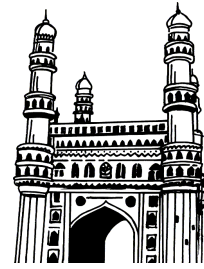


***Rahul's ✓
Topper's Voice***







M.C.A.

II Year III Sem

(Osmania University)

Latest Edition

SOFTWARE QUALITY AND TESTING

-  **Study Manual**
-  **List of Important Definitions**
-  **Important Questions**
-  **Solved Model Papers**

- by -

WELL EXPERIENCED LECTURER

Price
₹. 249-00



Since 1986

Rahul PublicationsTM

Hyderabad. Cell : 9391018098, 9505799122.

All disputes are subjects to Hyderabad Jurisdiction only

Subjects List

Common

- Software Engineering
- Computer Networks
- Artificial Intelligence
- Web Technologies

Professional Elective-I

- Software Quality & Testing
- Distributed Systems
- Internet of Things
- Image Processing

Professional Elective-II

- Network Security
- Cyber Security
- Information Retrieval System
- Natural Language Processing

In spite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

© No part of this publication should be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publisher

Sole Distributors :

Cell : 9391018098, 9505799122

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

C O N T E N T S

SOFTWARE QUALITY AND TESTING

STUDY MANUAL

List of Important Definitions	III - IV
Important Questions	V - VIII
Unit - I	1 - 14
Unit - II	15 - 42
Unit - III	43 - 60
Unit - IV	61 - 86
Unit - V	87 - 124

SOLVED MODEL PAPERS

Model Paper - I	125 - 126
Model Paper - II	127 - 128
Model Paper - III	129 - 130

SYLLABUS

UNIT - I

The Software Quality Challenge, Introduction Software Quality Factors, The Components of the Software Quality Assurance System – Overview, Development and Quality Plans.

UNIT - II

Integrating Quality Activities in the Project Life Cycle, Assuring the Quality of Software Maintenance Components, CASE Tools and their effect on Software Quality, Procedure and Work Instructions, Supporting Quality Devices, Configuration Management, Documentation Control, Project Progress Control.

UNIT - III

Software Quality Metrics, Costs of Software Quality, Quality Management Standards - ISO 9000 and Companion ISO Standards, CMM, CMMI, PCMM, Malcom Balridge, 3 Sigma, 6 Sigma, SQA Project Process Standards – IEEE Software Engineering Standards.

UNIT - IV

Building a Software Testing Strategy, Establishing a Software Testing Methodology, Determining Your Software Testing Techniques, Eleven – Step Software Testing Process Overview, Assess Project Management Development Estimate and Status, Develop Test Plan, Requirements Phase Testing, Design Phase Testing, Program Phase Testing, Execute Test and Record Results, Acceptance Test, Report Test Results, Test Software Changes, Evaluate Test Effectiveness.

UNIT - V

Testing Client / Server Systems, Testing the Adequacy of System Documentation, Testing Web-based Systems, Testing Off – the – Shelf Software, Testing in a Multiplatform Environment, Testing Security, Testing a Data Warehouse, Creating Test Documentation, Software Testing Tools, Taxonomy of Testing Tools, Methodology to Evaluate Automated Testing Tools, Load Runner, Win Runner and Rational Testing Tools, Java Testing Tools, JMetra, JUNIT and Cactus.

Contents

UNIT - I

Topic	Page No.
1.1 The Software Quality Challenge	1
1.2 Introduction Software Quality Factors	5
1.3 The Components of the Software Quality Assurance	8
1.4 System Overview	10
1.5 Development and Quality Plans	11

UNIT - II

2.1 Integrating Quality Activities in the Project Life Cycle	15
2.2 Assuring the Quality of Software Maintenance Components	21
2.3 CASE Tools and Their Effect on Software Quality	24
2.4 Procedure and Work Instructions	25
2.5 Supporting Quality Devices	28
2.6 Corrective and Preventive Actions	34
2.7 Configuration Management	36
2.8 Documentation Control	40
2.9 Project Progress Control	40

UNIT - III

3.1 Software Quality Metrics	43
3.1.1 Costs of Software Quality	43
3.2 Quality Management Standards - ISO 9000 and Companion ISO Standards	45
3.3 CMM	46
3.4 CMMI	49
3.5 PCMM	51
3.6 Malcom Balridge	52
3.7 3 Sigma	53
3.8 6 Sigma	54
3.9 SQA Project Process Standards	59
3.10 IEEE Software Engineering Standards	59

