, Dec. 18) 1-9

1.3.1 Difference between Effective Testing and Exhausting Testing 1-10

1.3.2 Domain of Input Data 1-11

1.4 Software Failure Case Studies 1-13

1.4.1 Top 8 Mega Software Failures 1-13

1.5 Software Testing Terminology 1-15
1.5.1 The Most Common Software Testing Terminology 1-15
1.6 Software Testing Life Cycle (STLC) 1-19
UQ. Explain STLC in details. (MU - May 16, Dec. 17, May 18, Dec. 18) 1-19
1.6.1 Requirement Phase Testing 1-20
1.6.2 Test Planning in STLC 1-21
1.6.3 Test Case Development Phase 1-21
1.6.4 Test Environment Setup 1-22
1.6.5 Test Execution Phase 1-22
1.6.6 Test Cycle Closure 1-23
1.7 Classification of Software Bugs 1-23
UQ. Classify different types of bugs based on Software development lifecycle. (MU - May 17, Dec. 19) 1-23
UQ. Classify bugs based in SDLC ? (MU - May 19) 1-23
1.8 Defect/Bug Life Cycle in Software Testing 1-25
UQ. What is Life cycle of Bug? Explain Life Cycle of Bug and Defect states of bug ? (MU - Dec. 17, Dec. 18, Dec. 19) 1-25
1.8.1 What is Defect Life Cycle ? 1-25
1.8.2 Defect Life Cycle Testing Process 1-26
1.9 Software Testing Methodology 1-27
1.9.1 Waterfall Method 1-28
1.9.2 Agile Methodology 1-29
1.9.3 Iterative Development 1-30
1.10 Verification and Validation 1-31
UQ. Discuss the benefits of verification and validation in a project. (MU - May 17) 1-31
UQ. Discuss verification and validation activities. (MU - May 16, May 18, Dec. 19) 1-31
1.10.1 Verification and Validation 1-31
1.10.2 Verification vs Validation : Key Difference 1-32
UQ. Differentiate between verification and validation. (MU - Dec. 17, Dec. 19) 1-32
1.10.3 Verification Requirements 1-33
1.10.3.1 Verification Requirement Properties 1-33
1.11 Verification of High Level Design and Low Level Design Validation 1-36
UQ. Explain Verification in high level and low level design. (MU - May 18) 1-36
1.11.1 Difference between High Level Design and Low Level Design 1-36
▶ **Chapter Ends** 1-37

1.1 INTRODUCTION

Introduction to Testing Methodologies

GQ. What are Software Testing Methodologies ?

- Testing methodologies in software engineering are testing strategies, approaches or methods used to test a specific product to ensure its usability.
- It makes sure that product works as per the given specifications and has no side effects when used outside the design parameters.
- Software testing methodologies encompass everything from unit testing to integration testing and specialized form of testing like security or performance testing.

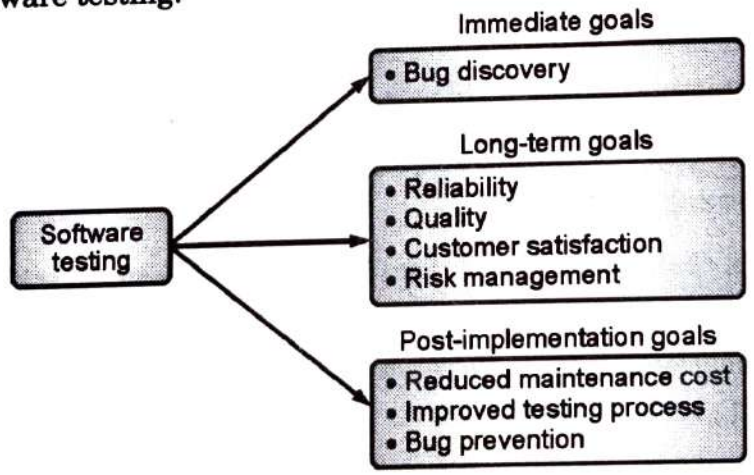
1.2 GOALS OF SOFTWARE TESTING

The goals of software testing may be classified into three major categories, as shown in Fig. 1.2.1.

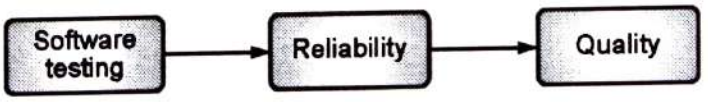
1.2.1 Short-term or Immediate Goals

These goals are immediate results after performing testing. These goals may be set in the individual phases of SDLC.

Bug discovery : The immediate goal of testing is to find errors at any stage of software development. More the bugs discovered at an early stage, better will be the success rate of software testing.



(1A1)**Fig. 1.2.1 : Software Testing Goals**



(1A2)**Fig. 1.2.2 : Testing produces reliability and quality**

1.2.2 Long-term Goals

These goals affect the product quality in the long run when one cycle of the SDLC is complete. Some of them are discussed here.

(I) Quality

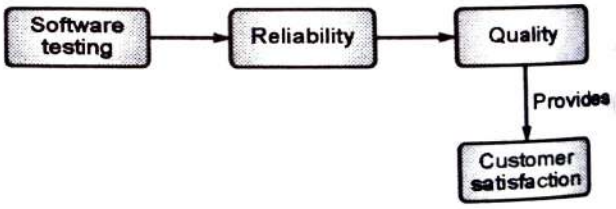
- Since the software is also a product, its quality is primary from the users' point of view. Thorough testing ensures superior quality. Therefore, the first goal of understanding and performing the testing process is to enhance the quality of the software product.
- Though quality depends on various factors, such as correctness, integrity, efficiency, etc., reliability is the major factor to achieve quality. The software should be passed through a rigorous reliability analysis to attain high-quality standards.
- Reliability is a matter of confidence that the software will not fail, and this level of confidence increases with rigorous testing. The confidence in the reliability, in turn, increases the quality, as shown in Fig. 1.2.2.

(II) Customer satisfaction

- From the user's perspective, the prime concern of testing is customer satisfaction only. If we want the customer to be satisfied with the software product, then testing should be complete and thorough.
- Testing should be complete in the sense that it must satisfy the user for all the specified requirements mentioned in the user manual, as well as for the unspecified requirements, which are otherwise understood.
- A complete testing process achieves reliability, which enhances the quality, and quality, in turn, increases customer satisfaction, as shown in Fig. 1.2.3.

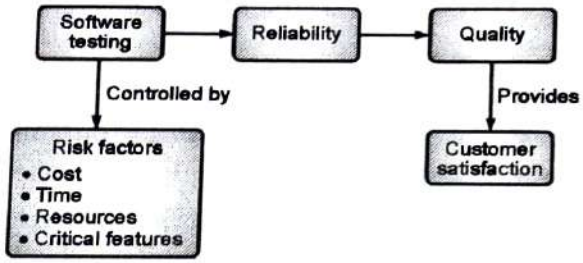
(III) Risk management

- Risk is the probability that undesirable events will occur in a system.
- These undesirable events will prevent the organization from successfully implementing its business initiatives.
- Thus, the risk is basically concerned with the business perspective of an organization.



(1A3)**Fig. 1.2.3 : Quality leads to customer satisfaction**

- Risks must be controlled to manage them with ease. Software testing may act as a control, which can help in eliminating or minimizing risks (Refer Fig. 1.2.4).
- Thus, managers depend on software testing to assist them in controlling their business goals.



(1A4)**Fig. 1.2.4 : Testing controlled by risk factors**

1.2.3 Post-implementation Goals

These goals are important after the product is released. Some of them are discussed here.

(I) Reduced Maintenance Cost

- The maintenance cost' of any software product is not its physical cost, as the software does not wear out.
- The only maintenance cost in a software product is its failure due to errors. Post-release errors are costlier to fix, as they are difficult to detect.
- Thus, if testing has been done rigorously and effectively, then the chances of failure are minimized and, in turn, the maintenance cost is reduced.

(II) Improved software testing process

- A testing process for one project may not be successful and there may be scope for improvement.
- Therefore, the bug history and post-implementation results can be analyzed to find out snags in the present testing process, which can be rectified in future projects. Thus, the long-term post-implementation goal is to improve the testing process for future projects.

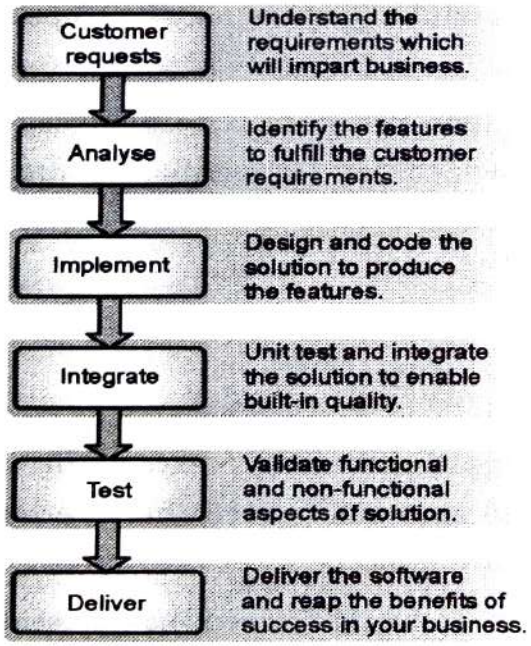
(III) Bug prevention

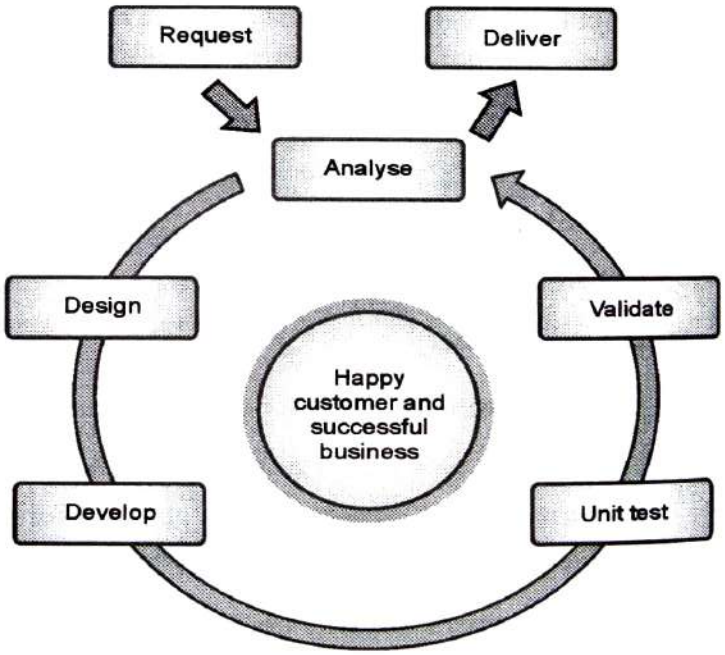
- It is the consequent action of bug discovery.
- From the behavior and interpretation of bugs discovered, everyone in the software development team gets to learn how to code safely such that the bugs discovered are not repeated in later stages or future projects.
- Though errors cannot be prevented to zero, they can be minimized. In this sense, bug prevention is a superior goal of testing.

1.2.4 Software Testing Definitions

UQ. What is software testing ? **MU - May 17, May 19**

- Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client.
- Software testing is carried out in a systematic manner with the intent of finding defects in a system. It is required for evaluating the system. As the technology is advancing we see that everything is getting digitized.
- You can access your bank online, you can shop from the comfort of your home, and the options are endless. Have you ever wondered what would happen if these systems turn out to be defective?
- One small defect can cause a lot of financial loss. It is for this reason that software testing is now emerging as a very powerful field in IT.
- Although like other products software never suffers from any kind of wear or tear or corrosion but yes, design errors can definitely make your life difficult if they go undetected.
- Regular testing ensures that the software is developed as per the requirement of the client. However, if the software is shipped with bugs embedded in it, you never know when they can create a problem and then it will be very difficult to rectify defect because scanning hundreds and thousands of lines of code and fixing a bug is not an easy task.
- You never know that while fixing one bug you may introduce another bug unknowingly in the system.





(1A5)**Fig. 1.2.5 : Software Testing Process**

(3) Planning

- All testing activities require planning. It is important to outline a test plan that will give in details about how each activity will be carried out.
- Test plan is also required to ensure that all aspects of the software are covered thoroughly and there is no repetition of testing process so that time and effort is not wasted.
- The latest trend now is to involve the testing team in specification writing process. It is important that the testing team understands the requirements of the client clearly as the entire development is based on the requirement defined by the client.
- Anything that is not in line with the requirement is a defect. So, the testing team should have a clear idea about what the final outcome of running software should be like. As a matter of fact it is important to start writing test cases in parallel to specification writing.
- This will help the testers analyze whether all the requirements are testable or not. When you write test cases in parallel to specification writing process you will think critically about the specifications and you will know if there is an issue with the requirement or if there is something that cannot be developed.

1.3 EFFECTIVE SOFTWARE TESTING VS. EXHAUSTIVE SOFTWARE TESTING

UQ. Explain effective and exhaustive software testing. **MU - Dec. 16, Dec. 18**

1] **Exhaustive or complete software testing** means that every statement in the program and every possible path combination with every possible combination of data must be executed.

- However, soon, we will realize that exhaustive testing is out of scope. **That is why the questions arise :**
  - (i) When are we done with testing? or
  - (ii) How do we know that we have tested enough?
- There may be many answers to these questions with respect to time, cost, customer, quality, etc. This section will explore why exhaustive or complete testing is not possible.

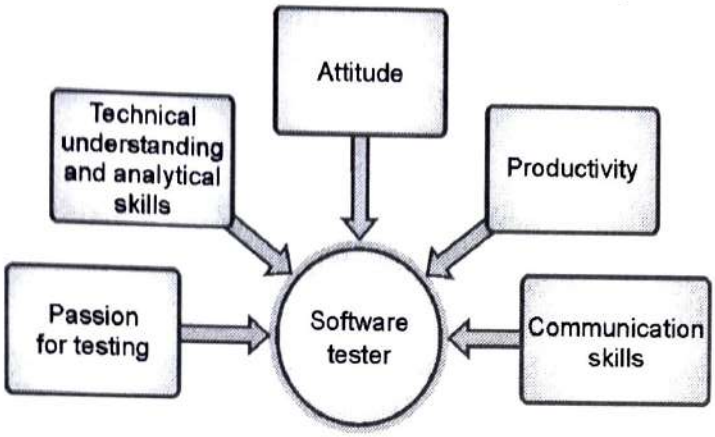
2] **We should concentrate on effective testing** that emphasizes efficient techniques to test the software so that important features will be tested within the constrained resources.

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.

1.2.5.1 Software Testing Summarization

- Software testing is required to check the reliability of the software. Software testing ensures that the system is free from any bug that can cause any kind of failure.
- Software testing ensures that the product is in line with the requirement of the client. It is required to make sure that the final product is user friendly.
- At the end software is developed by a team of human developers all having different viewpoints and approach. Even the smartest person has the tendency to make an

error. It is not possible to create software with zero defects without incorporating software testing in the development cycle.

- No matter how well the software design looks on paper, once the development starts and you start testing the product you will definitely find lots of defects in the design. You cannot achieve software quality without software testing.
- Even if testers are not involved in actual coding they should work closely with developers to improve the quality of the code. For best results it is important that software testing and coding should go hand in hand.

1.2.5.2 Software Testing Overview

(1) Defects

- Defects arise in software due to many reasons. As a matter of fact it is said that every software application has some defects embedded in it but not every defect is a threat to the system.
- There is a lot that can be accomplished with the help of software testing. Testing helps in evaluating the quality of software.
- There are many reasons why software testing has gained so much of importance in the field of information technology.
- Firstly, testing helps in reducing the overall cost of the software development project. If testing is ignored in the initial development stages to save a small amount of money then it may turn out to be a very expensive matter later because as you move on with development process it becomes more and more difficult to trace back defects and rectifying one defect somewhere can introduce another defect in some other module.

(2) Requirement

- When software is delivered to the client there are no arguments about the variation from the original requirements. Software testing helps in strengthening the market reputation of a company. Well tested software is of good quality and good quality means better feedback and reviews.
- In order to achieve best results it is important to organize all your testing efforts and this is what this Software Testing Training provided by International Software Test Institute is all about. Software testing cannot be fruitful without proper planning.
- To live up to the expectations of the client it is important to plan every step carefully. A lot of things need to be considered in order plan your testing efforts.
- Software testing should be planned keeping budget, schedule and performance in mind in order to achieve best results.

(3) Planning

- All testing activities require planning. It is important to outline a test plan that will give in details about how each activity will be carried out.
- Test plan is also required to ensure that all aspects of the software are covered thoroughly and there is no repetition of testing process so that time and effort is not wasted.
- The latest trend now is to involve the testing team in specification writing process. It is important that the testing team understands the requirements of the client clearly as the entire development is based on the requirement defined by the client.
- Anything that is not in line with the requirement is a defect. So, the testing team should have a clear idea about what the final outcome of running software should be like. As a matter of fact it is important to start writing test cases in parallel to specification writing.
- This will help the testers analyze whether all the requirements are testable or not. When you write test cases in parallel to specification writing process you will think critically about the specifications and you will know if there is an issue with the requirement or if there is something that cannot be developed.

1.3 EFFECTIVE SOFTWARE TESTING VS. EXHAUSTIVE SOFTWARE TESTING

UQ. Explain effective and exhaustive software testing. **MU - Dec. 16, Dec. 18**

1] **Exhaustive or complete software testing** means that every statement in the program and every possible path combination with every possible combination of data must be executed.

- However, soon, we will realize that exhaustive testing is out of scope. **That is why the questions arise :**

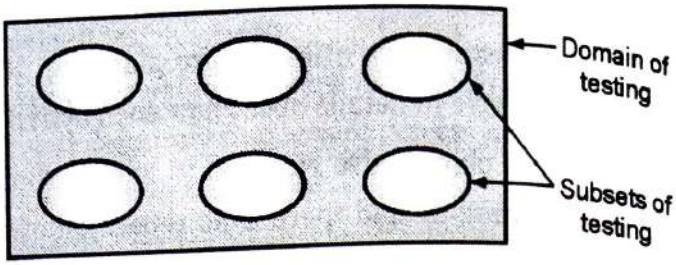
  (i) When are we done with testing? or

  (ii) How do we know that we have tested enough?

- There may be many answers to these questions with respect to time, cost, customer, quality, etc. This section will explore why exhaustive or complete testing is not possible.

2] **We should concentrate on effective testing** that emphasizes efficient techniques to test the software so that important features will be tested within the constrained resources.

- The testing process should be understood as a domain of possible tests (see Fig. 1.3.1) there are subsets of these possible tests. However, the domain of possible tests becomes infinite, as we cannot test every possible combination.



(1A7)**Fig. 1.3.1 : Testing domain**

- This combination of possible tests is infinite, that is, the processing resources and time are not sufficient for performing tests.
- Computer speed and time constraints limit the possibility of performing all the tests. **Complete testing** requires the organization to invest a long time, which is not cost-effective. Therefore, testing must be performed on selected subsets that can be performed within the constrained resources.
- This selected group of subsets (not the whole domain of testing) makes software testing effective.
- **Effective testing** can be enhanced if subsets are selected based on the factors that are required in a particular environment.

1.3.1 Difference between Effective Testing and Exhausting Testing

Sr. No.	Effective testing	Exhaustive testing
1.	Effective testing emphasizes efficient techniques to test the S/Ws/w so that important features will be tested within the constrained resources.	Exhaustive or complete testing means that energy statement in the program of every possible path combination with every possible combination of data must be executed.
2.	It is a practical method	It is not possible to perform complete testing
3.	It is feasible because: a) It checks for s/w reliability and no Bugs in the final product b) It tests in each phase c) It uses constrained resources	It is not feasible because: a) Achieving deadlines b) Various possible outputs c) Timing constraints d) No. of possible test environments.
4.	It is cost effective	It is not cost effective
5.	It is less complex and less time consuming	It is complex and time consuming
6.	It is adopted such that critical test cases are concerned first	It corners all the test cases

Domain of Possible Inputs to the Software is too Large to Test

Even if we consider the input data as the only part of the domain of testing, we are not able to test the complete input data combination.

1.3.2 Domain of Input Data

The domain of input data has four sub-parts :

1. Valid inputs	2. Invalid inputs
3. Edited inputs	4. Race condition inputs

(Refer Fig. 1.3.2)

1. Valid inputs

- It seems that we can test every valid input on the software. Let us look at a very simple example of adding two-digit two numbers.
- Their range is from – 99 to 99 (total 199). So the total number of test case combinations will be

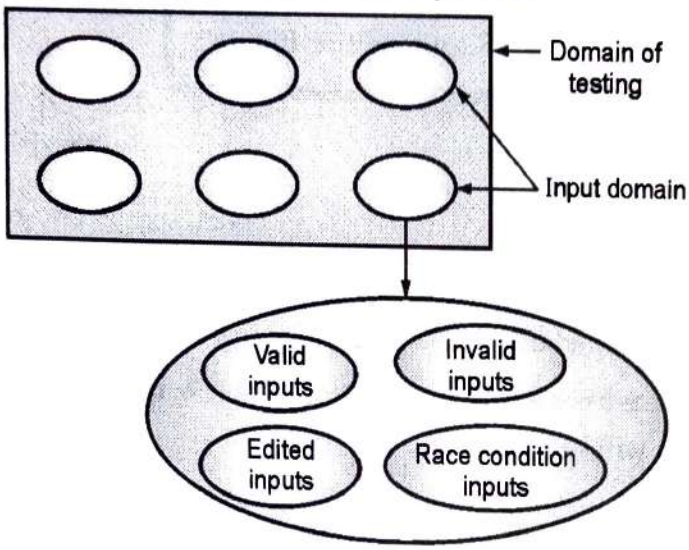
  $$199 \times 199 = 39601.199 \times 199 = 39601$$

- Further, if we increase the range from two digits to four-digits, then the number of test cases will be 399,960,001. Most addition programs accept 8 or 10 digit numbers or more. How can we test all these combinations of valid inputs?
- When we test software with valid data, it is known as positive testing. Positive testing is always performed keeping in view the valid range or limits of the test data in test cases.

2. Invalid inputs

- Testing the software with valid inputs is only one part of the input sub-domain.
- There is another part, invalid inputs, which must be tested for testing the software effectively.
- When we test software with invalid data, it is known as negative testing.
- Negative testing is always performed keeping in view that the software must work properly when it is passed through an invalid set of data.
- Thus, negative testing basically tries to break the software. The important thing, in this case, is the behavior of the program as to how it responds when a user feeds invalid inputs.
- A set of invalid inputs is also too large to test.

- If we consider again the example of adding two numbers, then the following possibilities may occur:
  (i) Numbers out of range
  (ii) Combination of alphabets and digits
  (iii) Combination of all alphabets
  (iv) Combination of control characters
  (v) Combination of any other key on the keyboard.



(1A8)**Fig. 1.3.2 : Input domain for testing**

3. Edited inputs

- If we can edit inputs at the time of providing them to the program, then many unexpected input events may occur. For example, you can add many spaces in the input, which are not visible to the user.
- It can be a reason for the non-functioning of the program. In another example, it may be possible that a user is pressing a number key, then Backspace key continuously and finally after some time, presses another number key and Enter. Its input buffer overflows and the system crashes.
- The behavior of users cannot be judged. They can behave in a number of ways, causing a defect in testing a program. That is why edited inputs are also not tested completely.

4. Race condition inputs

- The timing variation between two or more inputs is also one of the issues that limit the testing. For example, there are two input events, A and B.
- According to the design, A precedes B in most of the cases. However, B can also come first in rare and restricted conditions. There is the race condition, whenever B proceeds A.

- 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.

(8) Sony Pictures Entertainment

- The computer network infiltration of one of the major Hollywood movie studio, Sony Pictures entertainment where hundreds of confidential documents were stolen and uploaded on file sharing sites highlights the need of more investment in the network security and thoroughly testing the security aspects.
- This incident could cost Sony millions of dollars and some embarrassment if some hidden data is revealed.
- Businesses spend an average of €514,000 per IT failure, but 50% of these incidents occurred because of software coding errors or failed IT changes and are "avoidable", according to KPMG research.
- At Cigniti, we assist organizations to achieve defect free software applications, accelerate the software life cycle while assuring quality. Global businesses rely on Cigniti's proprietary IP (ETAS) and colocated software testing expertise to avoid software failures.

1.5 SOFTWARE TESTING TERMINOLOGY

1.5.1 The Most Common Software Testing Terminology

There is so much information out there about software testing that it can be hard to know where to start.

First time testers need to learn the terminology as fast as possible but there are so many different sources of information.

We have attempted to take the top software testing terminology, alphabetize it, and define it in simple terms. While a doozy of info, hopefully this can be a guide while learning the ropes of the trade.

Sr. No.	Parameter	Description
(1)	**Acceptance criteria**	The predefined and described set of requirements or conditions that must be met in order for the feature to be released to the public.
(2)	**Ad-hoc testing**	Informal testing that does not have any documentation, tickets or planning.
(3)	**Agile**	Not a person who moves quickly, but rather software development that focuses on tasks at hand to be broken down into short phases of work with frequent reappraisals of the work and adaption of the work.

Sr. No.	Parameter	Description
(4)	**Automated testing**	A testing technique that uses an automation testing tool to write test scripts and automate any of the repetitive tasks, comparing the actual outcome with the expected outcome.
(5)	**Black-box testing**	A testing technique that is conducted by a tester not knowing the structure or design of the product whatsoever.
(6)	**Blocker**	This is a bug that is deemed absolutely critical to fix before release of the product. If it is not fixed, it can possibly ruin the entire product or feature launch.
(7)	**Bugs**	Bugs are problems, issues, or defects that operate in the program code. These can be minor or major errors, but regardless they need to be fixed before the release.
(8)	**Bug reports**	This is a formal way to document bugs, but each testing company will have their own version. They will need these reports to be able to give details about the issue to the developers.
(9)	**Component testing**	Component testing is a testing technique that focuses separately on small elements of a system.
(10)	**Configuration testing**	Another testing technique that ensures that the system or product can operate under various software and hardware conditions, such as a web application being able to properly work in different browsers.
(11)	**Defects**	Defects are issues that do not meet the acceptance criteria. It might not be a bug, but rather a design or content issue that does not match the requirements set forth by the client.
(12)	**Expected outcome**	This is what the system feature or product is supposed to be doing. The observed or actual outcome is then compared against the expected outcome to show any deviations from the acceptable criteria.
(13)	**Exploratory testing**	Exploratory testing means that there is not a test written, rather a tester checks the system based on their knowledge of the system and will execute tests based on that.

Sr. No.	Parameter	Description
(14)	**Fail/Failure/Failed**	The feature or component did not meet the expected outcome.
(15)	**Feature**	Changes made to a system or product in order to add new functionalities or modify already existing ones.
(16)	**Life-cycle testing**	Life-cycle testing is a range of activities that are implemented during the testing process to ensure that the software quality goals are achieved.
(17)	**Load testing**	Load testing is a type of testing that measures the performance of the system under a certain load.
(18)	**Manual testing**	Manual testing is the testing process of manually testing software for bugs and defects where a tester is the 'end-user'.
(19)	**Mobile-Device testing**	Mobile-Device testing is software testing on mobile devices to ensure that works on both the hardware and software.
(20)	**Negative testing**	Negative testing is the method of testing where invalid information is inputted into the system in order to see if it can handle incorrect or unwanted results.
(21)	**Observed outcome**	Observed outcome is what the tester encounters within the system or product. It is compared to the expected outcome to see if it matches or deviates away from it.
(22)	**Pass/Passed**	The feature or component meets the expected outcome.
(23)	**Positive testing**	Positive testing is a testing technique that shows that the system/product in a test does what it is supposed to do.
(24)	**Priority**	The level that is assigned to the ticket. Each company will have their own rating system but usually is along the lines of critical being the highest priority and low being the lowest priority.
(25)	**Performance testing**	Performance testing is the type of testing conducted to determine the systems strength under a specific workload. It checks the responsiveness and security of the system.

Sr. No.	Parameter	Description
(26)	**Quality**	Quality measures the design of the software, how well the actual application conforms to that software, and how the software executes its purpose.
(27)	**Quality Assurance**	Quality Assurance is the methodical monitoring and evaluation of the software system to ensure that the minimum requirements and standards are being met.
(28)	**Regression testing**	A full system testing usually conducted before the new release of a system in order to find any new bugs or defects or see if previous ones were fixed.
(29)	**Release**	Release is the new version of the software system that can either be for the testers or for the clients.
(30)	**Requirements**	Requirements is all of the evidence that contains information about the feature. This allows for software developers to build and the testers to be able to test the right functionalities.
(31)	**Smoke testing**	Smoke testing is a quick testing form that allows testers to quickly check major features and functions either right before or after a release.
(32)	**Sprint s**	**Sprint s** is the amount of time that is given for the QA process, usually has to do with the number of tasks that a team needs to complete.
(33)	**Stress testing**	Stress testing is testing that is showing how the system reacts to certain workload situations that will exceed the systems specified requirements. It shows where the system will fail in its resources.
(34)	**Test case**	Test cases are a set of structured steps or script that tell the tester exactly how the feature of function of the system should work. It should usually contain expected results and the conditions associated with these.
(35)	**Test environment**	Test environment is the technical environment in which the software tester will be conducting and running the tests.
(36)	**Test suites**	Test suites are the test cases that are compiled together for the system testing.

Sr. No.	Parameter	Description
(37)	**Users**	Users are the people who will end up using the software.
(38)	**User acceptance testing**	User acceptance testing also known as UAT, is one of the last phases of testing. It is the process of ensuring that the software works for the user in the real world.
(39)	**User experience**	User experience comes from the experience of the person that is using the software or product and focuses on various aspects, but especially how the usability, overall design, and functionality.
(40)	**White-box testing**	White-box testing occurs when the tester has previous knowledge of the system that is being tested and is familiar with the structure. The opposite of this is black-box testing.

1.6 SOFTWARE TESTING LIFE CYCLE (STLC)

UQ. Explain STLC in details. **MU - May 16, Dec. 17, May 18, Dec. 18**

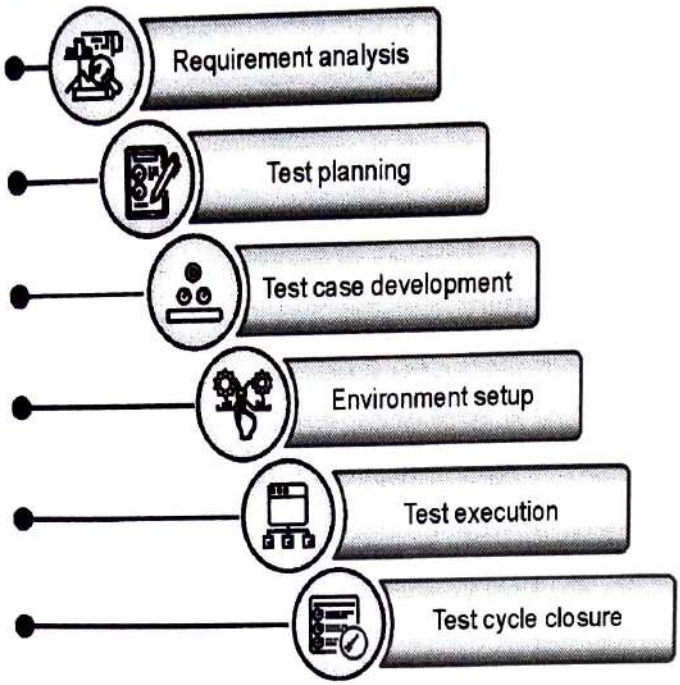
Note : (Refer Section 1.6.1 to 1.6.6)

- **Software Testing Life Cycle (STLC)** is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities.
- Contrary to popular belief, Software Testing is not just a single/isolate activity, i.e. testing. It consists of a series of activities carried out methodologically to help certify your software product.

STLC Phases

There are following six major phases in every Software Testing Life Cycle Model (STLC Model) :

1 Requirement Analysis	2. Test Planning
3. Test case development	4. Test Environment setup
5. Test Execution	6. Test Cycle closure



(1A9)**Fig. 1.6.1 : STLC Model Phases**

Each of these stages has a definite Entry and Exit criteria, Activities and Deliverables associated with it.

☞ What is Entry and Exit Criteria in STLC ?

- **Entry Criteria :** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- **Exit Criteria :** Exit Criteria defines the items that must be completed before testing can be concluded

You have Entry and Exit Criteria for all levels in the Software Testing Life Cycle (STLC).

In an Ideal world, you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. So, for this tutorial, we will focus on activities and deliverables for the different stages in STLC life cycle. Let's look into them in detail.

1.6.1 Requirement Phase Testing

Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail. Requirements could be either functional or non-functional. Automation feasibility for the testing project is also done in this stage.

(I) Activities in Requirement Phase Testing

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

(II) Deliverables of Requirement Phase Testing

- RTM
- Automation feasibility report. (if applicable)

1.6.2 Test Planning in STLC

Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.

(I) Test Planning Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

(II) Deliverables of Test Planning

- Test plan /strategy document.
- Effort estimation document.

1.6.3 Test Case Development Phase

The **Test Case Development Phase** involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units.

(I) Test Case Development Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

(II) Deliverables of Test Case Development

- Test cases/scripts
- Test data

1.6.4 Test Environment Setup

Test Environment Setup decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. Test team may not be involved in this activity if the development team provides the test environment. The test team is required to do a readiness check (smoke testing) of the given environment.

(I) Test Environment Setup Activities

(1) Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.

(2) Setup test Environment and test data

(3) Perform smoke test on the build

(II) Deliverables of Test Environment Setup

(1) Environment ready with test data set up

(2) Smoke Test Results.

1.6.5 Test Execution Phase

Test Execution Phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.

(I) Test Execution Activities

(1) Execute tests as per plan

(2) Document test results, and log defects for failed cases

(3) Map defects to test cases in RTM

(4) Retest the Defect fixes

(5) Track the defects to closure

(II) Deliverables of Test Execution

(1) Completed RTM with the execution status

(2) Test cases updated with results

(3) Defect reports

1.6.6 Test Cycle Closure

- **Test Cycle Closure** phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results.
- Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

(I) Test Cycle Closure Activities

(1) Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality.

(2) Prepare test metrics based on the above parameters.

(3) Document the learning out of the project.

(4) Prepare Test closure report.

(5) Qualitative and quantitative reporting of quality of the work product to the customer.

(6) Test result analysis to find out the defect distribution by type and severity.

(II) Deliverables of Test Cycle Closure

(1) Test Closure report

(2) Test metrics

(3) STLC Phases along with Entry and Exit Criteria

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".

1.8 DEFECT/BUG LIFE CYCLE IN SOFTWARE TESTING

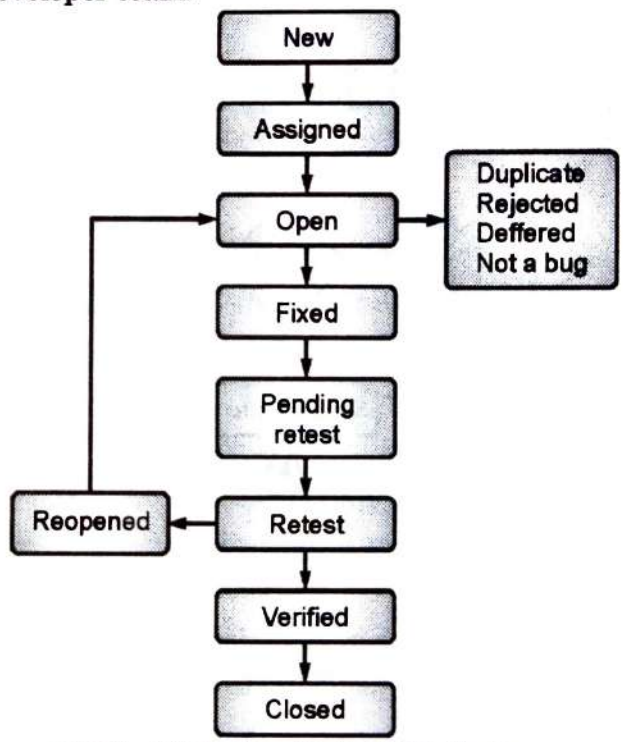
UQ. What is Life cycle of Bug? Explain Life Cycle of Bug and Defect states of bug ?
MU - Dec. 17, Dec. 18, Dec. 19

1.8.1 What is Defect Life Cycle ?

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

Defect Status

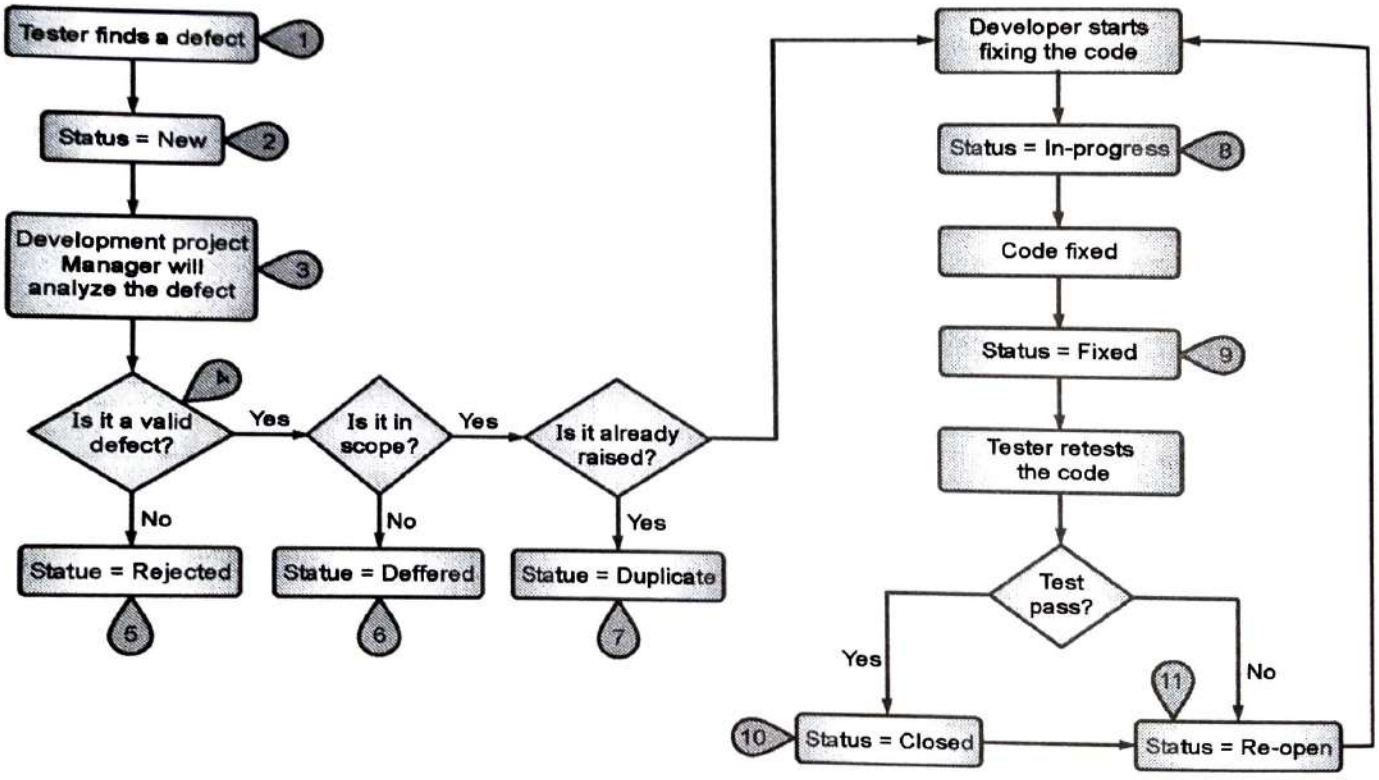
- **Defect Status** or Bug Status in defect life cycle is the present state from which the defect or a bug is currently undergoing. The goal of defect status is to precisely convey the current state or progress of a defect or bug in order to better track and understand the actual progress of the defect life cycle.
- The number of states that a defect goes through varies from project to project. Fig. 1.8.1 shows 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."



(1A10)**Fig. 1.8.1 : Bug/Defect Life Cycle**

- **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".

1.8.2 Defect Life Cycle Testing Process



(1A11)**Fig. 1.8.2 : Defect Life Cycle**

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 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.**

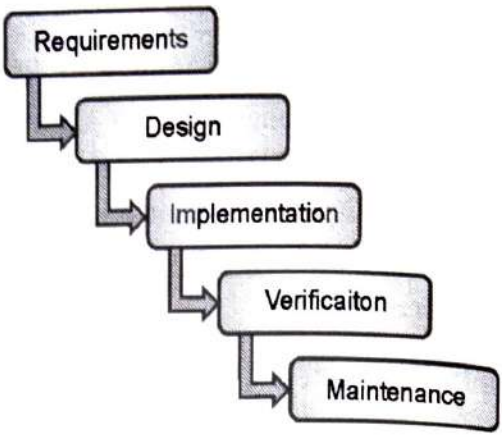
1.9 SOFTWARE TESTING METHODOLOGY

Three Software Testing Methodologies to Consider

- In today's tech landscape, staying ahead of your competition and delivering quality consistently are the two differentiators that make an app-first company succeed.
- So it's no surprise that teams are always on the lookout for the best possible testing approaches that will improve their QA strategy and influence the quality of their product. We all want to keep our strategies fresh and see the quality of our software get better and better as a result.
- Software testing methodologies are the processes used to ensure you deliver a well-tested product at speed and keep up with lightning-quick SDLCs.
- But how do you choose which software testing methodology is right for you? What does each one entail, and what are the benefits? Read on to find out.

1.9.1 Waterfall Method

This software development model is sequential. The next step only begins after the previous step is completed. The process might look a little something like in Fig. 1.9.1.



(1A12)**Fig. 1.9.1 : Waterfall Model**

(1) **Requirements:** This is all about gathering the requirements for the product or update. What are the key functions? How should it behave ?

(2) **Design :** Decide how the code will be written. Focus on the design of the product

(3) **Implementation :** Build the product. Write the code.

(4) **Verification :** Test, Test Test.

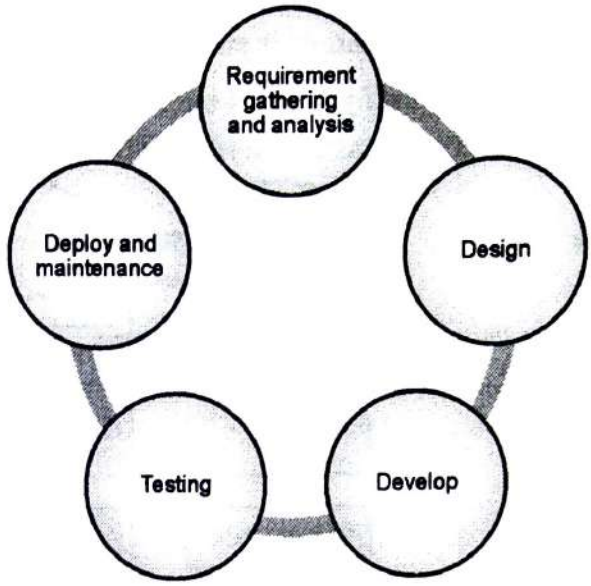
(5) **Maintenance :** Release the product into the wild. Then be ready for any bug fixes, updates or changes.

What are the benefits ?

- In reality, this method doesn't allow a lot of wiggle room.
- For that reason, it is extremely useful for small projects where requirements are clearly defined, For anything bigger, like a full product launch, the waterfall methodology can actually be very restrictive.
- If you want to release a simple update, with a clear set of instructions behind it, however, waterfall can help.
- What's more, testing is only the fourth step out of five, pushed right down the priority list. This method does not priorities QA, and, especially when 80% of bugs are introduced at the design stage of the typical SDLC, leaving quality as an afterthought could be a costly mistake.
- As software testing methodologies go, it might not be your preferred option if you need to launch a high-quality product.

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.

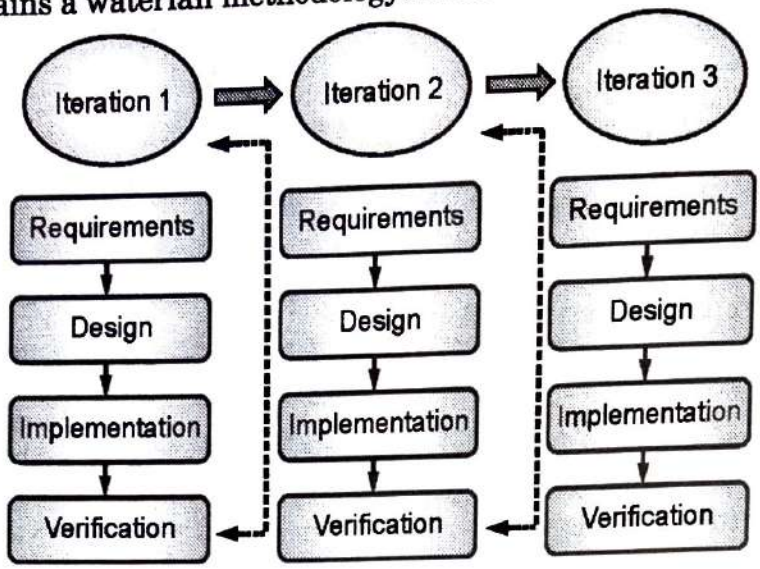
The involvement of QA from the word 'go' means that your product will be well-tested and better quality as a result.

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
  - Convenience for the work of developers and managers who are constantly in touch with a client or a product owner
  - Perfect for start-ups and fast-growing projects."

1.9.3 Iterative Development

- In iterative development, a large project is broken down into smaller, manageable chunks. Each 'chunk' is subjected to a number of iterations of the waterfall model.
- It's almost as if you run a number of different SDLCs within a wider project, and each iteration contains a waterfall methodology. It looks a little bit like this :



(1A14)**Fig. 1.9.3 : Iterative Development**

- As soon as one iteration is completed, the entirety of the software is subjected to testing (verification). Then, feedback from the tests is incorporated into the next cycle. As the iterations progress, the time spent testing can be reduced as a result of experience gained from previous iterations.
- This means that you have more flexibility to test earlier on in the process and test each iteration thoroughly, rather than conducting a huge amount of testing right at the end of your SDLC.

What are the benefits ?

- A considerable benefit of using this method is that testing feedback is available at the end of each iteration. This means that you are testing more regularly than in a method like waterfall, and allowing the results to inform your further decision making.
- Although iterative development does not quite abide by the rules of continuous testing, it does bear similarities to the agile methodology. You test earlier on in the SDLC and incorporate feedback from your tests into the development process. This means you are placing more emphasis on the value of QA and allowing it to influence part of your decision making.

Choosing the right software testing methodology for you

- There are so many methodologies to choose from when it comes to software development. When it comes to the testing part of the process, you need to consider your requirements, project size, scope and budget.
- For smaller projects, where the scope is clear, methods like waterfall can be extremely beneficial. That's because your team are following a straightforward process, with a degree of understanding about where the process will lead.
- But typically, for bigger projects, agile methodologies can have some strong benefits.
- Introducing testing early into your SDLC means you will catch bugs earlier on, and enables you to incorporate testing feedback into the design and build stages. This will help you achieve a better quality product overall, as you shift your focus towards making QA a priority.
- Each project is unique. So consider your options, assess your scope, and use our guide to decide whether software testing methodology is right for you.

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.

Key Difference

- Verification process includes checking of documents, design, code and program whereas Validation process includes testing and validation of the actual product.
- Verification does not involve code execution while Validation involves code execution.
- Verification uses methods like reviews, walkthroughs, inspections and desk-checking whereas Validation uses methods like black box testing, white box testing and non-functional testing.
- Verification checks whether the software confirms a specification whereas Validation checks whether the software meets the requirements and expectations.
- Verification finds the bugs early in the development cycle whereas Validation finds the bugs that verification cannot catch.
- Verification process targets on software architecture, design, database, etc. while Validation process targets the actual software product.
- Verification is done by the QA team while Validation is done by the involvement of testing team with QA team.
- Verification process comes before validation whereas Validation process comes after verification.

1.10.2 Verification vs Validation : Key Difference

UQ. Differentiate between verification and validation. **MU - Dec. 17, Dec. 19**

Sr. No.	Verification	Validation
1.	The verifying process includes checking documents, design, code, and program	It is a dynamic mechanism of testing and validating the actual product
2.	It does ***not*** involve executing the code	It always involves executing the code
3.	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
4.	Whether the software conforms to specification is checked	It checks whether the software meets the requirements and expectations of a customer
5.	It finds bugs early in the development cycle	It can find bugs that the verification process can not catch

Sr. No.	Verification	Validation
6.	Target is application and software architecture, specification, complete design, high level, and database design etc.	Target is an actual product
7.	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.
8.	It comes before validation	It comes after verification

1.10.3 Verification Requirements

- A *verification requirement* provides the rules of verification for a piece of data (verifiable data item). There are many variables included in these rules including where and how the rules apply at runtime.
- For example whether the verification engine needs to apply the rules to participant level data or to a specific case.
- Again using date of birth as an example of a verifiable data item, for some organizations the rules may be to verify this piece of data once and therefore verification engine applies the rules within participant manager.
- For other organizations the rules may require that date of birth is verified at a program level and therefore the verification engine applies the rules to a specific case - see 3.5.4 Verification Requirement Usages for further information.

1.10.3.1 Verification Requirement Properties

The following is an overview of the properties that can be set on a verification requirement.

(1) Due Date and Warning Date

- A number of properties exist for setting a due date on verification. The due days property specifies the number of days after a particular event that verification should fall due.
- Administrators can also specify whether the number of due days should be calculated from the date the case was created or from the date evidence was inserted or received.

- The warning days property specifies how many day's prior notice a caseworker will receive before the verification due date. If no warning date is specified, a caseworker will not receive a warning before the verification due date. Note that due date processing for verification requirements uses workflow functionality. For more information on expiry date and due date workflow processing, see Deadline Management.

(2) Level

- This property indicates the level of verification that must be achieved in order to consider data verified. Evidence will not be considered verified unless a verification item with the appropriate level is received.
- For example, if a verification requirement specifies a level 5 verification item (such as an original birth certificate) then providing a level 1 item (a photocopy of a birth certificate) will not satisfy the verification requirement.
- Alternatively, a combination of verification items that form a verification group of level 5 can be provided to satisfy the verification requirement.

(3) From and To Dates

- These properties indicate the period during which this verification requirement is effective. Note that these properties interact with the effective dates of verification item utilizations and the effective dates of evidence in order to determine the verifications that a caseworker can perform.
- For example, a requirement to verify an income amount might be defined as effective from January to December.
- However, one verification item may be defined as effective from January to July (e.g., a payslip), while another is defined to be effective from July to December (e.g., a tax return).
- The date that the income evidence is active determines which verification item is necessary to satisfy the verification requirement.

(4) Minimum Items

- This property specifies the minimum number of verification items that must be provided before data can be considered verified.
- For example, if the minimum item specified is 2, then the verification requirement will be considered satisfied if at least two verification items or verification groups are provided. When all the verification items specified in a verification group are provided, the Verification Engine will consider it to be a single item.

- A combination of verification items and groups can also be provided to satisfy the minimum number of verification items of a verification requirement.

(5) Mandatory

- This property indicates whether or not the verification requirement is mandatory. A mandatory verification requirement means that evidence and cases associated with the verification may not be activated until the rules defined for the verification have been met.
- When the mandatory property is not set, the verification requirement is optional and therefore the evidence associated with the verification can be activated even if the evidence has not yet been verified.

(6) Client-Supplied

- This property indicates if it is the case participant's responsibility to supply the verification items. This property could be used during communications between the organization and the client to ensure that a client is not asked to supply a verification item that should be sourced elsewhere.
- Note there is no system processing associated with this property, it is used for informational purposes only for the user.

(7) Re-verification

- This property allows users to specify the Cúram Verification engine's response to changes to Active evidence.
- The following list provides the names and impact of the settings for this property. Note that re-verification property does not apply to participant evidence.

(8) Recertify Always

- If a caseworker changes Active evidence, no previously met verification requirements are carried over to the new In Edit evidence.
- The new In Edit record must then be recertified.

(9) Recertify If Changed

- If caseworker changes Active evidence, and the value entered for the verifiable data item or any dependent data items has not changed, the existing verification information on the Active record is copied to the new In Edit record.
- If the value entered for the data item or any dependent data items has changed, then no verification information is copied from the Active record.

(10) Never Recertify

If a caseworker changes Active evidence, the verification information on the Active record is always copied to the In Edit record.

1.11 VERIFICATION OF HIGH LEVEL DESIGN AND LOW LEVEL DESIGN VALIDATION

UQ. Explain Verification in high level and low level design. **MU - May 18**

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.

1.11.1 Difference between High Level Design and Low Level Design

Sr. No.	High level design	Low level design
1.	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.
2.	High Level Design in short called as HLD.	Low Level Design in short called as LLD.
3.	It is also known as macro level/system design.	It is also known as micro level/detailed design.

Sr. No.	High level design	Low level design
4.	It describes the overall description/architecture of the application.	It describes detailed description of each and every module.
5.	High Level Design expresses the brief functionality of each module.	Low Level Design expresses details functional logic of the module.
6.	It is created by solution architect.	It is created by designers and developers.
7.	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.
8.	It is created first means before Low Level Design.	It is created second means after High Level Design.
9.	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 HLD the output criteria is program specification and unit test plan.

Chapter Ends...

❑❑❑

2.1 DYNAMIC TESTING

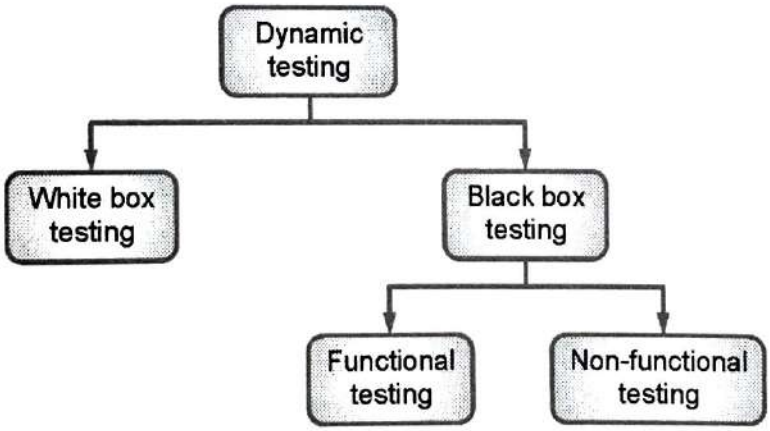
GQ. What is Dynamic Testing ?

- The name itself suggests that it is "Dynamic" in nature, which means altering. This is the kind of testing that we do with altering values or conditions **by executing the code**.
- This includes testing the request in real time by giving inputs and groping the result or the output value of performance.

Dynamic Testing Example

- The easiest instance to understand this is the login functionality of any application, like Google's gmail.com. If we are generating an account, and a password for the account, you would have certain rules for creation a strong password.
- For instance, 8 typescripts long, needs to must a capital letter and at least one distinct character.
- These are unknown but dissimilar conditions or limitations and if the user inputs any value that earnings from these rules, the proposition should whichever warn or reject.
- If you are testing this functionality as an instance, you would input all the environments required to test this and then approve the output.
- You would also input the non-working limits, for instance, input a 4 character password and settle if there is an error that is thrown. This is all part of dynamic testing.

2.1.1 Types of Dynamic Testing



(1B1)**Fig. 2.1.1 : Types of Dynamic Testing**

Dynamic Testing is Approximately Confidential into Two Types

1. White box testing

White box testing looks at the internal mechanisms of the code. For this thoughtful of testing, the tester should know the code increasing, evaluation and be able to interpret the code.

GQ. What is white box testing function?

White box testing is a software evaluating method used **to examine the internal structure, design, coding and inner-working of software**. Developers use this testing method to verify the flow of inputs and outputs through the application, improving usability and design and strengthening security.

2. Black box testing

Black box testing features at only the functionality of the Application Under Test (AUT). This does not require the tester to recognize the application details or be able to interpret the inner devices of the code. This is the type of testing normally done by the QA subdivision.

GQ. What is black box testing function ?

Black box testing is used **to test the system against external factors responsible for software failures**. This testing approach focuses on the input that goes into the software, and the output that is produced. The testing team does not cover the inside details such as code, server logic, and development method.

2.1.2 Advantage and Disadvantage of Dynamic Testing

Advantages

1. Dynamic testing is thorough, which aspects at in depth functionality of the submission so the quality is of uppermost standards.
2. Dynamic testing process is well established and hence the presentation is tested from users and business perspective thus increasing the quality standards.
3. Complex imperfections can be caught which may have runaway the review processes.
4. Dynamic testing can be automatic using tools.

Disadvantages

1. Since dynamic testing surveys a complex detailed procedure, it takes time.

2. It is time consuming and it costs more money to the organizations because of the need for the resources and time.
3. Dynamic testing is usually performed after coding is completed and hence defects are discovered later in the lifecycle

2.1.3 Black Box Testing Vs. White Box Testing : Key Alterations

GQ. What is Black Box testing ?

- In Black box testing, a sample doesn't have any substantial about the internal employed of the software system.
- Black box difficult is a high level of testing those emphases on the performance of the software. It includes testing from an external or end user viewpoint.
- Black box testing can be functional to almost every level of software testing: unit, integration, system, and taking.

GQ. What is White Box testing ?

- White box testing is a testing method which checks the internal operative of the system. In this method, testing is based on attention of code statements, subdivisions, paths or conditions.
- White Box testing is measured as low level testing. It is also called glass box, obvious box, clear box or code base testing.
- The white box Testing method shoulders that the path of the logic in a unit or database is known.

Key Difference

- In Black Box, testing is done deprived of the information of the internal structure of database or submission whereas in White Box, testing is done with information of the internal construction of program.
- When we associate Black box and White box stimulating, Black Box test doesn't necessitate software design material however the White Box test requires programming knowledge.
- Black Box challenging has the main goal to test the performance of the software whereas White Box testing has the main goal to test the internal process of the system.
- Associating White box testing and Black box testing, Black Box testing is focused on external or end user perception whereas White Box testing is intensive on code structure, environments, paths and offices.

- Black Box test delivers low granularity reports however the White Box test delivers high granularity reports.
- Associating Black box testing vs White box testing, Black Box testing is a not time overriding process while White Box testing is a time consuming process.

2.1.4 Differentiate between White Box and Black Box Testing

UQ. Differentiate between white box and black box testing. **MU - Dec. 16, Dec. 17**

Table 2.1.1

Parameter	Black Box testing	White Box testing
Definition	It is a testing method which is used to test the software without the knowledge of the internal structure of program or application.	It is a testing method in which internal structure is known to the tester.
Alias	It also known's as data driven, box testing, data, and functional testing.	It is also called structural testing, clear box testing, code based testing, or glass box testing.
Base of Testing	Testing is based on external expectations; internal performance of the application is unidentified.	Internal working is known, and the tester can test consequently.
Usage	This type of testing is ideal for higher levels of testing like System Testing, Receipt testing.	Testing is best suited for a lower level of testing like Unit Testing, Integration testing.
Programming knowledge	Programming knowledge is not needed to attain Black Box testing.	Programming information is required to thorough White Box testing.
Implementation knowledge	Implementation knowledge is not necessitating doing Black Box testing.	Complete understanding needs to implement White Box testing.
Automation	Test and programmer are reliant on each other, so it is tough to automate.	White Box testing is informal to automate.
Objective	The main objective of this challenging is to check what functionality of the system under test.	The main objective of White Box testing is done to check the excellence of the code.
Basis for test cases	Testing can start after formulating obligation requirement document.	Testing can start after formulating for Detail design document.

Parameter	Black Box testing	White Box testing
Tested by	Performed by the end user, designer, and tester.	Usually done by tester and developers.
Granularity	Granularity is low.	Granularity is high.
Testing method	It is based on experimental and error method.	Data province and internal limitations can be tested.
Time	It is less thorough and time overriding.	Comprehensive and laborious method.
Algorithm test	Not the best method for procedure testing.	Best suited for algorithm testing.
Code Access	Code access is not essential for Black Box Testing.	White box testing involves code access. Thus, the code could be stolen if testing is outsourced.
Benefit	Well suitable and efficient for large code subdivisions.	It allows eliminating the extra lines of code, which can bring in unseen imperfections.
Skill level	Low expert testers can test the application with no knowledge of the operation of programming language or working system.	Need an expert tester with vast experience to achieve white box testing.
Techniques	• Correspondence separating is Black box testing procedure is used for Black box testing. • Equivalence separating divides input values into valid and invalid partitions and selecting conforming values from each divider of the test data. • Boundary value analysis • checks limitations for input values.	• Statement Coverage, Branch coverage, and Path reporting are White Box testing technique. • Statement Coverage authorizes whether every line of the code is executed at least once. • Branch coverage legalizes whether each branch is executed at least once. • Path coverage method tests all the paths of the program.

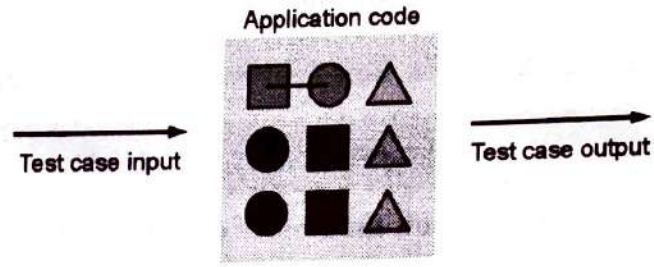
2.1.5 White Box Testing

UQ. Is white box testing really necessary ? Give reasons. **MU - May 19**

1] White box testing is a method that allows testers to review and confirm the inner workings of a software system its code, substructure, and additions with external systems.

2] White box testing is an essential part of automated build processes in a modern Continuous Integration / Continuous Delivery (CI / CD) growth pipeline.

3] White box testing is often referenced in the context of Static Application Security Testing (SAST), a method that payments source code or binaries mechanically and delivers feedback on bugs and possible weaknesses.



(1B2)Fig. 2.1.2 : White Box Testing

- White box testing affords inputs and observes outputs, since the inner workings of the code.

2.1.6 White Box Testing Pros and Cons

Sr. No.	Pros	Cons
1.	Ability to attain comprehensive code coverage	Requires a large effort to automate
2.	Easy to mechanize	Sensitive to deviations in code base, automation necessitates expensive conservation
3.	Reduces message overhead between testers and developers	Cannot test predictable functionality that does not exist in the codebase
4.	Allows for incessant improvement of code and expansion practices	Cannot test from the user's perspective

2.1.7 What Does White Box Testing Focus On ?

White box tests can focus on determining any of the following difficulties with an application's code :

- **Security gaps and vulnerabilities :** Checking to see if security best practices were applied when coding the application, and if the code is helpless to known security threats and adventures.
- **Broken or poorly structured paths :** Recognizing provisional logic that is terminated, broken or inefficient.
- **Expected output :** Performing all possible inputs to a function to see if it always returns the predictable result.
- **Loop testing :** Examination single loops, concatenated loops and nested loops for efficiency, conditional logic, and correct treatment of local and worldwide variables.

- **Data Flow Testing (DFT) :** Following variables and their values as they pass through the code to find variables that are not correctly prepared, declared but never used, or imperfectly operated.

2.1.8 Black Box Testing : Borderline Value Analysis

GQ. What are the types of errors noticed by black box testing?

As all we know the most of **errors** occurs at boundary of the input values.

This method efforts to find errors in the succeeding groupings :

- Incorrect or missing function.
- Interface **errors.**
- **Errors** in data structures or exterior database access.
- Performance or presentation **errors.**
- Initialization and termination **errors.**

A] Boundary value analysis (BVA)

- Boundary value examination is the most normally used test case design method for black box testing. As all we know the most of errors occurs at boundary of the input values.
- This is one of the methods used to find the error in the limitations of input values rather than the centre of the input value range.

B] Correspondence class dividing

- The correspondence class separating is the black box test case design method used for writing test cases. This method is used to decrease huge set of thinkable inputs to small but similarly operative inputs.
- This type of method is used to find errors in the data structure and can test the comprehensive set of program in divider.

Similarly, you may ask, what are the types of errors in software testing ?

Collective Classes of Software Errors

1. **Functionality Errors :** Functionality is a way the software is planned to behave.
2. **Communication Errors :** These faults occur in communication from software to end user.
3. **Missing command errors :** This materializes to occur when an predictable command is missing.
4. **Syntactic Error :** Also, what is black box testing with example?

2.1.9 Comparison of Black Box and White Box Testing

Black Box Testing	White Box Testing
The main focus of black box testing is on the validation of your functional necessities.	White Box Testing (Unit Testing) authenticates internal structure and working of your software code

GQ. What is black box approach ?

- **Black box** testing is a **method** of software testing that observes the functionality of an application without peering into its internal constructions or workings.
- This **method** of test can be functional practically to every level of software testing: unit, integration, system and getting.

GQ. What is white box testing and black box testing ?

- **Definition: Black Box Testing** is a software **testing** method in which the internal structure/ design/ application of the item being tested is NOT.

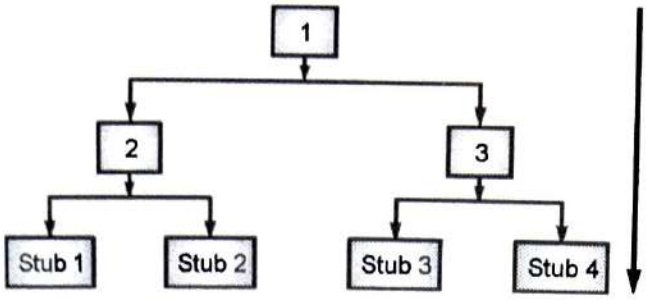
Known to the **tester White Box Testing** is a software **testing** method in which the internal structure/ design/ application of the item being tested is known to the **tester**.

2.1.10 Top down Integration Testing

UQ. Differentiate between top down and bottom up testing. **MU - Dec. 16**

- It is also known as incremental combination testing.
- The higher level mechanisms are first complete after which the lower level modules are tested.
- Once it is done, they are combined.
- The higher level components are the highest modules and the lower level components are the sub modules.
- It uses stubs to simulate the sub components.
- If the sub module has not been fully progressive, the stub acts like a supplementary to it.
- It is valuable in cases where significant defect occurs at the top of the program.
- The main module is intended first and then the sub modules or subroutines are called from it.
- It is performed on structure or process leaning programming languages.
- It is a simple testing technique.
- It works on big to small mechanisms.
- Stub components need to be created.

Here's the flow diagram for stub

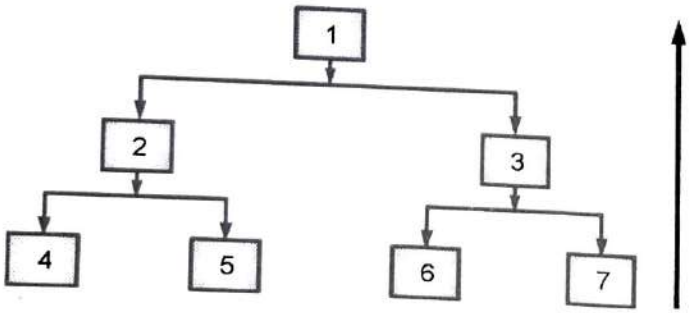


(1B3)**Fig. 2.1.3 : Top-down Integration Testing**

☞ Bottom up Integration Testing

- The lower level modules are tested first afterward which the higher level modul are experienced.
- The lower level modules are the sub-modules and higher level modules are th main modules.
- It uses test drivers to initiate and pass the required data to the sub-modules.
- The test drivers are used to simulate the main module.
- If the foremost module hasn't been established yet, the driver works as additional to it.
- It is beneficial in cases where the crucial flaws are recognized at the bottom of th program.
- It is generally applied on object-oriented programming languages.
- It is highly complex and data intensive.
- It works on small to big mechanisms.
- The driver modules need to be formed.

Here's the flow diagram



(1B4)**Fig. 2.1.4 : Bottom-up Integration Testing**

UEx. 2.1.1 MU - Dec. 16

A program reads three numbers, A, B, and C, with a range [1, 50] and prints the largest number. Design test cases for this program using equivalence class testing technique.

☑ **Soln. :** First we partition the domain of input as valid input values and invalid values. so we get following classes –

I1I1 = { <A,B,C> : 1 ≤ A ≤ 50 }

I2I2 = { <A,B,C> : 1 ≤ B ≤ 50}

I3I3 = { <A,B,C> : 1 ≤ C ≤ 50}

I4I4 = { <A,B,C> : A < 1}

I5I5 = { <A,B,C> : A > 50}

I6I6 = { <A,B,C> : B < 1}

I7I7 = { <A,B,C> : B > 50}

I8I8 = { <A,B,C> : C < 1 }

I9I9 = { <A,B,C> : C > 50}

Now test cases can be intended from the above derived modules taking one case from each class such that the test case insurances maximum valid input classes and detached test cases for each invalid class.

Test case id	A	B	C	Expected Result	Classes covered test cases
1	13	25	36	C is greater	I1,I2,I3I1,I2,I3
2	0	13	45	invalid input	I4I4
3	51	34	17	invalid input	I5I5
4	29	0	18	invalid input	I6I6
5	36	53	32	invalid input	I7I7
6	27	42	0	invalid input	I8I8
7	33	21	51	invalid input	I9

Another set of equivalence classes based on some possibilities of 3 integers A, B, C.

I1I1 = { < A,B,C > : A>B, A>C }

I2I2 = { < A,B,C > : B>A , B>C }

I3I3 = { < A,B,C > : C>A , C>B }

I4I4 = { < A,B,C > : A = B, A + C}

I5I5 = { < A,B,C > : B = C, A + B }

I6I6 = { < A,B,C > : A = C, C = B}

I7I7 = { < A,B,C > : A = B = C }

Test case ID	A	B	C	Expected Result	Classes covered test cases
1	25	13	13	A is greater	I1,I5I1,I5
2	25	40	25	B is greater	I2,I6I2,I6
3	24	24	37	C is greater	I3,I4I3,I4
4	25	25	25	All are equal	I7

UEx. 2.1.2 MU - Dec. 18

A program reads an integer number within the range [1,100] and determines whether It is a prime number or not. Design the test cases for this program using boundary value analysis.

☑ **Soln. :**

- Boundary value analysis (BVA) is based on testing the boundary standards of valid and invalid partitions. The Performance at the edge of each correspondence divider is more likely to be incorrect than the performance within the partition, so limitations are an area where testing is likely to produce defects.
- Every partition has its maximum and minimum values and these maximum and minimum values are the boundary standards of a partition.
- A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.
- Tests can be planned to cover both valid and invalid boundary values. When planning test cases, a test for each boundary value is chosen.
- For each boundary, we test +/-1 in the least important digit of either side of the boundary.
- Boundary value investigation can be applied at all test levels.
- Instance on Boundary Value Analysis Test Case Design Technique:
  Accept, we have to test a field which accepts
  Age 18 – 56

AGE [Enter Age] * Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min – 1)	**Valid (min, + min, – max, max)**	**Invalid (max + 1)**
17	18, 19, 55, 56	57

Minimum boundary value is 18

Maximum boundary value is 56

Valid Inputs: 18,19,55,56

Invalid Inputs: 17 and 57

Test case 1: Enter the value 17 (18 – 1) = Invalid

Test case 2: Enter the value 18 = Valid

Test case 3: Enter the value 19 (18 + 1) = Valid

Test case 4: Enter the value 55 (56 – 1) = Valid

Test case 5: Enter the value 56 = Valid

Test case 6: Enter the value 57 (56 + 1) = Invalid

Example

- Undertake we have to test a manuscript field (Name) which receives the length between 6-12 characters.

Name | Enter Name | * Accepts characters length (6 - 12)

BOUNDARY VALUE ANALYSIS		
Invalid (min – 1)	**Valid (min, + min, – max, max)**	**Invalid (max + 1)**
5 characters	6, 7, 11, 12 characters	13 characters

Minimum boundary value is 6

Maximum boundary value is 12

Valid text length is 6, 7, 11, 12

Invalid text length is 5, 13

Test case 1: Text length of 5 (min-1) = Invalid

Test case 2: Text length of exactly 6 (min) = Valid

Test case 3: Text length of 7 (min+1) = Valid

Test case 4: Text length of 11 (max-1) = Valid

Test case 5: Text length of exactly 12 (max) = Valid

Test case 6: Text length of 13 (max+1) = Invalid

1. Test cases using Boundary value checking (BVC)

Since these is one variable, the total number of test cases will be 4n + 1 = 5

In this instance, the set of minimum and maximum values is publicized below:

Min value = 1

Min value = 2

Max value = 100

–

Max value = 99

Nominal ...

2.1.11 What is Software Testing Technique ?

UQ. Explain with example software testing. **MU - Dec. 17**

Software Testing Techniques benefit you design enhanced test cases. Meanwhile thorough testing is not possible; Manual Testing Methods help moderate the number of test cases to be performed while cumulative test coverage. They help identify test circumstances that are otherwise difficult to identify.

Boundary Value Analysis (BVA)

- Boundary value analysis is based on testing at the limitations between partitions. It includes maximum, minimum, inside or outside boundaries, characteristic values and error values.
- It is usually seen that a large number of errors occur at the limitations of the defined input values somewhat than the centre. It is also recognized as BVA and gives a selection of test cases which workout bounding values.
- This black box testing technique matches correspondence partitioning. This software testing method base on the principle that, if a system works well for these specific values then it will work effortlessly well for all values which comes between the two boundary values.

Guidelines for Boundary Value analysis

- If an input condition is restricted between values x and y, then the test cases should be calculated with values x and y as well as values which are above and below x and y.
- If an input condition is a large number of values, the test case should be industrialized which need to work out the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.
- Apply strategies 1 and 2 to output conditions. It gives an output which reproduces the minimum and the maximum values predictable. It also tests the below or above values.

Example

Input condition is valid between 1 to 10

Boundary values 0,1,2 and 9,10,11

2.2 EQUIVALENCE CLASS TESTING

UQ. A program reads three numbers n1, n2, n3 in the range < 100 to 100 and prints the smallest number. Design test cases for this program using equivalence class testing technique. **MU - May 16**

Equivalence Class Partitioning

- Equivalent Class Separating allows you to divide set of test condition into a partition which should be measured the same.
- This software testing method divides the input domain of a program into programs of data from which test cases should be considered.
- The concept behind this method is that test case of a illustrative value of each class is equal to a test of any other value of the similar class. It permits you to identify valid as well as offensive equivalence classes.

Example

Input conditions are valid between

1 to 10 and 20 to 30

Hence there are five equivalence classes

--- to 0 (invalid)

1 to 10 (valid)

11 to 19 (invalid)

20 to 30 (valid)

31 to --- (invalid)

You select values from each class, i.e.,

– 2, 3, 15, 25, 45

```
int n1,n2,n3;
if(n1<n2 &amp;amp;&amp;amp;=""" n1<n3)<="""" p="""">
{
System.out.println("The smallest number is "+n1);
}
else if(n1<n3)< p="""">
{
System.out.println("The smallest number is"+n2);
}
else
{
System.out.println("The smallest number is"+n3);
}
```

Test cases for this program using equivalence...

2.2.1 State Table based Testing

- You can use State Table **to regulate invalid system transitions**. In a State Table, all the valid situations are listed on the left side of the table, and the proceedings that cause them on the top.
- Each cell signifies the state system will move to when the consistent event occurs.
- For the State Table, each and each valid state are providing on the left hand side of the table, also events that make them occur are on above.
- All of the cells signify the system of state would finally move to whenever the conforming event takes place.

	Correct Password	Incorrect Password
S1) Start	S6	S3
S2) 1st Try	S6	S3
S3) 2ND Try	S6	S4
S4) 3rd Try	S6	S5
S5) 4th Try	S6	S7
S6) Access		
S7) Close Application		

All the valid states are listed on the left side of the table while invalid states on the right side

- Let us take an instance. When you are in the D1 state, then you enter the correct password, then you are stimulated to D6 state which is Access Decided State.
- In case we have entered in the incorrect password in the first try then you would be enthused to D3 state or given a 2nd Try.
- In a comparable manner, we can predict the residual states.
- We have painted two of the conditions that are unacceptable with the help of this technique.
- Undertake that we are in the state S6, denotation, we are logged in into the application previously, and subsequently we open the other instance of the reservation of flight, then input either right or wrong passwords for that same agent only.
- The answer of the system for such situation has to get testing done.

2.2.2 Cause Effect Graphing CFG based Testing

UQ. Differentiate between Control Flow Graph [CFG] and Data Flow Graph [DFG] **MU - Dec. 18**

Control Flow

1. Process Oriented
2. Doesn't achieve or pass data between mechanisms.
3. It functions as a task coordinator
4. In control flow tasks necessitates completion (Success, failure or completion)
5. Synchronous in nature, this means, task necessitates completion before touching to next task. If the tasks are not associated with each other but still they are synchronous in nature.
6. Tasks can be performed both parallel and serially
7. Three types of control flow elements in SSIS 2005
   (i) **Containers :** Provides structures in the packages
   (ii) **Tasks :** Provides functionality in the packages
   (iii) **Precedence Constraints :** Connects containers, executable and tasks into an ordered control flow.
   (iv) We can control the arrangement implementation for tasks and also specify the circumstances that tasks and containers run.
8. It is likely to include nested containers as SSIS Architecture supports nesting of the containers. Control flow can contain multiple levels of nested containers.

Data Flow

1. Streaming in nature
2. Information oriented
3. Passes data between other components
4. Transformations work together to achieve and process data. This means first set of data from the source may be in the final terminus step while at the similar time other set of data is still graceful. All the transformations are doing work at the similar time.
5. Three types of Data Flow components
   (i) **Sources :** Excerpts data from the numerous sources (Database, Text Files etc)
   (ii) **Transformations :** Cleans, modify, merge and reviews the data
   (iii) **Destination :** Loads data into terminuses like database, files or in memory datasets.

☞ Error predicting

- Testing is the process of classifying defects, where a defect is any alteration between actual and predictable results.
- "A mistake in coding is called Error, error found by tester is called Defect, defect recognized by growth team then it is called Bug, build does not meet the supplies then it Is Failure."

☞ Defect

- It can be basically defined as a variance between predictable and definite. The defect is an error found AFTER the application goes into production.
- It normally refers to numerous troubles with the software products, with their external behavior or with its internal features.
- In other words, a Defect is a difference between predictable and actual results in the context of testing. It is the deviance of the customer requirement.

Defect can be categorized into the following :

- **Wrong :** When supplies are implemented not in the right way. This defect is a alteration from the given specification. It is Wrong!
- **Missing :** A condition of the customer that was not satisfied. This is a modification from the conditions, an indication that a specification was not implemented, or a requirement of the customer was not noted appropriately.
- **Extra :** A prerequisite integrated into the product that was not expected by the end customer. This is always a alteration from the condition, but maybe an quality desired by the user of the product. However, it is measured a defect because it's a variance from the current requirements.

UQ. Describe the change between failure, fault and error. **MU - May 18**

- **Error :** An error is a fault, misconstruction, or misinterpretation on the part of a software developer. In the grouping of the designer, we contain software engineers, computer operator, analysts, and testers. For example, a creator may misinterpret a enterprise representation, or a computer operator might type a variable name incorrectly – leads to an Error. It is the one that is generated because of the wrong login, loop or syntax. The error typically arises in software; it indications to a change in the functionality of the program.
- **Failure :** A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development, Failures are usually observed by testers.

- **Fault :** An improper step, procedure or data meaning in a computer program that causes the program to perform in an accidental or
- **Bug :** A bug is the result of a coding error. An Error found in the expansion atmosphere before the product is shipped to the customer. A software design error that causes a program to work poorly, produce incorrect results or crash. An error in software or hardware that causes a program to malfunction. A bug is the terms of Tester.
- **Surprising manner :** A fault is presented into the software as the result of an error. It is an difference in the software that may cause it to behave imperfectly, and not according to its description. It is the result of the error.
- The software industry can still not agree on the meanings for all the above. In spirit, if you use the term to malicious one precise thing, it may not be unspoken to be that thing by your spectators.

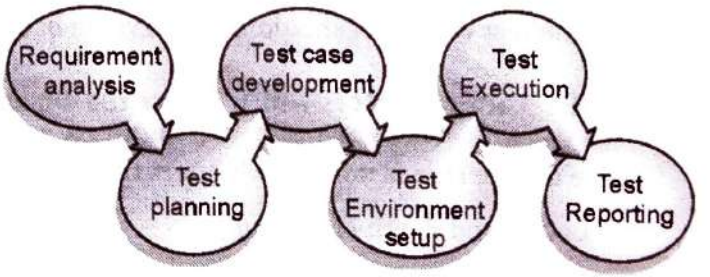
2.2.3 What is Lifecycle?

UQ. Explain the software Testing Life Cycle. **MU - Dec. 16**

- Lifecycle in the modest term refers to the classification of changes from one form to other forms. These fluctuations can happen to any tangible or imperceptible things.
- Each entity has a lifecycle from its initiation to retire / demise.
- In a parallel fashion, Software is also an entity. Just like emerging software includes a sequence of steps, testing also has stages which should be performed in a confident sequence.
- This phenomenon of performing the testing doings in a systematic and intentional way is called testing life cycle.

GQ. What Is Software Testing Life Cycle (STLC) ?

- Software Testing Life Cycle mentions to a testing process which has detailed steps to be executed in a certain sequence to ensure that the excellence goals have been met.
- In the STLC process, each activity is approved out in a intentional and systematic way.
- Each phase has dissimilar goals and deliverables. Different governments have different phases in STLC; however, the basis remains the same.



(1B5)**Fig. 2.2.1 : Software Testing Life Cycle**

Below are the phases of STLC

1. Requirements phase	2. Planning Phase
3. Analysis phase	4. Design Phase
5. Implementation Phase	6. Execution Phase
7. Conclusion Phase	8. Closure Phase

1. Requirement Phase

- Throughout this phase of STLC, analyze and study the requirements. Have suggesting sessions with other groups and try to find out whether the necessities are testable or not.
- This phase helps to categorize the likelihood of the testing. If any feature is not testable, interconnect it during this phase so that the extenuation strategy can be planned.

2. Planning Phase

- In applied scenarios, Test preparation is the first step of the testing process. In this phase, we recognize the doings and capitals which would help to meet the testing purposes. During development we also try to recognize the metrics, the method of assembly and pursuing those metrics.
- On what basis the development is done? Only requirements?
- The answer is NO. Requirements do form one of the bases but there are 2 other very imperative factors which impact test planning. These are :
  - Test strategy of the organization.
  - Risk analysis / Risk Organization and mitigation.

3. Analysis Phase

- This STLC phase describes "WHAT" to be tested. We essentially identify the test conditions through the necessities document, product risks, and other test bases.
- The test condition should be observable back to the condition.

2.2.4 Various Factors which Affect the Documentation of Test Conditions

- There are various factors which affect the documentation of test conditions :
  - Levels and depth of testing
  - The difficulty of the product
  - Product and project risks

- Software growth life cycle involved
- Test management
- Skills and information of the team.
- Obtainability of the stakeholders.

- We should try to write down the test conditions in a complete way. For instance, for an e-commerce web application, you can have a test complaint as "User should be able to type a payment". Or you can feature it out by saying "User should be able to make expense through NEFT, debit card, and credit card".
- The most imperative advantage of writing the complete test complaint is that it growths the test treatment since the test cases will be written on the basis of the test complaint; these details will trigger to write more complete test cases which will finally increase the coverage.
- Also, recognize the exit standards of the testing, i.e. regulate some conditions when you will stop the testing.

1. Design Phase

This phase describes "HOW" to test. This phase includes the subsequent tasks :

- Detail the test condition. Break down the test circumstances into multiple sub conditions to increase coverage.
- Identify and get the test data
- Identify and set up the test situation.
- Create the requirement traceability metrics
- Create test treatment metrics.

2. Implementation Phase

- The main task in this STLC phase is of formation of the thorough test cases. Prioritize the test cases also recognize which test case will developed part of the reversion suite. Before confirming the test case, It is significant to carry out the review to ensure the precision of the test cases. Also, don't forget to take the sign off of the test cases before actual implementation starts.
- If your project includes automation, identify the applicant test cases for mechanization and proceed for scripting the test cases. Don't forget to review them!

3. Execution Phase

- As the name proposes, this is the Software Testing Life Cycle phase where the definite execution takes place. But before you start your execution, make sure that your entry standard is met.

- Implement the test cases, log deficiencies in case of any inconsistency. Concurrently fill your traceability metrics to track your progress.

4. Conclusion Phase

- This STLC phase essences on the exit criteria and reportage. Dependent on your project and shareholders choice, you can resolve on reportage whether you want to send out a daily report of the weekly report, etc.
- There are dissimilar types of reports (DSR – Daily status report, WSR – Weekly status reports) which you can send, but the imperative point is, the content of the report variations and depends upon whom you are transfer your reports.
- If Project managers belong to testing related then they are more attentive in the technical aspect of the project, so contain the technical clothes in your report (number of test cases passed, failed, defects raised, severity 1 defects, etc.).
- But if you are reporting to upper shareholders, they might not be attentive in the technical things so report them about the risks that have been moderated through the testing.

5. Closure Phase

Tasks for the closure doings include the following :

- Check for the conclusion of the test. Whether all the test cases are performed or mitigated purposely.
- Check there is no severity 1 defects opened.
- Do lessons learned meeting and create lessons learned text. (Include what went well, where the scope of improvements is and what can be enhanced)

2.3 WHITE BOX TESTING TECHNIQUES

2.3.1 White Box Testing

- **White Box Testing** is software testing technique in which interior structure, design and coding of software are verified to verify flow of input output and to advance design, usability and safety.
- In white box testing, code is visible to instances so it is also called Clear box testing, open box testing, glowing box testing, Code based testing and Glass box testing.
- It is one of two parts of the Box Testing method to software testing. Its supplement, Black box testing, encompasses testing from an outside or end user type viewpoint. On the other pointer, White box testing in software engineering is based on the internal mechanisms of a submission and alternates commonly internal testing.

- The term "White Box" was used since of the see finished box thought. The clear box or White Box name signifies the aptitude to see complete the software's outside shell (or "box") into its inner mechanisms. Likewise, the "black box" in "Black Box Testing" signifies not being able to see the inner devices of the software so that only the end user knowledge can be tested.

GQ. What do you verify in White Box Testing ?

- White box testing includes the testing of the software code for the subsequent:
  - Internal security holes
  - Damaged or poorly organized paths in the coding processes
  - The flow of precise inputs through the code
  - Predictable output
  - The functionality of provisional loops
  - Testing of each statement, object, and function on an separate basis
- The testing can be done at system, addition and unit levels of software development. One of the basic areas of white box testing is to verify a employed flow for an application.
- It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the predictable output, you have encountered a bug.

☞ White Box Testing Techniques

- A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It categorizes areas of a program that are not trained by a set of test cases. Once gaps are recognized, you create test cases to verify tentative parts of the code, thereby increasing the quality of the software product
- There are mechanical tools presented to complete Code coverage examination. Underneath are a few attention analysis methods a box tester can use:
  - **Statement Coverage :** This method necessitates every possible statement in the code to be tested at least once throughout the testing process of software engineering.
  - **Branch Coverage :** This method checks every possible path (if else and other provisional loops) of a software application.
- Apart from above, there are frequent coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each method has its own merits and challenges to test (cover) all parts of software code.

- **Consuming Statement and Branch coverage you generally attain 80 90% code coverage which is sufficient.**

Types of White Box Testing

White box testing includes several testing kinds used to evaluate the usability of an application, block of code or precise software package. There are listed below –

- **Unit Testing :** It is often the first type of testing done on an presentation. Unit testing is achieved on each unit or block of code as it is industrialized. Unit Testing is basically done by the programmer.

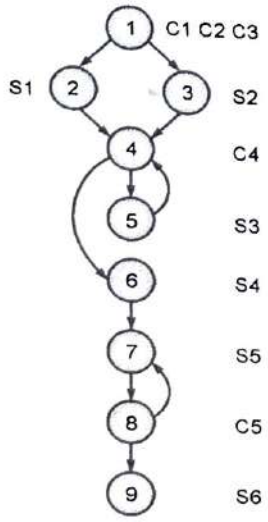
  As a software designer, you improve a few lines of code, a single function or an object and test it to make sure it works before current Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs recognised in this stage are inexpensive and easy to fix.

- **Testing for Memory Leaks :** Memory leaks are important causes of slower running submissions. A QA specialist who is knowledgeable at detecting memory leaks is essential in cases where you have a slow running software application.

UEx. 2.3.1 **MU - May 16**

Draw control flow graph and find cyclomatic complexity for the following PDL if(c1 or c2 and c3) s1:
Else s2:
While (c4) s3:
S4;
Do s5; while (c5);
S6

Soln. :



(1B6)**Fig. P. 2.3.1 : Control Flow Graph**

Cyclomatic complexity can be given as,

$V(G) = E - N + 2P = 11 - 9 + 2 = 4$

UEx. 2.3.2 **MU - May 16**

Design test cases to find maximum of 4 nos

☑ **Soln. :**

Steps	Description	Input Data	Expected Result
1	Enter 4 numbers	0 to 9	(For eg: 10,15,2,8) Inputs A...
2	Enter numbers and alphab...	0 to 9; A to Z; a to z	Invalid inputs
3	Enter numbers and special...	0 to 9; ~,!,@,#,$,%,^,&,*,(,),_,-	Invalid inputs

(1) Needs

White Box Testing is essential because it helps to test the following:

- It is executed at different levels such as system, integration and unit level of software development.
- One primary goal of White Box Testing is to verify the working of an application.

(2) Logic coverage criteria

- White-box testing provides the degree to which tests cover the logic of the software program. Exhaustive white box testing implies that each path along the program is executed; but rigorous testing is unachievable goal for a program with loops.

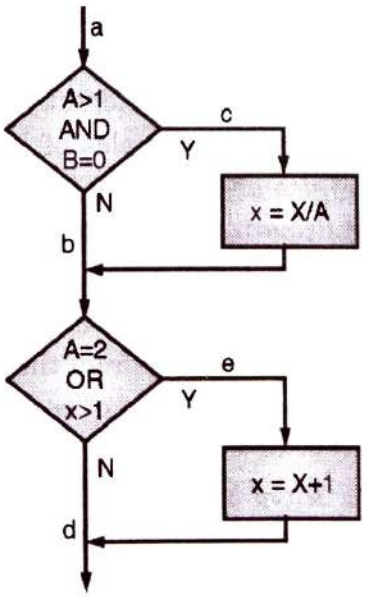


Fig. P. 2.3.2 : Small program to be checked

- If you refuse to conduct path testing, it may seem that it would be worth executing each statement through the program at least once. Unfortunately though, this is a weak criterion for a good back-box test. Suppose that the Figure above shows a small program to be tested. An equivalent program written in a procedural language, PL / 1, follows :

```
M: PROCEDURE (A, B, X);
IF ((A > 1) & (B = 0)) THEN DO;
X = X/A;
END;
IF ((A = 2) | (X > 1)) THEN DO;
X = X + 1;
END;  END;
```

- You could execute each statement by creating a single test, which traverses the path ***ace***. In other words, if you set A = 2, B = 0, and X = 3 at point ***a***, each statement would be executed once (in fact, X can take any integer value >1).
- Unfortunately, this criterion is worse than you may think. For instance, let the first decision be an ***or***, not an ***and***. If that is the case, this error will not be found. Let the second decision in the program be X> 0; if so, this error will not be found. In addition, there is a path in which X does not change (path abd). If there is an error, then it will not be detected. Thus, the statement coverage criterion is a poor one that it is usually not used.
- Using services of game testing company you become able to play video games free of charge! All you need you do is to control their quality by finding defects and bugs in them.
- Decision coverage is a stronger logic coverage criterion. Based on this criterion, a fair number of test cases should be written, for each decision to have true and false value at least one time. Put it differently, each branch changes direction must be executed at least once. Examples of branch statements or transition operators are *do-while, if-else or switch-case* statements (or COBOL PERFORM UNTIL statements). Multipath **go to** statements can be qualified in some programming languages, for instance, such as FORTRAN.
- Decision coverage usually meets statement coverage criterion. As each statement is on some path which emanates either from the transition operator or from the entry point of the program, the execution of each statement is a must given that each branch direction is executed.
- Nevertheless, there are three exceptions. The first – the program has no decisions. The second: programs or subroutines have several entry points; this operator can only be executed if the program is executed at the appropriate entry point. The third one – operators are within ON-units; execution of each branch direction does not necessarily cause the execution of all ON-units.

(3) Basis path testing

- We'll define basic path testing in this post, as well as the benefits of adopting this white box method for a program's source code.
- To assure the quality of the product you're delivering, it's vital to have good test coverage in software engineering. For designing test cases based on the logical path or flow of a program, basis path testing employs a white box testing technique.
- Basis path testing is similar to path testing in that it finds all of the code's possible executable paths. It is, however, more efficient in that it eliminates redundant tests and provides maximum coverage, and so takes less time.

Understanding A White Box Testing Method

- The White Box and Black box testing methods are used in the Box Testing approach to software testing. To summarize, the white box method examines software's internal structure, coding, and design, whereas the black box method examines it from the perspective of an end-user or an outsider.
- Clear Box testing, Structural testing, Open Box testing, Code-Based testing, Transparent Box testing, and Glass Box testing are all phrases used to describe white box testing. It got its name from the tester's ability to look past the software's exterior box and into its inner workings. The developer has access to the code.
- Black box testing, on the other hand, refers to the inability to see inside the inner workings of a system, allowing just the end-user experience to be evaluated. We'll concentrate on the former for the purposes of this post.
- The fundamental goal of white-box testing is to ensure that input and expected output flows through an application, as well as to improve usability, design, and security. It can be done at the development phase's system, integration, or unit levels.
- This method, when implemented, seeks to validate an application's process flow. Testing is done to find faults with the code using a set of preset inputs and expected outcomes. A problem occurs when a certain input fails to deliver the desired outcome.

Defining Path Testing

- It is vital to understand path testing before learning about basis path testing. The term Path refers to the flow of execution, or the order in which commands and conditions are executed in a specific order. In other words, it's the path a process takes to get from one place to another.

- To the already specified independent paths, an independent path adds at least one new process, command, or condition.
- In the source code of every software program, there are multiple entry and exit points. Path testing ensures that no problems are detected during the execution of processes via a program sequence by verifying these places in the source code. The method can be set up to execute all or a subset of the pats.
- The difficulty arises in complicated programs when the number of entrance and exit points to be tested adds up. This could take days or even weeks, depending on the quantity of source code to be tested. This is where basic path testing comes in handy because it decreases the number of test cases required.

What Is Basis Path Testing ?