Topic**Page No.****UNIT - IV**

4.1	Building a Software Testing Strategy	61
4.2	Establishing a Software Testing Methodology	66
4.3	Determining Your Software Testing Techniques	70
4.4	Eleven Step Software Testing Process Overview	73
4.5	Assess Project Management Development Estimate and Status	74
4.6	Develop Test Plan	76
4.7	Requirements Phase Testing	76
4.8	Design Phase Testing	77
4.9	Program Phase Testing	79
4.10	Execute Test and Record Results	80
4.11	Acceptance Test	83
4.12	Report Test Results	85
4.13	Test Software Changes	85
4.14	Evaluate Test Effectiveness	86

UNIT - V

5.1	Testing Client / Server Systems	87
5.2	Testing the Adequacy of System Documentation	92
5.3	Testing Web-based Systems	93
5.3.1	Testing Off the Shelf Software	95
5.4	Testing in a Multiplatform Environment	96
5.5	Testing Security	97
5.6	Creating Test Documentation	102
5.7	Software Testing Tools	106
5.8	Taxonomy of Testing Tools	108
5.9	Methodology to Evaluate Automated Testing Tools	109
5.10	Load Runner	112
5.11	Win Runner	115
5.12	Rational Testing Tools	116
5.13	Java Testing Tools	117
5.14	JMetra	119
5.15	JUNIT and Cactus	121

LIST OF IMPORTANT DEFINITIONS

UNIT - I

1. Software is Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
2. Software quality factors, also known as software quality characteristics or software quality attributes, are the key aspects of software that are used to evaluate its overall Quality.
3. Quality Assurance is also known as QA Testing. QA is defined as an activity to ensure that an organization is providing the best product or service to the customers.
4. Software Quality Assurance (SQA) is a system that ensures the quality of software products by identifying and fixing issues throughout the software development process. SQA is a critical part of the software development process and helps companies build high-quality products that meet customer expectations.
5. A development plan is a document that outlines the activities involved in the software development process. This plan provides of roadmap for the entire development team including project managers, developers, and testers, to follow throughout the project life cycle.

UNIT - II

1. The prototyping model is based on replacement of one or more SDLC model phases by an evolutionary process, where software prototypes are used for communication between the developer and the users and customers. Prototypes are submitted to user representatives for evaluation.
2. The spiral model, as revised by Boehm (1988, 1998), offers an improved methodology for overseeing large and more complex development projects displaying higher prospects for failure, it combines an iterative model that introduces and emphasizes risk analysis and customer participation into the major elements of SDLC and prototyping methodologies.
3. Testability requirements deal with the testing of an information system as well as with its

operation. Testability requirements related to software operation include automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the software system are in working order and to obtain a report about the detected faults.

4. Maintenance plans should be prepared for all customers, external and inter-nal. The plan should provide the framework within which maintenance provision is organized.
5. Computer-Aided Software Engineering tools are software applications that support various activities in the software development process, such as requirements management, design, coding, testing, and maintenance.

UNIT - III

1. Software quality metrics are quantitative measures used to evaluate and assess the quality of software products or processes.
2. ISO 9000 is defined as a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements needed to maintain an efficient quality system.
3. Key Process Areas defines the basic requirements that should be met by a software process to satisfy the KPA and achieve that level of maturity.
4. Capability Maturity Model Integration (CMMI) is a successor of CMM and is a more evolved model that incorporates best components of individual disciplines of CMM like Software CMM, Systems Engineering CMM, People CMM, etc.
5. The PCMM is a working framework used for the organization in defining the maturity structure for improving and developing the skill set of people who work within the organization.
6. Six Sigma is a methodology used by most organizations for process improvement, and It is a statistical concept that aims to define the variation found in any process. Six Sigma is a process of producing high and improved quality output.