- Basis path testing can achieve maximal path coverage with the fewest number of test cases by using a white box method.
- The smallest amount of test cases is used to execute every possible block of code in a program. It accomplishes this by determining the number of independent paths that must be tested so that the number of test cases necessary may be precisely defined, maximizing the coverage of each test case.
- The effectiveness of basis path testing is that it ensures complete branch coverage without the requirement to test all conceivable paths. As previously said, this can be time-consuming and expensive. Another testing method is branch coverage, which ensures that every branch extending from each decision point is examined at least once. This way, all of the code branches can be checked to ensure that none of them cause the application to behave strangely. As a result, base path testing is seen as a combination of path and branch testing techniques.

The following are some of the most important considerations for this method:

- Technique of white box testing
- Often used at the unit level by developers to test code in a software application.
- A method of structural testing for a program's source code.
- Aids in the evaluation of an application's internal workings.
- Calculates the number of tests to run using cyclomatic complexity.
- Defines separate test cases for each independent path to be run.
- When assessing source code, it ensures that all branches are covered.

☞ Steps for Carrying Out Testing

The steps for carrying out this testing technique are as follows :

- Creating a control flow diagram to determine the various program pathways

- Calculating the number of independent paths using a method called cyclomatic complexity, which we'll go over later.
- Define the set of testable base pathways.
- Create test cases to evaluate each path's software flow.

Cyclomatic Complexity

- Cyclomatic complexity is a software statistic and an important part of the foundation path testing procedure. Software metric is a numerical measurement of a software attribute's time, quality, size, and cost.
- In this scenario, cyclomatic complexity is used to assess the complexity of a program by finding all of the processes' independent routes.
- Thomas McCabe created the measure in 1976, based on a control flow model of a program. His concept presents a depiction of a program's control flow using a flow graph made up of nodes and edges. The processing jobs are represented by nodes, and the flow between them is controlled by edges. Independent pathways add a new process to the program flow, while nodes are the entry and exit points of processes in the program sequence. They have at least one edge that hasn't been traversed by any other path.
- A mathematical representation of the cyclomatic complexity of program code can be calculated as follows:

  $V(G) = E - N + 2$

  Where, E = number of edges; N = number of nodes; $V(G) = P + 1$

  Where, P = number of predicate nodes (nodes that contain conditions)

Once the number of paths or conditions has been calculated, the number of tests to be written is known. For example, 3 paths will mean that at least one test should be generated to cover each path.

The properties of cyclomatic complexity are as follows:

- V(G) is the highest number of independent paths shown in the graph
- V(G) is always greater than or equal to 1
- If V(G) is equal to 1 then G will have one path
- Ideally, minimize the complexity score to 10 – the higher the score, the more complex the code

Designing Test Cases

- Because basic path testing is a white box method for generating test cases that deals with the inner workings of the code, it's critical that the tester is familiar with the application's source code before creating test cases.

- Not only should the testers be familiar with the application's programming languages, but they should also be adept at secure coding techniques. Security is frequently a driving force for software testing, and the tester should be able to see any code that is vulnerable to malicious assault.
- Moving on to test cases, testers might create more code to check the application's source code for flow and structure. Writing and executing code to test the flow of processes is greatly easier when separate branches of code are isolated using basic path testing.
- Using the white box technique ensures that at least one statement is executed during testing. Each independent linear path within the program is tested resulting in a number of test cases equal to the program's cyclomatic complexity score.

☞ Advantages of Basis Path Testing

(1) Reduces the number of tests that are duplicated.

(2) It helps to validate other white-box procedures because it is one of the basic white-box testing methods.

(3) Test cases are focused on program logic.

(4) Analytical test case design is preferred over arbitrary test case design.

(5) For test cases involving the basis set, program statements will be performed at least once.

(4) Graph Matrices

- A **graph matrix** is a data structure that can assist in developing a tool for automation of path testing. Properties of graph matrices are fundamental for developing a test tool and hence graph matrices are very useful in understanding software testing concepts and theory.

GQ. What is a Graph Matrix ?

- A graph matrix is **a square matrix whose size represents the number of nodes in the control flow graph**. If you do not know what control flow graphs are, then read this article. Each row and column in the matrix identifies a node and the entries in the matrix represent the edges or links between these nodes. Conventionally, nodes are denoted by digits and edges are denoted by letters.

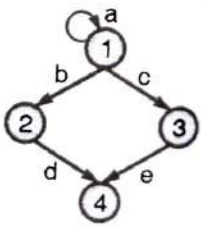


Fig. 2.3.1

Let's convert this control flow graph into a graph matrix. Since the graph has 4 nodes, so the graph matrix would have a dimension of 4 X 4. Matrix entries will be filled as follows :

- (1, 1) will be filled with 'a' as an edge exists from node 1 to node 1

- (1, 2) will be filled with 'b' as an edge exists from node 1 to node 2. It is important to note that (2, 1) will not be filled as the edge is unidirectional and not bidirectional
- (1, 3) will be filled with 'c' as edge c exists from node 1 to node 3
- (2, 4) will be filled with 'd' as edge exists from node 2 to node 4
- (3, 4) will be filled with 'e' as an edge exists from node 3 to node 4

The graph matrix formed is shown below:

	1	2	3	4
1	a	b	c	
2				d
3				e
4				

Connection Matrix

- A connection matrix is a matrix defined with edges weight. In simple form, when a connection exists between two nodes of control flow graph, then the edge weight is 1, otherwise, it is 0. However, 0 is not usually entered in the matrix cells to reduce the complexity.

  For example, if we represent the above control flow graph as a connection matrix, then the result would be :

	1	2	3	4
1	1	1	1	
2				1
3				1
4				

- As we can see, the weight of the edges is simply replaced by 1 and the cells which were empty before are left as it is, i.e., representing 0.
- A connection matrix is used to find the **cyclomatic complexity of the control graph**. Although there are three other methods to find the cyclomatic complexity but this method works well too.
- Following are the steps to compute the cyclomatic complexity:
  1. Count the number of 1s in each row and write it in the end of the row
  2. Subtract 1 from this count for each row (Ignore the row if its count is 0)
  3. Add the count of each row calculated previously

4. Add 1 to this total count
5. The final sum in Step 4 is the cyclomatic complexity of the control flow graph

Let's apply these steps to the graph above to compute the cyclomatic complexity.

	1	2	3	4	
1	1	1	1		3 – 1 = 2
2				1	1 – 1 = 0
3				1	1 – 1 = 0
4					Ignore
	Cyclomatic complexity = 2 + 0 + 0 + 1 = 3				

We can verify this value for cyclomatic complexity using other methods:

- **Method 1 : Cyclomatic complexity**

  = e - n + 2 * P

  Since here,

  e = 5

  n = 4

  and, P = 1

  Therefore, cyclomatic complexity,

  = 5 - 4 + 2 * 1

  = 3

- **Method-2 : Cyclomatic complexity**

  = d + P

  Here,

  d = 2

  and, P = 1

  Therefore, cyclomatic complexity,

  = 2 + 1

  = 3

- **Method 3 : Cyclomatic complexity**

  = number of regions in the graph

  Region 1: bounded by edges b, c, d, and e

  Region 2: bounded by edge a (in loop)

  Region 3: outside the graph

  Therefore, cyclomatic complexity,

  = 1 + 1 + 1

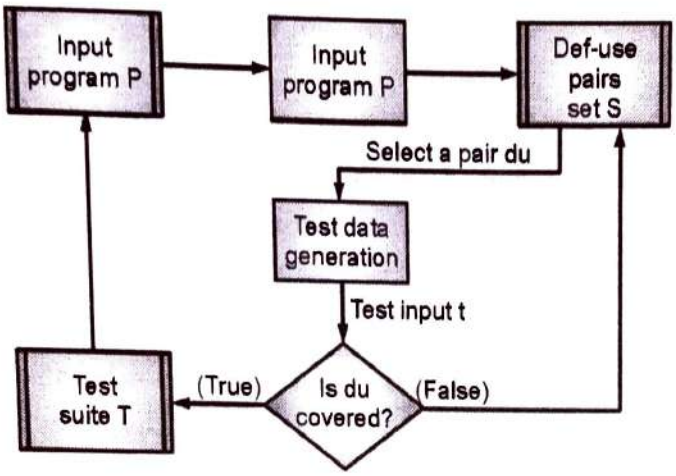
  = 3

2.3.2 Loop Testing, Data Flow Testing

UQ. Explain static Data Flow testing with example.

MU - May 16

- Data Flow Testing is a precise strategy of software testing that focuses on data variables and their values. It types use of the control flow graph.
- When it comes to classification Data flow testing will can be measured as a type of white box testing and organizational types of testing. It keeps a check at the data getting points by the variables and its usage points. It is done to cover the path testing and outlet testing gap.
- The process is directed to detect the bugs because of the improper usage of data variables or data values. For e.g. Initialization of data variables in software design code, etc.



(1B7)**Fig. 2.3.2 : Loop testing and Data flow testing**

GQ. What is Data flow Testing ?

- The programmer can complete frequent tests on data values and variables. This type of testing is mentioned to as data flow testing.
- It is completed at two abstract levels: static data flow challenging and dynamic data flow testing.
- The static data flow testing process contains analyzing the source code without affecting it.
- Static data flow testing revelations possible imperfections known as data flow difference.
- Dynamic data flow recognizes program paths from source code.

Let us understand this with the help of an example

1. read x;	
2. If(x>0)	(1, (2, t), x), (1, (2, f), x)
3. a= x-F1	(1, 3, x)
4. if(c<=0),	(1, (4, t), x), (1, (4, f), x)
5. if (x<1)	(1, (5, t), x), (1, (5, f), x)
6. x=x+1, (go to 5)	(1, 6, x)
else	
7. a=x+1	(1, 7, x)
8. print a;	(6,(5, f)x), (6,(5,t)x)
	(6, 6, x)
	(3, 8, a), (7, 8, a).

x= 1

Path – 1, 2, 3, 8

Output = 2

- If we consider **x = 1**, in step 1; x is allocated a value of 1 then we move to step 2 (since, x > 0 we will move to statement **3 (a = x+1)** and at end, it will go to statement 8 and print x = 2.
- For the second path, we assign x as 1

  Set x = – 1

  Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

Output = 2

- x is set as 1 then it goes to step 1 to allocate x as 1 and then moves to step 2 which is false as x is smaller than 0 (x > 0 and here x = – 1). It will then move to step 3 and then jump to step 4; as 4 is true (x <= 0 and their x is less than 0) it will jump on 5 (x < 1) which is true and it will move to step 6 **(x = x + 1)** and here x is improved by 1.

  So,

  x = – 1 + 1

  x = 0
- x become 0 and it goes to step 5 (x < 1), as it is true it will jump to step

  6 (x = x + 1)

  x = x + 1

  x = 0 + 1

  x = 1
- x is now 1 and jump to step 5 (x < 1) and now the condition is false and it will jump to step 7 (a = x + 1) and set a = 2 as x is 1. At the end the value of a is 2. And on step 8 we get the output as 2.

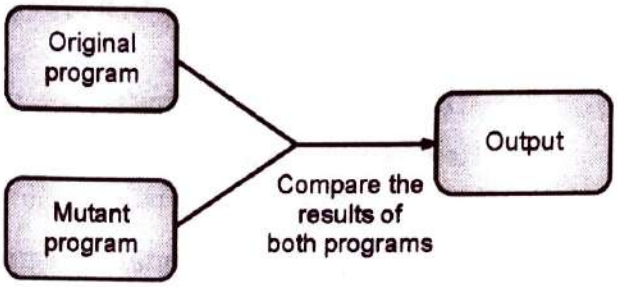
2.3.3 Mutation Testing

UQ. Explain Mutation testing with the help of example. **MU - Dec. 16**

- Mutation testing, also known as code mutation testing, is a form of **white box testing** in which testers change specific components of an application's source code to ensure a software test suite will be able to detect the changes.
- Transformation testing is related to alteration a program in small ways. It emphases to help the tester develop effective tests or locate faults in the test data used for the program.

History of Mutation Testing

- *Richard Lipton* proposed the alteration testing in 1971 for the first time.
- Though high cost condensed the use of mutation testing but now it is widely used for languages such as Java and XML.



(1B8)**Fig. 2.3.3 : Mutation testing**

Mutation Testing is a White Box Testing

- Mutation testing can be applied to design models, stipulations, databases, tests, and XML. It is a mechanical testing technique, which uses the structure of the code to guide the testing process.
- It can be described as the process of rewriting the source code in small ways in order to remove the redundancies in the source code.

Objective of Mutation Testing

The detached of mutation testing is:

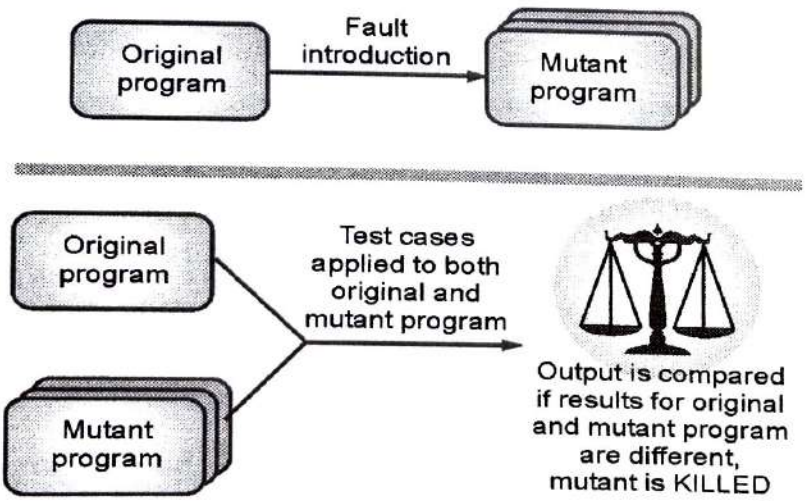
1. To classify pieces of code that is not tested appropriately.
2. To identify hidden defects that can't be detected consuming other testing methods.
3. To discover new types of errors or bugs.
4. To compute the mutation score.
5. To education error propagation and state infection in the program.
6. To measure the excellence of the test cases.

GQ. What is Mutation Testing? Clarify Mutation Testing Procedure?

GQ. Justify how mutation testing is operative to payment the quality of software with an instance

- **Mutation Testing** is a type of software testing in which positive statements of the source code are altered/mutated to form if the test suitcases are able to find errors in source code.
- The goal of Mutation Testing is confirming the quality of test cases in terms of strength that it should fail the mutated source code.
- The alterations made in the mutant program should be kept tremendously small that it does not affect the overall objective of the program.
- Mutation Testing is also called Fault based testing approach as it includes making a fault in the program and it is a type of White Box Testing which is mostly used for Unit Testing
  - What is Mutation Testing?
  - How to perform Mutation Testing?
  - How to Create Mutant Programs?
  - What to change in a Mutant Program?
  - Types of Mutation Testing
  - Mutation Score
  - Compensations of Mutation Testing
  - Disadvantages of Mutation Testing

How to execute Mutation Testing ?



(1B9)**Fig. 2.3.4 : Mutation Analysis**

Following are the steps to implement mutation testing (mutation analysis)

- **Step 1 :** Faults are presented into the source code of the program by creating many forms called mutants. Each mutant should comprise a single fault, and the goal is to cause the mutant version to fail which demonstrates the efficiency of the test cases.
- **Step 2 :** Test cases are practical to the original program and also to the mutant program. A Test Case should be satisfactory, and it is tweaked to notice liabilities in a program.
- **Step 3 :** Associate the results of an original and mutant program.
- **Step 4 :** If the original program and mutant programs produce the different output, then that the mutant is killed by the test case. Hence the test case is good sufficient to detect the change between the original and the mutant program.
- **Step 5 :** If the original program and mutant program produce the same output, Mutant is kept alive. In such suitcases, more operative test cases need to be formed that kill all mutants.

☞ How to Create Mutant Programs ?

A mutation is nobody but a single syntactic alteration that is made to the program statement. Each mutant program should change from the original program by one mutation.

Original Program	Mutant Program

If (x>y)
 	
Print "Hello"
 	
Else
 	
Print "Hi"
	If(x<y)
	
	Print "Hello"
	
	Else
	
	Print "Hi"

GQ. What to change in a Mutant Program ?

There are numerous methods that could be used to produce mutant programs. Let's look at them

Automation of Mutation Testing

Mutation testing is tremendously time overwhelming and complex to perform manually. To speed up the process, it is sensible to go for automation tools. Automation tools reduce the cost of testing as well.

☞ Types of Mutation Testing

In Software Engineering, Mutation testing could be essentially considered into 3 types– statement mutation, decision mutation, and value mutation.

1. **Statement Mutation :** Designer cut and pastes a part of a code of which the outcome may be a elimination of some lines
2. **Value Mutation :** Standards of primary parameters are adapted

   In this type of testing the values are different to detect errors in the program. Basically, a small value is changed to a larger value or a larger value is changed to a reduced value. In this testing essentially constants are changed.

Example

Initial Code:

```
int mod = 1000000007;
int a = 12345678;
int b = 98765432;
int c = (a + b) % mod;
```

Changed Code:

```
int mod = 1007;
int a = 12345678;
int b = 98765432;
int c = (a + b) % mod;
```

Decision Mutations : In decisions mutations are logical or arithmetic operators are altered to detect errors in the program.

Example

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Changed Code:

```
if(a > b)
 c = 10;
else
 c = 20;
```

Statement Mutations:

In statement mutations a statement is deleted or it is substitutes by some other statement.

Example

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Changed Code:

3. **Decision Mutation :** Control statements are to be altered

In statement mutations a statement is deleted or it is substitutes by some other statement.

Example

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Changed Code:

```
if(a < b)
 d = 10;
else
 d = 20;
```

☞ Rewards of Mutation Testing

1. It brings a good level of error detection in the program.
2. It discovers uncertainties in the source code.

☞ Disadvantages of Mutation Testing

1. It is highly costly and time overriding.
2. It is important for Black Box Testing.

☞ Mutation Score

- The mutation score is distinct as the percentage of killed mutants with the total number of mutants.

$$\text{Mutation Score} = \frac{\text{Killed Mutants}}{\text{Total number of Mutants}} * 100$$

- Test cases are mutation passable if the score is 100%. Experimental results hav shown that mutation testing is an effective approach for computing the suitability o the test cases. But, the main drawback is that the high cost of producing the mutant and performing each test case against that mutant program.

☞ Advantages of Mutation Testing

Subsequent are the advantages of Mutation Testing :

1. It is a powerful method to attain high coverage of the source program.
2. This testing is capable systematically testing the mutant program.
3. Mutation difficult brings a good level of error detection to the software develope
4. This method discovers uncertainties in the source code and has the capacity detect all the faults in the program.
5. Customers are helped from this testing by receiving a most dependable and sta system.

☞ Disadvantages of Mutation Testing

On the other side, the subsequent are the disadvantages of Mutant testing :

1. Mutation testing is tremendously costly and time overwhelming since there many mutant programs that need to be produced.
2. Since its time overwhelming, it's fair to say that this testing cannot be d without an automation tool.
3. Each mutation will have the same number of test cases than that of innovative program. So, a great number of mutant programs may need to tested in contradiction of the original test suite.
4. As this method includes source code changes, it is not at all applicable for B Box Testing.

2.3.4 Static Testing

UQ. What is static testing? Explain the types of static testing. **MU - May 17, May 1**

- **Static Testing** is a software testing technique which is used to check imperfectio software application without executing the code. Static testing is done to avoid er at an early stage of development as it is easier to identify the errors and solve errors.
- It also helps discovery errors that may not be found by Dynamic Testing. complement is Dynamic Testing which forms an application when the code is Refer to this tutorial for a thorough difference between static and dynamic testing

The two main kinds of static testing methods are

- **Manual examinations :** Manual inspections include analysis of code done manually, also known as **Reviews.**
- **Automated analysis using tools :** Automated analyses are essentially static analysis which is done using tools.
  - What is Static Testing?
  - What is Testing Review?
  - Why Static Testing?
  - What is Tested in Static Testing
  - How Static Testing is Performed
  - Static Testing Techniques
  - Tools used for Static Testing
  - Tips for Effective Static Testing Process

☞ What is Testing Review?

- A review in a Static Testing is a process or meeting directed to find the potential defects in the project of any program.
- Another meaning of review is that all the team members get to know about the progress of the project and infrequently the diversity of decisions may result in unresolved suggestions.
- Documents are straight examined by people and differences are sorted out.
- Reviews can further be classified into four parts :
  - Casual reviews
  - Walkthroughs
  - Technical review
  - Examinations

During the Review process four types of participants that take part in testing are:

- **Moderator :** Achieves entry check, follow up on rework, training team member, schedule the meeting.
- **Author :** Takes accountability for fixing the defect found and improves the quality of the document
- **Scribe :** It does the classification of the defect during a review and appears the review meeting

- **Reviewer :** Check material for faults and inspects
- **Manager :** Decide on the implementation of appraisals and guarantees the review process objectives are met.

Types of defects which can be easier to find during static testing are :

- Unconventionalities from values
- Non maintainable code
- Design defects
- Missing supplies
- Inconsistent border specifications

Usually, the defect exposed during static testing are due to security vulnerabilities, undeclared variables, boundary violations, syntax violations, inconsistent interface, etc.

2.3.5 V Model

UQ. What are the features of V testing model? Explain in detail. **MU - May 19**

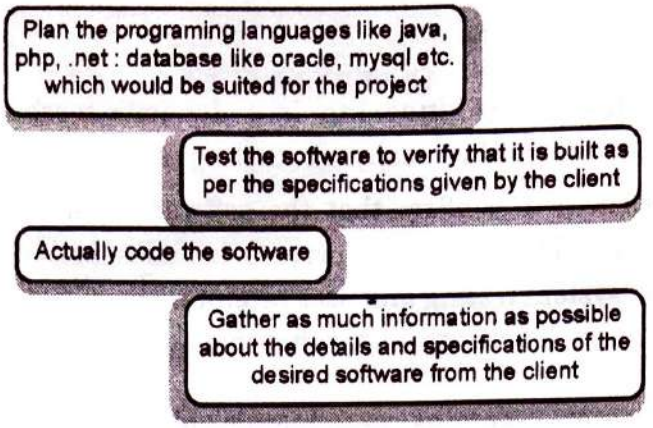
- **V Model** is a highly controlled SDLC model in which there is a testing phase parallel to each development phase.
- The V model is an postponement of the waterfall model in which testing is done on each stage parallel with expansion in a sequential way. It is known as the Validation or Verification Model.

Key Software Engineering Terms

- **SDLC :** SDLC is Software Development Life Cycle. It is the arrangement of activities carried out by Developers to design and develop high-quality software.
- **STLC :** STLC is Software Testing Life Cycle. It consists of a series of doings carried out by Testers organizationally to test your software product.
- **Waterfall Model :** Waterfall model is a consecutive model divided into dissimilar phases of software growth activity. Each stage is considered for execution the specific activity. Testing phase in waterfall model starts only after application of the system is done.

Example to Appreciate the V Model

- Suppose, you are assigned a task, to develop a custom software for a client. Now, nevertheless of your practical background, try and make an tasteful guess about the sequence of steps you will follow, to realize the task.



(1B10)**Fig. 2.3.5 : Example of V Model**

The correct sequence would be

Different phases of the Software Development Cycle	Activities performed in each stage
Requirement Meeting stage	Gather as much information as thinkable about the details & conditions of the anticipated software from the client. This is nothing but the Necessities gathering stage.
Design Stage	Plan the programming language like Java, PHP, .net; database like Oracle, MySQL, etc. Which would be suitable for the project, also some high level functions & architecture.
Build Stage	After the design stage, it is build stage, that is nothing but essentially code the software
Test Stage	Next, you test the software to verify that it is built as per the conditions are given by the client.
Deployment stage	Deploy the application in the individual setting
Maintenance stage	Once your system is ready to use, you may necessitate to change the code later on as per customer demand

2.4 VALIDATION ACTIVITIES : UNIT VALIDATION

GQ. What is Confirmation in Software Testing ?

- **Verification in Software Testing** is a process of examination documents, design, code, and program in order to check if the software has been built according to the supplies or not.
- The main goal of verification process is to confirm quality of software application, design, architecture etc. The confirmation process includes activities like reviews, walk through and review.

GQ. What is Validation in Software Testing ?

- **Validation in Software Engineering** is a dynamic mechanism of testing and validating if the software product really meets the exact wants of the customer or not.
- The process assistances to confirm that the software fulfils the wanted use in an suitable environment. The validation process includes activities like unit testing, integration testing, system testing and user receipt testing.

Key Difference

- Verification process contains examination of documents, design, code and program while Validation process contains testing and validation of the actual product.
- Verification does not include code execution while Validation involves code execution.
- Verification uses approaches like reviews, walkthroughs, examinations and desk checking while Validation uses approaches like black box testing, white box testing and non-functional testing.
- Verification forms whether the software confirms a requirement whereas Validation checks whether the software chances the necessities and expectations.
- Verification finds the bugs initial in the growth cycle whereas Validation finds the bugs that verification cannot catch.
- Equating validation and verification in software testing, Verification process boards on software architecture, design, database, etc. while Validation process targets the actual software product.
- Verification is done by the QA team while Validation is done by the participation of testing team with QA team.
- Likening Verification vs Validation testing, Verification process originates before validation whereas Validation process comes after verification.

2.4.1 Integration Testing : What is, Types, Top down and Bottom up Example

UQ. Explain Unit testing and Integration testing **MU - Dec. 19**

What is Integration Testing ?

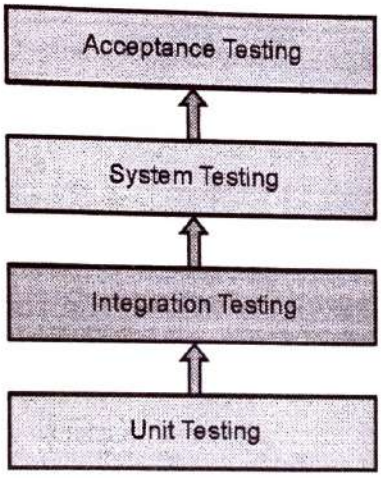
- **Integration Testing** is defined as a type of testing where software components are combined logically and tested as a group. A typical software project contains of multiple software modules, coded by dissimilar programmers.
- The resolution of this level of testing is to representation defects in the interaction between these software components when they are integrated Integration Testing

emphases on checking data message amongst these components. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.

- What is Integration Testing?
- Why do Integration Testing?
- Instance of Integration Test Case
- Approaches, Strategies, Methodologies of Integration Testing
- Big Bang Approach:
- Incremental Approach
- What is Stub and Driver?
- Bottom up Integration
- Top down Integration:
- Hybrid / Sandwich Integration
- How to do Integration Testing?
- Brief Account of Integration Test Plans:
- Entry and Exit Criteria of Addition Testing
- Best Performs / Strategies for Integration Testing

GQ. Why do we need integration testing ?

☞ Why do we need Integration Testing ?



(1B12)**Fig. C1 : Integration Testing**

Although each software module is unit tested, faults still exist for various reasons like

- A Module, in overall, is calculated by an different software developer whose sympathetic and programming logic may differ from other programmers.

- Addition Testing becomes required to verify the software modules work in unity
- At the time of module increase, there are wide likelihoods of change in necessities by the clients. These new requirements may not be unit tested and hence scheme integration Testing converts necessary.
- Lines of the software modules with the database could be mistaken
- External Hardware interfaces, if any, could be erroneous
- Insufficient exception handling could cause issues.

☞ Example of Integration Test Case

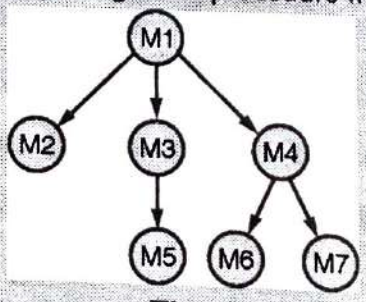
- Integration Test Case varies from other test cases in the sense it **focuses mostly on the interfaces & flow of data/information between the components**. Here priority is to be given for the **integrating links** rather than the unit functions which are previously tested.
- Example Integration Test Cases for the subsequent scenario : Request has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is unified logically.
- Here do not essence much on the Login Page testing as it's previously been done in Unit Testing. But check how it's linked to the Mail Box Page.
- Similarly Mail Box : Check its addition to the Delete Mails Module

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the boundary link between the Login and Mailbox module	Enter login identifications and click on the Login button	To be focused to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Designated email should appear in the Deleted/Trash folder

UQ. Explain various approaches in integration testing.

☞ Various Approaches in Integration Testing

UQ. Perform top-down and bottom up integration procedure from the following system hierarchy.



(1B11)**Fig. 2.4.1**

MU - May 19

- When the modules are under the process of development, the developers develop some interfaces and integrate the module with the help of those interfaces. Here integration is the process of assembling unit-tested modules. We need to test the following aspects, which have not been addressed previously while independently testing the modules:
- Interfaces: To ensure "interface integrity," the transfer of data between modules is tested. When data is passed to another module, by way of a call, there should not be any loss or corruption of data. The loss or corruption of data can happen due to mis-match or differences in the number or order of calling and receiving parameters.
- Module combinations may produce a different behavior due to combinations of data that are not exercised during unit testing. Global data structures, if used, may reveal errors due to unintended usage in some module.
- Integration Testing falls under the category of white box testing.

Following four approaches are generally adopted by the developers while integrating the modules.

To illustrate the methodology, let us consider the following arrangement of modules say M1, M2, M3 , M7:

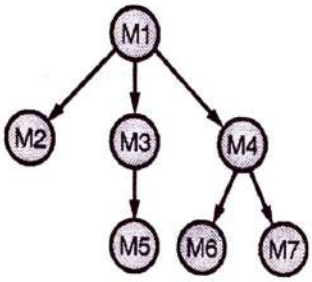


Fig. 2.4.2

(1) Top Down Approach (TDA)

- Involves development of all the topmost / parent modules beforehand followed by due integration with child modules.

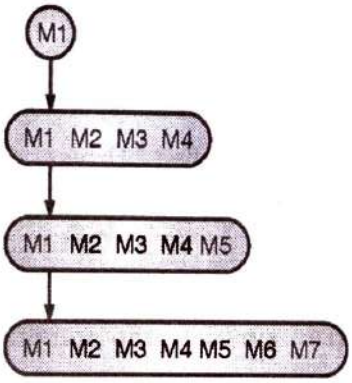


Fig. 2.4.3

- This approach uses a Program called "Stub". While integrating the modules with this Top Down Approach, if at any stage it is found that some mandatory module is missing, then in such an event that particular module is replaced with a temporary program known as "Stub".
- The testing starts with M1. To test M1 in isolation, communications to modules M2, M3 and M4 needs to be simulated by the tester by some means, since these modules may not be ready yet. To simulate responses of M2, M3 and M4 whenever they are to be invoked from M1, "stubs" are created.
- Simple applications may require stubs, which would simply return the control to their superior modules. More complex situation demand stubs to simulate a full range of responses, including parameter passing. Stubs may be individually created by the tester or they may be provided by a software testing harness, which is a sort of software specifically designed to provide a testing environment.
- In the above illustration, M1 would require stubs to simulate the activities of M2, M3 and M4. The integration of M3 would require a stub or stubs for M5 and M4 would require stubs for M6 and M7. Elementary modules i.e. the modules which do not call subordinates would not require any stubs.

☞ **Advantages of Top down Approach**

(a) This approach is advantageous if all major flaws are captured towards the top of the program

(b) Early skeletal program allows demonstrations and boosts the morale

☞ **Disadvantages of Top down Approach**

(a) Stub modules are essential

(b) Test conditions my be impossible, or very difficult, to create

(c) Observation of test output is more difficult, as only simulated values will be used initially. For the same reason, program correctness can be misleading

(2) Bottom Up Approach (BUA)

- Involves development of all the elementary modules or child modules beforehand followed by due integration with the corresponding parent modules.
- This approach uses a Program called "Driver". While integrating the modules with this Bottom Up Approach, if at any stage it is found that some mandatory module is missing, then in such an event that particular module is replaced with a temporary program known as "Driver".
- If M5 is ready, we need to simulate the activities of its superior, M3. Such a "driver" for M5 would simulate the invocation activities of M3. As with the stub, the complexity of a driver would depend upon the application under test.

- The driver would be responsible for invoking the module under test, it could be responsible for passing test data and it might be responsible for receiving output data. Here as well, the driving function can be provided through a testing harness or may be created by the tester as a program.

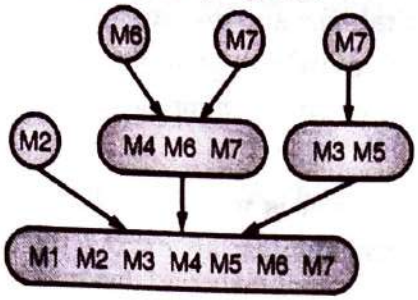


Fig. 2.4.4

- For the above-mentioned Bottom-Up example, drivers must be provided for modules M2, M5, M6, M7, M3 and M4. However there is no need for a driver for the topmost Module, M1.

Advantages of Bottom Up Approach

(a) This approach is advantageous if all major flaws are captured towards the bottom of the program

(b) Test conditions are easier to create

(c) Observations of test results are easier since "live" data is used from the beginning.

Disadvantages of Bottom Up Approach

(a) Driver modules are essential

(b) The program as an entity does not exist until the last module is added

(3) Big Bang Approach

- Once all the modules are ready after testing individually; the approach of integrating them finally at once is known as big bang approach.

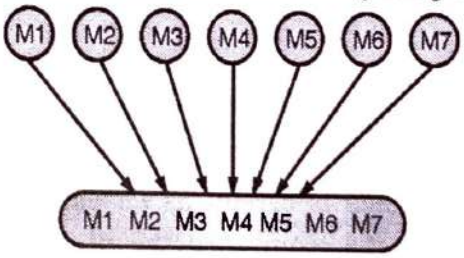


Fig. 2.4.5

- Though Big Bang approach seems to be advantageous when we construct independent module concurrently, this approach is quite challenging and risky as

we integrate all modules in a single step and test the resulting system. Locating interface errors, if any, becomes difficult here.

(4) Hybrid Approach

- To overcome the limitations and to exploit the advantages of Top-down and Bottom-up approaches, a hybrid approach in testing is used. As the name suggests, it is a mixture of the two approaches like Top Down approach as well as Bottom Up approach.
- In this approach the system is viewed as three layers consisting of the main target layer in the middle, another layer above the target layer, and the last layer below the target layer.
- The Top-Down approach is used in the topmost layer and Bottom-Up approach is used in the lowermost layer. The lowermost layer contains many general-purpose utility programs, which are helpful in verifying the correctness during the beginning of testing.
- Testing converges for the middle level target layers are selected on the basis of system characteristics and the structure of the code. The middle level target layer contains components using the utilities.

2.4.2 Methods, Policies, Practices of Addition Testing

Software Engineering describes variety of approaches to implement Integration testing, viz.

- Big Bang Approach
- Incremental Approach: which is further alienated into the following
  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach - Combination of Top Down and Bottom Up

Below are the dissimilar strategies, the way they are executed and their confines as well advantages.

Big Bang Testing

- **Big Bang Testing** is an Integration testing method in which all the components or modules are combined together at once and then tested as a unit.
- This mutual set of components is careful as an entity while testing. If all of the machineries in the unit are not accomplished, the integration process will not execute.

Advantage

1. Convenient for small systems.

Disadvantages

1. Fault Localization is problematic.
2. Given the sheer number of boundaries that need to be tested in this method, some interfaces link to be tested could be missed easily.
3. Since the Combination testing can originate only after "all" the modules are considered, the challenging team will have less time for implementation in the testing phase.
4. Since all components are tested at once, high risk critical components are not inaccessible and tested on priority. Peripheral modules which deal with user borders are also not remote and tested on importance.

Incremental Testing

- In the **Incremental Testing** method, testing is done by integrating two or more modules that are rationally related to each other and then tested for proper effective of the application.
- Then the other related modules are combined incrementally and the process remains until all the logically related modules are integrated and confirmed successfully.
- Incremental Method, in turn, is carried out by two different Methods :
  - Bottom Up
  - Top Down

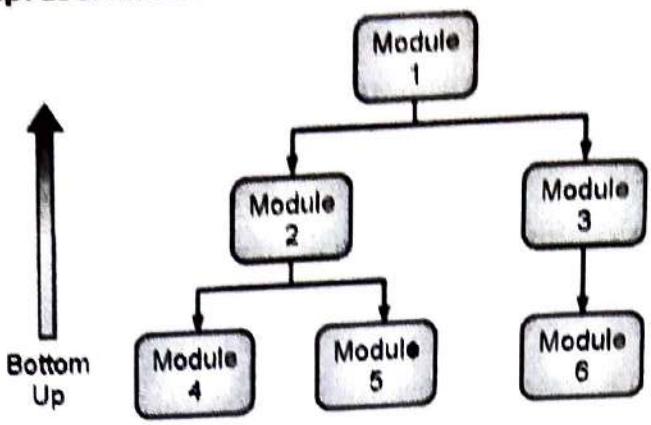
Stubs and Drivers

- **Stubs and Drivers** are the dummy programs in Integration testing used to simplify the software testing action.
- These packages act as alternates for the absent models in the testing. They do not implement the complete programming logic of the software module but they simulate data message with the calling module while testing.
  - **Stub** : Is called by the Module under Test.
  - **Driver** : Calls the Module to be tested.

2.4.3 Bottom-up Integration Testing

- **Bottom up Integration Testing** is a strategy in which the inferior level components are tested first.
- These tested components are then further used to simplify the testing of higher level modules.
- The process endures until all components at top equal are tested. Once the lower level components are tested and combined, then the next level of modules are formed.

Diagrammatic Representation



(1B13)Fig. 2.4.6 : Bottom up Integration Testing

Advantages

1. Fault localization is easier.
2. No time is missed waiting for all modules to be advanced unlike Big-bang approach

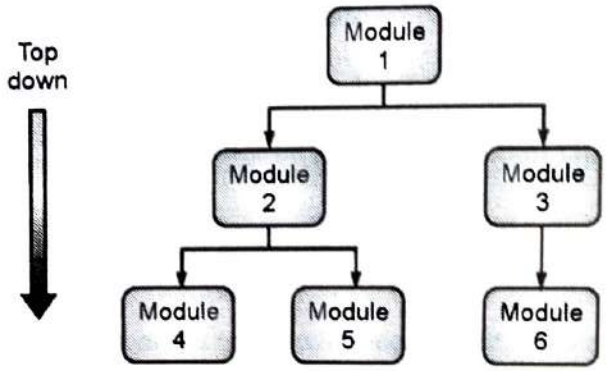
Disadvantages

1. Critical modules (at the top level of software architecture) which control the flow of presentation are tested last and may be prone to defects.
2. An early prototype is not possible

2.4.4 Top-down Integration Testing

- **Top Down Integration Testing** is a method in which integration testing takes place from top to bottom following the control flow of software system.
- The higher level modules are tested first and then lower level components are tested and combined in order to check the software functionality. Stubs are used for testing if some modules are not ready.

Illustrative Illustration



(1B14)Fig. 2.4.7 : Top down Integration Testing

Advantages

1. Fault Localization is informal.
2. Opportunity to attain an primary prototype.
3. Critical Modules are tested on priority; major strategy flaws could be found and fixed first.

Disadvantages

1. Needs many Stubs.
2. Components at a lower level are tested imperfectly.

2.4.5 Integration, Function, System

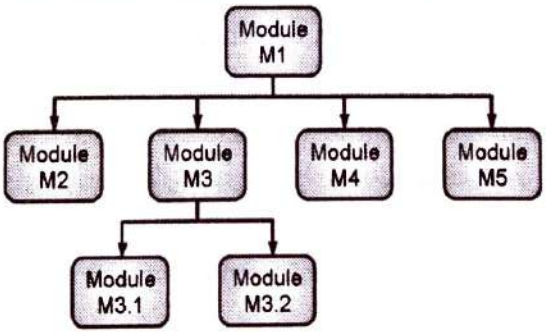
UQ. Explain different modules of incremental Integration Testing Methods. ? **MU - Dec. 19**

What is Incremental Testing ?

- Incremental Testing, also known as Incremental Integration Testing, is one of the methods of Integration Testing and integrates its fundamental concepts.
- It is like a test which associations Module and Integration testing strategy.
- In this testing, we test each module exclusively in unit testing phase, and then elements are integrated incrementally and tested to confirm smooth interface and interaction between modules.
- In this method, every module is mutual incrementally, i.e., one by one till all modules or components are added logically to make the compulsory application, instead of integrating the whole system at once and then execution testing on the end product.
- Integrated modules are tested as a group to confirm successful integration and data flow between modules.
- As in integration difficult, the primary focus of exploit this testing is to check interface, combined links, and flow of information between components.
- This process is recurrent till the modules are collective and tested successfully.

Example : Let's appreciate this concept with an example:

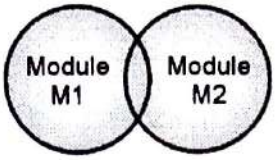
System or software application comprises of following Modules:



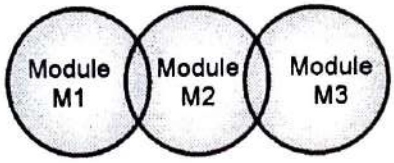
(1B15)**Fig. 2.4.8 : Incremental Testing**

2.4.6 Incremental Integration Testing Approach

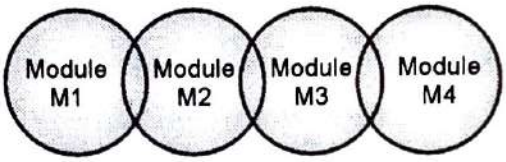
- Each Module i.e. M1, M2, M3, etc. are tested independently as part of unit testing.
- Modules are combined incrementally i.e. one by one and tested for effective interaction.



(a) Module M1 andM2



(b) Module M1, M2 and M3



(c) Module M1, M2, M3 M4

(1B16)Fig. 2.4.9 : Module M1, M2, M3, M4

- In Fig. 2.4.9(a), Module M1 & Module M2 are mutual and tested
- In Fig. 2.4.9(b), Module M3 is added and tested
- In Fig. 2.4.9(c), Module M4 is added and testing is done to make sure all works together successfully
- Rest of the Modules are also added incrementally at each step and tested for successful integration

2.4.7 Objective of Incremental Test

- To ensure that different modules work together successfully after integration
- Identify the defects earlier and in each phase. This gives developers an edge to identify where the problem is. Like if testing after M1 and M2 are integrated is successful but when M3 is added, the test fails; this will help the developer in segregating the issue
- Issues can be fixed in early phase without much rework and in less cost

Incremental Integration Testing Methodologies

- Before we start with this topic, I will like to give a brief intro about **Stubs and drivers** since we will be using these terms often.
- Stubs and drivers are pseudo code or dummy code used in Integration or component testing when one or more modules are not developed but are required to test some other module.
- **Stubs** are used in Top-down testing approach and are known as "called programs". Stubs help simulate the interface between lower lever modules which are not available or developed.
- **Drivers** are used in Bottom-up testing approach and are known as "calling programs". Drivers help simulate the interface between top level modules which are not developed or available.
- A question that might occur to some of us is why not wait till all the application modules are developed instead of using stub/driver before starting testing?
- The simple answer is that it increases the project execution time since testers will be sitting idle till all the modules are developed. Also, this makes the root analysis of defect difficult. This type of testing approach is known as Big-Bang Integration testing.

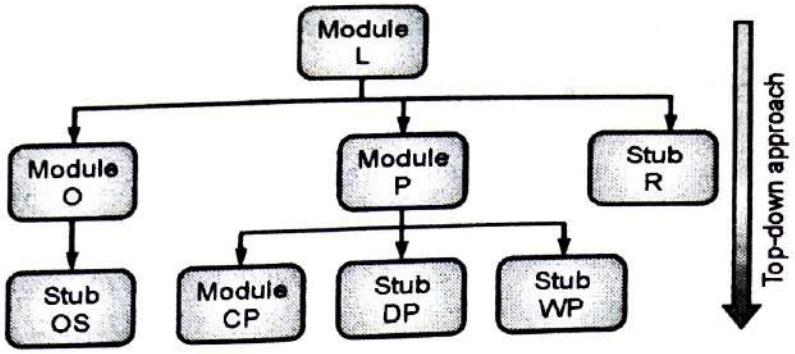
Now that we have covered Stubs and drivers, let's move on to **different methodologies of Incremental Testing :**

Top Down

- As the name suggests, testing takes place from top to bottom, i.e., from the central module to sub module. Modules framing the top layer of application are tested first.
- This approach follows the structural flow of the application under testing. Unavailable or not developed modules or components are substituted by stubs.

Let's understand this with an example :

- **Module :** Website Login aka L
- **Module :** Order aka O
  - Module Order Summary aka OS (Not yet developed)
- **Module :** Payment aka P
  - Module Cash Payment aka CP
  - *Module Debit / Credit Payment aka DP (Not yet developed)*
  - Module Wallet Payment aka WP (Not yet developed)
- **Module :** Reporting aka R (Not yet developed)



(1B17)**Fig. 2.4.10 : Top down Incremental Integration Testing Approach**

Following test cases will be derived:

- **Test Case1 :** Module L and Module O will be integrated and tested
- **Test Case2 :** Module L, O and P will be integrated and tested
- **Test Case3 :** Module L, O, P and R will be integrated and tested.

  And so on other test cases are derived.

2.4.8 Object Oriented Testing Issues

UQ. What are Issues in Object Oriented Testing ?

MU - Dec. 16, Dec. 18, May 16, May 17, May 18

- Testing in an OO context must address the basics of testing a base class and the code that uses the base class. Factors that affect this testing are inheritance and dynamic binding.
- Therefore, some systems are harder to test (e.g., systems with inheritance of implementations harder than inheritance of interfaces) and OO constructs such as inheritance and dynamic binding may serve to hide faults
- Use the OO Metrics techniques to assess the design and identify areas that need special attention in testing for adequate coverage. For example, depth of tree complicates testing and number of methods that have to be tested in a class and a lack of cohesion in methods means more tests to ensure there are no side effects among the disjoint methods.

OO Testing Strategies

While there are efforts underway to develop more automated testing processes from test models of the object model characteristics (for example states, data flows, or associations), testing is still based on the creation of test cases and test data by team members using a structural (White Box Testing) and/or a functional (See Black Box Testing) strategy.

2.4.9 Acceptance Testing

UQ. Acceptance testing. MU - Dec. 16, May 18

GQ. Differentiate system testing and acceptance testing.

System Testing

- System Testing is done to check whether the software or product meets the specified requirements or not. It is done by both testers and developers.
- It contains the Testing System testing, Integration Testing. It is done through more positive and negative test cases.

Acceptance Testing

- Acceptance Testing is done after the system testing. It is used to check whether the software meets the customer requirements or not.
- Acceptance testing is used by testers, stakeholders as well as clients. It includes only Functional Testing and it contain two testing Alpha Testing and Beta Testing.
- Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not. **Standard Definition of Acceptance Testing :**
- It is a formal testing according to user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers or other authorized entities to determine whether to accept the system or not.

Difference between System Testing and Acceptance Testing

Sr. No	System Testing	Acceptance Testing
1.	System testing is done to check whether the software or product meets the specified requirements or not.	Acceptance testing is the type of testing which is used to check whether the software meets the customer requirements or not.
2.	System testing is used by developers as well as testers.	Acceptance testing is used by testers, stakeholders as well as clients.
3.	System Testing is both functional and non-functional testing.	Acceptance testing is only functional testing.
4.	System Testing is the constitute of System and integration testing.	Acceptance testing is the constitute of alpha and beta testing.

Sr. No	System Testing	Acceptance Testing
5.	System testing is done before the Acceptance testing.	Acceptance testing is done after the System testing.
6.	System testing is the constitute of positive as well as negative test cases.	Acceptance Testing is the constitute of positive test cases.
7.	In system testing, system is checked for dummy inputs.	In acceptance testing, system is checked for random inputs.

2.4.10 Alpha and Beta Testing

Alpha testing

- Alpha testing is a user acceptance testing that is done by in house testers before the application goes live, objective is to perform a final round of testing and identify every type of issue which may have missed in previous rounds of testing.
- It is done by the testing team and issues are fixed immediately. The black box & white box testing approach is performed while performing Alpha testing.

Beta Testing

- What is Beta Testing? Beta testing is an opportunity for real users to use a product in a production environment to uncover any bugs or issues before a general release.
- Beta testing is the final round of testing before releasing a product to a wide audience.

UQ. Explain entry and exit criteria for Alpha and Beta Testing. **MU - May 16, May 18**

- Alpha testing is performed by testers who are usually internal employees of the organization.
- Beta testing is performed by clients who are not part of the organization.
- Alpha testing is performed at developer's site.
- Beta testing is performed at end-user of the product.

Entry criteria for alpha testing

1. All features are complete/testable (no urgent bugs).
2. High bugs on primary platforms are fixed/verified.
3. 50% of minimum bugs on primary platforms are fixed/verified.
4. All the features have been tested on primary platforms.
5. Performance has been measured/compared with the previous releases (user function).
6. Usability testing and feedback (ongoing).
7. Aloha sites are ready for installation.

Exit criteria for alpha testing

1. Get responses/feedback from the customers.
2. Create a report of any serious bugs.
3. Notify bug-fixing issues to the developers.

Entry criteria for beta testing

1. Positive responses from the alpha sites.
2. Customer bugs in the alpha testing have been addressed.
3. There are no such fatal errors that can affect the functionality of the software.
4. Secondary platform compatibility testing is complete.
5. Regression tests corresponding to the hug fixes have been done.
6. Beta sites are ready for installation.

Exit criteria for beta testing

1. Get responses/feedback from the beta testers.
2. Create a report of all serious bugs.
3. Notify bug-fixing issues to the developers.

UQ. How Alpha testing differs from Beta testing ? **MU - May 16, May 18**

2.4.11 Alpha Vs Beta Testing

Sr. No.	Alpha Testing	Beta Testing
1.	First stage of user acceptance testing.	Second stage of user acceptance testing.
2.	Objective is to make sure that application is working as expected, all functional and performance issues are closed, and application is ready for Beta testing.	To validate that application is satisfying customer needs and requirements completely, customer accept the system by validating all the business scenarios with approval that now system is ready to be launched to its real end users.
3.	It is performed by In-house testing team.	It is performed by end users.
4.	It is done in the premises where entire team is involved [test and development], it is called lab testing.	It is done at customer premises where no one from development and testing team is present, it is called real time testing.

Sr. No.	Alpha Testing	Beta Testing
5.	Alpha Testing requires long execution cycle depends on what and how many issues are uncovered.	Beta Testing requires only few weeks of execution.
6.	It is done after System testing, when testing team has completed functional, performance and compatibility testing, and system testing is completed, Alpha testing is performed. If any issue is found, on the basis of issue severity issue is fixed and again retesting and regression testing is done. Or Application goes with Open known Issues for Beta testing.	Beta testing is done after Alpha testing is done, with a testing sign off, when all major issues are closed and system is in ready state for use.
7.	It requires test environment.	It doesn't require test environment.
8.	Both White box and Black box testing involves in this.	Mainly Black box testing involves in this.
9.	Major issues are fixed immediately in Alpha testing.	Issues are collected from the end users and fix them later in Beta testing.
10.	Testing, retesting and regression testing is done, means multiple rounds of testing is performed to find issues.	Beta testing is the final testing and it is done on final tested environment and it is done only once.
11.	While Alpha testing main testable feature is in-depth functionality, compatibility, performance and usability. Here security and reliability is not a major testing concern.	While Beta testing main high level business scenarios, application reliability, disaster recovery, application security and usability is the main testing concern, all stakeholders should be satisfied with the application functionality and quality.
12.	If any high severity issue comes while Alpha testing, cost to fixing is relatively less then Beta testing phase.	If any high severe issue comes in Beta testing then cost of fixing is very high, plus company creditability is also at risk.
13.	System Testing is done before Alpha Testing.	Alpha Testing is done before Beta Testing.

Sr. No.	Alpha Testing	Beta Testing
14.	Beta testing is done after Alpha testing.	Product is released to the public after Beta Testing.
15.	Functionality, usability are tested in alpha testing. Reliability and Security testing are not performed in-depth during alpha testing.	Functionality, Usability, Reliability, Robustness, Security testing are performed during beta testing.
16.	The build released for Alpha Testing is called Alpha Release.	The build released for Beta Testing is called Beta Release.
17.	It evaluates the quality of the product. It answers "Does the product work?"	It evaluates the customer satisfaction. It answers "Do customers like the product?"
18.	It ensures whether the application is ready of beta testing or not.	It ensures whether the application is ready for the production launch.
19.	Goal is to find bugs.	Goal is to collect feedback from beta testers and evaluate them.
20.	Stakeholders for this testing are In-house developers, Quality Assurance Team.	Stakeholders for this testing are Product Management, Quality Management, and User Experience teams.

2.5 REGRESSION TESTING

UQ. Comment on regression testing process. MU - May 18

UQ. Regression testing produces quality software? Justify with reasons. MU - May 19

2.5.1 Progressive vs. Regressive, Regression Testing Produces Quality Software

GQ. What is Regression Testing ?

GQ. Define problem, Regression testing techniques, Objectives of regression testing.

- **Regression Testing** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

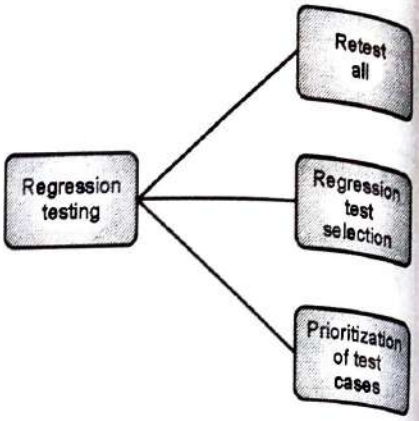
Regression testability, Regression testing types

Need of Regression Testing

- The **Need of Regression Testing** mainly arises whenever there is requirement to change the code and we need to test whether the modified code affects the other part of software application or not.
- Moreover, regression testing is needed, when a new feature is added to the software application and for defect fixing as well as performance issue fixing.

How to do Regression Testing

- In order **to do Regression testing** process, we need to first debug the code to identify the bugs. Once the bugs are identified, required changes are made to fix it, then the regression testing is done by selecting relevant test cases from the test suite that covers both modified and affected parts of the code.
- Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression Testing can be carried out using the following techniques:



(1B18)**Fig. 2.5.1 : Regression Testing**

Retest All

This is one of the methods for Regression Testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

Regression Test Selection is a technique in which some selected test cases from test suite are executed to test whether the modified code affects the software application or not. Test cases are categorized into two parts, reusable test cases which can be used in further regression cycles and obsolete test cases which cannot be used in succeeding cycles.

Prioritization of Test Cases

Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Selecting test cases for regression testing

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the Test Case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

UQ. Compare progressive and regressive testing. **MU - May 18**

- Progression testing focuses on new functionality and proving that it works as per the requirements. Whereas regression testing focuses on proving that existing functions of the application are not broken from the addition of new code.
- During testing, regression, and progression testing have been critical. Especially when you are dealing with systems that have serious impacts on customers.

What is progression testing ?

- Is reality it's another word for saying functional testing. Some people also refer to this as incremental testing, depending on what type of testing model you're using.
- Essentially, it's testing new functionality in a methodical approach. To make sure that the new functionality matches the desired requirements.
- I say sometimes referred to as the "incremental model" because it depends on what model of testing is being used. For example, you could be using the V-model.
- However, whatever your model of choice is, progression testing still comes down to testing new functionality against your requirements.

What is regression testing ?

- Regression testing is different from progression testing. Ultimately, with regression testing, we are making sure that we have not broken anything when we introduce new code.

Why do we need regression testing ?

- Regression testing is needed because mistakes or errors in code can be expensive to clean up post-go-live.

- Typically you'd need to use regression techniques for the following situations:
  - If you have a new defect fix.
  - New functionality (Such as the calculator example above).
  - Performance improvement.
  - Routine maintenance release.

All of these potential changes can provide a risk of damaging existing functionality which can have adverse cost effects to the business if they are not tested.

2.5.2 Regression Testing Tools

UQ. Explain the Regression testing types. **MU - Dec. 16, Dec. 17, May 17**

- If your software undergoes frequent changes, regression testing costs will escalate. In such cases, Manual execution of test cases increases test execution time as well as costs.
- Automation of regression test cases is the smart choice in such cases. The extent of automation depends on the number of test cases that remain re-usable for successive regression cycles.

Following are the most important tools used for both functional and regression testing in software engineering :

- **Selenium** : This is an open source tool used for automating web applications. Selenium can be used for browser-based regression testing.
- **Quick Test Professional (QTP) :** HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses VBScript language for automation. It is a Data-driven, Keyword based tool.
- **Rational Functional Tester (RFT) :** IBM's rational functional tester is a Java tool used to automate the test cases of software applications. This is primarily used for automating regression test cases and it also integrates with Rational Test Manager.
- **Regression Testing and Configuration Management :** Configuration Management during Regression Testing becomes imperative in Agile Environments where a code is being continuously modified.
- To ensure effective regression tests, observe the following :
  - Code being regression tested should be under a configuration management tool
  - No changes must be allowed to code, during the regression test phase. Regression test code must be kept immune to developer changes.
  - The database used for regression testing must be isolated. No database changes must be allowed.

2.5.3 Difference between Re-Testing and Regression Testing

- Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed, Defect needs to be re-opened. If fixed, Defect is closed.
- Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.
- Also, check out the complete list of differences over here.

Challenges in Regression Testing

(1B19)**Fig. 2.5.2 : Regression Testing**

Following are the major testing problems for doing regression testing :

- With successive regression runs, test suites become fairly large. Due to time and budget constraints, the entire regression test suite cannot be executed
- Minimizing the test suite while achieving maximum Test coverage remains a challenge
- Determination of frequency of Regression Tests, i.e., after every modification or every build update or after a bunch of bug fixes, is a challenge.

2.5.4 Progressive Testing

- **Progressive Testing** is also known as Incremental Testing. In Software Testing Incremental Testing refers to test modules one after another. When in an application parent-child modules are tested related modules to it needs to be tested first.
- Let's understand Progressive Testing/Increment Testing in more elaborately way by going little bit deep into it. This Increment Testing is considered as sub testing technique which comes under Integration Testing. Actually this testing acts as an approach/strategy to perform integration testing on software product rather as direct testing activities.
- After completion of Unit Testing over each individual components of the software, then the Integration Testing is performed to ensure proper interface and interaction between components of system. Incremental testing or Progressive testing is treated as partial phase of Integration testing.
- First it performs Integration testing on standalone components thereafter it goes on integrating components and performs integration testing over them accordingly.

- As the components are integrated in an incremental manner that's why also it is termed as Incremental Testing.

2.6 SELF LEARNING TOPICS

(1) Select the test cases (positive and negative scenarios) for the selected system and

(2) Design Test cases for the system using any two studied testing techniques.

What is Software Testing Technique?

Software Testing Techniques help you design better test cases. Since exhaustive testing is not possible; Manual Testing Techniques help reduce the number of test cases to be executed while increasing test coverage. They help identify test conditions that are otherwise difficult to recognize.

In this tutorial, you will learn 5 important software testing techniques :

(1) Boundary Value Analysis (BVA)

- Boundary value analysis is based on testing at the boundaries between partitions. It includes maximum, minimum, inside or outside boundaries, typical values and error values.
- It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the center. It is also known as BVA and gives a selection of test cases which exercise bounding values.
- This black box testing technique complements equivalence partitioning. This software testing technique base on the principle that, if a system works well for these particular values then it will work perfectly well for all values which comes between the two boundary values.

Guidelines for Boundary Value analysis

(1) If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.

(2) If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.

(3) Apply guidelines 1 and 2 to output conditions. It gives an output which reflects the minimum and the maximum values expected. It also tests the below or above values.

Example :

- Input condition is valid between 1 to 10
- Boundary values 0,1,2 and 9,10,11

Equivalence Class Partitioning

- Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same. This software testing method divides the input domain of a program into classes of data from which test cases should be designed.
- The concept behind this technique is that test case of a representative value of each class is equal to a test of any other value of the same class. It allows you to Identify valid as well as invalid equivalence classes.

Example :

- Input conditions are valid between 1 to 10 and 20 to 30
- Hence there are five equivalence classes
  (1) --- to 0 (invalid)
  (2) 1 to 10 (valid)
  (3) 11 to 19 (invalid)
  (4) 20 to 30 (valid)
  (5) 31 to --- (invalid)

You select values from each class, i.e.,

– 2, 3, 15, 25, 45

(2) Decision Table Based Testing

- A decision table is also known as to Cause-Effect table. This software testing technique is used for functions which respond to a combination of inputs or events. For example, a submit button should be enabled if the user has entered all required fields.
- The first task is to identify functionalities where the output depends on a combination of inputs. If there are large input set of combinations, then divide it into smaller subsets which are helpful for managing a decision table.
- For every function, you need to create a table and list down all types of combinations of inputs and its respective outputs. This helps to identify a condition that is overlooked by the tester.

Following are steps to create a decision table:

- Enlist the inputs in rows
- Enter all the rules in the column
- Fill the table with the different combination of inputs
- In the last row, note down the output against the input combination.

Example : A submit button in a contact form is enabled only when all the inputs are entered by the end user.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Input								
Name	F	T	F	T	F	T	F	T
Email	F	F	T	T	F	F	T	T
Message	F	F	F	F	T	T	T	T
Output								
Submit	F	F	F	F	F	F	F	T

(3) State Transition

In State Transition technique changes in input conditions change the state of the Application Under Test (AUT). This testing technique allows the tester to test the behavior of an AUT. The tester can perform this action by entering various input conditions in a sequence. In State transition technique, the testing team provides positive as well as negative input test values for evaluating the system behavior.

☞ Guideline for State Transition

(1) State transition should be used when a testing team is testing the application for a limited set of input values.

(2) The technique should be used when the testing team wants to test sequence of events which happen in the application under test.

Example

- In the following example, if the user enters a valid password in any of the first three attempts the user will be able to log in successfully.
- If the user enters the invalid password in the first or second try, the user will be prompted to re-enter the password. When the user enters password incorrectly 3rd time, the action has taken, and the account will be blocked.

State transition diagram

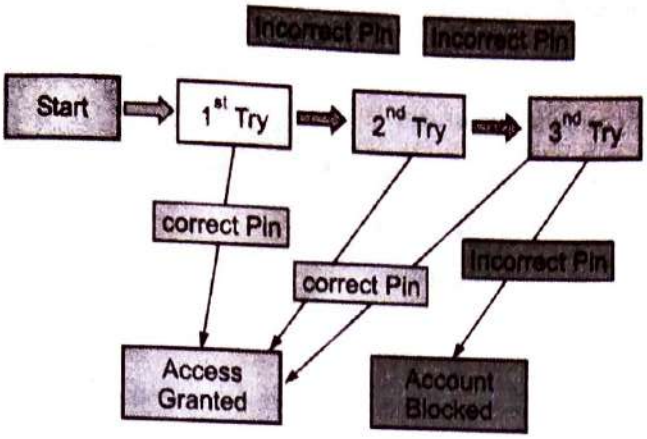


Fig. 2.6.1

In this diagram when the user gives the correct PIN number, he or she is moved to Access granted state. Following Table is created based on the diagram above-

State Transition Table

	Correct PIN	Incorrect PIN
S1) Start	S5	S2
S2) 1st attempt	S5	S3
S3) 2nd attempt	S5	S4
S4) 3rd attempt	S5	S6
S5) Access Granted	-	-
S6) Account blocked	-	-

In the above-given table when the user enters the correct PIN, the state is transitioned to Access granted. And if the user enters an incorrect password, he or she is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

(4) Error Guessing

- **Error Guessing** is a software testing technique based on guessing the error which can prevail in the code. The technique is heavily based on the experience where the test analysts use their experience to guess the problematic part of the testing application. Hence, the test analysts must be skilled and experienced for better error guessing.
- The technique counts a list of possible errors or error-prone situations. Then tester writes a test case to expose those errors. To design test cases based on this software testing technique, the analyst can use the past experiences to identify the conditions.

☞ Guidelines for Error Guessing

(1) The test should use the previous experience of testing similar applications

(2) Understanding of the system under test

(3) Knowledge of typical implementation errors

(4) Remember previously troubled areas

(5) Evaluate Historical data and Test results

☞ Conclusion

- Software testing Techniques allow you to design better cases. There are five primarily used techniques.
- Boundary value analysis is testing at the boundaries between partitions.
- Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same.
- Decision Table software testing technique is used for functions which respond to a combination of inputs or events.
- In State Transition technique changes in input conditions change the state of the Application Under Test (AUT)
- Error guessing is a software testing technique which is based on guessing the error which can prevail in the code.

Chapter Ends...

□□□

3.1 TEST MANAGEMENT : TEST ORGANIZATION

UQ. What are Key elements of Test Management ? **MU - Dec. 19**

There are a lot of test management tools out there. Some of them are made good, some of them are made great. Here are 6 key features that constitute a *great* piece of test management software:

1. Attractive user interface and easy-to-use design
2. Allow for multiple projects and user permissions
3. Traceability
4. Facilitates scheduling and organization
5. Monitoring and metrics
6. Flexibility

1. Attractive User Interface And Easy-To-Use Design

- If you're creating an application, you already know the importance of a clean, attractive, and intuitive user interface. It just makes everything more pleasant, right ?
- Your team members and devs will appreciate being able to glance at the UX and know exactly where everything is, how to access popular functions, and where their most needed data is stored. Complexity does not equate to a smarter or more advanced platform. Simple is beautiful.

2. Allow For Multiple Projets And User Permissions

- As you'll know, good software development requires teamwork from a range of individuals, including the product owner, project management professionals, product manager, testers, and coders, and all of whom need access to different information depending on their status. Some stakeholders will only require read-only access for reporting and analysis, while others will need full admin access to everything in the platform's backend.
- The capability to host multiple projects should be a given, too, even if it comes at an additional cost. **Scalability is key** when it comes to finding the right test management tool. You never want to be held back by simple product limitations.

3. Traceability

- With testing software, being able to trace every piece of the work and every change that is made is invaluable. Test management tools should support tracking and traceability, ensuring team accountability while also amassing useful data that you can refer back to later.

- Good test case management tools let you follow every change, from the very first test case through all the repeated cycles, and easily tell you who is responsible for each change. Not only does this let everyone on the project know where they stand, it also provides accountability for every person involved.

4. Facilitates Scheduling And Organization

- Planning and scheduling are crucial aspects of efficient software testing. One of the most important tasks in software testing management is figuring out how long the project will take, and you can't do this without effective scheduling, planning, requirements management, and organization.
- Planning includes defining the scope, approach, and resources needed to complete the project, as well as creating a schedule for the work needed. Without a good test plan, the project is unlikely to proceed smoothly and this becomes even more obvious with larger scale projects.
- Therefore, great test management tools should enable you to organize. This means figuring out how the testing and development teams are going to execute test plans and test suites. Test management tools should help with every part of this process, ensuring that the testing itself runs smoothly and helping with planning and management issues as they arise.

5. Monitoring And Metrics

- A test management solution should also help with monitoring and metrics. These are important parts of the testing process that ensure the project is running as it should, on time and on budget. The best test management tools will allow you to configure exactly what you are tracking and what the data visualizations look like.
- Test management software helps you define the project's goal and keep on track as it progresses. It compares the actual real-time progress with planned expectations, and records and reports any problems.

6. Flexibility

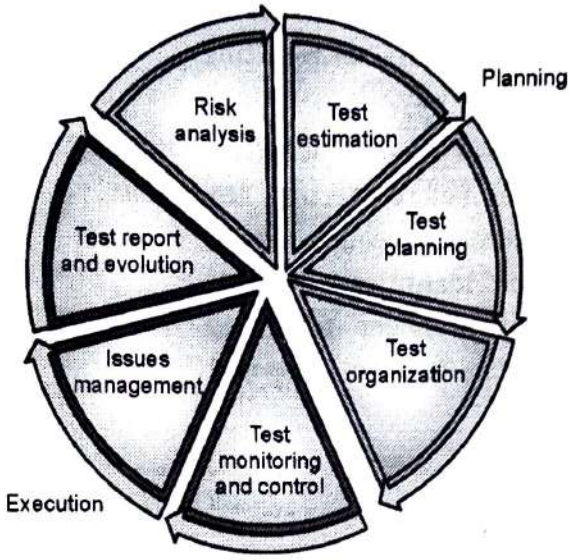
- Something every test automation tool needs is flexibility. Every project is different and should be treated as such. Test management software that combines transparency, traceability, and flexibility in an easy-to-use, clean, and attractive interface is the ultimate goal.
- Flexibility is especially important if your team works on a variety of projects with different methodologies, such as Agile testing, DevOps, or Scrum, and you need something that works for everyone. When you can't find a single testing tool that ticks all the boxes, using more than one tool is always an option.

Test Management

- **Test Management** is a process of managing the testing activities in order to ensure high quality and high-end testing of the software application.
- The method consists of organizing, controlling, ensuring traceability and visibility of the testing process in order to deliver the high quality software application. It ensures that the software testing process runs as expected.

Test Management Phases

- This topic **briefly** introduces Test Management Process and shows you an **overview** of Test Management Phases.
- You will learn more details about each Test Management Phases in the next articles.



(1C1)**Fig. 3.1.1 : Test Management Phases**

Test Management Process

- **Test Management Process** is a procedure of managing the software testing activities from start to the end.
- The test management process provides planning, controlling, tracking and monitoring facilities throughout the whole project cycle.
- The process involves several activities like test planning, designing and test execution. It gives an initial plan and discipline to the software testing process.

- **There are two main Parts of Test Management Process :**

Planning

1. Risk Analysis
2. Test Estimation
3. Test Planning
4. Test Organization

Execution

1. Test Monitoring and Control
2. Issue Management
3. Test Report and Evaluation

3.1.1 Structure of Testing Group, Test Planning, Detailed Test Design and Test Specification

UQ. Explain the structure of testing group. **MU - Dec. 19**

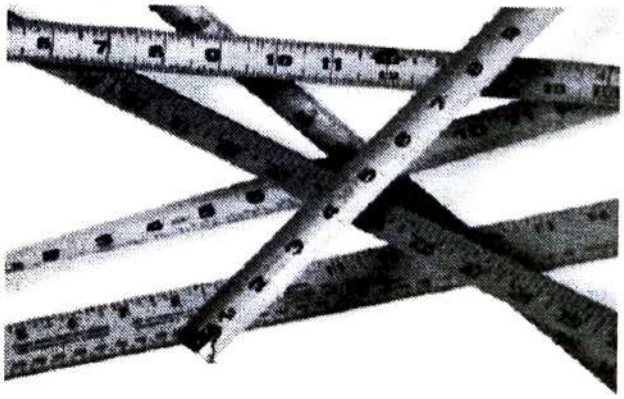
GQ. What are the components of a test plan? Illustrate test plan hierarchy with a neat diagram.

Planning Risk Analysis and Solution

- **Risk** is the potential loss (an undesirable outcome, however not necessarily so) resulting from a given action or an activity.
- Risk Analysis is the first step which Test Manager should consider before starting any project. Because all projects may contain risks, early risk detection and identification of its solution will help Test Manager to **avoid** potential loss in the future & save on project cost.

(1C2)**Fig. 3.1.2 : Risk Analysis**

Test Estimation



(1C3)**Fig. 3.1.3 : Test Estimation**

- An estimate is a forecast or prediction. Test Estimation is approximately determining **how long** a task would take to complete.
- Estimating effort for the test is one of the **major** and **important** tasks in Test Management.

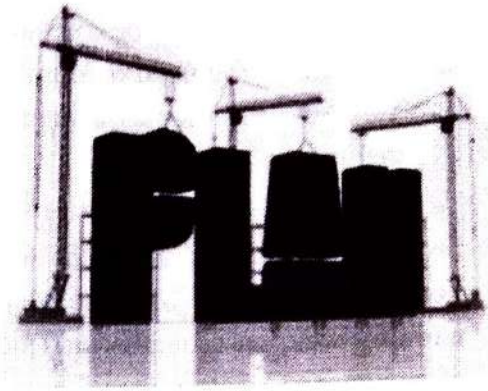
Benefits of correct estimation

1. Accurate test estimates lead to better planning, execution and monitoring of tasks under a test manager's attention.
2. Allow for more accurate scheduling and help realize results more confidently.

UQ. What is Test Planning? Explain Different components of Test plan document. **MU - Dec. 19**

Test Planning

- A Test Plan can be defined as a document describing the **scope**, **approach**, **resources**, and **schedule** of intended Testing activities.
- A project may **fail** without a complete Test Plan. Test planning is particularly important in large software system development.
- In software testing, a test plan gives **detailed** testing information regarding an upcoming testing effort, including:
  - Test Strategy
  - Test Objective
  - Exit /Suspension Criteria
  - Resource Planning
  - Test Deliverables



(1C4)**Fig. 3.1.4 : Test Planning**

Test Organization in Software Testing

GQ. What is Test Organization in Software Testing?

- **Test Organization in Software Testing** is a procedure of defining roles in the testing process. It defines who is responsible for which activities in testing process.
- Test functions, facilities and activities are also explained in the same process.
- The competencies and knowledge of the people involved are also defined however everyone is responsible for quality of testing process.

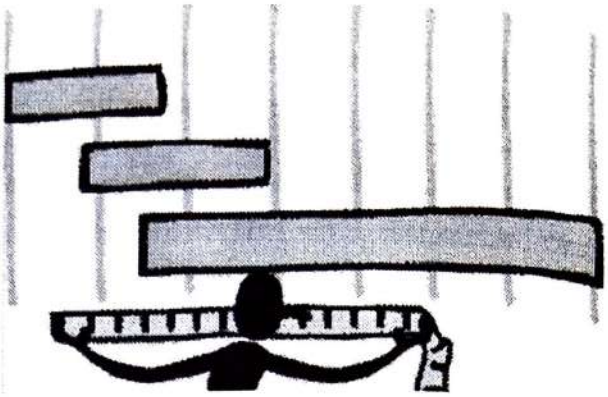
(1C5)**Fig. 3.1.5 : Test Organization**

- Now you have a Plan, but how will you stick to the plan and execute it? To answer that question, you have **Test Organization** phase.
- Generally speaking, you need to organize an effective Testing Team. You have to assemble a skilled team to run the ever-growing testing engine effectively.

☞ Execution

Test Monitoring and Control

- What will you do when your project runs **out of resource** or **exceeds** the time schedule? You need to Monitor and Control Test activities to bring it back on schedule.
- Test Monitoring and Control is the process of overseeing all the metrics necessary to ensure that the project is running well, on schedule, and not out of budget.



(1C6)**Fig. 3.1.6 : Test Monitoring**

☞ Monitoring

- Monitoring is a process of **collecting**, **recording**, and **reporting** information about the project activity that the project manager and stakeholder needs to know
- To Monitor, Test Manager does following activities
  - **Define** the project goal, or project performance standard
  - **Observe** the project performance, and compare between the actual and the planned performance expectations
  - **Record** and **report** any detected problem which happens to the project

☞ Controlling

- Project Controlling is a process of using data from monitoring activity to bring actual performance to planned performance.
- In this step, the Test Manager takes action to correct the deviations from the plan. In some cases, the plan has to be **adjusted** according to project situation.

☞ Issue Management

- As mentioned in the beginning of the topics, all projects may have **potential** risk. When the risk happens, it becomes an **issue**.
- In the life cycle of any project, there will be always an **unexpected** problems and questions that crop up. For an example :
  - The company cuts down your project budget

- Your project team lacks the skills to complete project
- The project schedule is too tight for your team to finish the project at the deadline.

- Risk to be avoided while testing :
  - **Missing** the deadline
  - **Exceed** the project budget
  - **Lose** the customer trust
- When these issues arise, you have to be ready to deal with them – or they can potentially affect the project's outcome.

UEx. 3.1.1 MU - Dec. 17, May 18

A program reads an integer number within the range [0,200] or [1,100] and determines whether it is an even number or not. Design test cases for this program using BVC, robust and worst case testing methods.

☑ **Soln. :**

1. **Test cases using Boundary value checking (BVC)**

Since these is one variable, the total number of test cases will be 4n + 1 = 5

In this example, the set of minimum and maximum values is shown below :

Min value = 1

+

Min value = 2

Max value = 100

-

Max value = 99

Nominal ...

2. **Test cases using BVC**

Since there is only one variable for finding number is even or not, the total test cases will be

4n + 1 = 5

∴ min value = 0

maximum value = 200

minimum + value = 1

maximum – value = 199

nominal value = 100

☞ Integration Testing Types

- Once the planning, designing and management of software is completed and the programmers and developers are ready with their completed codes and software structure, the testing team takes over the platform of **Software Development Life Cycle (SDLC)**.
- Testing, which is the most crucial and necessary part of software development process is executed by the team of testers in every major as well as minor stage of software development process. It ensures product quality, functionality, effectiveness, compatibility and more.
- Moreover, the main aim of **software testing** is to validate whether the end product meets the requirements of the client or not. During the process of development a software or application goes through several types of testing, which tests and examines different features, qualities and functionality of the software.
- Integration Testing is a **type of software testing**, which is performed on software to determine the flow between two or more modules by combining them. Integration testing makes sure that the interactions between different components of the software is completed smoothly without any complication.

☞ What is Integration Testing ?

- The term **'integrate'** as defined by Cambridge Dictionary means, "To combine two or more things in order to become more effective". This definition of the term integrate defines the basic purpose of integration testing, which is a software development process where program units are combined and tested as groups in multiple ways.
- In this context, a unit is defined as the smallest testable part of an application. In other words, **Integration Testing** is the phase in software testing, wherein individual software modules are combined and tested as a group. It occurs after **unit testing** and before **validation testing**.
- Furthermore, when software application is tested through integration testing, it exposes the problems and issues with the interfaces among program components before trouble occurs in real world execution.
- This type of testing takes its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
- Moreover, integration testing is a component of **extreme programming**, which is a pragmatic method of software development that takes meticulous approach to build a product by means of continual testing and revision.

What are the Different Types of Integration Testing ?

UQ. Explain the different types of Incremental Integration Testing Methods.

MU - Dec. 18, May 19

- From testing codes in proper context to combining and testing the components or modules of the software, integration testing is immensely useful.
- It is a type of software testing that not only ensures the quality of the software, but also makes sure that the software is tested in the production like environment before it is released for the use of the end users and clients.
- Integration testing, which is also known as **Integration and Testing (I&T)**, is performed through various types of approaches that are either executed by the integration tester or by a test team.
- As stated before, in this type of software testing program units are combined and tested as groups in multiple ways, hence each and every type of integration testing carries a lot of significance as they help testers in determining the effectiveness as well as the functionality of the software. Moreover, the various types of integration testing can be categorised into two different groups that are mentioned below:

1. Incremental Integration Testing

- In **Incremental Integration Testing**, the developers integrate the modules one by one using stubs and drivers to uncover defects in the software program.
- Here, each module has a definitive role to play in the project or product structure and has clearly defined dependencies, which can be known only at the runtime.
- The most important quality of incremental integration testing is that the defects are found early in a smaller assembly, when it is relatively easy to detect the root cause of the same.
- Incremental Integration Testing is performed on software through the following approaches :

a. Top-Down Integration Approach

In top-down approach the testing takes place from top to bottom, following the control flow or architectural structure. Additionally, components or systems are substituted by stubs.

Advantages

1. The tested product is extremely consistent in this approach as the integration testing is performed in an **environment** that is similar to the real world environment.
2. The stubs in top-down approach can be written in lesser time compared to drivers as they are simpler to author.

3. Here, fault **localization** is easier.
4. In top-down approach the critical modules are tested on priority, which further helps testers in finding major design flaws as early as possible.

☞ **Disadvantages**

1. It requires several stubs.
2. The modules at the lower level of the software are tested inadequately.
3. The basic functionality of the software is tested at the end of the cycle.

b. Bottom-Up Approach Integration Approach

In this approach of incremental integration testing, all the modules at the lower level of the software are tested with higher modules until all the modules are tested. Moreover, the **bottom up testing** takes place from the bottom of the control flow to upwards. Unlike top-down approach, here the components or systems are substituted with driver.

☞ **Advantages**

1. No time is wasted waiting for all modules to be developed.
2. In this approach development and testing can be done together so that the product or application will be efficient and as per the customer specifications.

☞ **Disadvantages**

1. The key interface defects are caught at the end of the cycle.
2. Test drivers are required to be created for modules at all levels except the top control.
3. The critical modules, which control the flow of the application are tested last and may be prone to defects.

2. Non-Incremental Integration Testing

- Whenever the relationship between the modules is not clear, non-incremental integration testing or big bang integration is executed.
- In this case the data is created in one module and is combined with all the other modules to check as well as test the flow of data between them. It is because of this that non-incremental integration testing is also known as Big Bang Integration.
- **Big Bang Integration Testing :** In this type of integration testing approach, most of the developed modules are coupled together to form a complete software system or a major part of the system, which is then used for integration testing.
- This method is very effective for saving time in the **integration testing process**. However, if the test cases and their results are not properly recorded, the entire integration process will be more complicated and may prevent the testing team form getting their desired goals or results of integration testing.

☞ **Advantages**

1. Here all components of the software are integrated at once.
2. Convenient for small systems.
3. Helps testers in saving integration testing time.

☞ **Disadvantages**

1. Fault localisation is difficult in this approach of integration testing.
2. Given the number of interfaces that are required to be tested in this approach, some of the interface links can be missed easily.
3. As all modules are tested at once, high risk critical modules are not isolated and tested in priority.

3. Sandwich Testing

- **Sandwich testing** is a culmination of both incremental as well as non incremental integration testing, wherein Bottom-Up approach is focused on middle to top layer, Top-Down approach is concerned about layers from middle to downwards and the Big Bang approach is followed for the middle layer.
- This type of testing combines the advantages of all the three approaches and is mainly used to test large projects.

☞ **Advantages**

1. Sandwich approach is very useful for large enterprises and huge projects that further have several subprojects.
2. When development follows a spiral model and the module itself is as large as a system, then one can use sandwich testing.
3. Top-Down and Bottom-Up approach both start as per development schedule.
4. Units are tested and brought together to make a system.
5. Integration is done downwards.
6. The resources that are required are immense and big team perform both top-down and bottom-up method of testing at a time or one after the other.

☞ **Disadvantages**

1. As both Top-Down and Bottom-Up approaches are executed on the software, the cost of testing is very high.
2. It cannot be used for smaller systems with huge interdependence between the modules.
3. It only makes sense when the individual subsystem is as good as completed system.
4. Different skill sets are required for testers at different levels.

Conclusion

- From the above discussion we can conclude that the importance of Integration testing and its types in extremely high during software development process. Integration testing, which is a type of software testing, combines and tests units in groups through various ways as well as testing types.
- The main purpose of this level of testing is to expose faults and other discrepancies, which are found when interactions are happening between integrated units. Moreover, to achieve desired goals and to detect all the faults in the integrated units, the test drivers and **test stubs** are used in integration testing, which assist testers with the process of testing. Hence, if a tester wants to have a streamline development process, where getting new features into production is fast and easy, then they should implement integration testing.
- In short, upon the completion of unit testing, the units or modules are integrated, which gives rise to integration testing, which aims at verifying the **function**, **performance**, and **reliability** between the modules that are integrated.

3.2 SOFTWARE METRICS : NEED, DEFINITION AND CLASSIFICATION OF SOFTWARE METRICS

UQ. Explain the need of software metrics. **MU - May 16, May 17, Dec. 19**

UQ. What is the need of software measurement? Explain different size metrics. **MU - May 17, May 18, Dec. 18, May 19**

Software Measurement and Metrics

Software Measurement

- A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process.
- Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement

Software is measured to :

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement Metrics

There are 2 types of software measurement:

1. **Direct Measurement**

   In direct measurement the product, process or thing is measured directly using standard scale.

2. **Indirect Measurement**

   In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Metrics

A metrics is a measurement of the level that any impute belongs to a system product or process. There are 4 functions related to software metrics :

1. Planning 2. Organizing
3. Controlling 4. Improving

Classification of Software Metrics

There are 2 types of software metrics :

1. **Product Metrics :** Product metrics are used to evaluate the state of the product, tracing risks and under covering prospective problem areas. The ability of team to control quality is evaluated.
2. **Process Metrics :** Process metrics pay particular attention on enhancing the long term process of the team or organisation.
3. **Project Metrics :** Project matrix is describes the project characteristic and execution process.
   - Number of software developer
   - Staffing pattern over the life cycle of software
   - Cost and schedule
   - Productivity

3.2.1 Test Design Preparedness Metrics

UQ. Explain test design preparedness metrics. **MU - Dec. 19**

The following metrics can be used to represent the level of preparedness of test design :

1. **Preparation Status of Test Cases (PST)**
   - A test case can go through a number of phases or states, such as draft and review, before it is released as a valid and useful test case.

- Thus it is useful to periodically monitor the progress of test design by counting the test cases lying in different states of design - create, draft, review, released and deleted.
- It is expected that all the planned test cases that are created for a particular project eventually move to the released state before the start of test execution.

2. Average Time Spent (ATS) In Test Case Design

- It is useful to know the amount of time it takes for a test case to move from its initial conception, that is, create state, to when it is considered to be usable, that is, released state.
- This metric is useful in allocating time to the test preparation activity in a subsequent test project. Hence it is useful in test planning.

3. Number of Available Test (NAT) Cases

- This is the number of test cases in the released state from the existing projects.
- Some of these test cases are selected for regression testing in the current test project.

4. Number of Planned Test (NPT) Cases

- This is the number of test cases that are in the test suite and ready for execution at the start of system testing.
- This metric is useful in scheduling test execution. As testing continues, new unplanned test cases may be required to be designed.
- A large number of new test cases compared to NPT suggest that initial planning was not accurate.

5. Coverage of a Test Suite (CTS)

- This metric gives the fraction of all requirements covered by a selected number of test cases or a complete test suite.
- The CTS is a measure of the number of test cases needed to be selected or designed to have good coverage of system requirements.

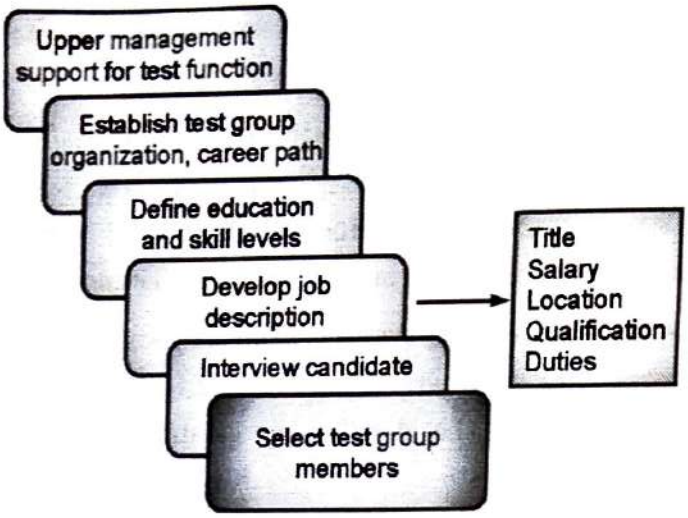
3.2.2 Building a Testing Group

- It was mentioned that organizing, staffing, and directing were major activities required to manage a project and a process. These apply to managing the testing process as well. Staffing activities include filling positions, assimilating new personnel, education and training, and staff evaluation.
- Directing includes providing leadership, building teams, facilitating communication, motivating personnel, resolving conflicts, and delegating authority.

- Organizing includes selecting organizational structures, creating positions, defining responsibilities, and delegating authority.
- Hiring staff for the testing group, organizing the testing staff members into teams, motivating the team members, and integrating the team into the overall organizational structure are organizing, staffing, and directing activities your organization will need to perform to build a managed testing process.
- Establishing a specialized testing group is a major decision for an organization. The steps in the process are summarized in Fig. 3.2.1. To initiate the process, upper management must support the decision to establish a test group and commit resources to the group.
- Decisions must be made on how the testing group will be organized, what career paths are available, and how the group fits into the organizational structure.
- When hiring staff to fill test specialist positions, management should have a clear idea of the educational and skill levels required for each testing position and develop formal job descriptions to fill the test group slots.
- When the job description has been approved and distributed, the interviewing process takes place. Interviews should be structured and of a problem-solving nature. The interviewer should prepare an extensive list of questions to determine the interviewee's technical background as well as his or her personal skills and motivation.
- Zawacki has developed a general guide for selecting technical staff members that can be used by test managers. Dustin describes the kinds of questions that an interviewer should ask when selecting a test specialist. When the team has been selected and is up and working on projects, the team manager is responsible for keeping the test team positions filled (there are always attrition problems).
- He must continually evaluate team member performance. Bartol and Martin have written a paper that contains guidelines for evaluation of employees that can be applied to any type of team and organization .
- They describe four categories for employees based on their performance :

  (i) retain, (ii) likely to retain,

  (iii) likely to release (iv) release.
- For each category, appropriate actions need to be taken by the manager to help employee and employer.



(1C7)**Fig. 3.2.1 : Steps informing a test group**

☞ Structure of test group

UQ. Explain structure of Testing Group. **MU - May 16**

- It is important for a software organization to have an independent testing group. The group should have a formalized position in the organizational hierarchy. A reporting structure should be established and resources allocated to the group.
- Will eventually need to upgrade their testing function to the best case scenario which is a permanent centralized group of dedicated testers with the skills described earlier in this chapter.
- This group is solely responsible for testing work. The group members are assigned to projects throughout the organization where they do their testing work.
- When the project is completed they return to the test organization for reassignment. They report to a test manager or test director, not a project manager. In such an organization testers are viewed as assets.
- They have defined career paths to follow which contributes to long-term job satisfaction. Since they can be assigned to a project at its initiation, they can give testing support throughout the software life cycle. Because of the permanent nature of the test organization there is a test infrastructure that endures.
- There is a test knowledge base of test processes, test procedures, test tools, and test histories (lessons learned).
- Dedicated staff is responsible for maintaining a test case and test harness library
- A test organization is expensive, it is a strategic commitment. Given the com nature of the software being built, and its impact on society, organizations n realize that a test organization is necessary and that it has many benefits.

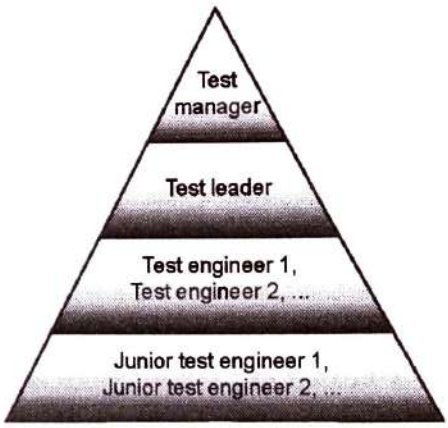
- By investing in a test organization a company has access to a group of specialists who have the responsibilities and motivation to :
  - maintain testing policy statements;
  - plan the testing efforts;
  - monitor and track testing efforts so that they are on time and within budget;
  - Measure process and product attributes;
  - provide management with independent product and process quality information;
  - design and execute tests with no duplication of effort;
  - automate testing;
  - Participate in reviews to insure quality; are meet.
- The duties of the team members may vary in individual organizations. The following gives a brief description of the duties for each tester that are common to most organizations.

The Test Manager

- In most organizations with a testing function, the test manager (or test director) is the central person concerned with all aspects of testing and quality issues.
- The test manager is usually responsible for test policy making, customer interaction, test planning, test documentation, controlling and monitoring of tests, training, test tool acquisition, participation in inspections and walkthroughs, reviewing test work, the test repository, and staffing issues such as hiring, firing, and evaluation of the test team members. He or she is also the liaison with upper management, project management, and the quality assurance and marketing staffs.

The Test Lead

- The test lead assists the test manager and works with a team of test engineers on individual projects. He or she may be responsible for duties such as test planning, staff supervision, and status reporting.
- The test lead also participates in test design, test execution and reporting, technical reviews, customer interaction, and tool training.



(1C8)**Fig. 3.2.2 : Test team hierarchy**

The Test Engineer

- The test engineers design, develop, and execute tests, develop test harnesses, and set up test laboratories and environments.

- They also give input to test planning and support maintenance of the test and defect repositories.

The Junior Test Engineer

- The junior test engineers are usually new hires. They gain experience by participating in test design, test execution, and test harness development.
- They may also be asked to review user manuals and user help facilities defect and maintain the test and defect repositories.

3.3 TESTING METRICS FOR MONITORING AND CONTROLLING THE TESTING PROCESS : ATTRIBUTES AND CORRESPONDING METRICS

- **Definition : Attribute :** An **attribute** is a property, qualitative or quantitative, of software products or processes.
- **Definition : Product attribute, process attribute**: A **product attribute** is an attribute that characterizes a software product or set of products. A **process attribute** is an attribute that characterizes a software-related process, such as development, maintenance, documentation, management, or multiple instances of such a process.

- Examples of attributes include reliability (a product attributes, non-quantitative) and total project cost (process, quantitative).
- A metric is simply a quantitative attribute:

- **Definition : Metric**: A **metric** is an attribute whose values are numbers (integers or reals), expressed relative to a certain **unit** specified as part of the metric definition.

- Examples of metrics include the number of source lines of a program (product) and the total cost of a project (process). The metric tool provides by default a set of metrics, they are available in the Metric Evaluation tab of the metric tool
- Attributes other than metrics will be called "qualitative":

- **Definition : Qualitative attribute :** A qualitative attribute is an attribute other than a metric.

- An example of qualitative attribute is the reliability of a software product.
- The "process" vs. "product" distinction carries over to metrics :

- **Definition : Product metric, process metric**: A metric is a **product metric** if it is a product attributes, a **process metric** if it is a process attribute.

- "Relevance", as defined in the previous section, suggests that the purpose of metrics is to help us gain information about attributes that are of direct interest to us. Often these will be qualitative; for example we may want to estimate the reliability of our software. Metrics provide us with numerical values that can serve to assess or predict such attributes.

Applying a metric will give us measures :

- **Definition : Measure** : A **measure** is the value of a metric for a certain process or product.

- **Test Monitoring and Test Control** is basically a management activity. **Test Monitoring** is a process of evaluating and providing feedback on the "currently in progress" testing phase. **Test Control** is an activity of guiding and taking corrective action based on some metrics or information to improve efficiency and quality.

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

- Point 1 and 2 basically talk about Test Reporting, which is an important part of Test Monitoring. Reports should be precise and should avoid "long stories". It is important here to understand that the content of the report differs for every stakeholder.

- Points 3 and 4 talks about the metrics. Following metrics can be used for Test Monitoring :
  - Test Coverage Metric
  - Test Execution Metrics (Number of test cases pass, fail, blocked, on hold)
  - Defect Metrics
  - Requirement Traceability Metrics
  - Miscellaneous metrics like level of confidence of testers, date milestones, cost, schedule and turnaround time.

- Test Control involves guiding and taking corrective measures activity, based on the results of Test Monitoring. Test Control examples include :
  - Prioritizing the Testing efforts
  - Revisiting the Test schedules and Dates

- Reorganizing the Test environment
- Re prioritizing the Test cases/Conditions

- Test Monitoring and Control go hand in hand. Being primarily a manager's activity, a Test Analyst contributes towards this activity by gathering and calculating the metrics which will be eventually used for monitoring and control.