LIST OF IMPORTANT DEFINITIONS

UNIT - IV

1. A software testing strategy document is a comprehensive plan that outlines the testing approach, goals, scope, resources, and timelines for a software project.
2. Testing methodologies are the strategies and approaches used to evaluate a particular product to ensure it performs as expected and is easy to use.
3. In White Box Testing, testers verify systems they are deeply acquainted with, sometimes even ones they have created themselves. No wonder white box testing has alternate names like open box testing, clear box testing, and transparent box testing.
4. Functional tests are designed and run to verify every function of a website or app. It checks that each function works in line with expectations set out in corresponding requirements documentation.
5. Each individual component is tested before the developer pushes it for merging. Unit tests are created and run by the devs themselves.
6. Test Execution is the process of executing the tests written by the tester to check whether the developed code or functions or modules are providing the expected result as per the client requirement or business requirement. Test Execution comes under one of the phases of the Software Testing Life Cycle

UNIT - V

1. Client-server testing is a testing approach designed to verify the accurate and secure exchange of data between the client and server, guaranteeing that requests and responses are synchronized correctly.
2. Web testing is a software testing technique to test web applications or websites for finding errors and bugs. A web application must be tested properly before it goes to the end-users.
3. Security testing is a type of software testing that focuses on evaluating the security of a system or application.
4. A test strategy outlines the approach, goals, and standards for testing activities within a project. It provides a structured framework for defining testing scope, objectives, methodologies, and tools, ensuring consistency and alignment with project requirements.
5. Load Runner is one of the oldest performance testing tools on the market. It is used to test the behavior and performance of applications under load.

Important Questions

UNIT - I

1. What is software ? Explain about the types of software.

Ans :

Refer Unit-I, Page No. 1, Q.No. 1

2. Discuss the causes of software error.

Ans :

Refer Unit-I, Page No. 3, Q.No. 3

3. Write about the strategies to over come quality assurance testing challenges.

Ans :

Refer Unit-I, Page No. 5, Q.No. 5

4. What is McCall's Software Quality Model?

Ans :

Refer Unit-I, Page No. 6, Q.No. 7

5. What are the components of Software Quality assurance?

Ans :

Refer Unit-I, Page No. 8, Q.No. 11

6. Write the overview of a Software Quality Assurance.

Ans :

Refer Unit-I, Page No. 10, Q.No. 14

7. Explain the objectives of a development plan and a quality plan?

Ans :

Refer Unit-I, Page No. 12, Q.No. 17

UNIT - II

1. What are the Factors affecting intensity of quality assurance activities in the development process?

Ans :

Refer Unit-II, Page No. 19, Q.No. 2

2. What are the various requirements of software quality.

Ans :

Refer Unit-II, Page No. 21, Q.No. 5

3. What are the various performance controls for software maintenance.

Ans :

Refer Unit-II, Page No. 24, Q.No. 8

4. Explain the difference between procedures and work instructions?

Ans :

Refer Unit-II, Page No. 27, Q.No. 13

5. What are the uses of checklists in software quality Checklists.

Ans :

Refer Unit-II, Page No. 30, Q.No. 15

6. Explain the importance of staff training and certification to improve the software quality.

Ans :

Refer Unit-II, Page No. 31, Q.No. 16

7. Explain about various corrective and preventive actions to improve the software quality.

Ans :

Refer Unit-II, Page No. 34, Q.No. 17

8. Explain about software configuration management plans.

Ans :

Refer Unit-II, Page No. 38, Q.No. 19

9. Explain the components of project progress control?

Ans :

Refer Unit-II, Page No. 40, Q.No. 22

UNIT - III

1. Explain briefly about software quality Metrics.

Ans :

Refer Unit-III, Page No. 43, Q.No. 1

2. Write about the costs of software quality.

Ans :

Refer Unit-III, Page No. 43, Q.No. 2

3. Write ISO 9000 standards for quality management.

Ans :

Refer Unit-III, Page No. 45, Q.No. 3

4. What is the Capability Maturity Model (CMM)? Explain.

Ans :

Refer Unit-III, Page No. 46, Q.No. 4

5. What is Capability Maturity Model Integration (CMMI)?

Ans :

Refer Unit-III, Page No. 49, Q.No. 6

6. What is PCMM? Explain various methods of PCMM.

Ans :

Refer Unit-III, Page No. 51, Q.No. 9

7. Explain the 6 Key Principals of Six Sigma.

Ans :

Refer Unit-III, Page No. 55, Q.No. 13

8. Explain about IEEE Software engineering standards.

Ans :

Refer Unit-III, Page No. 59, Q.No. 16

UNIT - IV

1. What is to Include in a Test Strategy Document? Explain.

Ans :

Refer Unit-IV, Page No. 61, Q.No. 2

2. Differentiate between test strategy and test plan.

Ans :