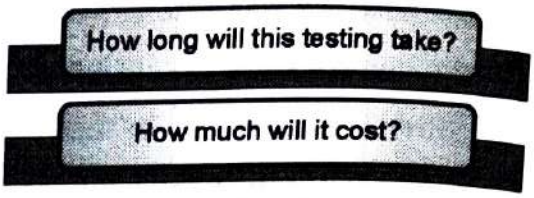
3.3.1 Estimation Model for Testing Effort

What is Software Test Estimation?

Test Estimation is a management activity which approximates **how long** a Task would take to complete. Estimating effort for the test is one of the **major** and **important** tasks in Test Management.

Why Test Estimation?

Two questions you can expect from your clients when discussing potential test engagements are shown in Fig. 3.3.1.



(1C9)**Fig. 3.3.1 : Test Estimation**

What to Estimate?

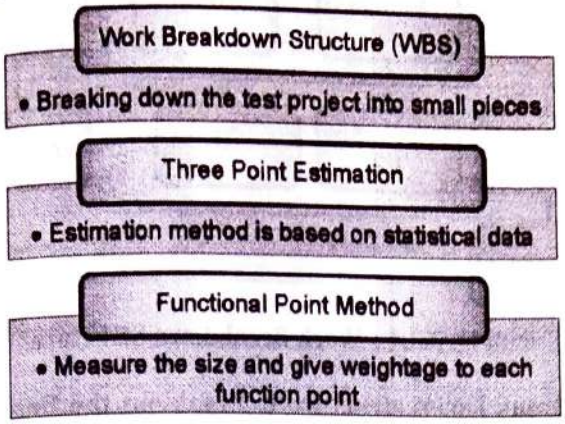
- **Resources :** Resources are required to **carry out** any project tasks. They can b people, equipment, facilities, funding, or anything else capable of definition required for the completion of a project activity.
- **Times :** Time is the most valuable resource in a project. Every project has a deadline to delivery.
- **Human Skills :** Human skills mean the **knowledge** and the **experience** of the Team members. They affect to your estimation. For example, a team, whose members have low testing skills, will take more time to finish the project than the one which has high testing skills.
- **Cost :** Cost is the project **budget**. Generally speaking, it means **how much money** it takes to finish the project.

How to estimate ?

List of Software Test Estimation Techniques :

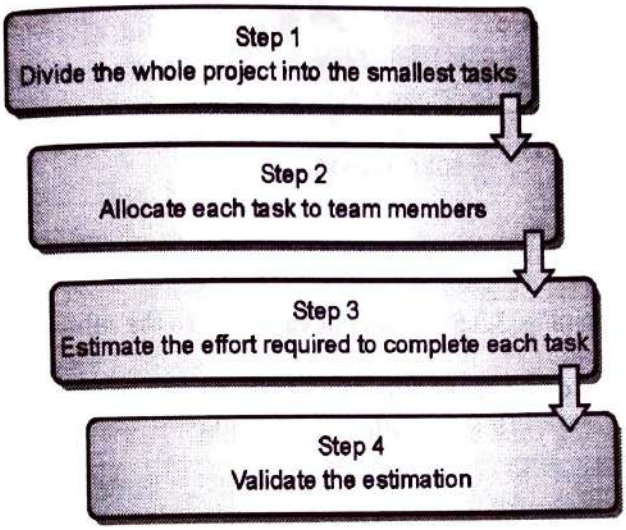
- Work Breakdown Structure
- 3-Point Software Testing Estimation Technique
- Wideband Delphi technique
- Function Point/Testing Point Analysis

- Use – Case Point Method
- Percentage distribution
- Ad-hoc method



(1C10)**Fig. 3.3.2 : Software Test Estimation Techniques**

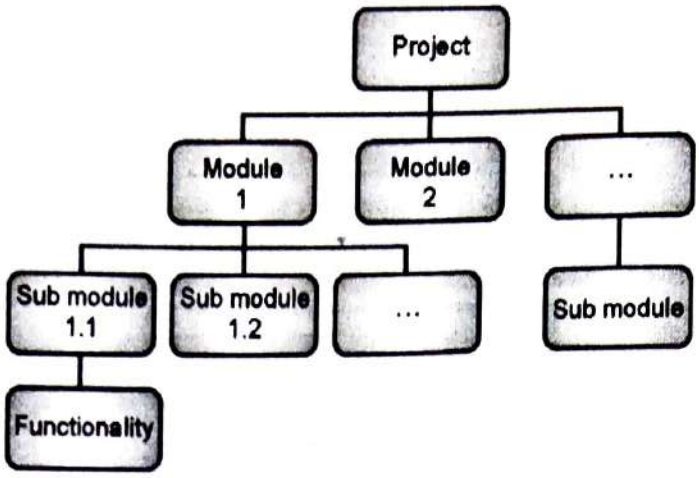
Following is the 4 Step process to arrive at an estimate



(1C11)**Fig. 3.3.3 : Estimation 4 Step process**

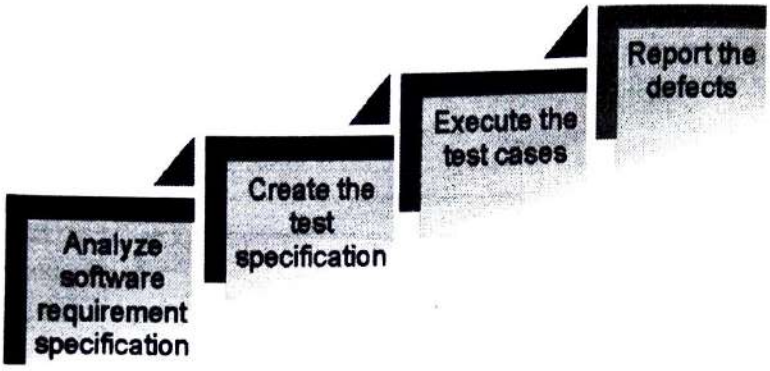
Step 1 : Divide the whole project task into subtasks

- Task is a piece of work that has been given to someone. To do this, you can use the **Work Breakdown Structure** technique.
- In this technique, a complex project is divided into modules. The modules are divided into sub-modules. Each sub-module is further divided into functionality. It means divide the whole project task into the **smallest** tasks.



(1C12)**Fig. 3.3.4 : Work Breakdown Structure**

- **Use the Work Break Down structure to break out the Bank project into 5 smaller tasks**



(1C13)**Fig. 3.3.5 : break out each task to the subtask**

- **After that, you can break out each task to the subtask. The purpose of th activity is creating task as detailed as possible.**

Task	Sub task
Analyze software requirement specification	Investigate the soft requirement specs
	Interview with the developer & other stakeholders to know more about the website
Create the Test Specification	Design test scenarios
	Create test cases
	Review and revise test cases
Execute the test cases	Build up the test environment
	Execute the test cases
	Review test execution results
Report the defects	Create the Defect reports
	Report the defects

Step 2 : Allocate each task to team member

In this step, each task is assigned to the **appropriate** member in the project team. You can assign task as follows

Task	Members
Analyze software requirement specification	All the members
Create the test specification	Tester/Test Analyst
Build up the test environment	Test Administrator
Execute the test cases	Tester, Test Administrator
Report defects	Tester

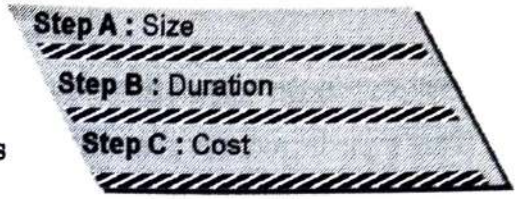
Step 3 : Effort Estimation for Tasks

There are 2 techniques which you can apply to estimate the effort for tasks

- Functional Point Method
- Three Point Estimation

Method 1 - Function Point Method

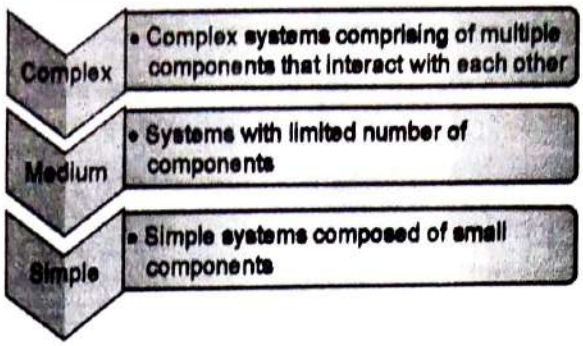
In this method, the Test Manager estimates Size, Duration, and Cost for the tasks.



(1C14)**Fig. 3.3.6 : Effort Estimation for Tasks**

Step A : Estimate size for the task

- In Step 1, you already have broken the whole project task into small task by using WBS method.
- Now you estimate the size of those tasks. Let's practice with a particular task **"Create the test specification"**
- The size of this task depends on the functional size of the system under test.
- The functional size reflects the **amount** of functionality that is relevant to the user. The more **number** of functionality, the more **complex** system is.
- Prior to start actual estimating tasks effort; functional points are divided into three groups like **Complex, Medium Simple** as following :



(1C15)**Fig. 3.3.7 : Estimate size for the task**

- Based on the complex of software functions, the Test Manger has to give enough weightage to each functional point. For example

Group	Weightage
Complex	5
Medium	3
Simple	1

Test estimation best practices

This topic introduces general tips on how to estimate Testing accuracy.

Add some buffer time

- Many unpredictable things may happen to your project, such as a talented team member quits his job suddenly, the testing takes more time than estimated to complete... etc.
- That why you need include some buffer in your estimation. Having a buffer in the estimation enables to cope for any delays that may occur.

Account Resource planning in estimation

- What should you do if some members in your team take long leaves? It may delay the project. Resource planning in estimation plays a key role.
- The availability of resources will help to make sure that the estimations are realistic. Here you have to consider the leaves for your team member, generally long leaves.

Use the past experience as reference

- Experiences from past projects play a vital role while preparing the time estimates. Because some project may be some similarity, you can reuse the past estimation.

- For example, if you use to do a project like testing a website, you can learn from that experience, try to avoid all the difficulties or issues that were faced in past projects.

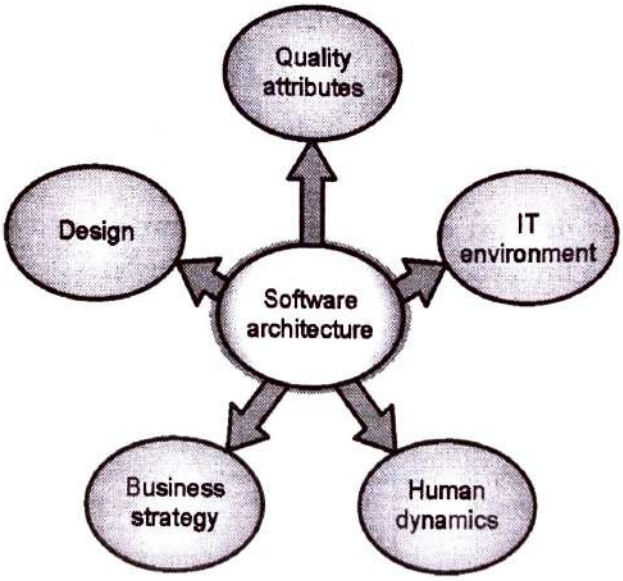
Stick to your estimation

- Estimation is just estimate because it may go wrong. In early stages of the project, you should frequently re-check the test estimations and make modification if needed.
- We should not extend the estimation after we fix it, unless there are major changes in requirement, or you have to negotiate with customer about the re-estimation

3.3.2 Architectural Design, Information Flow Matrix used for Testing

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.

We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.



(1C16)**Fig. 3.3.8 : Architecture of a System**

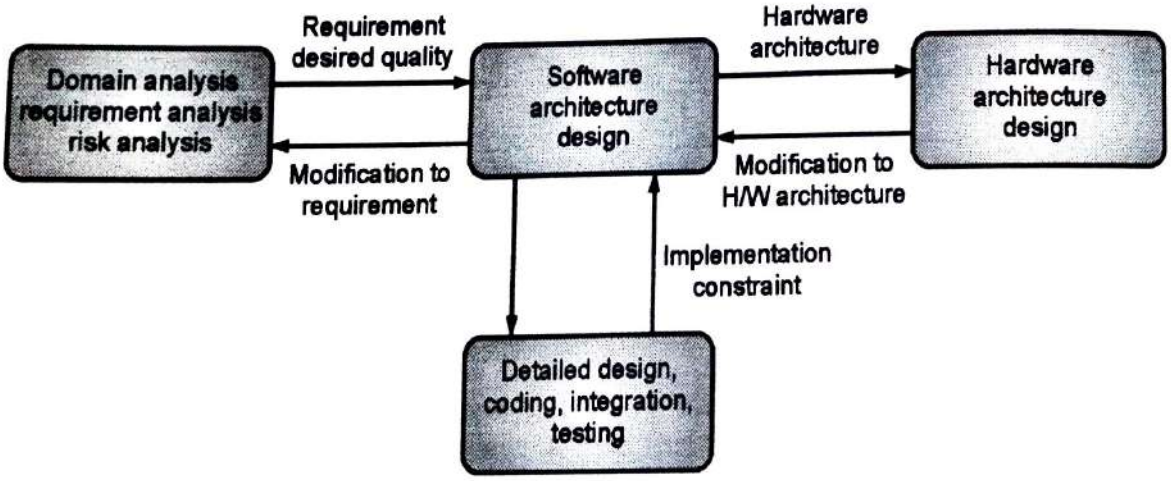
Software Architecture

- Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of
  - Selection of structural elements and their interfaces by which the system is composed.
  - Behavior as specified in collaborations among those elements.
  - Composition of these structural and behavioral elements into large subsystem.
  - Architectural decisions align with business objectives.
  - Architectural styles guide the organization.

☞ Software Design

- Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows :
  - To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
  - Act as a blueprint during the development process.
  - Guide the implementation tasks, including detailed design, coding, integration, and testing.
- It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



(1C17)**Fig. 3.3.9 : Software design provides a design plan**

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows –

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations –

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design.

Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas :

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoff among themselves.

☞ Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process, assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

☞ Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

☞ Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

☞ Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

☞ Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans
- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

☞ Business Processes and Information Flow

The Information Flow Model

- While business process models are intended to capture the details of the tasks and interactions with a process, it is useful to augment the business process model with the corresponding information flow.

- This will expose how both information and control are shared and propagated through the business application.
- It is useful to have a method for describing the way data propagates through a system, and this section describes some aspects of a high-level information flow model.
- An information flow model distinguishes the discrete processing stages within the process, describes how information flows through that system, characterizes the kinds of data items that flow through the process, and captures the type or method of data access.
- This model is valuable because it provides a basis for distinguishing between data dependencies, control dependencies, and artificially imposed implementation dependencies, which in turn can lead toward flow optimization, identification of bottlenecks, finding locations for insertion of data validation monitors, inserting data collection points for later analysis, and opportunities for increased business analysis points.

Information Flow : Processing Stages

In an information flow model, we distinguish discrete processing stages. Although the following list is by no means complete, we can characterize each information flow stage as one of these classes.

- **Supply**, representing external data suppliers provides
- **Acquisition**, representing the point at which existing data instances are acquired
- **Transformation**, representing the point where a data instance is modified to conform to another processing stage's expected representative format
- **Creation**, the point at which new data instances are created
- **Process**, representing points at which system state is modified as a result of input data
- **Store**, in which a data instance is stored in a persistent system
- **Packaging**, in which data is collated, aggregated, and/or summarized
- **Switch/route**, where a set of rules is used to determine where and how to route data instances
- **Decision point**, which is a point at which a data consumer (real or automated) is solicited for a decision
- **Deliver**, the delivery point for data that is meant to be consumed
- **Consume**, the presentation point for information presented by the system

3.3.3 Function Point and Test Point Analysis

UQ. Explain Function Point and Test Point Analysis ? MU - Dec. 16, Dec. 17, Dec. 18, May 19, Dec. 19

UQ. Write notes on Test suite minimization problem? MU - May 16, Dec. 19

- A **Function Point (FP)** is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing.
- For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are :

ISO Standards

- **COSMIC** : ISO/IEC 19761:2011 Software engineering. A functional size measurement method.
- **FiSMA** : ISO/IEC 29881:2008 Information technology - Software and systems engineering - FiSMA 1.1 functional size measurement method.
- **IFPUG** : ISO/IEC 20926:2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.
- **Mark-II** : ISO/IEC 20968:2002 Software engineering - Ml II Function Point Analysis - Counting Practices Manual.
- **NESMA** : ISO/IEC 24570:2005 Software engineering - NESMA function size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis.

Object Management Group Specification for Automated Function Point

- **Object Management Group (OMG),** an open membership and not-for-profit computer industry standards consortium, has adopted the Automated Function Point (AFP) specification led by the Consortium for IT Software Quality. It provides a standard for automating FP counting according to the guidelines of the International Function Point User Group (IFPUG).
- **Function Point Analysis (FPA) technique** quantifies the functions contained within software in terms that are meaningful to the software users. FPs consider the number of functions being developed based on the requirements specification.

- **Function Points (FP) Counting** is governed by a standard set of rules, processes and guidelines as defined by the International Function Point Users Group (IFPUG). These are published in Counting Practices Manual (CPM).

Functions

There are two types of functions :

- Data Functions
- Transaction Functions

Data Functions

There are two types of data functions :

- Internal Logical Files
- External Interface Files

Data Functions are made up of internal and external resources that affect the system.

Internal Logical Files

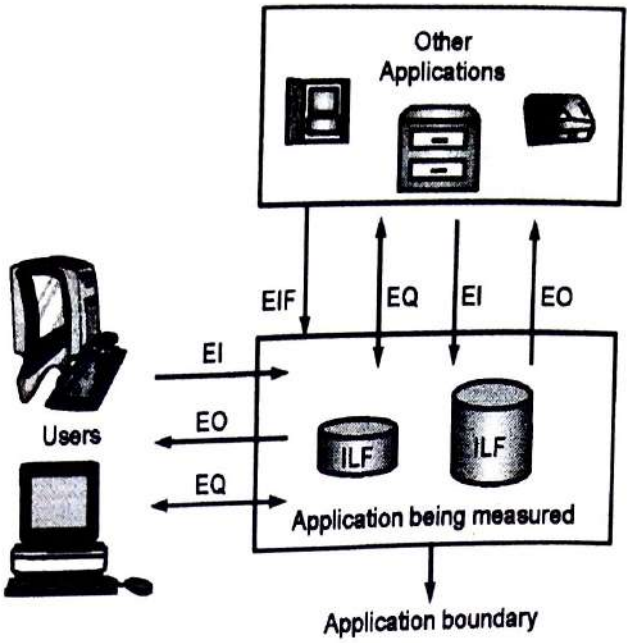
- **Internal Logical File (ILF)** is a user identifiable group of logically related data or control information that resides entirely within the application boundary.
- The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted. An ILF has the inherent meaning that it is internally maintained, it has some logical structure and it is stored in a file.

(Refer Fig. 3.10.10)

External Interface Files

- External Interface File (EIF) is a user identifiable group of logically related data or control information that is used by the application for reference purposes only.
- The data resides entirely outside the application boundary and is maintained in an ILF by another application.
- An EIF has the inherent meaning that it is externally maintained, an interface has to be developed to get the data from the file.

(Refer Fig. 3.3.10)



(1C16)Fig. 3.3.10 : Application boundary, data functions, transction functions

Transaction Functions

- There are three types of transaction functions.

1. External Inputs
2. External Outputs
3. External Inquiries

- Transaction functions are made up of the processes that are exchanged between the user, the external applications and the application being measured.

1. External Inputs

External Input (EI) is a transaction function in which Data goes "into" the application from outside the boundary to inside. This data is coming external to the application.

- Data may come from a data input screen or another application.
- An EI is how an application gets information.
- Data can be either control information or business information.
- Data may be used to maintain one or more Internal Logical Files.
- If the data is control information, it does not have to update an Internal Logical File. (Refer Fig. 3.3.11)

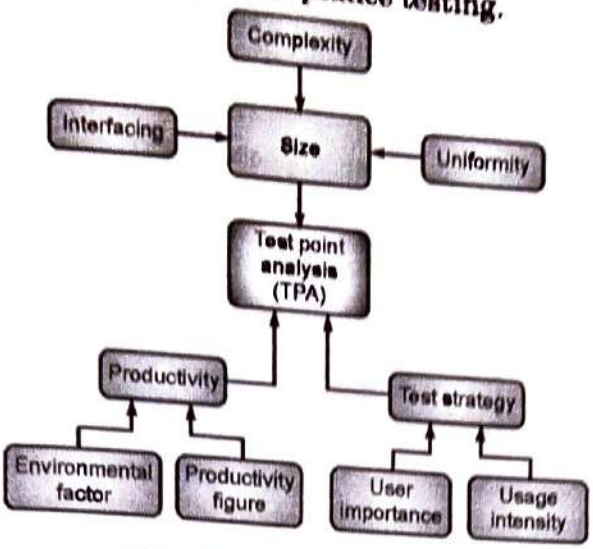
2. External Outputs

- External Output (EO) is a transaction function in which data comes "out" of the system. Additionally, an EO may update an ILF.

- The data creates reports or output files sent to other applications. (Refer Fig. 3.3.11)

3. External Inquiries

- External Inquiry (EQ) is a transaction function with both input and output components that result in data retrieval. (Refer Fig. 3.3.11)
- The function point analysis productivity factor covers the white box testing; it does not cover system testing or acceptance testing.



(1C19)**Fig. 3.3.11 : TPA parameters**

1) Size

- Size of an information system is determined mainly by the number of function points assigned to it.
- Other factors are Complexity; complexity relates to the number of conditions in a function. More conditions almost always mean more test cases and therefore a greater volume of testing work.
- Interfacing; the degree of interfacing of a function is determined by the number of data sets maintained by a function and the number of other functions, which make use of those data sets.
- Interfacing is relevant because these "other" functions will require testing if the maintenance function is modified.
- Uniformity; it is important to consider the extent to which the structure of a function allows it to be tested using existing or slightly modified specifications, i.e. the extent to which the information system contains similarly structured functions.

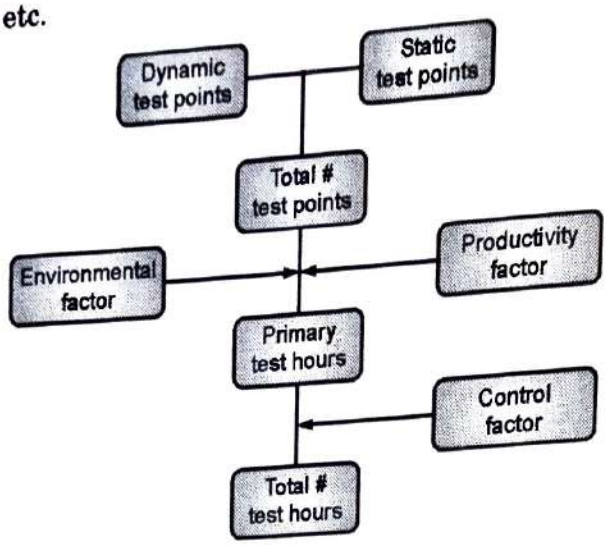
2) Strategy

- The importance attached to the various quality characteristics for testing purposes and the importance of the various subsystems and/or functions determine the test strategy.

- Any requirement importance is from two perspectives: one is the user importance and the other is the user usage.
- Depending on these two characteristics a requirement rating can be generated and a strategy can be chalked out accordingly, which also means that estimates vary accordingly.

3) Productivity

- Productivity has two important aspects: environment and productivity figures. Environmental factors define how much the environment affects a project estimate.
- Environmental factors include aspects such as tools, test environments, availability of test ware, etc.
- While the productivity figures depend on knowledge, how many senior people are on the team, etc.



(1C20)Fig. 3.3.12 : Overview thest point anaysis procedure

3.4 EFFICIENT TEST SUITE MANAGEMENT : MINIMIZING THE TEST SUITE AND ITS BENEFITS

Test Case Design

- Basically test design is the act of creating and writing test suites for testing software.
- Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities.
- But when we come to make a test case we need to be very specific.
- In fact now we need the exact and detailed specific input.
- But just having some values to input to the system is not a test, if you don't know what the system is supposed to do with the inputs, you will not be able to tell that whether your test has passed or failed.

- Test cases can be documented as described in the IEEE 829 Standard for Test Documentation.
- The IEEE 829 Standard for Test Documentation consists of different documents covering : Test plan, Test Design Specification, Test Case Specification, Test Procedure Specification, Test Item Transmittal Report, Test Log, Test Incidental Report, Test Summary Report.

3.4.1 Test Suite Minimization Problem, Test Suite Prioritization Its Type

Why does a test suite grow?

- There may be unnecessary test cases in the test suite including both obsolete and redundant test cases.
- For example a change in a program causes a test case to become obsolete.
- A test case is redundant if other test cases in test suite provide the same coverage of the program.
- Thus due to obsolete and redundant test cases, the size of a test suite continues to grow unnecessarily.
- Test engineers measures the extent to which a criterion is satisfied in terms of Coverage.
- Coverage is measured in terms of the requirements that are imposed.
- Partial Coverage is defined as the percentage of requirements that are satisfied.
- Coverage Criteria is used as a stopping point to decide when a program is sufficiently tested.
- **Minimizing the test suite and its benefits :** A test suite can sometimes grow to an extent that it is nearly impossible to execute. In this case, it becomes necessary to minimize the test cases such that they are executed for maximum coverage.
- The reasons why minimization is important are :
  - Release date of the product is near.
  - Limited staff to execute all the test cases.
  - Limited test equipment's or unavailability of testing tools minimizing the test suite has the following Benefits :
    1. Redundant test cases will be eliminated.
    2. Lower costs by reducing a test suite to a minimal subset.
- A reduction in the size of test suite decreases both the overhead of maintaining the test suite and the number of test cases that must be rerun after changes are made to the software, thereby reducing the cost of regression testing.

- **Test Suite Prioritization :** The purpose of prioritization is to reduce the number of test cases based on some criteria, while aiming to select the most appropriate tests.
- **The different priorities are :**
  - **Priority1 :** The test cases that must be executed otherwise there may be worse consequences for the release of the product.
  - **Priority 2 :** The test cases may be executed, if time permits.
  - **Priority 3 :** The test case is not important prior to the current release. It may be tested shortly after the release of the current version of the software.
  - **Priority 4 :** The test case is never important as its impact is nearly negligible.
- **Types of test case prioritization :** General Test Case Prioritization: In this prioritization, we prioritize the test cases that will be useful over a succession of subsequent modified versions of P (Program or Product), without any knowledge of the modified versions.
- **Version Specific Test Case Prioritization :** Here, we prioritize the test cases such that they will be useful on a specific version P' of P. Prioritization Techniques: Coverage Based Test Prioritization Risk Based Prioritization using Relevant Slices Prioritization based on Requirements
- **Coverage Based Test Prioritization :** Total Statement based Prioritization: This prioritization orders the test cases based on the total number of statements covered. It counts the number of statements covered by test cases and orders them in descending order. For example, if T1 covers 5 statements, T2 covers 7 statements and T3 covers 14 statements, then the order of prioritization is T3, T3, T1.

Additional Statement Coverage Prioritization

Statement	Statement Coverage		
	Test case 1	Test case 2	Test case 3
1	x	x	x
2	x	x	x
3		x	x
4			x
5			
6		x	
7	x	x	
8	x	x	
9	x	x	

- This technique iteratively selects a test case T1, that yields the greatest statement coverage, then selects a test case which covers a statement uncovered by T1.
- **Total Branch coverage Prioritization :** In this prioritization, the criteria to order are to consider condition branches in a program instead of statements. Thus it is the coverage of each possible outcome of a condition in a predicate.
- The test case which will cover maximum branch outcomes will be ordered first. For example, in the following diagram the order will be test cases 1,2,3.

Branch Statement	Branch Coverage		
	Test case 1	Test case 2	Test case 3
Entry to while	x	x	x
2-true	x	x	x
2-false	x		
3-true		x	
3-false	x		

- Total Fault-Exposing-Potential (FEP) Prioritization: Statement and branch coverage prioritization ignore a fact about test cases and faults.
- Thus the ability of a test case to uncover a fault is called the fault exposing potential.
- To obtain an approximation of a test case, an approach was adopted using manual analysis.
- The approach is: Given program P and test suite T, First create a set of mutants N = {n1, n2,.....nm} for P, noting which statement sj in P contains each mutant.
- Next, for each test case ti ϵT, execute each mutant version nk of P on ti , noting whether ti kills that mutant.
- Calculate the FEP(s,t) as the ratio :

$$FEP(s,t) = \frac{\text{Mutants of sj killed}}{\text{Total number of mutants of sj}}$$

- To perform total FEP prioritization, given these FEP(s,t) values, calculate an award value for each test case ti ∈T, by summing the FEP(sj, ti) values for all statements sj in P. Given these award values, we prioritize test cases by sorting them in order of descending award value.

Statement	FEP(s,t) Coverage		
	Test case 1	Test case 2	Test case 3
1	0.5	0.5	03
2	0.4	0.5	0.4
3		0.01	0.4
4		1.3	
5			
6	0.3		
7	0.6		0.1
8		0.8	0.2
9			0.6
Award values	1.8	3.11	2.0

- **Risk Based Prioritization :** Risk Based technique is to prioritize the test cases based on some potential problems which may occur during the project.
- The two components of risk based technique are: Probability of occurrence / likelihood : It indicates the probability of occurrence of a problem.
- **Severity of impact / failure impact :** If the problem has occurred, how mu impact does it have on the software? A risk analysis table consists of the following columns : Problem ID, Potential Problem, Uncertainty factor, Severity of impact, Risk Exposure. For example, the problems in the given table can be prioritized in the order of P5, P4, P2, P3 and P1 based on the risk exposure.

Problem ID	Potential Problem	Uncertainty Factor	Risk Impact	Risk Exposure
P1	Specification ambiguity	2	3	6
P2	Interface problems	5	6	30
P3	File corruption	6	4	24
P4	Databases not synchronized	8	7	56
P5	Unavailability of modules for integration	9	10	90

- **Prioritization using Relevant Slices :** Execution Slice: The set of statements executed under a test case is called Execution Slice of the program.

Example

```
S1: read(a,b);
S2: result;
S2: if(a>=100 || b>=200)
S3: result=a+b; S4: else
S5: result=a-b;
S6: return result;
```

Test Cases	a	b	result
1	150	250	400
2	50	200	250
3	200	50	250
4	20	20	0

- **Dynamic Slice :** The set of statements executed under a test case and have an effect on the program output under that test case is called Dynamic Slice of the program with respect to the output variables. Example:

```
S2: sum=0, I;
S2: if(a==0)
    S3: print a;
S4: else
    S5: print b;
        S6: else if
        S7: {
            S8: sum=a+b;
            S9: I=50;
            S10: Print(I);
            S11: }
S12: return(sum);
```

- **Relevant Slice :** The set of statements that were executed under a test case and did not affect the output, but have potential to affect the output produced by a test case is known as Relevant Slice of the program.
- For example, in the above code, statements S2 , S4 have the potential to affect the output, if modified. Prioritization based on Requirements: have applied requirement engineering approach for prioritizing the system test cases.
- It is known as PORT (Prioritization of Requirements for Test).

- **They have considered the four factors for analysing and measuring the criticality of requirements :**
  - Customer-Assigned priority of requirements
  - Requirement Volatility
  - Developer-perceived implementation complexity
  - Fault proneness of requirements

3.4.2 Techniques and Measuring Effectiveness

UEx. 3.4.1 MU - Dec. 19, Dec. 16

what is measuring effectiveness of prioritized test suits? Consider a program with 5 faults and a test suite of 5 test cases as shown in table below.

Calculate APFD for this program.

	T1	T2	T3	T4	T5
F1			X		X
F2		X	X	X	
F3	X				X
F4			X	X	X
F5	X			X	

Soln. :

Measuring the effectiveness of a Prioritized Test Suite

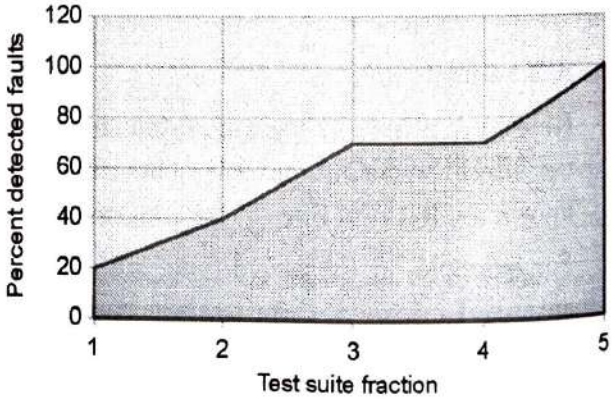
Developed APFD (Average percentage of faults detected) metric that measures the weighted average of the percentage of faults detected during the execution of the test suite. Its value ranges from 0 to 100 where higher value means faster fault detection rate.

APFD is calculated as given below.

$$APFD = 1 - \frac{(TF1 + TF2 + ... + TFm)}{nm} + 1/2n$$

Where :

- **T Fi** is the position of the first test in Test suite T that exposes fault i
- m is the total number of faults exposed in the system or module under T
- n is the total number of test cases in T



(1C21)**Fig. P. 3.4.1 : Prioritized Test Suite**

	T1	T2	T3	T4	T5
F1			X	X	
F2	X		X		X
F3		X		X	
F4	X		X	X	
F5		X		X	X

$$APFD = 1 - \frac{(3+1+2+1+2)}{5*5} + (1/2) * 5$$

3.5 SELF-LEARNING TOPICS

Design quality matrix for your selected system

(1) Software metrics can be classified into three categories

- **Product metrics** : Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** : These characteristics can be used to improve the development and maintenance activities of the software.
- **Project metrics** : This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.
- Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.
- **Software quality metrics** are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.
- Software quality metrics can be further divided into three categories :
  - Product quality metrics
  - In-process quality metrics
  - Maintenance quality metrics

Product Quality Metrics

This metrics include the following :

- Mean Time to Failure
- Defect Density

- Customer Problems
- Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

(2) Customer Problems

- It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.
- The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**.
- PUM = Total Problems that customers reported (true defect and non-defect oriented problems) for a time period + Total number of license months of the software during the period

  Where,

Number of license-month of the software	=	Number of install license of the software	×	Number of months in the calculation period

- PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year.

Customer Satisfaction

- Customer satisfaction is often measured by customer survey data through the five-point scale :
  - Very satisfied
  - Satisfied
  - Neutral
  - Dissatisfied
  - Very dissatisfied

- Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys.
- Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis. For example –
  - Percent of completely satisfied customers
  - Percent of satisfied customers
  - Percent of dis-satisfied customers
  - Percent of non-satisfied customers

Usually, this percent satisfaction is used.

(3) In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes –

- Defect density during machine testing
- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

☞ Defect density during machine testing

- Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field.
- Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.
- This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested.
- It is especially useful to monitor subsequent releases of a product in the same development organization.

☞ Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following –

- The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.

- The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
- The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately.
- This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

Phase-based defect removal pattern

- This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.
- Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software.
- The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.
- With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

Defect removal effectiveness

It can be defined as follows :

$$\text{DRE} = \frac{\text{Defect removed during a development phase}}{\text{Defects latent in the product}} \times 100\ \%$$

- This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used for the front-end and **phase effectiveness** for specific phases.
- The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

(4) Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix backlog and backlog management index

- Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available.
- It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.
- Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problems arrived during the month}} \times 100\,\%$$

- If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

Fix response time and fix responsiveness

- The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.
- The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

Percent delinquent fixes

It is calculated as follows –

$$\text{Percent Delinquent Fixes} = \frac{\text{Number of fixes that exceeded the response time criteria by ceverity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$$

CHAPTER 4

Module 4

Test Automation

University Prescribed Syllabus

4.1 **Automation and Testing Tools :** Need, Categorization, Selection and cost in testing tool,

4.2 Guidelines for testing tools.

4.1 Introduction to Automation and Testing Tools 4-2
4.1.1 Type of Testing 4-3
4.2 JIRA 4-4
4.3 Bugzilla 4-7
4.4 Test Director 4-8
4.5 IBM Rational 4-13
4.6 Brief History of the Selenium Project 4-14
4.6.1 Open Source Selenium 4-14
4.6.2 Selenium's Tool Suite 4-16
4.6.3 Selenium IDE 4-16
UQ. What is Selenium's IDE explain in detail. (MU - Oct. 18) 4-16
4.6.4 Selenium RC 4-18
UQ. What is Selenium's RC explain in detail. (MU - May 18) 4-18
4.6.5 Selenium Web-Driver 4-19
UQ. What is Selenium's Web-Driver explain in detail. (MU - May 19) 4-19
4.6.6 Selenium Grid 4-21
UQ. What is Selenium's Grid explain in detail. (MU - May 18) 4-21
4.7 Self-learning Topics 4-22
4.7.1 Write Down Test Cases, Execute and Manage Using Studied Tools 4-22
▶ **Chapter Ends** 4-28

4.1 INTRODUCTION TO AUTOMATION AND TESTING TOOLS

GQ. What Is Automation and Testing Tools ?

- Automation Testing or Test Automation is a software testing technique that performs using special automated testing software tools to execute a test case suite.
- On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.
- The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports.
- Software Test Automation demands considerable investments of money and resources.
- Successive development cycles will require execution of same test suite repeatedly. Using a test automation tool, it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. This improved ROI of Test Automation.
- The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual testing altogether.
- In the software testing world, there are two types of testing techniques - manual and automated. Both aim to execute the test case and then compare the actual outcome with the expected result.
- Manual testing is a testing technique that requires significant human effort to ensure a software solution does everything that it is supposed to do.
- While great for specific test cases, excessive use of manual testing proved to be inefficient over time. Modern software development consists of successive development cycles that require repetitive executions of the same groups of tests (also known as test suites). This process can be extremely mentally-taxing and time-consuming if performed manually.
- Therefore, test automation was designed to tackle this very problem. Since its inception, many types of automation testing have been introduced, and robust testing tools have allowed testers to be more hands-off by simplifying writing and replaying test suites, which frees up valuable resources and boosts business productivity. This article will cover the most popular types of test automation any tester should know to be well-prepared for their testing journey.

What Can be Automated ?

- Quite a lot actually! Typically, automated testing is divided into the type of testing, the type of tests, and the phase of testing.
- (Please note that some of these may overlap and the classification is often dependent on workflow and preferences).

4.1.1 Type of Testing

There are 2 main types, functional and non-functional:

1. **Functional :** Which tests the real-world, business application of a software solution? For example, a ride-sharing app likes. Uber must be able to connect end users to drivers when all conditions are met, at the bare minimum.
2. **Non-functional :** This tests the remaining requirements of the software (for example performance, security, data storage, etc.) With the ride-sharing example, this type of testing will ensure that the app is fast and efficient when performing its most essential functions, like connecting end users to drivers in this case.

Type of tests

Aside from the types of automation testing; Smoke Tests, Integration Tests, Regression Tests, Security Tests, Performance Tests, Acceptance Tests, etc. are also common in the field of test automation.

1. **Smoke Tests**

   Smoke tests are a type of Functional test that only covers the most crucial features of a software solution to ensure that it could be further tested without "catching fire," hence the name Smoke Tests.

2. **Integration Tests**

   Integration tests take all the individual pieces and functionalities of a software solution and test them together as a whole to guarantee smooth operation between all of them.

3. **Regression Tests**

   Regression tests run a combination of Functional and Non-functional tests to check if the software has "regressed" after a given change.

4. **Security Tests**

   Security tests cover Functional and Non-functional tests that screen the software for any vulnerability. They reveal weaknesses and any potential exploit in a system.

5. Performance Tests

Performance tests are often Non-functional tests that help testers evaluate criteria like responsiveness and stability as the software handles load and stress.

6. Acceptance Tests

Acceptance tests are Functional tests that determine how acceptable the software is to the end-users. This is the final test a solution must pass before it could be released.

☞ Phase of testing

1. Unit

As the name implies, this phase tests the individual components, or units, of software. Unit testing is the very first phase of testing, usually done manually by developers before handing the software off to testers, but it could also be automated.

2. API

Application Programming Interface (or API for short) acts as the "middleman" between all of the systems that your software uses, and thus, is then tested after the development process to ensure smooth integration between systems and software. This phase of testing is fairly flexible; it could be conducted either before or after the UI phase, which we will go over shortly, and by either the development or the testing team.

3. UI

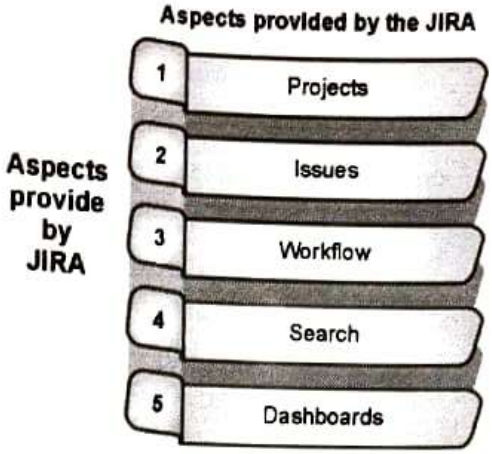
Last but not least, User Interface (AKA UI) is what the end users see and interact with and, thus, is usually tested at the very end of the process. This phase of testing is run by testers after the UI of the application has been drafted for the most authentic replication of user experience possible. This is where the business logic of the software is examined and optimized, which also falls under the Functional test classification.

4.2 JIRA

GQ. *What is Jira as testing tool?*

- JIRA is a software testing tool developed by the Australian Company Atlassian. It is a bug tracking tool that reports all the issues related to your software or mobile apps. The word JIRA comes from the Japanese word, i.e., "Gojira" which means Godzilla.
- JIRA tutorial provides basic and advanced concepts of JIRA tool. Our JIRA tutorial is designed for beginners and professionals.

- JIRA is one of the most widely used open source testing tool used in manual testing
- Our JIRA tutorial includes all topics of a testing tool such as Features, installation, issues, workflows, components, reports, etc.
- JIRA is based on the Agile methodology and the current version of the Jira is 6.



(1D1)**Fig. 4.2.1**

The following are the useful aspects provided by the Jira :

1. Projects	2. Issue	3. Workflow
4. Search	5. Dashboards	

1. **Projects :** It is used to manage the defects very effectively.
2. **Issue :** It is used to track and manage the defects/issues.
3. **Workflow :** Processes the Issue/Defect life cycle. Suppose we have a business requirement, we create the technical design and from the technical design, we create the test cases. After creating the test cases, coding is done, and then testing is performed on the project. This design workflow is possible by using Jira.
4. **Search :** Find with ease. Suppose we have done with a project at the beginning of the December and its version is 1.0. Now, we move to version 1.1 and completed at the end of December. What we are doing is that we are adding new versions. Through Jira, we can get to know that what happened in the earlier versions, how many defects occurred in the earlier projects and the learning we achieve from the earlier projects.
5. **Dashboards :** Dashboard is a display which you see when you log in to the Jira. You can create multiple dashboards for multiple projects. You can create the personal dashboard and can add the gadgets in a dashboard so that you can keep track of the assignments and issues that you are working on.

Why JIRA ?

JIRA tool is used because of the following reasons :

- Plan, Track and Work Faster
- JIRA is a bug-tracking tool mainly used to track, organize, and prioritize the bugs, newly added features, improvements for certain software releases.

- Projects are subdivided into issues and issues can be of multiple types such as bug, new feature, improvement, and documentation tasks.
- When the release date of software comes near, then software developers need to focus on the remaining issues which are to be fixed before the specified date. It also becomes difficult for the QA to maintain the status of the documentation, i.e., sometimes it becomes hard to keep track of everything.
- JIRA is a good choice for handling the above issues. It enables software developers to track issues and improvements. It manages the projects as well as maintain the technical documentation.

The main source of information

- JIRA is the primary source of information for the next software release. On JIRA, the whole team of the software developers can plan for the new features which are to be added and bugs to be fixed in the next release.
- It also helps the QA team in writing the technical documentation. Through JIRA, the QA team can check the status of each feature that is newly added by the software developers, and according to that, they can plan how to document for the new version.

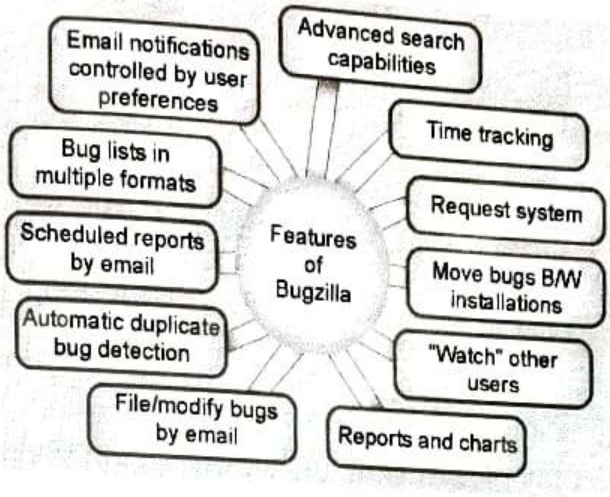
Organize the documentation tasks

- JIRA tool is used to organize the documentation tasks. It is useful in grouping the multiple tasks by using the component functionality, and even you can create your own documentation. In this way, you can create a structured way of documentation.
- Track the progress of our documentation
- It is a very useful tool in tracking the progress of our documentation. JIRA tool provides a very important feature, i.e., pie chart macro. In the pie chart macro, you can view tasks such as Open tasks, Closed tasks, Resolved tasks.
- Helps to meet the deadlines of a documentation release.
- You can define the specific due date or deadline for the release of documentation, and even you can configure the JIRA tool with the notifications so that you can finish your documentation in time.
- Measures the time spent on documentation
- JIRA tool does not have the default functionality for measuring the time spent on documentation. JIRA tool is bundled with the Tempo Timesheets, which measures how much time has been spent on the documentation.
- Provides feedback faster. JIRA tool provides the Confluence pages where you can connect to the issues in just a few clicks. If something needs to be updated, then you can create the issues directly from the Confluence page.

4.3 BUGZILLA

GQ. What is Bugzilla as testing tool?

- Bugzilla is a bug tracking tool that helps to track the issues related to their product.
- Bugzilla tool is written in Perl language, and it uses MySQL database.
- It is a bug tracking tool. However, it can also be used as a test management tool because it can be linked with other Test case management tools such as Quality Center, Testlink, etc.



(102)**Fig. 4.3.1**

- The following are the **features of a Bugzilla** :
  1. Advanced Search capabilities
  2. Bugzilla has two forms of search:
- It has Google-like bug search which is simple to use, and it also searches the full text of a bug.
- It provides you a very advanced search system where you can create any type of search that you want such as time-based searches (For example, you want to see the list of bugs whose priority has been changed since last two days).
- **Email Notifications controlled by user preferences**

  You will get an email if any changes are made in the Bugzilla, and the notifications that you get on which bug is totally controlled by the user preferences.
- **Reports and Charts**

  When you search for the bugs, then you can get the bug lists in multiple formats such as Atom, iCalendor format.

  The iCalendor format is used when you are using the time tracking feature in Bugzilla. There are even more formats available in Bugzilla such as printable format that contains the details of all the bugs, CSV format used for importing bug list into spreadsheets.

- **Scheduled reports by email**

  Bugzilla has a system that will send you, users or a group that you specify the results of a search on a schedule that you have mentioned.

- **Automatic Duplicate Bug Detection**

  When you are filing a new bug, and when you type the summary for the bug, then the system looks for similar bugs. If the system finds the similar bugs, then it allows the user to add themselves in the CC list of one of those bugs instead of creating a new one.

- **File/Modify Bugs by email**

  You can send an email to Bugzilla to create a new bug or modify the existing bug. You can even attach the files to Bug.

- **Time Tracking**

  Bugzilla also provides the feature of time tracking. You can determine how many hours a bug will take to get fixed and you can also even track the hours that you need to spend on the bug. You can also set the deadline by which the bug needs to be fixed.

- **Request System**

  The request system is a way of asking other users to do something with a bug. The user can either grant the request or deny the request, and Bugzilla will keep track of the answer.

- **Bugs Between Installations**

  Bugzilla has the ability to move the bug from one Bugzilla installation to another. In Bugzilla, a bug can also be moved across different versions.

- **"Watch" Other Users**

  Bugzilla allows you to watch other users. You will also get an email that the user gets from Bugzilla. This feature is useful when some developer goes for leave then other developer needs to handle the bug.

- **Reports and Charts**

  Bugzilla provides an advanced reporting system. If you want to know how the database of a bug looks like, then you can create the two fields across the X-axis and Y-axis. With the help of search criteria, you can limit the information of a bug. You can also export these reports in CSV format.

4.4 TEST DIRECTOR

GQ. What is Test Director ?

- It is a Global Test Management tool, the industry's first global test management solution.

- It helps organizations deploy high-quality applications more quickly and effectively.
- It has four modules:

1. Requirements
2. Test Plan
3. Test Lab
4. Defects

- These modules are seamlessly integrated, allowing for a smooth information flow between various testing stages. The completely Web-enabled TestDirector supports high levels of communication and collaboration among distributed testing teams, driving a more effective, efficient global application-testing process.
- **Web-based Site Administrator :** The Site Administrator includes tabs for managing projects, adding users and defining user properties, monitoring connected users, monitoring licenses and monitoring TestDirector server information.
- **Domain Management :** TestDirector projects are now grouped by domain. A domain contains a group of related TestDirector projects, and assists you in organizing and managing a large number of projects. Enhanced Reports and Graphs Additional standard report types and graphs have been added, and the user interface is richer in functionality. The new format enables you to customize more features. Version Control Version control enables you to keep track of the changes you make to the testing information in your TestDirector project. You can use your version control database for tracking manual, WinRunner and QuickTest Professional tests in the test plan tree and test grid.
- **Collaboration Module :** The Collaboration module, available to existing customers as an optional upgrade, allows you to initiate an online chat session with another TestDirector user. While in a chat session, users can share applications and make changes.
- **TestDirector Advanced Reports Add-in :** With the new Advanced Reports Add-in, TestDirector users are able to maximize the value of their testing project information by generating customizable status and progress reports. The Advanced Reports Add-in offers the flexibility to create custom report configurations and layouts, unlimited ways to aggregate and compare data and ability to generate cross-project analysis reports.
- **Automatic Traceability Notification :** The new traceability automatically traces changes to the testing process entities such as requirements or tests, and notifies the user via flag or e-mail. For example, when the requirement changes, the associated test is flagged and tester is notified that the test may need to be reviewed to reflect requirement changes. Coverage Analysis View in Requirements Module The graphical display enables you to analyze the requirements according to test coverage status and view associated tests - grouped according to test status.
- **Hierarchical Test Sets :** Hierarchical test sets provide the ability to better organize your test run process by grouping test sets into folders.

- **Workflow for all TestDirector Modules :** The addition of the script editor to all modules enables organizations to customize TestDirector to follow and enforce any methodology and best practices. Improved Customization With a greater number of available user fields, ability to add memo fields and create input masks users can customize their TestDirector projects to capture any data required by their testing process. New rich edit option add color and formatting options to all memo fields.
- TestDirector Features and Benefits Supports the entire testing process. TestDirector incorporates all the following aspects of the testing process into a single browser-based application :
  - Requirements management
  - Planning
  - Scheduling
  - Running tests
  - Issue management
  - Project status analysis
- Leverages innovative Web technology Testers, developers and business analysts can participate in and contribute to the testing process by working seamlessly across geographic and organizational boundaries.
- **Uses industry-standard repositories :** TestDirector integrates easily with industry-standard databases such as SQL, Oracle, Access and Sybase.
- **Links test plans to requirements :** TestDirector connects requirements directly to test cases, ensuring that functional requirements have been covered by the test plan.
- **Integrates with Microsoft Office :** TestDirector can import requirements and test plans from Microsoft Office, preserving your investment and accelerating your testing process.
- **Manages manual and automated tests :** TestDirector stores and runs both manual and automated tests, and can help jumpstart a user's automation project by converting manual tests to automated test scripts.
- **Accelerates testing cycles :** TestDirector's TestLab manager accelerates the test execution cycles by scheduling and running tests automatically unattended, even overnight. The results are reported into TestDirector's central repository, creating an accurate audit trail for analysis.
- **Supports test runs across boundaries :** TestDirector allows testers to run tests on their local machines and then report the results to the repository that resides on a remote server.
- **Integrates with internal and third-party tools :** Documented COM API allows TestDirector to be integrated both with internal tools (e.g., WinRunner and LoadRunner) and external third-party lifecycle applications.

- **Enables structured information sharing :** TestDirector controls the information flow in a structured and organized manner. It defines the role of each tester in the process and sets the appropriate permissions to ensure information integrity.
- **Provides Analysis and Decision Support Tools :** TestDirector's integrated graphs and reports help analyze application readiness at any point in the testing process. Using information about requirements coverage, planning progress, run schedules or defect statistics, managers are able to make informed decisions on whether the application is ready to go live.
- **Provides easy defect reporting :** TestDirector offers a defect tracking process that can identify similar defects in a database.
- **Generates customizable reports :** TestDirector features a variety of customizable graphs and reports that provide a snapshot of the process at any time during testing. You can save your favorite views to have instant access to relevant project information.
- **Supports decision-making through analysis :** TestDirector helps you make informed decisions about application readiness through dozens of reports and analysis features. Provides Anytime, Anywhere Access to Testing Assets Using TestDirector's Web interface, testers, developers and business analysts can participate in and contribute to the testing process by collaborating across geographic and organizational boundaries.
- **Provides Traceability :** Throughout the Testing Process: TestDirector links requirements to test cases, and test cases to issues, to ensure traceability throughout the testing cycle. When requirement changes or the defect is fixed, the tester is notified of the change.
- **Integrates with Third-Party Applications :** Whether an individual uses an industry standard configuration management solution, Microsoft Office or a homegrown defect management tool, any application can be integrated into TestDirector. Through the open API, TestDirector preserves the users' investment in their existing solutions and enables them to create an end-to-end lifecycle-management solution.
- **Facilitates Consistent and Repetitive Testing Process :** By providing a central repository for all testing assets, TestDirector facilitates the adoption of a more consistent testing process, which can be repeated throughout the application lifecycle or shared across multiple applications or lines of business (LOB). Testing Process Test management is a method for organizing application test assets such as test requirements, test plans, test documentation, test scripts or test results to enable easy accessibility and reusability. Its aim is to deliver quality applications in less time. The test management process is the main principle behind Mercury Interactive's

TestDirector. It is the first tool to capture the entire test management process requirements management, test planning, test execution and defect management in one powerful, scalable and flexible solution.

- **Managing Requirements :** Requirements are what the users or the system needs. Requirements management, however, is a structured process for gathering, organizing, documenting and managing the requirements throughout the project lifecycle. Too often, requirements are neglected during the testing effort, leading to a chaotic process of fixing what you can and accepting that certain functionality will not be verified.
- In many organizations, requirements are maintained in Excel or Word documents, which makes it difficult for team members to share information and to make frequent revisions and changes. TestDirector supports requirements-based testing and provides the testing team with a clear, concise and functional blueprint for developing test cases.
- Requirements are linked to tests that is, when the test passes or fails, this information is reflected in the requirement records. You can also generate a test based on a functional requirement and instantly create a link between the requirement, the relevant test and any defects that are uncovered during the test run. Test Planning Based on the requirements, testers can start building the test plan and designing the actual tests.
- Today, organizations no longer wait to start testing at the end of the development stage, before implementation. Instead, testing and development begin simultaneously. This parallel approach to test planning and application design ensures that testers build a complete set of tests that cover every function the system is designed to perform.
- TestDirector provides a centralized approach to test design, which is invaluable for gathering input from different members of the testing team and providing a central reference point for all of your future testing efforts. In the Test Plan module, you can design tests manual and automated document the testing procedures and create quick graphs and reports to help measure the progress of the test planning effort.
- **Running Tests :** After you have addressed the test design and development issues and built the test plan, your testing team is ready to start running tests. TestDirector can help configure the test environment and determine which tests will run on which machines. Most applications must be tested on different operating systems, different browser versions or other configurations.
- In TestDirector's Test Lab, testers can set up groups of machines to most efficiently use their lab resources. TestDirector can also schedule automated tests, which saves testers time by running multiple tests simultaneously across multiple machines on the network.

- Tests with TestDirector can be scheduled to run unattended, overnight or when the system is in least demand for other tasks. For both manual and automated tests, TestDirector can keep a complete history of all test runs. By using this audit trail, testers can easily trace changes to tests and test runs.
- Managing Defects : The keys to creating a good defect management process are setting up the defect workflow and assigning permission rules.
- With TestDirector, you can clearly define how the lifecycle of a defect should progress, who has the authority to open a new defect, who can change a defect's status to "fixed" and under which conditions the defect can be officially closed.
- TestDirector will also help you maintain a complete history and audit trail throughout the defect lifecycle. Managers often decide whether the application is ready to go live based on defect analysis.
- By analyzing the defect statistics in TestDirector, you can take a snapshot of the application under test and see exactly how many defects you currently have, their status, severity, priority, age, etc.
- Because TestDirector is completely Web-based, different members of the team can have instant access to defect information, greatly improving communication in your organization and ensuring everyone is up to date on the status of the application.

4.5 IBM RATIONAL

GQ. What is IBM Rational ?

- IBM Rational Developer for i provides an integrated development environment (IDE) to create, maintain and modernize applications on the IBM i platform. It integrates developer tools such as search, edit, build and analysis, refactoring capabilities and debuggers with the widely used Eclipse framework for faster, easier application development and modernization. Rational Developer for i integrates with IBM Rational Team Concert for better application lifecycle planning and management.
- IBM Rational Developer for z System is an integrated development tool that you can use to design, develop, deploy, and test mainframe software applications, Web, and composite applications.
- IBM Rational Developer for z Systems provides a common workbench and tools for the following:
  - Development of COBOL, PL/I, C++, high-level assembler, Java and Web Services programs
  - Interactive access to z/OS systems
  - Support for IMS, CICS, Batch, USS, DB2 stored procedures, application servers
  - CICS and IMS application development for Web Services and SOA

- IBM Rational Developer for z Systems offers integration for a variety of source code management (SCM) tools in addition to providing a framework for creating your own SCM integration.

There are two IBM Rational Developer for z Systems offerings :

1. **Developer for z Systems**

   Provides z Systems developers with tools to make traditional mainframe development and integrated mixed workload faster and more efficient.

2. **Rational Developer for the Enterprise**

   Unified development tool for making traditional mainframe development, Web development, and integrated mixed workload or composite development faster and more efficient even across hardware platforms.

4.6 BRIEF HISTORY OF THE SELENIUM PROJECT

GQ. Brief History selection and cost in testing tool, guidelines for testing tools.

GQ. What is Selenium Project ?

- Selenium (the testing framework, not the mineral you get from eating clams) came from? Here's a short history of the technology, from its origins more than a decade ago as a proprietary tool through the present era of Web driver. Selenium originated in elder days - by which I mean 2004 - as a tool for testing web applications. It was developed by Jason Huggins, a programmer at Thought-Works.
- That Selenium originated at Thought-Works is interesting. While no one in 2004 was talking about Agile infrastructure, Thought-Works was the place where Martin Fowler made his career. Fowler went on to become one of the major thought leaders behind the migration to micro services. While Fowler can't take credit for Selenium, it seems fitting that the tool, which is an important part of automated testing for Develops-inspired workflows today, originated in the same place from which the Agile infrastructure revolution later emerged.

4.6.1 Open Source Selenium

- At first, Selenium was used only internally by Thought Works employees. But that changed by the end of 2004, when the tool was open-sourced. I don't know exactly when Selenium became an open source tool, since the earliest emails relating to the open-sourcing do not seem to exist anymore (at least not publicly). But it was apparently no later than late November 2004. That's when the first extant Selenium development emails were being exchanged, and people were talking about checking out the Selenium code via Subversion.
- Selenium at this point remained an imperfect tool. It suffered from some bugs that affected testing for certain browser environments. And thanks to the inefficiencies of the waterfall-style software development practices of the time, bug fixes were slow to reach users.

- As a Selenium user noted on November 29, 2004. Last but not least, the move placed Selenium within the rapidly growing stack of open source applications at the time. The early 2000s were the era when companies like Red Hat were showing that Linux and other software that was given away for free could have huge commercial value.
- It was also when open source web browsers, namely Mozilla Firefox, and word processors, such as Open Office, were giving closed-source tools a run for their money. And open source web browsers like Apache HTTP had already held a majority of market share for years. Against this backdrop, open-sourcing Selenium only made sense.

Selenium Grows Up

- Within a year of its release as an open source tool, Selenium had evolved significantly. By October 2005, developers were dreaming up ambitious "grand plans" for the tool. They envisioned adding sophisticated new features that would extend Selenium beyond its original mission as a basic web app testing tool. Those included things like support for testing framed applications and cross-platform testing.
- The evolution of Selenium during this period was helped by the fact that Jason Huggins, the original Selenium developer, moved in 2007 to Google, where he was able to continue work on the tool.

Selenium Meets Web-driver

- Simon Stewart developed another testing tool for web apps called Web Driver. Web Driver's debut signaled a desire for features that were not available in Selenium. And for a short time, the two tools competed. But in 2011, the projects were merged to form one web testing tool to rule them all.
- The combination of Selenium and Web Driver became Selenium 2.0, which debuted in July 2011. The new release paired the Web Driver APIs that are familiar to Selenium users today with the original Selenium feature set. Jason Huggins & Simon Stewart S

Selenium Present, Selenium Future

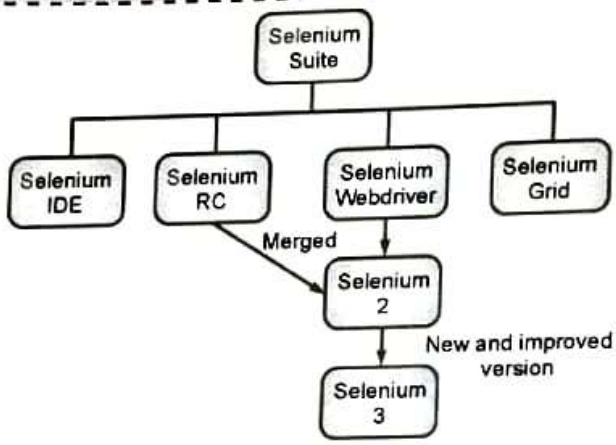
- The demands of automated testing continue to change. It's a safe bet that Selenium will, too. One important trend that is likely to shape Selenium development going forward is the demand for ever-more efficient automated testing.
- The Selenium ecosystem has offered some automation options for a while thanks to Selenium Grid and other tools. But as the movement increases pressure on development teams to test and deliver software even faster than they already do, techniques for speeding tests, such as by offloading them to the cloud and running them in parallel, will remain key. Shift-left testing has also become an important part of the automated testing conversation.
- Selenium is already well suited for shift-left testing, which refers to the practice of performing tests earlier in the development cycle, in order to identify bugs before they

slow development. But Selenium users have to choose to take advantage of Selenium in the right way for this purpose.

- Optimizing Selenium today is easier thanks to a rich ecosystem of plugins and integrations that simplify the task of working Selenium into the software delivery pipeline.
- Since development workflows are now more complex than they have ever been, and will probably grow yet more complex over time, the ecosystem surrounding Selenium is poised to remain essential in helping Selenium to remain relevant for modern application testing.

4.6.2 Selenium's Tool Suite

GQ. Define Selenium's Tool Suite. List and explain Core Components.



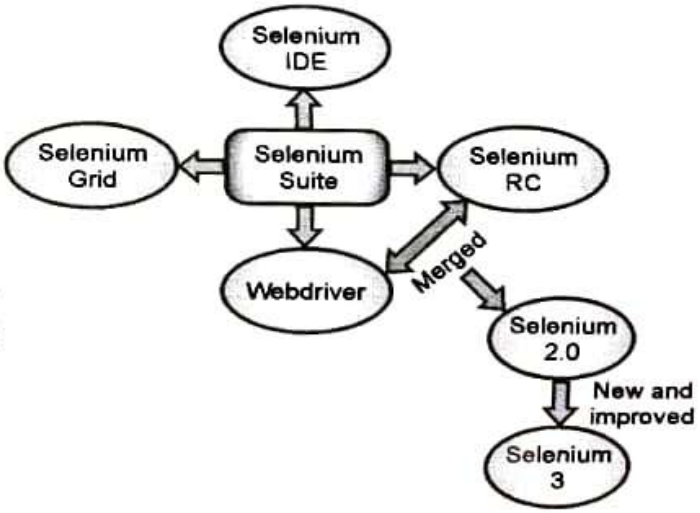
(103)Fig. 4.6.1 : Selenium Suite Tree Diagram

4.6.3 Selenium IDE

UQ. What is Selenium's IDE explain in detail. **MU - Oct. 18**

- Selenium IDE (Integrated Development Environment) is an open source web automation testing tool under the Selenium Suite. Unlike Selenium WebDriver and RC, it does not require any programming logic to write its test scripts rather you can simply record your interactions with the browser to create test cases. Subsequently, you can use the playback option to re-run the test cases.
- Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite. It is a browser plugin to record and playback the operations performed on the browser. Selenium IDE plugins are available for Chrome and Firefox browsers. It doesn't support the programming features. Selenium is the language which is used to write test scripts in Selenium IDE.

- As a Firefox plugin, Selenium Integrated Development Environment (IDE) can be used to create a test script prototype quickly and easily. It can record human testers' actions as a script while the tester runs the test case manually. Selenium IDE is a rapid prototyping tool for building test scripts within very less amount of time. It allows you to record, edit and debug the test case by providing the very simple to use components. This tool will be most helpful for beginners to learn the commands used by selenium while recording the test case. Although it was available as Firefox addon for a long time, it also available on chrome recently
- With this tool, you can easily record the test case and able to play back any number of time whenever you will require. You can easily export the recorded scripts as reusable scripts in one of any programming languages that support. Although it was very simple and easy to use tool for beginners, it's having some limitations. It only allows you to record and playback very simple test cases. It will not possible to test dynamic websites or web applications. It's neither be scripted using any programming logic not support data-driven testing.
- The recorded test script can be executed at a later point in time for the regression test automatically. This tool can access the browser's DOM elements with the use of JavaScript.
- It also provides a flexible interface for testers to create or update test cases. Thought Works Company introduces selenium IDE in 2006 andimplemented in the Firefox browser, which provides record and playback functionality to the test scripts.
- Selenium-IDE is the simplest tool of Selenium community. Selenium-IDE allows software testers to export recorded scripts in many languages like HTML, Java, Ruby, PHP, Python, C#, and Test-NG.
- Selenium–IDE supports six locators, i.e., - Id, Name, X Path, CSS Selector, Link Text, DOM.



(1D4)**Fig. 4.6.2 : Selenium Suite**

Pros

1. It is simple, easy to install and use.
2. Built-in test results reporting and help modules.
3. Test Cases can be exported to usable formats in the Selenium RC and WebDriver.

4. No previous knowledge of programming needed, though basic knowledge of HTML and DOM required.
5. Can easily export the recorded tests in different programming languages such as Ruby, Python, etc.

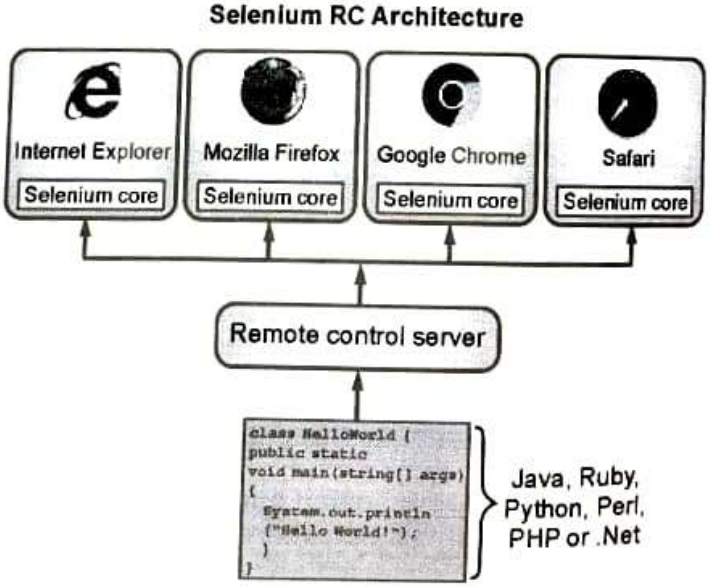
Cons

1. It is only available for Firefox.
2. The execution of test cases is slow as compared to RC and WebDriver.
3. Data-driven testing is not supported.
4. It is not able to test dynamic web applications.

4.6.4 Selenium RC

UQ. What is Selenium's RC explain in detail. **MU - May 18**

- Selenium RC is the main feature in the Selenium. A tester can use it to simulate user actions such as input data, submit a form, and click a button in web browsers.
- Selenium RC was the first tool used on selenium project. It was the core application written in Java as a programming language.
- This tool will accept commands for the browser via HTTP request. It consists of two components which are selenium RC server and RC client. Where RC server will communicate with HTTP/GET/POST request while the RC client will include programming codes.
- You will be able to write an automated test in most of the programming languages such as Java, JavaScript, Ruby, PHP, Python, and Perl and C #.
- Selenium RC is the first open-source tooling in selenium community which is introduced by Thought Works Company in 2004. Selenium RC doesn't have a record and playback features.
- Selenium RC cannot execute test script with selenium server. Selenium –RC supports multiple languages (java, C++, python), multiple operating systems (Windows, Linux), and multiple browsers (internet explorer, Google Chrome).



(105)Fig. 4.6.3 : Core Components of Selenium RC

- Although selenium RC was the main project for a long time until the selenium 2 was released. It was officially deprecated when selenium version 2 was released.
- It will support mostly in maintenance mode that will provide some feature that is not available in Selenium 2.
- When a web browser is being loaded in a test script, it injects a suite of JavaScript (JS) into it. Then use those JavaScript's programming to interact with the different web browsers.

Usage of Selenium RC

1. Tester writes a test case script with the supported programming language API.
2. The test script sends a command to the RC server.
3. RC server receives these commands and triggers selenium core from executing the commands and interacting with the browser page web elements.

Pros

1. It supports cross-browser testing.
2. It supports data-driven testing.
3. Execution speed is more as compared to IDE.
4. It supports conditional operations and iterations.

Cons

1. Slower execution speed as compared to Web-Driver.
2. Browser interaction is less realistic.
3. Programming knowledge required.

4.6.5 Selenium Web-Driver

UQ. What is Selenium's Web-Driver explain in detail. **MU - May 19**

- Web-Driver is the new feature added in the Selenium 2. It aimed to deliver an easy and helpful programming interface to resolve the limitations of Selenium RC programming API. Different from RC, Web-Driver uses browser native support to interact with the web pages.
- So different browsers have different web driver libraries and different features too. All these are decided by the web browser that runs the test cases. The implementation of Web-Driver is much more related to the web browser.
- So, are the following Web-Driver Drivers ?

1. **HttpUnit Driver :** This is one of the fastest and reliable Web-Driver implementations. Based on the HttpUnit, it can run across Linux, Windows, and Mac because of its pure java implementation.

2. **Firefox Driver :** It is easy to configure and use. It is being used to run the test scripts in the Firefox web browser and does not require extra configuration to use.
3. **Chrome Driver :** It is being used to run a test script on the Google Chrome web browser that needs more configurations to use.
4. **Internet Explorer Driver :** It is being used to run the test script in the Internet Explorer web browser that needs more configurations to use. It can only run in Windows OS, slower than the Chrome and Firefox Web Driver. Selenium Web-Driver is also called Selenium -2, and Google introduced it in 2008. Selenium Web-drivers is just a collection of core java interface. In comparison to Selenium RC, Selenium web driver is more powerful and faster tool because it directly calls to the web browser. Web-driver supports multiple browsers, multiple operating systems, and multiple languages.

- Selenium Web Driver (Selenium 2) is the successor to Selenium RC and is by far the most important component of Selenium Suite. Selenium Web Driver provides a programming interface to create and execute test cases. Test scripts are written in order to identify web elements on web pages and then desired actions are performed on those elements.
- Selenium WebDriver performs much faster as compared to Selenium RC because it makes direct calls to the web browsers. RC on the other hand needs an RC server to interact with the web browser.
- Since, WebDriver directly calls the methods of different browsers hence we have separate driver for each browser. Some of the most widely used web drivers include :
  - Mozilla Firefox Driver (Gecko Driver)
  - Google Chrome Driver
  - Internet Explorer Driver
  - Opera Driver
  - Safari Driver
  - HTML Unit Driver (a special headless driver)

☞ **Pros**

1. No separate components such as the RC server are needed.
2. Execution time is faster as compared to Web-Driver and RC.
3. It supports testing on different platforms such as Android, iOS, Windows, Mac, and Linux.

☞ **Cons**

1. No mechanism to track runtime messages.

2. Image testing is not available.
3. Prior knowledge of programming required.

4.6.6 Selenium Grid

UQ. What is Selenium's Grid explain in detail. MU - May 18

- With the Selenium Grid feature, test scripts can run on multiple machines at the same time, which reduces the total test scripts run time.
- Thus helps to find the bug more quickly because the test cases run more quickly. This is suitable for a large application with too many test scripts to run.
- You can also choose to run test scripts on different web browsers and on different machines. You can configure the browser version, Operating System, and machine to run the test case by using the Selenium RC capabilities.
- Selenium grid is the part of selenium version1 that combined with selenium RC to scale for large test suit and able to run tests on remote machines. We can execute multiple test cases at the same time on different remote machines.
- If you run your test cases on multiple environments, you will use the different remote machine to run the tests at the same time.
- While testing with selenium grid, one server acts as the hub where other machines contact the hub for obtaining browser access. This ability to run test cases on remote machine may be most helpful for spreading the testing load across several machines having different environments or platforms.
- So this feature of selenium grid will help you to speed up your test automation process. Selenium web driver is the latest addition among the tools in selenium tool suite. It is the upgraded version of Selenium RC.
- So it is much faster than selenium RC since it makes direct calls to the browsers. Unlike the selenium RC, selenium webdriver does not require any special server for running or executing test cases.
- Selenium webdriver supports all of the main browsers such as Mozilla Firefox, Google Chrome and Internet Explorer and Safari.
- It allows you to write test scripts on different programming languages such as Java, C#, Ruby, Python, Perl etc. Although it overcomes all the limitations of Selenium RC, generating a detailed test report is not possible with selenium webdriver yet.
- Selenium Grid is also an important tool of Selenium Suite, which allows us to run our tests script on different machines against different browsers simultaneously. Selenium Grid proceeds from the Hub-Node Architecture to achieve parallel execution of test scripts.

- Selenium Grid is divided into two parts-
  - **Grid-1 :** Grid-1 introduced by Thought works company in 2004.
  - **Grid-2 :** Google Company introduced it in 2008.

☞ **Pros**

1. Selenium Grid offers tools needed to diagnose the failures and rebuild a similar environment for the new test execution.
2. Selenium Grid saves time extremely as it uses the Hub-Node design.
3. It supports the simultaneous execution of test cases in multiple browsers and environments.

☞ **Cons**

1. The code executes only on the local machines where the test cases are launched.
2. Considerable efforts and time are must for the initial operation of parallel testing.
3. The remote machine only receives the browser control commands.

4.7 SELF-LEARNING TOPICS

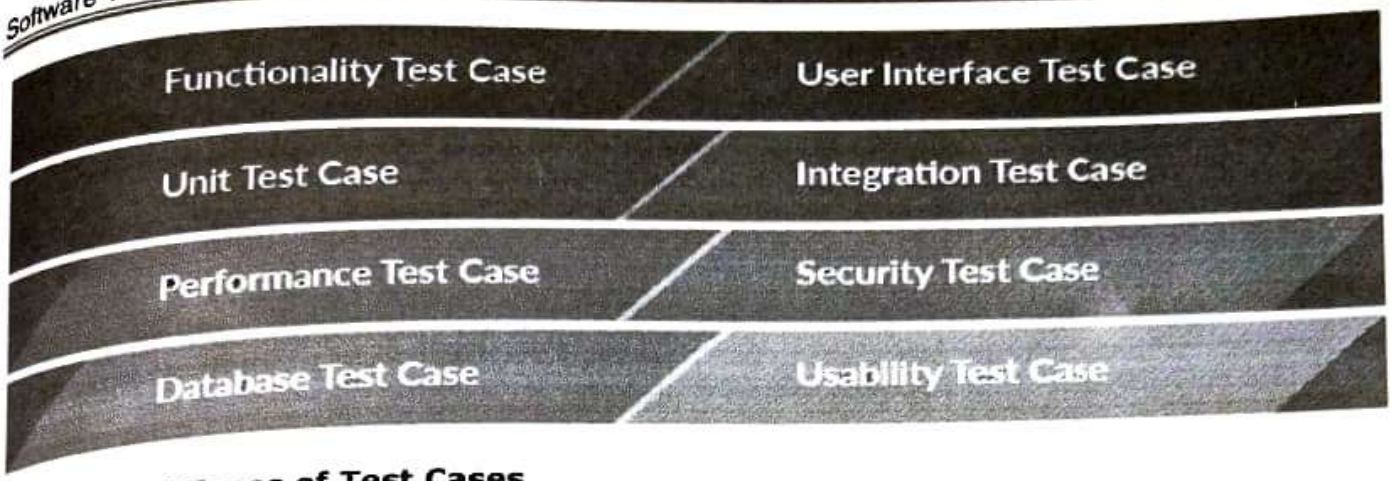
4.7.1 Write Down Test Cases, Execute and Manage Using Studied Tools

☞ **What is a Test Case in Software ?**

- A test case is exactly what it sounds like: a test scenario measuring functionality across a set of actions or conditions to verify the expected result. They apply to any software application, can use manual testing or an automated test, and can make use of test case management tools.
- A key thing to remember when it comes to writing test cases is that they are intended to test a basic variable or task such as whether or not a discount code applies to the right product on an e-commerce web page. This allows a software tester more flexibility in how to test code and features.

☞ **Test Script vs. Test Case**

- The difference between test cases vs. test scripts should also be clarified. A test script is a short program meant to test certain functionality. A test case is a document with steps to be completed as planned out ahead of time.
- Consider test cases as a meticulously planned trip and test scripts to be more like a quick trip to the grocery store.



Different Types of Test Cases

Test cases can measure many different aspects of code. The steps involved may also be intended to induce a Fail result as opposed to a positive expected result such as when a user inputs the wrong password on a login screen.

Some common test case examples would be the following :

Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Area should accommodate up to 20 characters	Input up to 20 characters	All 20 characters in the request should be appropriate	Pass or Fail
Security	Verify password rules are working	Create a new password in accordance with rules	The user's password will be accepted if it adheres to the rules	Pass or Fail
Usability	Ensure all links are working properly	Have users click on various links on the page	Links will take users to another web page according to the on-page URL.	Pass or Fail

Test cases can be applied to any number of features found in any given software. Some of the most popular include:

- API testing example – See it in action.
- UI testing example – See it in action.
- Unit testing example – See it in action.
- Load & performance testing example – See it in action.
- Security testing
- SQL queries
- Low-code application testing

☞ A Popular Test Case Example

- Test cases come in handy in a variety of software scenarios. Everything from banking to personal software requires a test case application. For example, if the goal is to have encrypted, sensitive data, the software needs to have features that work as intended.
- But functional testing is just one aspect of writing a test case. Software testing should robustly challenge every aspect of the code from performance to compatibility to security. That's why personal encryption software needs to be tested so thoroughly — especially when it comes to things like Web APIs.

☞ How to Write Software Test Cases

Writing test cases varies depending on what the test case is measuring or testing. This is also a situation where sharing test assets across dev and test teams can accelerate software testing. But it all starts with knowing how to write a test case effectively and efficiently.

Test cases have a few integral parts that should always be present in fields. However, every test case can be broken down into 8 basic steps.

▶ Step 1 : Test Case ID

Test cases should all bear unique IDs to represent them. In most cases, following a convention for this naming ID helps with organization, clarity, and understanding.

▶ Step 2 : Test Description

This description should detail what unit, feature, or function is being tested or what is being verified.

▶ Step 3 : Assumptions and Pre-Conditions

This entails any conditions to be met before test case execution. One example would be requiring a valid Outlook account for a login.

- **Step 4 : Test Data**

  This relates to the variables and their values in the test case. In the example of an email login, it would be the username and password for the account.

- **Step 5 : Steps to be Executed**

  These should be easily repeatable steps as executed from the end user's perspective. For instance, a test case for logging into an email server might include these steps:

  1. Open email server web page.
  2. Enter username.
  3. Enter password.
  4. Click "Enter" or "Login" button.

- **Step 6 : Expected Result**

  This indicates the result expected after the test case step execution. Upon entering the right login information, the expected result would be a successful login.

- **Step 7 : Actual Result and Post-Conditions**

  As compared to the expected result, we can determine the status of the test case. In the case of the email login, the user would either be successfully logged in or not. The post-condition is what happens as a result of the step execution such as being redirected to the email inbox.

- **Step 8 : Pass/Fail**

  Determining the pass/fail status depends on how the expected result and the actual result compare to each other.

  Same result = Pass

  Different results = Fail

Standard Unit Test Case Format

Each part of a well-written unit test will define several core aspects including:

1. Functions performed by the test
2. Data used in the test
3. Expected result from the test execution
4. Ensuring the test was executed in isolation from other parts of the codebase

It's important to know that the standard format of well-written tests is composed of the following parts:

- Meaningful test method name
- Controlled data or mocks to be used for testing
- Method or unit under test (the part of code we are testing)
- Applying an assertion
- Executing the unit test in isolation

```
/**
* Parasoft Jtest UTA : Test for isOverdrawn (int)
*
* @see examples.nbank.Account#isOverdrawn(int)
* @author bmcmullin
* /
@Test
public void testIsOverdrawnBalance500AssertTrue ( ) throws Throwable {
// Given
Customer customer = mockCustomer ( );
Int initial_balance = 0 ;
Account underTest = new Account (customer, initial_balnace) ;
// When
int balance = – 500 ;
boolean result = underTest.isOverdrawn (balance) ;
// Then
assertTrue (result);
}
```

☞ Is There a Test Case Template ?

As mentioned, there is a standard test case format. However, the test case template would likely vary from company to company and even from team to team. Instead, a test case template is the document with a list of test scenarios and subsequent test cases.

☞ Quality Test Case Example

Thogh test cases will vary based on the type of testing and overall field of testing, building a quality test case comes down to those few reliable items above. Remember: the name of the test method must include the method or unit under test and what is the expected outcome.

It should also be noted that each unit should be tested in isolation. In this case, "isolation" means keeping tests focused as much as possible in order to execute only the piece of the application we are testing for.

This example comes from a banking-related test case :

```
@Test
Public void testIsOverdrawnBalances500AssertTrue( )
```

With this method name, we know that this is a unit test that is:

- Testing the method 'isOverDrawn()'.
- The balanced used for the controlled data was 500.
- The expected result is true.

A meaningful method name allows anyone reviewing the results to understand what the unit test was testing for. Moreover, it signals the data to be tested, the expected result, and what was tested.

If the test fails, knowing the expected result is critical in allowing for easier troubleshooting and ensuring no regressions are introduced.

Test Case Data

- The data used needs to be enough to execute the test. For unit testing, we want to make it as simple as possible to test the most basic unit of our application. The data could be as simple as making a string or object variable for which you can control the data. Or a mock framework can be used for the test if a dependency is not available or you need that dependency to be in a specific state.
- Having just enough to test that one part if sufficient. You DO NOT need to configure every piece of the application for the test to run.
- All of this affects how the unit test will behave since this is the data being used for unit test execution. As such, this part of unit testing is the most time consuming as it requires some understanding of the code you are testing to know what data to use for testing.
- Keep it simple by using just the parts needed for the code being tested. Mocks are very useful in this phase as they allow you to control how methods from those objects will behave when interacting with your test.

For example, given the following data :

```
// Given
Customer customer  = mockCustomer ( ) ;
int initial_balnace = 0 ;
Account underTest = new Account (customer, initial_balance) ;
// When
int balance = – 500;
boolean result = underTest.isOverdrawn (balance) ;
```

- We avoided the "real customer class" by using a mock for the "customer class" for testing isolation. We do not want to introduce nor configure another object for this test as it adds another layer of maintainability for that object, and it is not affecting the result of the method under test.
- The next variable to be created is the "initial balance"—something known due to knowledge of the code. The next line shows the Account object being created along with the mock and the Initial Balance to prepare the method we are testing for with the data we just used.
- So in this example, the account object is configured with the mock customer since we do not care about the customer object's data and we passed an initial balance that we can control for our test.

- The next line defines the input as the method under test requires a number to be used. We defined the "balance" to be used in the method we are testing for. Then the method is executed with the result of the method being stored in our variable for us to use later.

☞ Applying an Assertion

Once the test can complete successfully (as in it runs from start to finish with no exceptions or errors), then it is time to apply an assertion to the unit test. Without the assertion, the unit test is meaningless since there is nothing you are enforcing to ensure it is working as intended.

Collecting coverage of what lines were executed does tell what was executed but it does not provide enough detail to determine the following:

- If the code is behaving as expected.
- If the code meets quality targets.
- If the data returned is the expected data.

An assertion can be as basic as :

```
\\ Then
assertTrue(result);
```

- As long as the unit test contains one assertion that is checking the method under test result, this is a meaningful unit test.

```
/**
 * Parasoft Jtest UTA: Test for isOverdrawn(int)
 *
 * @see examples.nbank.Account#isOverdrawn(int)
 * @author bmcmullin
 */
@Test
public void testIsOverdrawnBalance500AssertTrue() throws Throwable {

    // Given
    Customer customer = mockCustomer();
    int initial_balance = 0;
    Account underTest = new Account(customer, initial_balance);
    // When
    int balance = -500;
    boolean result = underTest.isOverdrawn(balance);

    // Then
    assertTrue(result);
}
```

Chapter Ends...

❑❑❑

Module 5

CHAPTER 5 Testing for Specialized Environment

University Prescribed Syllabus

5.1 Agile Testing, Agile Testing Life Cycle, Testing in Scrum phases, Challenges in Agile Testing

5.2 **Testing Object** Oriented Software : OOT Basics, Object-oritented Testing

5.1 Introduction to Agile Testing 5-3

GQ. What Is Agile Testing, Agile Testing Life Cycle ? 5-3

GQ. Explain Testing in Scrum phases, Challenges in Agile Testing. 5-3

5.1.1 What is Agile Testing ? 5-3

5.1.2 Agile Testing Vs. +9 Waterfall Testing 5-4

5.1.3 Agile Testing Principles 5-4

5.1.4 Agile Testing Activities 5-5

5.2 Web based System 5-6

GQ. Explain Web based Systems :

(i) Web based system, web technology evaluation

(ii) Traditional software and web based software

(iii) Challenges in testing for web based software

(iv) Testing web based testing 5-6

5.2.1 How to test Web Application ? 5-6

5.3 Self-learning Topics 5-10
5.3.1 UAT (Acceptance Testing) 5-11
5.3.2 DevOps 5-11
5.3.3 Regression Testing 5-12
5.3.4 Automation Testing 5-12
5.3.5 Integration Testing 5-13
5.3.6 User Testing 5-13
5.3.7 Accessibility Testing 5-14
5.3.8 Performance Testing 5-14
5.3.9 Selenium Testing 5-15
5.3.10 Scriptless Test Automation 5-15
5.3.11 Artificial Intelligence 5-16
5.3.12 Robotic Process Automation (RPA) 5-16
5.3.13 Infrastructure as Code (IaC) 5-17
5.3.14 Pen-Testing-as-a-Service (PTaaS) 5-17
5.4 Testing Object Oriented Software: OOT Basics, Object oriented Testing 5-18
5.4.1 Object Oriented Testing Basic 5-20
5.4.2 Object-Oriented Metrics 5-22
5.4.3 Object Oriented Testing 5-23
▶ **Chapter Ends** **5-26**

5.1 INTRODUCTION TO AGILE TESTING

GQ. What Is Agile Testing, Agile Testing Life Cycle ?
GQ. Explain Testing in Scrum phases, Challenges in Agile Testing.

- Agile is an iterative development methodology, where both development and testing activities are concurrent. Testing is not a separate phase; Coding and Testing are done interactively and incrementally, resulting in quality end product, which the meets customer requirements.
- Further, continuous integration results in early defect removal and hence time, effort and cost savings.

Agile Manifesto

The Agile Manifesto was published by a team of software developers in 2001, highlighting the importance of the development team, accommodating changing requirements and customer involvement.

The Agile Manifesto is

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value :

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

5.1.1 What is Agile Testing ?

- Agile Testing is a software testing practice that follows the principles of agile software development.
- Agile Testing involves all members of the project team, with special expertise contributed by testers. Testing is not a separate phase and is interwoven with all the development phases such as requirements, design and coding and test case generation. Testing takes place simultaneously through the Development Life Cycle.
- Furthermore, with testers participating in the entire Development Lifecycle in conjunction with cross-functional team members, the contribution of testers towards building the software as per the customer requirements, with better design and code would become possible.
- Agile Testing covers all the levels of testing and all types of testing.

5.1.2 Agile Testing Vs. +9 Waterfall Testing

- In a Waterfall Development methodology, the Development Life Cycle activities happen in phases that are sequential. Thus, testing is a separate phase and gets initiated only after the completion of the development phase.
- Following are the highlights of differences between Agile Testing and Waterfall Testing :

Agile Testing	Waterfall Testing
Testing is not a separate phase and occurs concurrently with development.	Testing is a separate phase. All levels and types of testing can begin only after the completion of development.
Testers and developers work together.	Testers work separately from developers.
Testers are involved in coming up with requirements. This helps in requirements mapping to the behaviors in the real world scenario and also framing the acceptance criteria. Also, logical Acceptance Test Cases would be ready along with the requirements.	Testers may not be involved in the requirements phase.
Acceptance Testing is done after every iteration and customer feedback is sought.	Acceptance Testing is done only at the end of the project.
Every iteration completes its own testing thus allowing regression testing to be implemented every time new functions or logic are released. Regression Testing can be implemented only after the completion of development.	
No time delays between coding and testing.	Usual time delays between coding and testing.

- Continuous testing with overlapping test levels. Testing is a timed activity and test levels cannot overlap.
- Testing is a best practice. Testing is often overlooked.

5.1.3 Agile Testing Principles

The principles of Agile testing are :

- **Testing moves the project forward :** Continuous testing is the only way to ensure continuous progress. Agile Testing provides feedback on an ongoing basis and the final product meets the business demands.

- **Testing is not a phase :** Agile team tests alongside the development team to ensure that the features implemented during a given iteration are actually done. Testing is not kept for a later phase.
- **Everyone tests :** In agile testing, the entire team including analysts, developers, and testers test the application. After every iteration, even the customer performs the User Acceptance Testing.
- **Shortening Feedback Loops :** In Agile Testing, the business team get to know the product development for each and every iteration. They are involved in every iteration. Continuous feedback shortens the feedback response time and thus the cost involved in fixing it is less.
- **Keep the Code Clean :** The defects are fixed as they are raised within the same iteration. This ensures clean code at any milestone of development.
- **Lightweight Documentation :** Instead of comprehensive test documentation,
- **Agile testers :** Use reusable checklists to suggest tests.
- Focus on the essence of the test rather than the incidental details.
- Use lightweight documentation styles/tools.
- Capture test ideas in charters for exploratory testing.
- Leverage documents for multiple purposes.
- **Leveraging one test artifact for manual and automated tests :** Same test script artifact can be utilized for manual testing and as an input for automated tests. This eliminates the requirement of Manual Test Documentation and then an equivalent Automation Test Script.
- **"Done Done," not just done :** In Agile, a feature is said to be done not after development but after development and testing.
- **Test-Last vs. Test Driven :** Test Cases are written along with the requirements. Hence, development can be driven by testing. This approach is called Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD). This is in contrast to testing as a last phase in Waterfall Testing.

5.1.4 Agile Testing Activities

- The Agile Testing Activities at Project Level are :
  - Release Planning (Test Plan)
  - For every Iteration,
  - Agile Testing Activities during an Iteration
  - Regression Testing
  - Release Activities (Test Related)

- The Agile Testing Activities during an iteration include :
  - Participating in iteration planning
  - Estimating tasks from the view of testing
  - Writing test cases using the feature descriptions
  - Unit Testing
  - Integration Testing
  - Feature Testing
  - Defect Fixing
  - Integration Testing
  - Acceptance Testing
  - Status Reporting on Progress of Testing
  - Defect Tracking

5.2 WEB BASED SYSTEM

GQ. Explain Web based Systems :
(i) Web based system, web technology evaluation
(ii) Traditional software and web based software
(iii) Challenges in testing for web based software
(iv) Testing web based testing

- Web Testing, or website testing is checking your web application or website for potential bugs before its made live and is accessible to general public. Web Testing checks for functionality, usability, security, compatibility, performance of the web application or website.
- During this stage issues such as that of web application security, the functioning of the site, its access to handicapped as well as regular users and its ability to handle traffic is checked.

5.2.1 How to test Web Application ?

In Software Engineering, the following testing types/technique may be performed depending on your web testing requirements.

1. Functionality Testing of a Website

- Functionality Testing of a Website is a process that includes several testing parameters like user interface, APIs, database testing, security testing, client and server testing and basic website functionalities. Functional testing is very convenient and it allows users to perform both manual and automated testing. It is performed to test the functionalities of each feature on the website.

- Web based Testing Activities includes : Test all links in your webpages are working correctly and make sure there are no broken links. Links to be checked will include -
  - Outgoing links
  - Internal links
  - Anchor Links
  - Mail To Links
- Test Forms are working as expected. This will include-
  - Scripting checks on the form are working as expected. For example- if a user does not fill a mandatory field in a form an error message is shown.
  - Check default values are being populated
  - Once submitted, the data in the forms is submitted to a live database or is linked to a working email address
  - Forms are optimally formatted for better readability
- Test Cookies are working as expected. Cookies are small files used by websites to primarily remember active user sessions so you do not need to log in every time you visit a website. Cookie Testing will include
  - Testing cookies (sessions) are deleted either when cache is cleared or when they reach their expiry.
  - Delete cookies (sessions) and test that login credentials are asked for when you next visit the site.
- Test HTML and CSS to ensure that search engines can crawl your site easily. This will include
  - Checking for Syntax Errors
  - Readable Color Schemas
  - Standard Compliance. Ensure standards such W3C, OASIS, IETF, ISO, ECMA, or WS-I are followed.
- Test business workflow- This will include
  - Testing your end - to - end workflow/ business scenarios which takes the user through a series of webpages to complete.
  - Test negative scenarios as well, such that when a user executes an unexpected step, appropriate error message or help is shown in your web application.
- Tools that can be used : QTP, IBM Rational, Selenium

2. Usability testing

- Usability Testing has now become a vital part of any web based project. It can be carried out by testers like you or a small focus group similar to the target audience of the web application.
  - **Test the site Navigation :** Menus, buttons or Links to different pages on your site should be easily visible and consistent on all webpages
  - **Test the Content :** Content should be legible with no spelling or grammatical errors.