Refer Unit-IV, Page No. 63, Q.No. 5

3. Explain how a test strategy document enhances project outcomes ?

Ans :

Refer Unit-IV, Page No. 64, Q.No. 6

4. What are Software Testing Techniques? Explain.

Ans :

Refer Unit-IV, Page No. 70, Q.No. 11

5. Discuss the Assess Project Management Development Estimate and Status.

Ans :

Refer Unit-IV, Page No. 74, Q.No. 14

6. Define design phase testing. State the objectives and key activities of design phase testing?

Ans :

Refer Unit-IV, Page No. 77 Q.No. 18

7. Define program phase testing. State its activities and objectives?

Ans :

Refer Unit-IV, Page No. 79, Q.No. 20

8. Discuss the various Ways to Perform Test Execution?*Ans :*

Refer Unit-IV, Page No. 81, Q.No. 25

9. What is Acceptance Testing? Explain various types of acceptance testing?*Ans :*

Refer Unit-IV, Page No. 83, Q.No. 29

UNIT - V**1. What are the Objectives of client server testing.***Ans :*

Refer Unit-V, Page No. 87, Q.No. 2

2. Define webtesting. State various types of web testing?*Ans :*

Refer Unit-V, Page No. 93, Q.No. 8

3. What is off the shelf software? State its disadvantages?*Ans :*

Refer Unit-V, Page No. 95, Q.No. 12

4. Cross-Platform Testing Best Practices.*Ans :*

Refer Unit-V, Page No. 96, Q.No. 14

5. Define security testing.Explain the goals of security testing?*Ans :*

Refer Unit-V, Page No. 97, Q.No. 17

6. State the advantages and disadvantages of security testing?*Ans :*

Refer Unit-V, Page No. 99, Q.No. 19

7. Discuss the benetifts and challenges to data warehouse testing?*Ans :*

Refer Unit-V, Page No. 100, Q.No. 21

8. Discuss briefly about various Software Testing Tools.*Ans :*

Refer Unit-V, Page No. 106, Q.No. 28

9. Describe the concept of Java testing tools?*Ans :*

Refer Unit-V, Page No. 117, Q.No. 34

UNIT I

The Software Quality Challenge, Introduction Software Quality Factors, The Components of the Software Quality Assurance System — Overview. Development and Quality Plans.

1.1

THE SOFTWARE QUALITY CHALLENGE

Q1. What is software ? Explain about the types of software.

Ans :

(Imp.)

According to the IEEE

1. Software is Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
2. A 'similar definition comes from ISO ISO definition (from ISO 9000-3) lists four components necessary to assure the quality of the software development process and years of maintenance:

Computer programs (code)

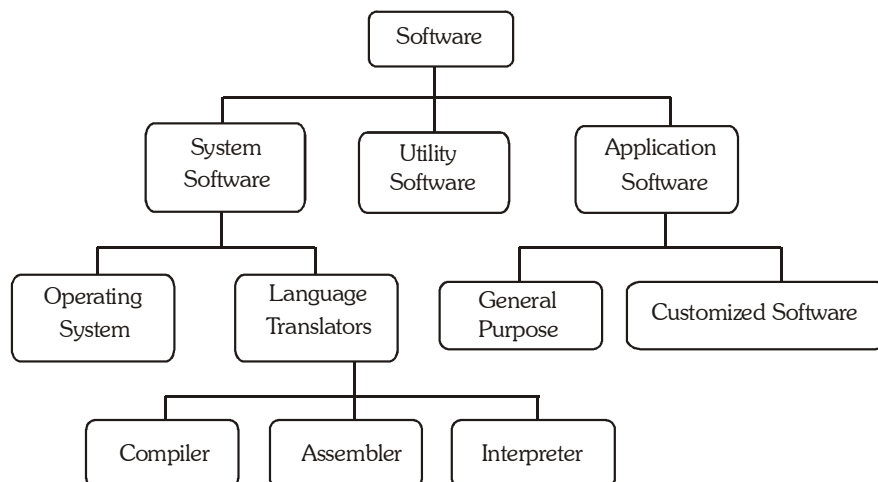
Procedures

Documentation

Data necessary for operating the software system.