- Images if present should contain an "alt" text

- Tools that can be used: Chalkmark, Clicktale, Clixpy and Feedback Army

3. Interface Testing

- Three areas to be tested here are - Application, Web and Database Server
  - **Application :** Test requests are sent correctly to the Database and output at the client side is displayed correctly. Errors if any must be caught by the application and must be only shown to the administrator and not the end user.
  - **Web Server :** Test Web server is handling all application requests without any service denial.
  - **Database Server :** Make sure queries sent to the database give expected results.
- Test system response when connection between the three layers (Application, Web and Database) cannot be established and appropriate message is shown to the end user.
- Tools that can be used : AlertFox, Ranorex

4. Database Testing

- Database is one critical component of your web application and stress must be laid to test it thoroughly. Testing activities will include :
  - Test if any errors are shown while executing queries
  - Data Integrity is maintained while creating, updating or deleting data in database.
  - Check response time of queries and fine tune them if necessary.
  - Test data retrieved from your database is shown accurately in your web application
- Tools that can be used: QTP, Selenium

5. Compatibility testing

- Compatibility tests ensures that your web application displays correctly across different devices. This would include :
  - **Browser Compatibility Test :** Same website in different browsers will display differently. You need to test if your web application is being displayed correctly across browsers, JavaScript, AJAX and authentication is working fine. You may also check for Mobile Browser Compatibility.
- The rendering of web elements like buttons, text fields etc. changes with change in Operating System. Make sure your website works fine for various combination of Operating systems such as Windows, Linux, Mac and Browsers such as Firefox, Internet Explorer, Safari etc.

Tools that can be used : NetMechanic

6. **Performance Testing**

- This will ensure your site works under all loads. Software Testing activities will include but not limited to :
  - Website application response times at different connection speeds
  - Load test your web application to determine its behavior under normal and peak loads
  - Stress test your web site to determine its break point when pushed to beyond normal loads at peak time.
  - Test if a crash occurs due to peak load, how does the site recover from such an event
  - Make sure optimization techniques like gzip compression, browser and server side cache enabled to reduce load times
- Tools that can be used : Loadrunner, JMeter

7. **Security testing**

- Security Testing is vital for e-commerce website that store sensitive customer information like credit cards. Testing Activities will include :
  - Test unauthorized access to secure pages should not be permitted
  - Restricted files should not be downloadable without appropriate access
  - Check sessions are automatically killed after prolonged user inactivity
  - On use of SSL certificates, website should re-direct to encrypted SSL pages.
- Tools that can be used: Babel Enterprise, BFBTester and CROSS

8. **Crowd Testing**

- You will select a large number of people (crowd) to execute tests which otherwise would have been executed a select group of people in the company. Crowdsourced testing is an interesting and upcoming concept and helps unravel many a unnoticed defects.
- Tools that can be used: People like you and me !!!. And yes , loads of them!
- This concludes the tutorial. It includes almost all testing types applicable to your web application.
- As a Web-tester its important to note that web testing is quite an arduous process and you are bound to come across many obstacles. One of the major problems you will face is of course deadline pressure.
- Everything is always needed yesterday! The number of times the code will need changing is also taxing. Make sure you plan your work and know clearly what is expected of you. Its best define all the tasks involved in your web testing and then create a work chart for accurate estimates and planning.

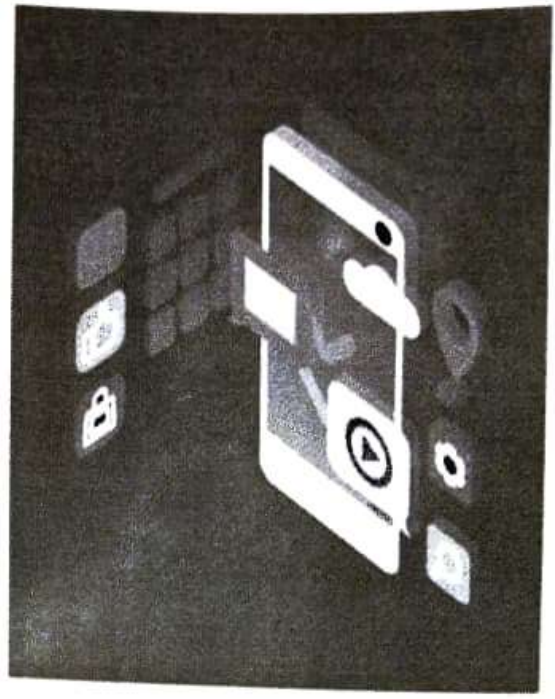
5.3 SELF-LEARNING TOPICS

Study the Recent Technical Papers on Software Testing for Upcoming Technologies (Mobile, Cloud, Blockchain, IoT)

- In recent years, there has been a great evolution in the field of software testing with new trends coming into IT industry services. The introduction of new technologies has brought the latest updates in software design, development, testing, and delivery.
- The top priority of businesses across the globe is cost optimization. In doing so, most of the IT leaders believe in the integration of the latest IT techniques for their organization.
- **Digital transformation** is another important point of focus for the industries and the businesses that are ranking high on cloud computing and business analytics. Factors like quality and reliability are being given major attention, which results in the reduction of software application errors, improving the security and application performance.
- Today, the companies are integrating their testing, earlier in the software development cycle, with **testing methods like Agile**. This also involves the establishment of the T-CoEs to match the testing mechanism with business development building products that are 'Ready for Business'.
- Some companies also hire independent testing companies for their software testing needs. In this way, they incur less cost on testing and do not even require in-house resources.
- There are several other **important trends in the software-testing** world. Thus, there is a strong need to adapt the latest testing trends for all the software industries in the world, which will help them to adapt to the requirements of the modern world.

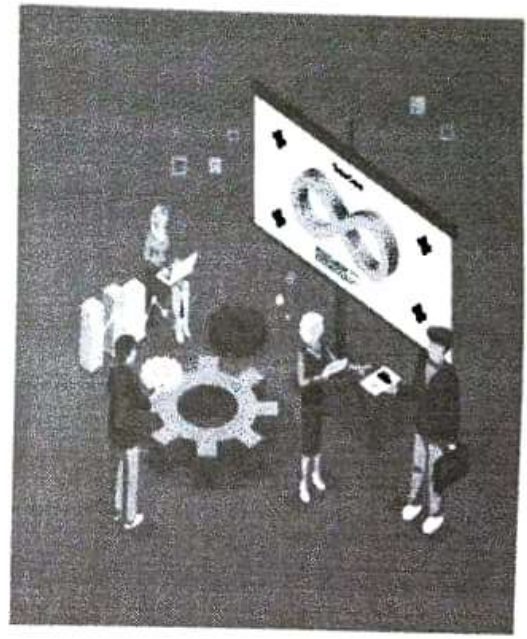
1. UAT (Acceptance Testing)	2. DevOps
3. Regression Testing	4. Automation Testing
5. Integration Testing	6. User Testing
7. Accessibility Testing	8. Performance Testing
9. Selenium Testing	10. Scriptless Test Automation:
11. Artificial Intelligence	12. Robotic Process Automation (RPA)
13. Infrastructure as Code (IaC)	14. Pen-Testing-as-a-Service (PTaaS)

5.3.1 UAT (Acceptance Testing)



- As soon as a product is developed, even before it is moved to production, the product owner will check its functionality and usability by performing the User acceptance testing.
- This is actually the final phase before launch where the stakeholders check if the product is as per their requirements and also check if there are any errors while moving ahead with the functionalities.
- Primarily, a **UAT** is an important and final phase to test whether the software is functioning as per the requirements.

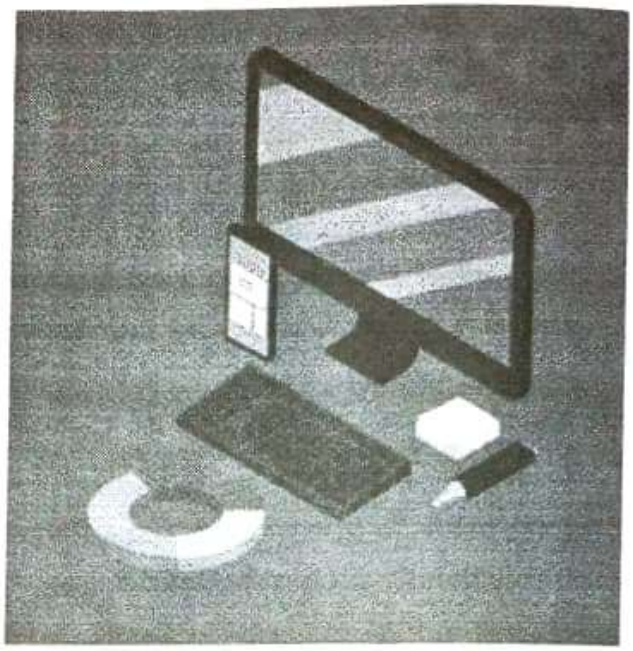
5.3.2 DevOps



- DevOps is a widely known practise of bringing development and operations teams together to bring an effective DevOps culture. This DevOps culture is about a shared collaboration between the development (Dev) and operation (Ops) teams.
- It is a modern code deployment approach that significantly helps in the collaboration and coordination among various teams and accelerates the software delivery process with faster releases.
- This process ensures effective feedback to deliver quality software and ensures improved customer satisfaction.

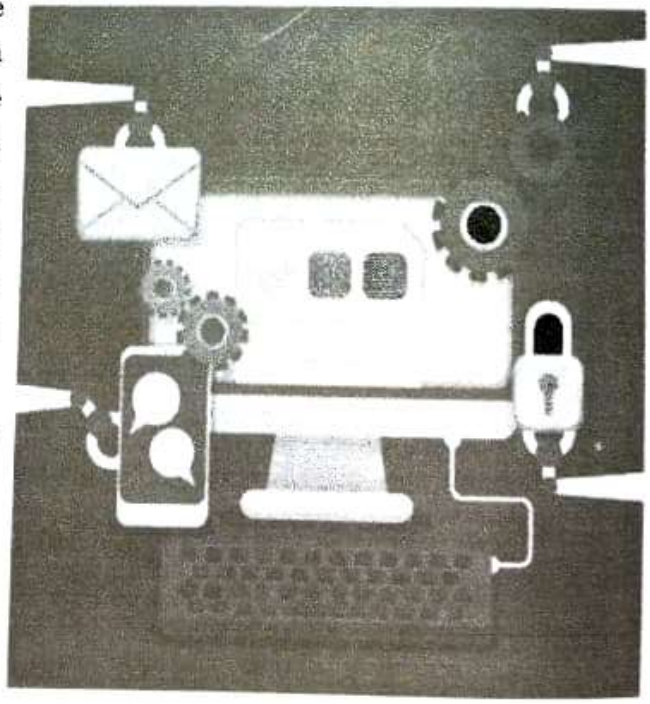
5.3.3 Regression Testing

- Regression testing is one of the software testing types that should be performed when there is a change made in the application or when there is a new feature added to the application.
- With this testing practice, tests are conducted to ensure and check the previously developed and tested software still performs well even after a change is made in the software.
- This is an effective functional testing type that should be taken up especially when there are continuous changes made in the application as this testing process checks for any new bug or error in the existing software and is more so a verification process for the software.



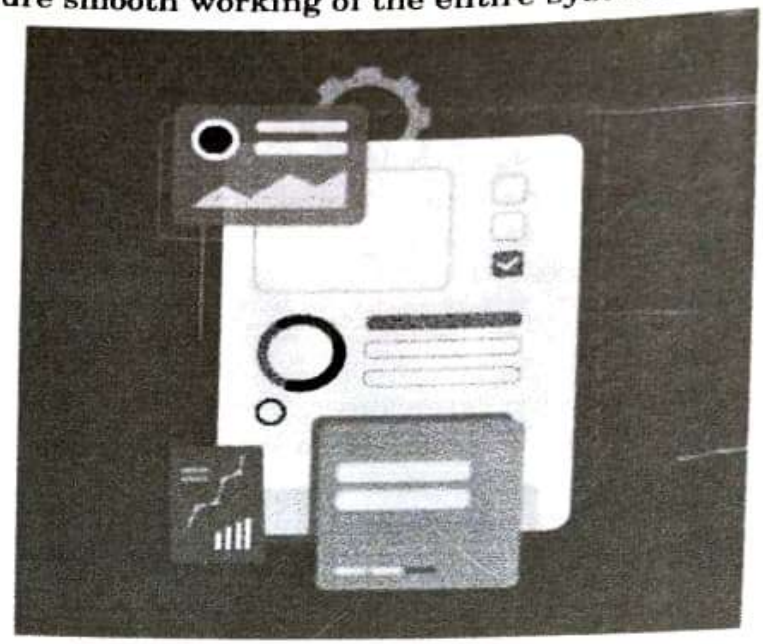
5.3.4 Automation Testing

- With today's enterprises adopting agile and DevOps processes, it becomes a mandate for these practices to leverage automation testing. Basically, Test automation is critical for continuous delivery (CD) and continuous testing (CT), as it can speed up the release cycles, increase test coverage and ensure quality software release.
- Software automation testing involves the usage of tools and test scripts to test the software, and these automated test results are more reliable. Hence, test automation speeds up the testing process, ensures faster releases and delivers accurate results.



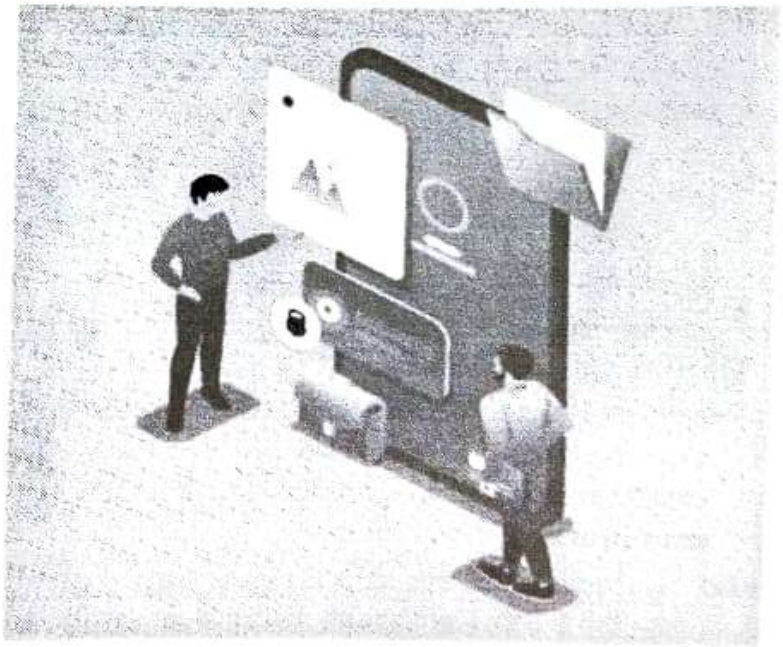
5.3.5 Integration Testing

- In Software testing, it is important that every system component gets integrated with the different application modules to ensure smooth working of the entire system.
- Enterprises following agile and DevOps should take up integration testing to ensure that the application modules function effectively when they are grouped together.
- Thus, Integration testing should be leveraged by businesses as there are numerous benefits with it such as the process helps to identify system-level issues such as module integration issues, broken databases, etc. and helps to identify them while developers resolve them at the earliest.

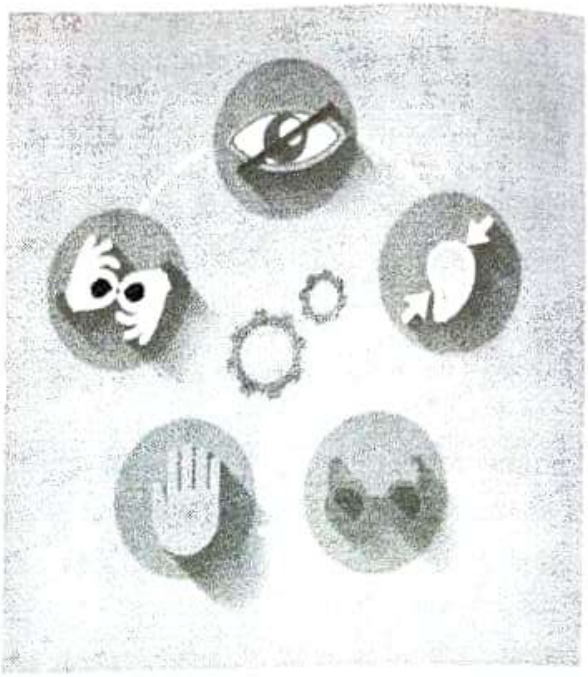


5.3.6 User Testing

- One of the important types of software testing that is gaining more popularity in recent years is user testing.
- This form of user testing refers to a technique wherein real users take up the role of testers to test the interface and functions of applications, websites, mobile applications, or services.
- In this method, the real users test the apps by considering various real-time use cases and the feedback from these users helps in improving the application for the end-users.
- This is a usability technique to gain valuable insights from users regarding how they feel about the product.



5.3.7 Accessibility Testing



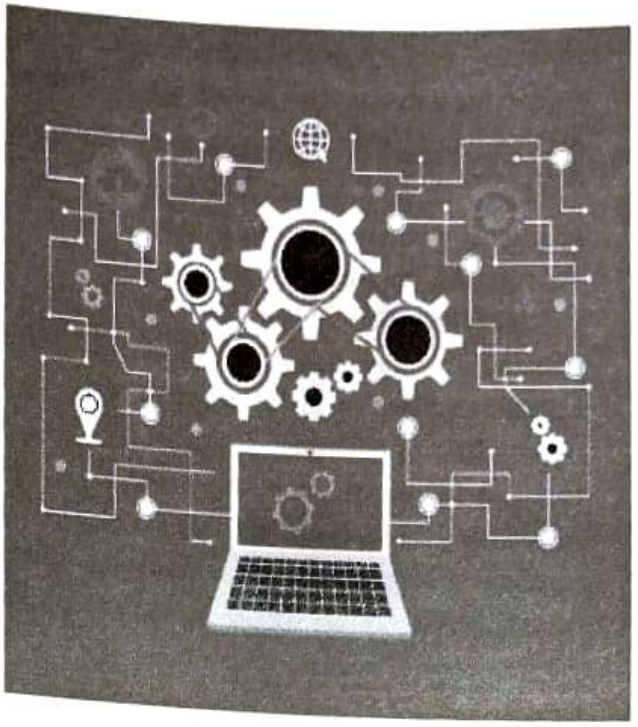
- In today's digital age with connected devices and mobile apps running in millions, there is a need for these mobile & web apps to be accessible to differently-abled people. However, there are various innovative products, applications and websites that are not made accessible to people with certain disabilities.
- But, today, it is a mandate that businesses should leverage accessibility testing. This sort of software testing not only focuses on verifying app usability, but it also makes sure that the application can be used by people with many disabilities, including visual, auditory, physical, speech, cognitive, language, learning, and neurological disabilities.

5.3.8 Performance Testing

- Today's businesses become successful only if their business-critical mobile and web applications perform well under varying loads and should essentially deliver great performance. If these business apps crash when numerous users tend to use it, then users will dump such apps and would never wish to get back to such apps.

- Only those mobile and web apps that perform seamlessly under varying loads are bound to deliver a great customer experience (CX). In order to get high-performing digital mobile and web apps, leverage performance testing from next-gen testing service providers.

5.3.9 Selenium Testing



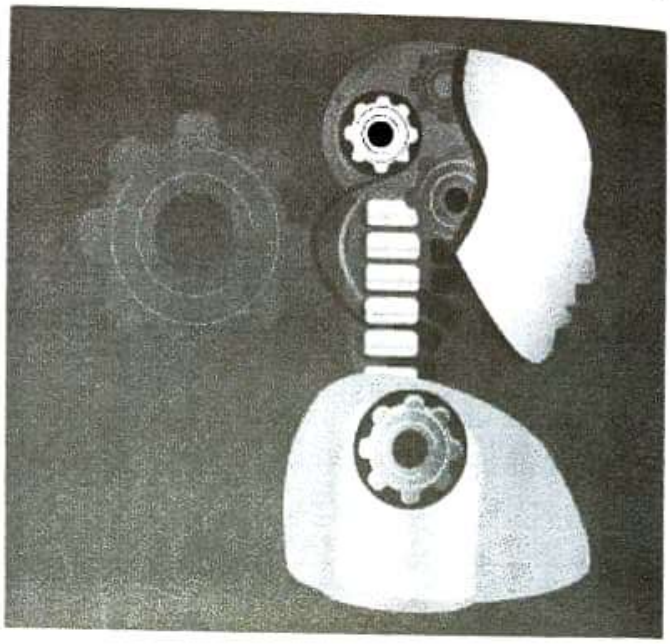
- Businesses adopting agile and DevOps processes should leverage test automation using various test automation tools for faster releases and to get quicker time-to-market. Selenium is one of the most commonly used test automation tool which is a lightweight tool and developer-friendly tool, commonly used for automating web applications.
- Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE).

5.3.10 Scriptless Test Automation

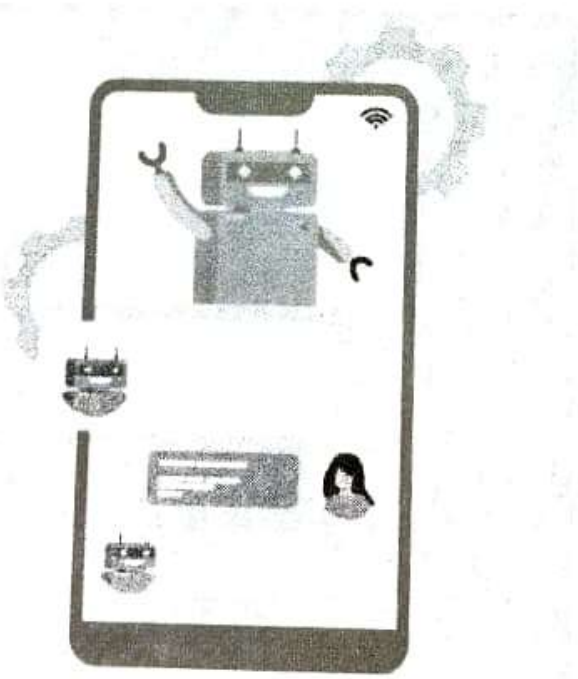
- In recent years, there were several changes in the world of technology. The same has happened in the software testing world where Test Automation has evolved to facilitate rapid software releases at the highest quality.
- Automation has always been interesting, as it reduces the mundane testing efforts and accelerates the testing process. However, the ROI is not always well anticipated.
- In order to maximize the scalability of test automation, 'Scriptless Test Automation' is introduced.
- Scriptless test automation enables the testers and business users to automate test cases without worrying about the coding. It helps to achieve faster results and reduces the time expended to understand the code.

5.3.11 Artificial Intelligence

- Software testing is the only premeditated way where an application can be observed under certain conditions and where testers can recognize the risks involved in the software implementation.
- Testing, on the other hand, is gradually transitioning to greater automation to ensure maximum precision and accuracy in the journey towards digital transformation. In an attempt to make the application foolproof, the world is turning towards Artificial Intelligence (AI). This implies that instead of manual testing and human intervention, we are moving towards a situation where machines will be slowly taking over.



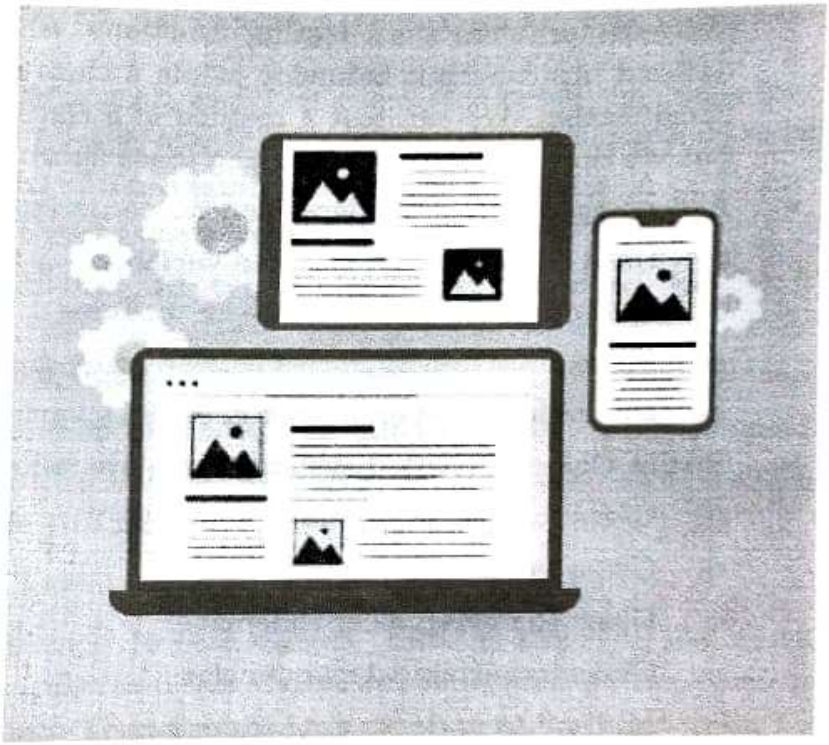
5.3.12 Robotic Process Automation (RPA)



- New and emerging technologies, such as Artificial intelligence (AI), cognitive computing, the Internet of Things (IoT), and machine learning are revolutionizing all industries.
- Some implementations like self-driving cars are set to change the digital world.
- Advances in software and AI world have paved the way for Robotic Process Automation (RPA).
- It is the most recent technology which has the capability to re-invent the business process management landscape.

5.3.13 Infrastructure as Code (IaC)

- The next-generation infrastructure management technologies are transforming the way we manage IT infrastructure.
- The extensive implementation of virtualization and cloud infrastructure has shifted the bottleneck from allocating servers to configuring them.
- The arduous process where it used to take a couple of weeks or months to assign a server, has been transformed into a process of a minute or two.



5.3.14 Pen-Testing-as-a-Service (PTaaS)

- A company's security stance is continuously changing in-line with the growing risks. A traditional penetration testing service is a point in time evaluation.
- However, **PTaaS involves a continuous cycle of testing and remediation**.
- It suggests that to combat the changing security stance of the company, there must be an on-going program of testing and management.

5.4 TESTING OBJECT ORIENTED SOFTWARE: OOT BASICS, OBJECT ORIENTED TESTING

- Software typically undergoes many levels of testing, from unit testing to system or acceptance testing. Typically, in-unit testing, small "units", or modules of the software, are tested separately with focus on testing the code of that module. In higher, order testing (e.g, acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.
- As information systems are becoming more complex, the object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of problems involved in testing classes, abstract classes, inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc. Testing classes is a fundamentally different problem than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these approaches
- According to Davis the dependencies occurring in conventional systems are:
  - Data dependencies between variables
  - Calling dependencies between modules
  - Functional dependencies between a module and the variable it computes
  - Definitional dependencies between a variable and its types.
- But in Object-Oriented systems there are following additional dependencies:
  - Class to class dependencies
  - Class to method dependencies
  - Class to message dependencies
  - Class to variable dependencies
  - Method to variable dependencies
  - Method to message dependencies
  - Method to method dependencies

Issues in Testing Classes

- Additional testing techniques are, therefore, required to test these dependencies. Another issue of interest is that it is not possible to test the class dynamically, only its instances i.e, objects can be tested. Similarly, the concept of inheritance opens various issues e.g., if changes are made to a parent class or superclass, in a larger system of a class it will be difficult to test subclasses individually and isolate the error to one class.

- In object-oriented programs, control flow is characterized by message passing among objects, and the control flow switches from one object to another by inter-object communication.
- Consequently, there is no control flow within a class like functions. This lack of sequential control flow within a class requires different approaches for testing.
- Furthermore, in a function, arguments passed to the function with global data determine the path of execution within the procedure. But, in an object, the state associated with the object also influences the path of execution, and methods of a class can communicate among themselves through this state because this state is persistent across invocations of methods. Hence, for testing objects, the state of an object has to play an important role.

Techniques of object-oriented testing are as follows:

1. **Fault Based Testing**

- This type of checking permits for coming up with test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that may lead to errors.). For all of these faults, a test case is developed to "flush" the errors out. These tests also force each time of code to be executed.
- This method of testing does not find all types of errors. However, incorrect specification and interface errors can be missed. These types of errors can be uncovered by function testing in the traditional testing model.
- In the object-oriented model, interaction errors can be uncovered by scenario-based testing. This form of Object oriented-testing can only test against the client's specifications, so interface errors are still missed.

2. **Class Testing Based on Method Testing**

This approach is the simplest approach to test classes. Each method of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques. Therefore all the methods of a class can be involved at least once to test the class.

1. **Random Testing**

It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories :

2. **Partition Testing**

This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.

3. **Scenario-based Testing**

It primarily involves capturing the user actions then stimulating them to similar actions throughout the test. These tests tend to search out interaction form of error.

5.4.1 Object Oriented Testing Basic

What is object-oriented testing?

- Object-Oriented testing is **a software testing process that is conducted to test the software using object-oriented paradigms like, encapsulation, inheritance, polymorphism, etc.** The software typically undergoes many levels of testing, from unit testing to system or acceptance testing
- Once a program code is written, it must be tested to detect and subsequently handle all errors in it. A number of schemes are used for testing purposes.
- Another important aspect is the fitness of purpose of a program that ascertains whether the program serves the purpose which it aims for. The fitness defines the software quality.

Testing Object-Oriented Systems

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing

In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing

This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing

System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques

Grey Box Testing

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are :

- **State model based testing :** This encompasses state coverage, state transition coverage, and state transition path coverage.
- **Use case based testing :** Each scenario in each use case is tested.
- **Class diagram based testing :** Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing :** The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing

The two main approaches of subsystem testing are :

- **Thread based testing :** All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing :** The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing

- **Alpha testing :** This is carried out by the testing team within the organization that develops software.
- **Beta testing :** This is carried out by select group of co-operating customers.
- **Acceptance testing :** This is carried out by the customer before accepting the deliverables.

Software Quality Assurance

Software Quality

Schulmeyer and McManus have defined software quality as "the fitness for use of the total software product". A good quality software does exactly what it is supposed to do and is interpreted in terms of satisfaction of the requirement specification laid down by the user.

Quality Assurance

Software quality assurance is a methodology that determines the extent to which a software product is fit for use. The activities that are included for determining software quality are :

- Auditing
- Development of standards and guidelines
- Production of reports
- Review of quality system

Quality Factors

- **Correctness :** Correctness determines whether the software requirements are appropriately met.
- **Usability :** Usability determines whether the software can be used by different categories of users (beginners, non-technical, and experts).
- **Portability :** Portability determines whether the software can operate in different platforms with different hardware devices.
- **Maintainability :** Maintainability determines the ease at which errors can be corrected and modules can be updated.
- **Reusability :** Reusability determines whether the modules and classes can be reused for developing other software products.

5.4.2 Object-Oriented Metrics

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

Project Metrics

Project Metrics enable a software project manager to assess the status and performance of an ongoing project. The following metrics are appropriate for object-oriented software projects :

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Number of subsystems

Product Metrics

Product metrics measure the characteristics of the software product that has been developed. The product metrics suitable for object-oriented systems are :

- **Methods per Class :** It determines the complexity of a class. If all the methods of a class are assumed to be equally complex, then a class with more methods is more complex and thus more susceptible to errors.
- **Inheritance Structure :** Systems with several small inheritance lattices are more well-structured than systems with a single large inheritance lattice. As a thumb rule, an inheritance tree should not have more than 7 (± 2) number of levels and the tree should be balanced.
- **Coupling and Cohesion :** Modules having low coupling and high cohesion are considered to be better designed, as they permit greater reusability and maintainability.
- **Response for a Class :** It measures the efficiency of the methods that are called by the instances of the class.

Process Metrics

Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time. They are used as indicators for long-term software process improvements. Some process metrics are :

- Number of KLOC (Kilo Lines of Code)
- Defect removal efficiency
- Average number of failures detected during testing
- Number of latent defects per KLOC.

5.4.3 Object Oriented Testing

OO Testing

- Class Testing : Equivalent to unit testing for conventional SW.
- The concept of a unit changes in the OO context.
- The class or object is the smallest testable unit.
- Testing must focus on each method of the class and the state behavior of the class.
- It is important to consider the state of the class because this will affect the behavior, methods, of the class.
- It is often difficult to determine the state of the object.
- You may have to introduce new methods to look at the state variables for testing purposes.

OO Class Testing Example

- A bank account class.
- It may have methods to open the account, deposit funds, withdraw funds and close the account.
- The states of the account include
  - open with positive balance,
  - open with negative or zero balance and
  - closed.
- How the methods behave depends on the state of the account.
  - An account with a zero or negative balance will not allow the customer to withdraw funds.
  - If positive it might allow customer to go to overdraft once.
  - You could introduce a new method to determine the if the account is open or closed and if balance is positive.

OO Testing

- **Integration Testing:**

  Because OO SW does not have a hierarchical control structure, conventional top-down or bottom-up integration strategies are not applicable.

Two approaches for OO Integration testing

- **Thread-based testing**
  - Integrate the set of classes required to respond to one input or event of the system.
  - Each thread is integrated and tested individually.
- **Use-based testing**
  - First construct and test the classes which use very few (if any) server classes (these are called the independent classes).
  - The next step is to test the next layer of classes, the dependent classes, that use the independent classes.

Validation Testing

- Conventional black-box testing derived from the analysis model can be used to test the OO software.
- Use cases are a good place to look when developing test cases.

Test cases and the Class hierarchy

- Methods that are redefined in a subclass must be tested because it represents a new design and new code.
- Methods that are inherited must also be tested. – A subset of the original tests can be executed to ensure it works in the derived class.
- Random Testing: When a variety of different operation sequences are randomly generated. – Keep in mind the behavior life sequence of the class.

Partition testing

- Very similar to equivalence class partitioning for conventional SW. Partition the input and output of the class and design test cases to exercise each class.
- 3 Examples :
  - State-based
  - Attribute-based
  - Category-based
- State-based partitioning : Look at the states for the class.
  - Determine which operations change the state of the class and which do not and design test cases to exercise the class.
  - Test each method, while object is in each state.
  - Design test cases to do this.
- Attribute-based partitioning
  - Look at the attributes of the class.
  - Partition methods into those that use the attribute, modify it and those that do not use it.
  - Design test cases for each partition.
- Category-based partitioning
  - Look at the methods for the class.
  - Partition the methods into categories based on their function.

Examples

- Initialization operations, computational, queries and termination operations.
- Design test cases for each partition.

Student Registration Example

- A student registration class
  - The methods include adding a class, dropping a class, transferring to a different section of the class and list classes.

- A student must first be registered with the university (opened as a student).
- Holds can be placed on a student and this effects whether he/she can register for a class.
- There are limits on how many credit hours a student may register for. – A student may graduate (close a student).

Student Registration Example

Random testing may generate : register, add class, transfer, add class, drop class, add class, hold, release hold, add class, ...

OO Test Case Design

- Given the differences between conventional and OO SW, test case design is slightly different.
- OO test cases should be defined in the following way :
  - Each test case should be uniquely identified and associated with the class to be tested. – The purpose of the test should be stated.
  - The following list should be developed for each test :
    1. List of object states to be tested.
    2. List of messages and operations (methods) to be tested
    3. List of exceptions that may occur as the object is tested.
    4. List of external conditions (think about system testing) that may change.
- Anything else needed for the test?

OO Test Case Design

- Instead of testing each component, you will test each class.
- Make sure you identify the class you are testing.
- You must test each method, in each state for the class. – This information will be included in the purpose/condition.
- If needed, add methods to check state of the object. – You could use the result of these methods in your expected outcome.

Chapter Ends...

❑❑❑

Module 6

CHAPTER 6

Quality Management

University Prescribed Syllabus

6.1 Software Quality Management, McCall's quality factors and Criteria,

6.2 ISO9000:2000, SIX Sigma

6.1 Software Quality Management 6-3

6.1.1 Software Quality Assurance (SQA) 6-3

6.2 ISO 9000 Standards 6-4

6.3 Software Quality Assurance Plan 6-6

6.4 Introduction to Total Quality Management ? 6-7

UQ. What is Total Quality Management ? (MU - Oct. 19) 6-7

6.5 ISO 9000 : 2000 6-8

6.5.1 Why ISO 9000 or 9001 ? 6-8

6.5.2 What is an ISO 9000 Certification ? 6-9

6.5.3 How to become ISO 9000 Certified ? 6-9

6.6 Capability Maturity Model (CMM) 6-10

6.6.1 What is Capability Maturity Model (CMM) Levels ? 6-10

6.6.2 Internal Structure of CMM 6-13

6.6.3 Limitations of CMM Models 6-14

6.6.4 Why Use CMM ? 6-14

6.7 Process Quality Metrics 6-15

6.7.1 Process Software Quality Maintenance Metrics 6-17

UQ. What is in process Software Quality Maintenance Metrics ? (MU - Nov./Dec. 18) 6-17

6.8 Ishikawa's 7 basic tools 6-18

6.8.1 Check Sheet / Checklist 6-19

6.8.2 Pareto Analysis 6-20

6.8.3 Fishbone Diagram 6-21

6.8.4 Scatter Diagram 6-22

6.8.5 Flowchart 6-22

6.8.6 Control Chart 6-23

6.9 Self-learning Topics 6-24

6.9.1 Case Studies to Identify Quality Attributes Relationships for Different Types of Applications (Web-based, Mobile-based, etc) 6-24

▶ **Chapter Ends 6-33**

6.1 SOFTWARE QUALITY MANAGEMENT

GQ. How to maintain Software Quality Management ?

- Software Quality Management (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.
- Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue.
- Software Quality Assurance is a kind of an Umbrella activity that is applied. There is no one universal definition of software quality. This is because of the complexity caused by the three or more participants affected by the quality of software, namely, customer, developer and stakeholders.
- The issue is whose views, expectations and aspirations are to be considered supreme. The majority hold that customer satisfaction should be the goal for measuring software quality.
- The customer may be satisfied though with software, the quality of which cannot be considered the best by other standards.
- The software quality definition is based on the following :
  - Customer focus and customer satisfaction
  - Functional and performance requirement
  - Ease of learning, use and maintainability
  - Adherence to development standards
- Customer satisfaction largely depends on meeting functional and performance requirements and ease of operations. Adhering to development standards ensures to a great extent the achievement of these goals.
- Software quality is defined as the quality that ensures customer satisfaction by offering all the customer deliverables on performance, standards and ease of operations. The definition is applicable for software as well as for a generic software product.

6.1.1 Software Quality Assurance (SQA)

- Software quality assurance is a planned effort to ensure that a software product fulfills these criteria and has additional attributes specific to the project, e.g., portability, efficiency, reusability, and flexibility.
- It is the collection of activities and functions used to monitor and control a software project so that specific objectives are achieved with the desired level of confidence. It is not the sole responsibility of the software quality assurance group but is determined by the consensus of the project manager, project leader, project personnel, and users.

- A formal definition of software quality assurance is that is 'the systematic activities providing evidence of the fitness for use of the total software product."
- Software quality assurance is achieved through the use of established guidelines throughout the software process.
- Software Quality Assurance has :
  - A quality management approach
  - Formal technical reviews
  - Multi testing strategy
  - Effective software engineering technology
  - Measurement and reporting mechanism

☞ **Goals, Attributes, and Metrics**

- **Requirement quality :** The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow. SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.
- **Design quality :** Every element of the design model should be assessed y the software team to ensure that it exhibits high quality and that the design itself conforms to requirements.
- **Code quality :** Source code and related work products must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- **Quality control effectiveness :** A software team should apply limited resources in a way that has the highest likelihood of achieving a high–quality result. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

6.2 ISO 9000 STANDARDS

GQ. What are different ISO 9000 Standards ?

- **ISO 9001 : 2008 Quality management systems -** Requirements is intended for use in any organization regardless of size, type or product (including service). It provides a number of requirements which an organization needs to fulfill to achieve customer satisfaction through consistent products and services which meet customer expectations. It includes a requirement for continual (i.e., planned) improvement of the Quality Management System, for which ISO 9004:2004 provides many hints.
- This is the only implementation for which third party auditors can grant certification. It should be noted that certification is not described as any of the 'needs' of an organization as a driver for using ISO 9001 but does recognize that it may be used for such a purpose.

- **ISO 9004:2000 Quality management systems** - Guidelines for performance improvements covers continual improvement. This gives you advice on what you could do to enhance a mature system.
- This document very specifically states that it is not intended as a guide to implementation.
- There are many more standards in the ISO 9001 series, many of them not even carrying "ISO 9000" numbers. For example, some standards in the 10,000 range are considered part of the 9000 group: ISO 10007: 1995 discusses configuration management, which for most organizations is just one element of a complete management system.
- The emphasis on certification tends to overshadow the fact that there is an entire family of ISO 9000 standards... Organizations stand to obtain the greatest value when the standards in the new core series are used in an integrated manner, both with each other and with the other standards making up the ISO 9000 family as a whole.
- Note that the previous members of the ISO 9000 series 9002 and 9003 have been integrated into 9001.
- In most cases, an organization claiming to be "ISO 9000 registered" is referring to ISO 9001.

(1F1)**Fig. 6.2.1 : ISO Standard Principles**

6.3 SOFTWARE QUALITY ASSURANCE PLAN

GQ. Explain in detail Software Quality Assurance Plan.

- The software quality assurance (SQA) plan is an outline of quality measures to ensure quality levels within a software development effort.
- The plan is used as a baseline to compare the levels of quality during development with the planned levels of quality. If the levels of quality are not within the planned quality levels management will respond appropriately as documented within the plan.
- The plan provides the framework and guidelines for development of understandable and maintainable code. These Ingredients help ensure the quality sought in a software project, A SQA plan also provides the procedures for ensuring that quality software will be produced or maintained in house or under contract.
- These procedures affect planning. Designing, writing, testing, documenting, strong, and maintaining computer software. It should be organized in this way because the plan ensures the quality of the software rather than describing specific procedures for developing and maintaining the software.
- In the management approval process, management relinquishes tight control over software quality to the SQA plan administrator in exchange for improved software quality. Software quality is often left to software developers. Quality is desirable. But management may express concern as to the cost of a formal SQA plan. Staff should be aware that management views the program as a means of ensuring software quality, and not as an end in itself.
- To address management concerns, software life cycle costs should be formally estimated for projects implemented both with and without a formal SQA plan. In general, implementing a formal SQA plan makes economic and management sense.
- The SQA Plan helps to lay down the steps towards the quality goals of the
- organization. A standard for SQA plan gives details and templates on all activities, which have become part of the standard and which ensure quality standards implementation.
- Testing activity needs Test Plan likewise SQA activity also needs a plan which is called SQA plan.
- The goal of SQA plan is to craft planning processes and procedures to ensure products manufactured, or the service delivered by the organization are of exceptional quality.
- During project planning, Test Manager makes an SQA plan where SQA audit is scheduled periodically.

- The documentation of SQA plan includes
  - Project plan
  - Models of data, classes and objects, processes, design, architecture
  - Software Requirement Specifications (SRS)
  - Test plans for testing SRS
  - Users help documentation, manuals, online help, etc.
  - Reviews and audits
- The SQA process is made up of several activities. Some are organization specific, some are software specific and some are customer specific. It helps effective application of methods, tools, test plans and standards towards the goal of software quality. It also include measures, measurement and metric for building a quality in the organization.

6.4 INTRODUCTION TO TOTAL QUALITY MANAGEMENT ?

UQ. What is Total Quality Management ? **MU - Oct. 19**

- Total quality management (TQM) is the continual process of detecting and reducing or eliminating errors in manufacturing, streamlining supply chain management, improving the customer experience, and ensuring that employees are up to speed with training. Total quality management aims to hold all parties involved in the production process accountable for the overall quality of the final product or service.
- TQM was developed by William Deming, a management consultant whose work had a great impact on Japanese manufacturing.
- While TQM shares much in common with the Six Sigma improvement process, it is not the same as Six Sigma. TQM focuses on ensuring that internal guidelines and process standards reduce errors, while Six Sigma looks to reduce defects.
- Total Quality Management (TQM) is a structured approach to overall organizational management. The focus of the process is to improve the quality of an organization's outputs, including goods and services, through continual improvement of internal practices. The standards set as part of the TQM approach can reflect both internal priorities and any industry standards currently in place.
- Industry standards can be defined at multiple levels and may include adherence to various laws and regulations governing the operation of the particular business. Industry standards can also include the production of items to an understood norm, even if the norm is not backed by official regulations.

6.5 ISO 9000 : 2000

GQ. Write in brief ISO 9000:2000.

- The ISO 9001 standard is a document that describes all of the requirements needed in order to create and maintain a quality management system as described in ISO 9000. This is a subtle difference between ISO 9000 and ISO 9001 that some fail to recognize.
- So, to explicitly point it out, the difference between the two (ISO 9000 vs 9001) is summarized as the definition of quality management system (ISO 9000) and requirements needed to meet that definition (ISO 9001).
- Both the ISO 9000 and 9001 standards are based on a number of quality management principles including a strong customer focus, the motivation, and implication of top management, the process approach and continual improvement. The seven quality management principles include the following as described by the ISO :

1. **Customer focus :** Quality management primarily focuses on meeting customer requirements and striving to exceed customer expectations.
2. **Leadership :** Helping leaders to establish unity of purpose and direction at all levels and to create conditions to engage members of the organization in achieving the organization's quality objectives.
3. **Engagement of people :** Obtaining and maintaining (at all levels throughout the organization) competent, empowered, and engaged people to enhance the organization's capability to create and deliver value.
4. **Process approach :** Delivering consistent and predictable results through the use of effective and efficient activities that are understood and managed as interrelated processes that function as a coherent system.
5. **Improvement :** Maintaining an ongoing, organization-wide focus on improvement.
6. **Evidence-based decision making :** Using the analysis and evaluation of data and information in the decision making process to produce desired results.
7. **Relationship management :** Managing the organization's relationships with related parties, such as partners or vendors, for sustained success.

- You can learn more about the quality management principles from the ISO's 2015 publication. Also, a copy of ISO 9001:2015 can be purchased at the ISO online store.

6.5.1 Why ISO 9000 or 9001 ?

- One misconception is that ISO 9000 or 9001 is only for manufactures or large organizations. As a principles-based standard, ISO 9001 can be applied to any organization regardless of what type or size it may be. The standard defines the requirements, but it does not dictate the method of application.

- The latest version of the standard has been specifically designed to be more accessible to organizations outside the manufacturing sector.
- As with anything, there are ISO 9000/9001 pros and cons. The application of ISO 9001 when implementing a quality management system can provide the following benefits the organizations :
  1. Clear understanding of your objectives and new business opportunities.
  2. Identifying and addressing the risks associated with your organization.
  3. Renewed emphasis on putting your customers first.
  4. Meeting the necessary statutory and regulatory requirements.
  5. Organizational and process alignment to increase productivity and efficiency.

6.5.2 What is an ISO 9000 Certification ?

- If you are researching the ISO 9000 requirements or how to become ISO 9000 certified, you should really be focused on ISO 9001. You see an organization cannot become ISO 9000 certified. First issued in 1987 and last updated in 2015, ISO 9001 is the standard that sets out the criteria for a quality management system and is also the only standard within ISO 9000 that an organization can certify to. Therefore, it is incorrect to say that an organization is ISO 9000 compliant.
- However, a business can be ISO 9001 certified or compliant. While an ISO 9001 certification is not regulatory requirement, ISO reports that "over one million companies and organizations in over 170 countries have certified to ISO 9001."
- An organization must demonstrate the following in order to be ISO 9001 certified :
  - The company follows the guidelines within the ISO 9001 standard;
  - The company meets its own requirements;
  - The company meets its customer requirements and statutory and regulatory requirements; and
  - The company maintains documentation of its performance.
- An ISO 9001 certification can enhance an organization's credibility as it shows customers that the organization's products and services meet quality expectations. Additionally, there are some instances where an ISO 9001 certification is required or legally mandated for businesses in some industries.

6.5.3 How to become ISO 9000 Certified ?

- The ISO 9001 certification process requires an organization to implement ISO 9001:2015 requirements. Once implemented, an organization must successfully complete registrar's audit to confirm that the organization system meets those requirements.

- The auditor will interview management and staff within the organization to determine whether or not they understand their role and responsibilities in complying with the ISO 9001 standards. The auditor will also examine the organization's documentation to validate compliance with the ISO 9001 requirements. The auditor will then prepare a detailed report that details the parts of the standard that the organization did not meet.
- The organization will need to agree to correct any problems within a specified time frame. The organization executes remedial activities to ensure that all problems are corrected. Once these gaps are addressed and confirmed by the auditor, the organization can then be certified.
- In order to maintain the ISO 9001 certification, the organization must continue with regular surveillance and recertification audits.

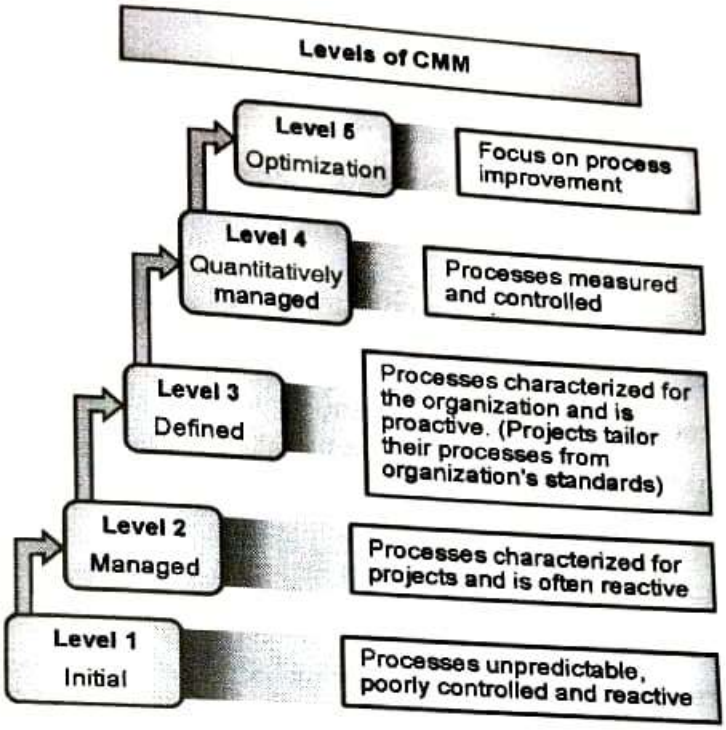
6.6 CAPABILITY MATURITY MODEL (CMM)

- Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.
- CMM was developed at the Software engineering institute in the late 80's. It was developed as a result of a study financed by the U.S Air Force as a way to evaluate the work of subcontractors. Later based on the CMM-SW model created in 1991 to assess the maturity of software development, multiple other models are integrated with CMM-I they are
- Capability Maturity Model (CMM) & CMM Levels: A Fool's Guide
- In this tutorial, we will learn,
  - What is Capability Maturity Model (CMM) Levels?
  - What happens at different levels of CMM?
  - How long does it Take to Implement CMM?
  - Internal Structure of CMM
  - Limitations of CMM Models
  - Why Use CMM?

6.6.1 What is Capability Maturity Model (CMM) Levels ?

- Initial
- Repeatable/Managed
- Defined
- Quantitatively Managed
- Optimizing

- Capability Maturity Model (CMM) & CMM Levels: A Fool's Guide
- What happens at different levels of CMM?
- Levels
  - Activities
  - Benefits



(1F2)Fig. 6.6.1

Level 1 Initial

- At level 1, the process is usually chaotic and ad hoc
- A capability is characterized on the basis of the individuals and not of the organization
- Progress not measured
- Products developed are often schedule and over budget
- Wide variations in the schedule, cost, functionality, and quality targets
- None. A project is Total Chaos

Level 2 Managed

- Requirement Management
- Estimate project parameters like cost, schedule, and functionality
- Measure actual progress
- Develop plans and process

- Software project standards are defined
- Identify and control products, problem reports changes, etc.
- Processes may differ between projects
- Processes become easier to comprehend
- Managers and team members spend less time in explaining how things are done and more time in executing it
- Projects are better estimated, better planned and more flexible
- Quality is integrated into projects
- Costing might be high initially but goes down overtime
- Ask more paperwork and documentation

☞ **Level-3 Defined**

- Clarify customer requirements
- Solve design requirements, develop an implementation process
- Makes sure that product meets the requirements and intended use
- Analyse decisions systematically
- Rectify and control potential problems
- Process Improvement becomes the standard
- Solution progresses from being "coded" to being "engineered"
- Quality gates appear throughout the project effort with the entire team involved in the process
- Risks are mitigated and don't take the team by surprise

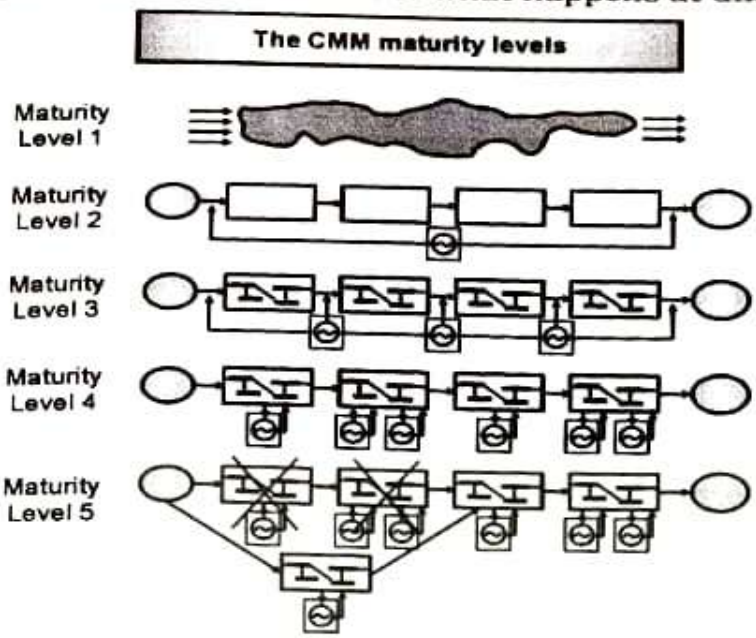
☞ **Level-4 Quantitatively Managed**

- Manages the project's processes and sub-processes statistically
- Understand process performance, quantitatively manage the organization's project
- Optimizes Process Performance across the organization
- Fosters Quantitative Project Management in an organization.

☞ **Level-5 Optimizing**

- Detect and remove the cause of defects early
- Identify and deploy new tools and process improvements to meet needs and business objectives
- Fosters Organizational Innovation and Deployment
- Gives impetus to Causal Analysis and Resolution

Fig. 6.6.2 gives a pictorial representation of what happens at different CMM level



(1F3)Fig. 6.6.2

Capability Maturity Model (CMM) & CMM Levels: A Fool's Guide

How long does it take to Implement CMM ?

CMM is the most desirable process to maintain the quality of the product for any software development company, but its implementation takes little longer than what is expected.

- CMM implementation does not occur overnight
- It's just not merely a "paperwork."
- Typical times for implementation is

  3-6 months -> for preparation

  6-12 months -> for implementation

  3 months -> for assessment preparation

  12 months ->for each new level

6.6.2 Internal Structure of CMM

- Each level in CMM is defined into key process area or KPA, except for level-1. Each KPA defines a cluster of related activities, which when performed collectively achieves a set of goals considered vital for improving software capability
- For different CMM levels, there are set of KPA's, for instance for CMM model-2, KPA are
  - REQM- Requirement Management

- PP- Project Planning
- PMC- Project Monitoring and Control
- SAM- Supplier Agreement Management
- PPQA-Process and Quality Assurance
- CM-Configuration Management

- Likewise, for other CMM models, you have specific KPA's. To know whether implementation of a KPA is effective, lasting and repeatable, it is mapped on following basis
  - Commitment to perform
  - Ability to perform
  - Activities perform
  - Measurement and Analysis
  - Verifying implementation

6.6.3 Limitations of CMM Models

1. CMM determines what a process should address instead of how it should be implemented.
2. It does not explain every possibility of software process improvement.
3. It concentrates on software issues but does not consider strategic business planning, adopting technologies, establishing product line and managing human resources.
4. It does not tell on what kind of business an organization should be in.
5. CMM will not be useful in the project having a crisis right now.

6.6.4 Why Use CMM ?

- Today CMM act as a "seal of approval" in the software industry. It helps in various ways to improve the software quality.
- It guides towards repeatable standard process and hence reduce the learning time on how to get things done
- Practicing CMM means practicing standard protocol for development, which means it not only helps the team to save time but also gives a clear view of what to do and what to expect
- The quality activities gel well with the project rather than thought of as a separate event
- It acts as a commuter between the project and the team
- CMM efforts are always towards the improvement of the process

SUMMARY

CMM was first introduced in late 80's in U.S Air Force to evaluate the work of subcontractors. Later on, with improved version, it was implemented to track the quality of the software development system.

The entire CMM level is divided into five levels.

- **Level 1 (Initial) :** Where requirements for the system are usually uncertain, misunderstood and uncontrolled. The process is usually chaotic and ad-hoc.
- **Level 2 (Managed) :** Estimate project cost, schedule, and functionality. Software standards are defined
- **Level 3 (Defined) :** Makes sure that product meets the requirements and intended use
- **Level 4 (Quantitatively managed) :** Manages the project's processes and sub-processes statistically
- **Level 5 (Maturity) :** Identify and deploy new tools and process improvements to meet needs and business objectives

6.7 PROCESS QUALITY METRICS

GQ. What is in process Quality Metrics ?

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes:

1. Defect density during machine testing
2. Defect arrival pattern during machine testing
3. Phase-based defect removal pattern
4. Defect removal effectiveness

1) Defect density during machine testing

- Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.
- This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

2) Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following:

- The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.
- The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
- The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

3) Phase-based defect removal pattern

- This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.
- Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.
- With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

4) Defect removal effectiveness

- It can be defined as follows :

$$\text{DRE} = \frac{\text{Defect removed during a development phase}}{\text{Defects latent in the product}} \times 100\%$$

- This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called early defect removal when used for the front-end and phase effectiveness for specific phases.
- The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

6.7.1 Process Software Quality Maintenance Metrics

UQ. What is in process Software Quality Maintenance Metrics ? MU - Nov./Dec. 18

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and fix responsiveness

$$\text{Percent delinquent fixes} = \frac{\text{Number of fixes that exceeded the response time criteria by ceverity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$$

1) **Fix quality**

- Fix backlog and backlog management index
- Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week.
- Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.
- Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problems arrived during the month}} \times 100\%$$

- If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.
- Fix response time and fix responsiveness
- The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.
- The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.
- Percent delinquent fixes
- It is calculated as follows :

$$\text{Percent Delinquent Fixes} = \frac{\text{Number of fires that exceeded the response time criteria by ceverity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$$

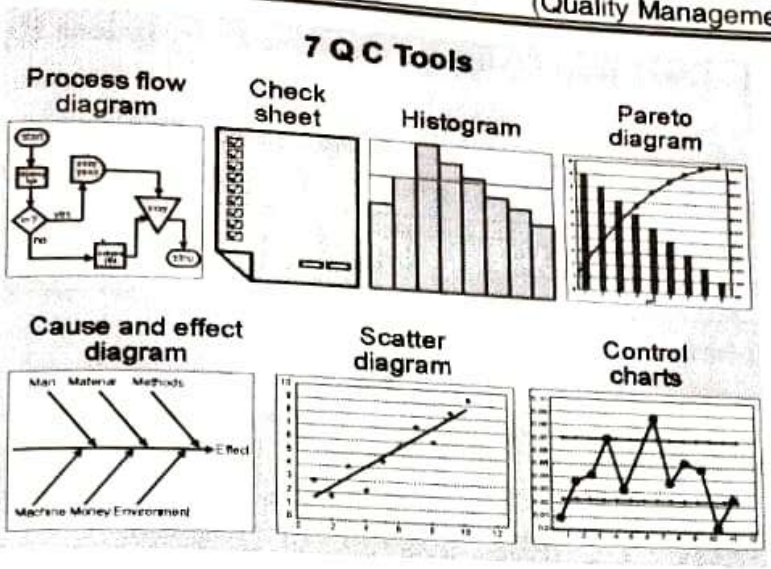
2) Fix Quality

- Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect.
- For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.
- A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.
- Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

6.8 ISHIKAWA'S 7 BASIC TOOLS

GQ. What are Ishikawa's 7 basic tools ?

- Dr. Kaoru Ishikawa was first total quality management guru, who has been associated with the development and advocacy of using the seven quality control (QC) tools in the organizations for problem solving and process improvements.
- Seven old quality control tools are a set of the QC tools that can be used for improving the performance of the production processes, from the first step of producing a product or service to the last stage of production.
- So, the general purpose of this paper was to introduce these 7 QC tools. This study found that these tools have the significant roles to monitor, obtain, analyze data for detecting and solving the problems of production processes, in order to facilitate the achievement of performance excellence in the organizations.
- There are seven basic quality tools, which can assist an organization for problem solving and process improvements. The first guru who proposed seven basic tools was Dr. Kaoru Ishikawa in 1968, by publishing a book entitled "Gemba no QC Shuho" that was concerned managing quality through techniques and practices for Japanese firms.
- It was intended to be applied for "self-study, training of employees by foremen or in QC reading groups in Japan. It is in this book that the seven basic quality control tools were first proposed. valuable resource when applying the seven basic tools (Omachonu and Ross, 2004).



(1F4)**Fig. 6.8.1 : Ishikawa7 Basic Tool for Quality Control**

- These seven basic quality control tools, which introduced by Dr. Ishikawa, are :

1. Check sheets	2. Graphs (Trend Analysis)
3. Histograms	4. Pareto charts
5. Cause-and-effect diagrams	6. Scatter diagrams
7. Control charts.	

- Fig. 6.8.1 indicates the relationships among these seven tools and their utilizations for the identification and analysis of improvement of quality.

6.8.1 Check Sheet / Checklist

GQ. Explain the Terms Checklists, Pareto diagrams, Histogram, Run Charts, Scatter diagrams, Control chart, Cause Effect diagram.

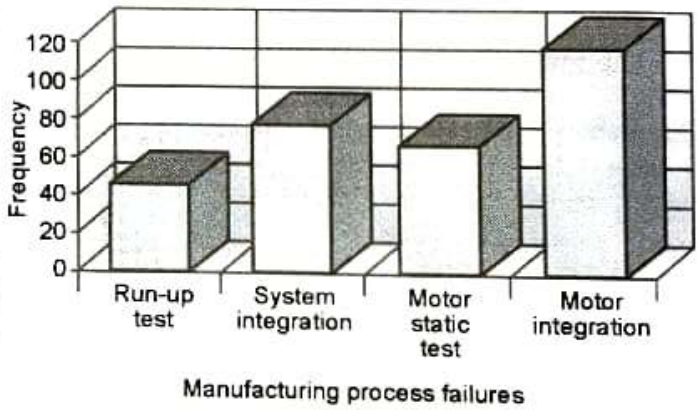
- Check sheets are simple forms with certain formats that can aid the user to record data in an firm systematically.
- Data are "collected and tabulated" on the check sheet to record the frequency of specific events during a data collection period. They prepare a "consistent, effective, and economical approach" that can be applied in the auditing of quality assurance for reviewing and to follow the steps in a particular process. Also, they help the user to arrange the data for the utilization later.
- The main advantages of check sheets are to be very easily to apply and understand, and it can make a clear picture of the situation and condition of the organization. They are efficient and powerful tools to identify frequently problems, but they don't have effective ability to analyze the quality problem into the workplace. The check sheets are in several, three major types are such as Defect-location check sheets; tally check sheets, and; defect-cause check sheets .

Table 6.8.1 : Check sheet (Tally) for telephone interruptions Histogram

Telephone Interruptions

Reason	Day					
	Mon	Tues	Wed	Thurs	Fri	Total
Wrong number	卌	卌	卌	||||	|||| ||	20
Info request	||	||	||	|	||	10
Boss	卌	卌	|||| ||	|	||||	19
Tots	12	6	10	8	13	49

- Histogram is very useful tool to describe a sense of the frequency distribution of observed values of a variable. It is a type of bar chart that visualizes both attribute and variable data of a product or process, also assists users to show the distribution of data and the amount of variation within a process.
- It displays the different measures of central tendency (mean, mode, and average). It should be designed properly for those working into the operation process can easily utilize and understand them. Also, a histogram can be applied to investigate and identify the underlying distribution of the variable being explored. Fig. 6.8.2 illustrates a histogram of the frequency of defects in a manufacturing process.

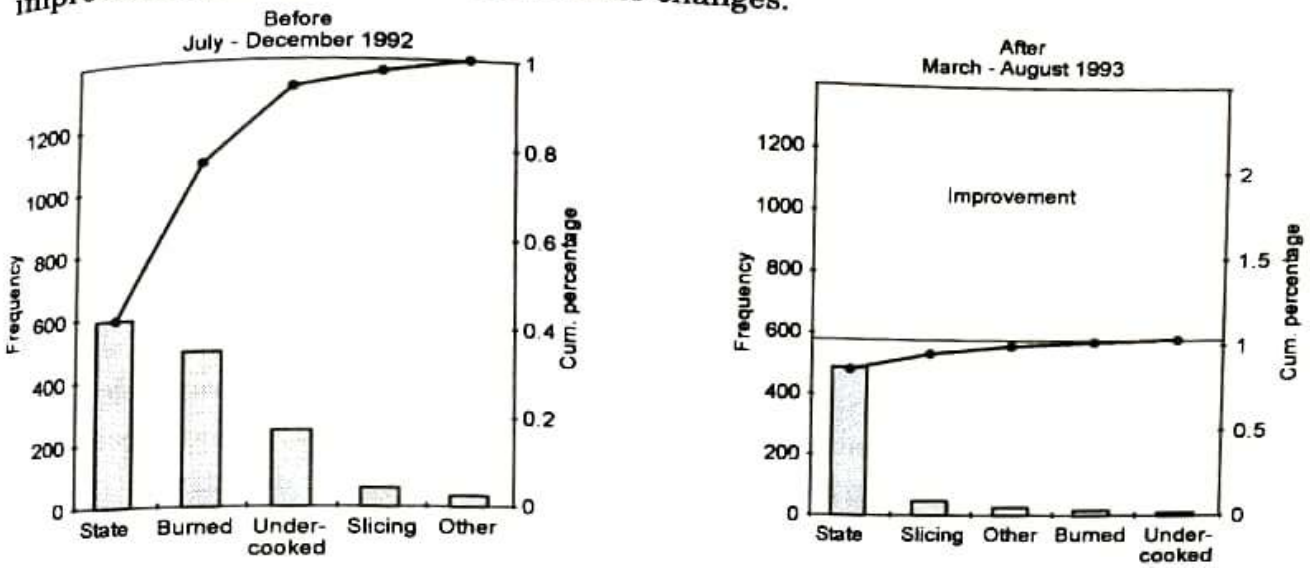


(1F5)**Fig. 6.8.2 : Histogram for variables**

6.8.2 Pareto Analysis

- It introduced by an Italian economist, named Vilfredo Pareto, who worked with income and other unequal distributions in 19th century, he noticed that 80% of the wealth was owned by only 20% of the population. later, Pareto principle was developed by Juran in 1950.
- A Pareto chart is a special type of histogram that can easily be apply to find and prioritize quality problems, conditions, or their causes of in the organization (Juran and Godfrey, 1998).

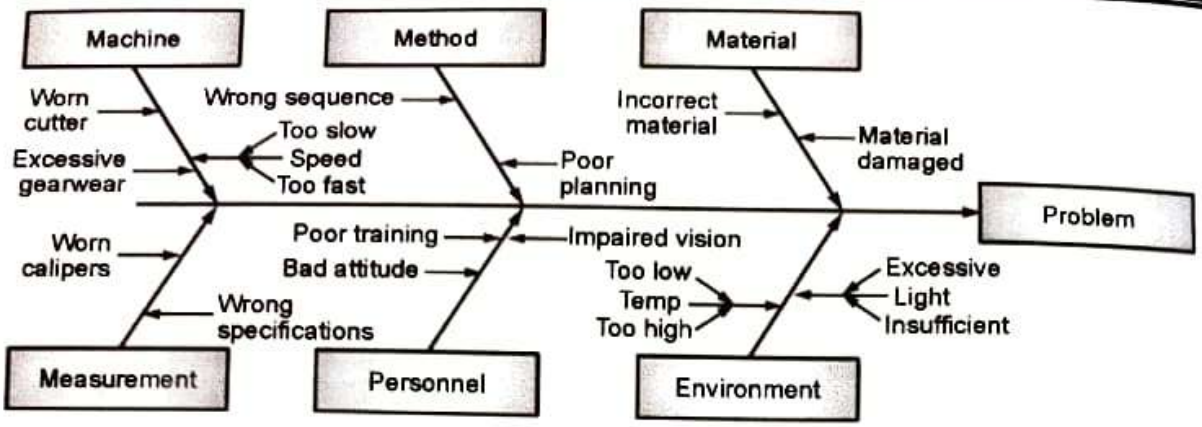
- On the other hand, it is a type of bar chart that shows the relative importance of variables, prioritized in descending order from left to right side of the chart.
- The aim of Pareto chart is to figure out the different kind of "nonconformity" from data figures, maintenance data, repair data, parts scrap rates, or other sources. Also, Pareto chart can generate a mean for investigating concerning quality improvement, and improving efficiency, "material waste, energy conservation, safety issues, cost reductions", etc., as Fig. 6.8.3 demonstrated concerning Pareto chart, it can able to improve the production before and after changes.



(1F6)**Fig. 6.8.3 : Pareto Charts**

6.8.3 Fishbone Diagram

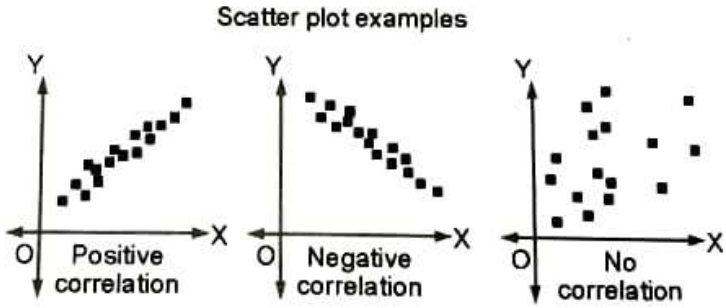
- Kaoru Ishikawa is considered by many researchers to be the founder and first promoter of the 'Fishbone' diagram (or Cause-and-Effect Diagram) for root cause analysis and the concept of Quality Control (QC) circles. Cause and effect diagram was developed by Dr. Kaoru Ishikawa in 1943.
- It has also two other names that are Ishikawa diagram and fishbone because the shape of the diagram looks like the skeleton of a fish to identify quality problems based on their degree of importance.
- The cause and effect diagram is a problem-solving tool that investigates and analizes systematically all the potential or real causes that result in a single effect.
- On the other hand, it is an efficient tool that equips the organization's management to explore for the possible causes of a problem.
- This diagram can provide the problem-solving efforts by "gathering and organizing the possible causes, reaching a common understanding of the problem, exposing gaps in existing knowledge, ranking the most probable causes, and studying each cause".
- The generic categories of the cause and effect diagram are usually six elements (causes) such as environment, materials, machine, measurement, man, and method, as indicated in Fig. 6.8.4. Furthermore, "potential causes" can be indicated by arrows entering the main cause arrow.



(1F7)**Fig. 6.8.4 : The cause and effect diagram (Fishbone Diagram)**

6.8.4 Scatter Diagram

- Scatter diagram is a powerful tool to draw the distribution of information in two dimensions, which helps to detect and analyze a pattern relationships between two quality and compliance variables (as an independent variable and a dependent variable), and understanding if there is a relationship between them, so what kind of the relationship is (Weak or strong and positive or negative).
- The shape of the scatter diagram often shows the degree and direction of relationship between two variables, and the correlation may revealed the causes of a problem. Scatter diagrams are very useful in regression modeling.
- The scatter diagram can indicate that there is which one of these following correlation between two variables :
  (a) Positive correlation;
  (b) Negative correlation, and
  (c) No correlation, as demonstrated in Fig. 6.8.5. An operation, so it is very useful to find and improve quality into process

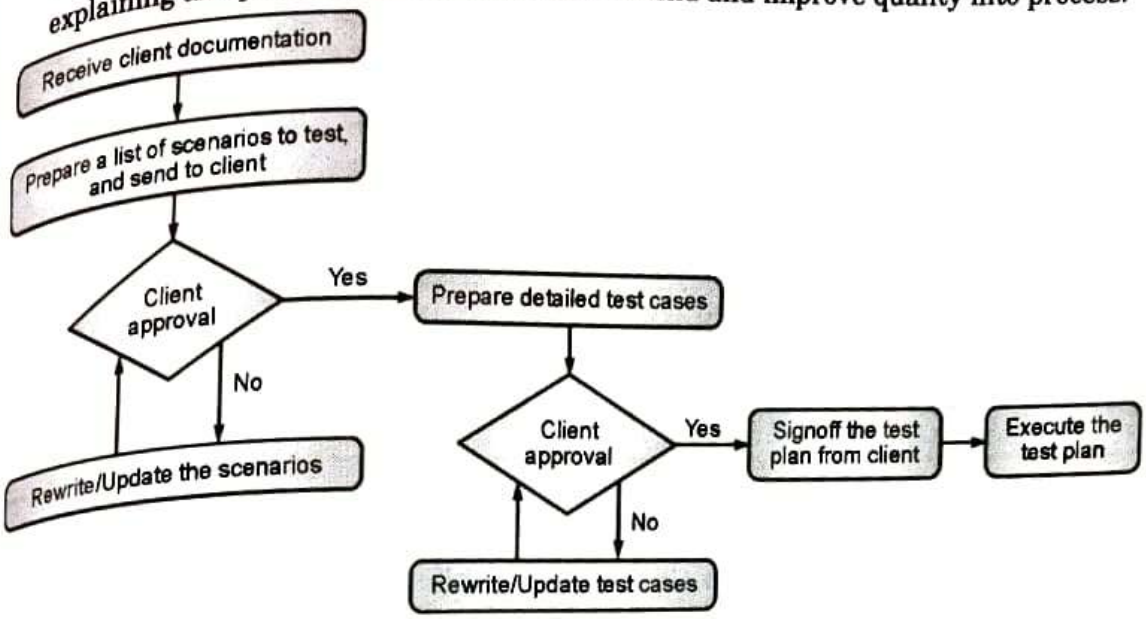


(1F8)**Fig. 6.8.5 : Scatter Diagram**

6.8.5 Flowchart

- Flowchart presents a diagrammatic picture that indicates a series of symbols to describe the sequence of steps exist in an operation or process.
- On the other hand, a flowchart visualize a picture including the inputs, activities, decision points, and outputs for using and understanding easily concerning the overall objective through process.

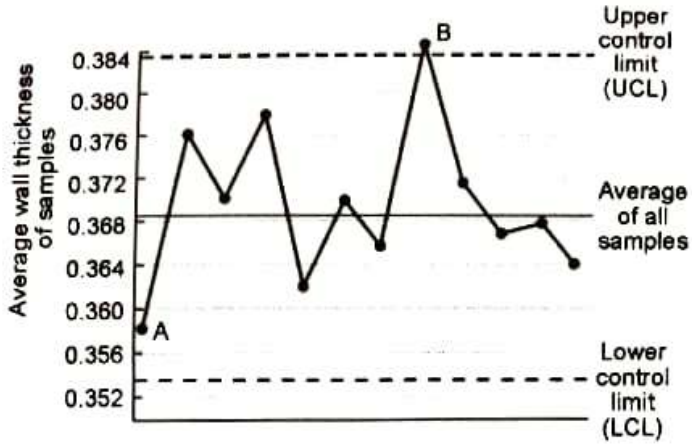
- This chart as a problem solving tool can apply methodically to detect and analyze the areas or points of process may have had potential problems by "documenting" and explaining an operation, so it is very useful to find and improve quality into process.



(1F9)**Fig. 6.8.6 : Flow Char Review Process**

6.8.6 Control Chart

- Control chart or Shewhart control chart was introduced and developed by Walter A. Shewhart in the 1920s at the Bell Telephone Laboratories, and is likely the most "technically sophisticated" for quality management.
- Control charts are a special form of "run chart that it illustrates the amount and nature of variation in the process over time". Also, it can draw and describe what has been happening in the process. Therefore, it is very important to apply control chart, because it can observe and monitor process to study process that is in "statistical control" (No problem with quality) accordant to the samplings or samplings are between UCL and LCL (Upper Control Limit (UCL) and the Lower Control Limit (LCL)).
- "Statistical control" is not between UCL and LCL, so it means the process is out of control, then control can be applied to find causes of quality problem, as shown in Fig. 6.8.7 that A point is in control and B point is out of control.
- In addition, this chart can be utilized for estimating "the parameters" and "reducing the variability" in a process.
- The main aim of control chart is to prevent the defects in process. It is very essentiality for different businesses and industries, the reason is that unsatisfactory products or services are more coasted than spending expenses of prevention by some tools like control charts .



(1F10)**Fig. 6.8.7 : The Shewhart control chart**

6.9 SELF-LEARNING TOPICS

6.9.1 Case Studies to Identify Quality Attributes Relationships for Different Types of Applications (Web-based, Mobile-based, etc)

- The mobile Internet promised comparable flexibility and cost efficiency to the normal web. However, experiences indicate that the development of mobile web applications needs to consider special challenges in the areas of usability, development efficiency and runtime consideration.
- The major challenge of mobile application development is the heterogeneity of mobile devices and web browsers installed on the devices. The differences in the form factors and input capabilities strongly influence the usability of an application. In addition the pre-installed browsers differ between the devices.
- Currently most mobile devices, either support WML or subset of (X)HTML. The different markup languages pose threats to application development that are either automatically or manually adapted to the different languages.
- Finally the wireless network connection causes additional threats, such as higher network delay, limited bandwidth or dropped connections. These challenges will be investigated in depth in the "Challenges for Mobile Application Development" section. These problems may be considered as quality problems. However, while several studies on quality for e-commerce systems have been done (i.e. [StXe01]) there are no such studies for mobile applications.
- An adapted quality model that investigates special issues in quality assurance is helpful in two ways. First the consideration of special quality features help implementers increasing the quality of their applications and leverage the acceptance of mobile web applications.

- Therefore quality assurance methods contribute in overcoming some of today's challenges in mobile application development. Second a quality model will support researchers in comparing their approaches in mobile web applications development and device independence.
- The major challenges to application development when building mobile web applications and maps the challenges to the well-known ISO 9126 quality attributes to the special needs of mobile web applications. The paper is organized as follows :
  - The first chapter introduces the ISO 9126 norm.
  - The second chapter describes the major challenges of mobile web applications development and links those challenges to the affected quality features.
  - Chapter three summarises the findings of chapter two and indicates the parts of the ISO 9126 norm that are especially affected. The paper concludes with an outlook on future work that is required in order to archive the described goals. Software quality is a complex concept containing a large number of quality criteria.
  - These quality criteria are usually interactive where preference outcome of one criterion over another criterion is always influenced by the remaining criteria. However, to avoid complexity researchers always tend to construct independent criteria which causing some bias effect in evaluation.
  - This paper presents a method to investigate and identify the relationships of quality criteria in the development of web-based applications (WBA). Experienced-based approach and an online survey were conducted to gather the intended relationships.
  - The results have shown that there exist relationships and interaction between the quality attributes.
- There are 6.8 billion people on the planet, 5.1 billion of whom own a cell phone. And today, an ever-growing percentage of these devices are smart phones.
- According to a recent Pew Research Center Study, the number of users accessing the Internet on their smart phones has more than doubled in the past 5 years, as has the number of users downloading and using mobile apps.
- Of those who use the Internet or email on their phones, more than a third go online primarily through their handheld devices.
- Indeed, mobile computing is becoming increasingly ubiquitous... and it's awesome. Except, of course, when it's not.
- As a mobile device user, few things are as frustrating and difficult to fat-finger navigate as a poorly designed mobile web app, or even a native app.
- And as a mobile app developer, few things can be as intensely irritating as striving to support as wide a range of mobile clients as possible, each of which has its own frustrating set of idiosyncrasies.

- Whether you choose to develop a mobile web, native, or hybrid app, the quest to support multiple mobile browsers, more exotic devices, and coming to grips with various platforms can be quite a gut wrenching experience indeed.
- This mobile web app development tutorial seeks to help you navigate different browsers and platforms.
- As a mobile device user, few things are as frustrating and difficult to fat-finger-navigate as a poorly designed mobile web or native app.
- And as a mobile app developer, few things can be as intensely irritating as striving to support as wide a range of mobile clients as possible, each of which has its own frustrating set of idiosyncrasies.
- Of course, not every developer today needs to worry about supporting mobile clients. But the increasingly omnipresent nature of mobile devices and applications strongly suggests that those who don't need to support mobile clients today will more than likely need to do so in the not-too-distant future. So if you're not already thinking about mobile app development, you probably should be.

☞ Mobile Web App vs. Native App vs. Hybrid App

- As is true with most technology selections, there's no one-size-fits-all answer when it comes to the type of mobile app to develop. There are numerous web app best practices to consider, not all of which are technical.
- Who is your target audience? Are they more likely to prefer a mobile web or a native app? What's the difference between native and hybrid apps ?
- What development resources do you have and which mobile technologies are they most familiar with? What is the licensing and sales model that you're envisioning for your product?
- Generally speaking (although there are always exceptions), the mobile web app route is faster and cheaper than the native mobile app route, especially when the objective is to support a wide range of devices.
- Conversely, there may be capabilities native to the mobile device (such as the movement sensor and so on) that are essential to your app, but which are only accessible via a native app (which would therefore make the mobile web app choice a non-starter for you).
- And beyond the old web apps vs. native apps question, a hybrid mobile app may be the right answer for you, depending on your requirements and resource constraints.
- Hybrid apps, like native apps, run on the device itself (as opposed to inside a browser), but are written with web technologies (HTML5, CSS and JavaScript) and typically underpinned by a hybrid app framework.

- More specifically, hybrid apps run inside a native container, and leverage the device's browser engine (but not the browser) to render the HTML and process the JavaScript locally.
- A web-to-native abstraction layer enables access to device capabilities that are not accessible in mobile web applications, such as the accelerometer, camera, and local storage.
- But whatever choice you make – whether it be a mobile web app, a native or hybrid app – be careful to adequately research and confirm your assumptions. As an example, for the purposes of this mobile web app development tutorial, you may have decided to develop a native mobile app for e-commerce to sell your products. But, according to Hubspot, 73% of smart phone users say they use the mobile web more than native apps to do their shopping... So, in this case, you may have bet on the wrong horse.
- But whatever choice you make – whether it be mobile web, native or hybrid app – be careful to adequately research and confirm your assumptions.
- And then, of course, there are the practical considerations of time and budget. As one of my favorite sayings goes, "faster, better, cheaper... pick any two". While time-to-market and cost constraints are of paramount importance in web application development, it's crucial not to compromise too heavily on quality in the process. It's quite difficult to recover the confidence of a user who has had a bad first experience.
- Indeed, mobile web, native, and hybrid apps are all radically different beasts, each with their own unique set of benefits and challenges. This mobile web development tutorial specifically focuses on methodologies and tools to employ, and pitfalls to avoid, in the development of highly functional, intuitive, and easy-to-use mobile web applications.
- A critical best practice in determining how to develop a mobile web application is to know your customer.

Mobile Web App Development Requires Detailed Planning

- Identifying your (or your customer's) requirements is one of the most essential best practices in app development, mobile or otherwise. Carefully research the targeted capabilities to determine if they are achievable in your mobile web app.
- It's quite frustrating, and highly unproductive, to realize that one or more of your essential client functions aren't supported, when you've already invested the time and resources to design the web-based interface and supporting infrastructure.
- Another common gotcha for mobile web app developer newbies is to assume that web-based code for a desktop browser will work "as is" in a mobile browser. Not. There most definitely are differences and, if you're not aware of them, they can definitely bite you.

- The HTML5 <video> tag's autoplay functionality, for example, doesn't work on mobile browsers. Similarly, the CSS transition and opacity properties are not supported (or at least are not consistently supported) in most mobile browsers nowadays. You will also have problems with some web API methods on a mobile platform, such as the Sound Cloud music streaming API that requires Adobe Flash which is not supported on most mobile devices.
- A common gotcha for mobile web app developer newbies is to ass-u-me that web-based code for a desktop browser will work "as is" in a mobile browser.
- A particularly complicating factor in mobile web application development is that the lifespan of mobile devices tends to be much shorter than that of desktop displays (the average lifespan of a cell phone in the U.S. is around 21 months). These shorter device life spans, accompanied by constant releases of new mobile devices and technologies, yield an ever-changing landscape of to-be-targeted devices.
- While working in a browser does somewhat alleviate this issue by shielding you from a number of device-specific issues, you will still need to design a browser-based view that supports many different screen resolutions (as well as adjusting appropriately for landscape and portrait orientations).
- Thought needs to be given as well to supporting Apple's Retina Displays (liquid crystal displays that have a pixel density high enough that the human eye is unable to discern individual pixels at a typical viewing distance). Several Apple products – including the iPhone, iPod Touch, iPad, MacBook Pro, iPad Mini and iPad Air - offer Retina displays.
- For a mobile web app in particular, it's important to be aware that a Retina display makes low resolution images (which are typically served to mobile devices) look fuzzy and pixelation can occur.
- The best app development solution in these cases is to have the server recognize that the request is coming from a Retina device and to then provide an alternate higher resolution image to the client.
- If you want to use some of the cool HTML5 stuff, remember to verify in advance that the functionality you're looking for is supported across the device landscape that your customers are likely to be using.
- For example, in iOS 6 and above, there is no support for the navigator getUserMedia functionality since the camera is only accessible through native apps. Two great resources for checking what's supported on specific devices and browsers are caniuse.com and html5test.com.

Remember to verify in advance that the functionality you're looking for is supported across the device landscape that your customers are likely to be using.

- CSS3 media queries can also help you provide customized content for each device. Here's some example code for capturing different device characteristics, such as pixel density, screen resolution, and orientation:

```
/* For lower than 700px resolutions */
@media (max-width: 700px) { ... }
/* Same as last but with the device orientation on land scape */
@media (max-width: 700px) and (orientation: landscape) { ... }
/* Including width and orientation you can add a media type clause,
in this case 'tv' */
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
/* for low resolution display with background-image */
.image {
    background-image: url(/path/to/my/image.png);
    background-size: 200px 300px;
    height: 300px;
    width: 200px;
    }
    /* for high resolution (Retina) display with background-image */
    @media only screen and (min--moz-device-pixel-ratio: 2),
    only screen and (-o-min-device-pixel-ratio: 2/1),
    only screen and (-webkit-min-device-pixel-ratio: 2),
    only screen and (min-device-pixel-ratio: 2) {
  -repeat;
 background-size: 200px 400px;
  /* rest of your styles... */
 }
}
```

Optimizing Your Mobile Web Application for Performance

- "OMG, this thing is so slow!" As a mobile web app developer, those are probably the very last words you ever want to hear from one of your users. You must therefore think carefully about how to reduce and optimize each byte and server transfer to reduce the user's wait time.
- It's unrealistic to expect that transfers will always be done over a WiFi network, and you should know that 60% of mobile web users say they expect a site to load on their mobile phone in 3 seconds or less (source).
- Similarly, Google found that, for every extra five seconds of load time, traffic dropped by 20% (and it is also worth noting that search engines look at load times as part of their calculation of page quality score).

- 60% of mobile web users say they expect a site to load on their mobile phone in 3 seconds or less.
- As a part of this mobile web app development tutorial, here are a few tips that can help optimize the performance of your mobile web applications and minimize latency:
- Image Optimization. Image load time is well-known to be one of the biggest performance issues affecting page load on mobile devices. Use of online image optimizers, such as smushit.com, can be helpful in addressing this issue.
- Code compression. Compressing your JavaScript and CSS files, depending on the amount of code you have, can potentially have a significant impact on performance.

☞ **Database queries**

- Some mobile device browsers don't accept as many cookies as desktop browsers do, which can result in the need to execute even more queries than usual. Server-side caching is therefore especially crucial when supporting mobile web app clients.
- Remember to employ the appropriate filters to preclude SQL query injection that could otherwise compromise the security of your site and server.
- Content delivery networks (CDN). If you are planning to provide lots of videos, images, audio files, or other types of media, use of a CDN is highly recommended. Some of the more common commercial CDNs include Amazon S3, Microsoft Windows Azure, and MaxCDN. The advantages of using a CDN are numerous and include:
- Improved download performance. Leveraging a CDN's resources enables you to distribute load, save bandwidth, and boost performance. The better CDNs offer higher availability, lower network latency, and lower packet loss.
- Moreover, many CDNs provide a globally distributed selection of data centers, enabling downloads to occur from a server closer to the user's location (resulting in fewer network hops and faster downloads).
- More concurrent downloads. Browsers typically limit the number of concurrent connections to a single domain, after which additional downloads are blocked until one of the previous downloads has completed.
- You can often see this limit in action when downloading many large files from the same site. Each additional CDN (on a different domain) allows for additional concurrent downloads.
- Enhanced analytics. Many commercial CDNs provide usage reports that can supplement your own website analytics and which may offer a better quantification of video views and downloads. GTmetrix, for example, has an excellent website reporting tool for monitoring and optimizing the sources loaded on your site.

Mobile Web App Development Tools

- "The right tools for the right job" is an age-old adage that applies as much to software development as it does to any other domain.
- This tutorial provides and introduction to some of the more popular and widely-used tools for mobile web app development, but bear in mind that there may very well be other tools that are the "right" ones for developing your mobile web application, depending on your requirements and available resources.

Choosing the Right JavaScript Mobile Web App Framework

- As mobile web app development tend to create many of the same common challenges – such as cross-browser compatibility and inconsistent HTML and CSS in mobile browsers - frameworks have been developed (based on HTML5 and CSS3) that are specifically designed to address these issues and to work as flawlessly as possible on a wide array of smart phones and tablets.
- Most of these mobile web app frameworks are lightweight, which helps facilitate fast mobile web browsing without compromising the look and feel of your site.
- Broadening our view beyond the mobile landscape, if there is a single popular JavaScript framework worth mentioning, it is jQuery. If you're familiar with the desktop version, I recommend trying jQuery Mobile for your mobile web app.
- It has a widget library that converts semantic markup into a gesture-friendly format, making operations easy on touch-screens. The latest version consists of a really lightweight code base that packs a punch with a lot of graphical elements that really can improve your UI.
- Another alternative, Sencha Touch, is rapidly gaining market share as well. It offers excellent performance overall and helps produce a mobile web user interface that largely looks and feels like a native one. Its full-featured widget library is based on Sencha's ExtJS JavaScript library.
- Here are some key differences to consider when comparing jQuery Mobile and Sencha Touch :
  - **Look and feel :** Generally speaking, the look and feel of a Sencha Touch app is crisper and superior to that of a jQuery mobile app, but it is important to remember that such reactions do tend to be highly subjective.
  - **Extensibility :** jQuery Mobile offers lots of 3rd party extensions and is inherently designed to be highly extensible, whereas Sencha Touch is currently much more of a "closed" framework.
  - **Device support :** jQuery Mobile currently targets a larger cross-section of devices than Sencha Touch.

- **HTML "vs." JavaScript.j Query** is largely HTML-centric (i.e., extending and manipulating existing HTML in JavaScript), whereas Sencha Touch coding is entirely JavaScript-based. (This is an example, incidentally, of the skill set of your development team being important to consider when making your technology selections.)
- **External dependencies** :. jQuery mobile requires jQuery and jQuery UI for DOM manipulation, whereas Sencha Touch has no external dependencies.
- **Learning curve** : Most developers find the ramp-up time with jQuery to be less than that of Sencha Touch, perhaps fueled by the large percentage of web developers who are already familiar with the standard jQuery libraries.

☞ Responsive Frameworks and Mobile Web Applications

- An increasing number of responsive frameworks have begun cropping up in recent years, with two of the currently most popular being Bootstrap and Foundation.
- In short, responsive frameworks simplify and streamline web-based responsive UI design and implementation, encapsulating the most common layouts and UI paradigms into a reusable, performance-optimized framework.
- Mostly based on CSS and JavaScript, many of these frameworks are open-source, free to download, and easily customizable.
- Unless you have a highly peculiar set of requirements, it is likely that use of one of these frameworks will reduce the level-of-effort to design and implement your mobile web application.

Examining the two leading options, Bootstrap and Foundation, a few of the key differences to consider include :

- **Targeted platforms** : While Bootstrap does support mobile, tablet, and desktop devices, it is primarily oriented toward desktop use. Foundation, on the other hand, is designed for essentially all screen sizes and types.
- **Browser compatibility** : Bootstrap is compatible with IE7 or higher, whereas Foundation is only compatible with IE9 or higher.
- **Diversity of layouts and components** : Bootstrap has a significantly larger collection of UI elements than is offered by Foundation.
- **Auto-resizing** : With Foundation, the grid shrinks and stretches according to the current browser height and width, whereas Bootstrap only supports a pre-defined set of grid sizes based on a standard set of screen sizes.

☞ Debugging and Testing Mobile Web Apps

- Debugging mobile web apps can be tricky and somewhat frustrating, especially if you need to scrounge around for different devices to test on, or install SDKs for a (typically imperfect) emulation of the targeted client platforms.

- In this context, one clear advantage of mobile web development (as compared with native app development) is that you can utilize standard browser-based developer tools to debug your application.
- Based on my personal preference for remote debugging, the one I recommend in this app development tutorial is Chrome with its DevTools. Other standard options include Firefox with Firebug or Opera's Dragonfly tools.
- When learning how to develop web applications, look toward Chrome and its DevTools.

☞ **Some of the reasons I prefer Chrome with its DevTools include**

- **Mobile emulator in Chrome's DevTools :** This is perhaps alone sufficient reason to select Chrome for debugging of mobile web apps.
- Key features include emulation of touch events, user agent spoofing, network bandwidth throttling, geolocation overrides, device orientation overrides, and CSS Media Type Emulation.

☞ **Interactive editor. Ability to edit JavaScript or CSS on-the-fly**

- Superior JavaScript debugger. Allows for DOM breakpoints and provides the ability to profile your JavaScript code execution time.
- Built-in JSON and XML viewers. Avoids the need for any plugins to inspect server responses.
- Support for the Android Debug Bridge (ADB) protocol directly over USB. Facilitates easy instantiation of a remote debugging session. (Here is a good tutorial by Google on how to start remotely debugging in Chrome.)
- Dynamic inspection of resources. Allows you to inspect your app's local data sources, including Indexed DB or Web SQL databases, local and session storage, cookies, and Application Cache resources. You can also quickly inspect your application's visual resources, including images, fonts, and style sheets.
- To test the layout and cross browsing compatibility of your web app, you can also use some helpful online tools, such as Browser Stack. Just enter the URL for your application, select the browser, version, and operating system, and you'll get the emulated view (and load speed) of your site in that environment.

Chapter Ends...

❑❑❑