Software can be broadly are categorized as :

- System Software
- Application Software
- Utility Software



Types of Software

1. **System Software** : System Software is the software that is directly related to coordinating computer operations and performs tasks associated with controlling and utilizing computer hardware. These programs assist in running application programs and are designed to control the operation of a computer system.

System software directs the computer what to do, when to do and how to do.

System software can be further categorized into :

- Operating System
- Language Translators
- **Operating System** : A user cannot communicate directly with the computer hardware, so the operating system acts as an interface between the user and the computer hardware.
- **Language Translators** : The special translator system software that is used to translate the program written in high-level language (or Assembly language) into machine code is called language processor or translator program.

2. **Application Software** : An application software is bought by the user to perform specific applications or tasks, say for example making a document or making a presentation or handling inventory or managing the employee database.
3. **Utilities Software** : A utility software is one which provides certain tasks that help in proper maintenance of the computer. The job of utility programs is to keep the computer system running smoothly.

Q2. Write about the classification of software errors.

Ans : (Imp.)

By their nature, there are such software glitches.

1. Functional Errors

If the behavior of the software does not meet the functional requirements, then this type of error is detected. They can be found through functional testing. For example, in a past test project that ran on an e-commerce website search engine,

we found a functional bug. When the user entered the product ID, it did not show any results, even though the search can be done by both the product name and the ID.

2. Performance Issues

Performance problems are primarily related to the response time and resource consumption of the software and its speed and stability. They are discovered during performance testing. When the system response time is X times longer than specified in the requirements, this is an issue for investigation and correction.

3. Usability Errors

The application becomes inconvenient to use because of the lack of usability, making it difficult for the user to work. Examples of usability bugs include a content layout that is difficult to scan or move or a very difficult and time-consuming registration process.

4. Compatibility Bugs

We do compatibility testing to identify compatibility malfunction. After all, an application with compatibility errors does not integrate properly with the third-party software, or does not work in certain network configurations, and does not show stable performance on certain hardware, operating systems, browsers, and particular devices.

5. Security Vulnerabilities

Security vulnerabilities are weak points that allow for a potential security attack. The most common security flaws in projects for which we conduct security testing are encryption bugs, XSS vulnerability, SQL injection vulnerabilities, weak authentication, buffer overflows, and logical errors in role-based access.

6. Software Errors by Severity

Software error classification by severity is based on the technical impact the bugs/errors have on the system. While testing the client's software, we can distinguish the following levels of severity:

7. Critical Errors

They always block the entire system's operation, and in this case, testing cannot be carried out. For example, when a critical bug occurs, a message sent by the application is constantly

returned, that an error prevents us from logging in.

8. Issues of High Severity

They affect the main functions of the application, without which its operation is impossible, and the application behaves completely differently from what is specified in its requirements.

9. Issues of Moderate Severity

They are detected when one of the functions does not behave as specified in the requirements. An example of such a bug is a broken link in the privacy policy section.

10. Errors or Bugs of Low Importance

An example can be the bug with the user interface. These bugs have no significant influence on the functionality, for example, a different color or size of the button, or ugly looking dashboard, squeezed on the tablet, while nice looking on the mobile screen.

Q3. Discuss the causes of software error.

Aus : (Imp.)

1. Faulty definition of requirements

The faulty definition of requirements usually caused by the client which not clear in giving the requirement list which can cause the software errors.

The most common errors are:

- Erroneous definition of requirements
- Absence of vital requirements
- Incomplete definition of requirement (Ex: The use cases often don't provide enough detail for developers to know what to build.
- Because the user do not derive specific software functional requirements

2. Client-developer communication failures

Communication is very important, especially when you work in team which include many different people.

There will be a lot of differentiation when delivering the material.

Misunderstanding in a project can caused a major failure moreover in a development process.

These 3 most failure in a client-developer communication failures based on Daniel Gallin;

- i) Misunderstanding of the client's instructions stated in the requirements document.
- ii) Misunderstanding of the client's requirements changes presented to the developer in written form during the development period.
- iii) Misunderstanding of the client's requirements changes presented orally to the developer during the development period.
- iv) Misunderstanding of the client's responses to the design problems presented by the developers.

3. Deliberate deviations from software requirements

In some cases, deviation are deliberately made by the developer from the documented requirements, this thing can caused software errors. The errors in this case usually by the changing of the products.

The most common deviation are:

- i) The developer reuses software modules
- ii) Due to the time or budget pressures
- iii) Due to the unapproved improvements (Ex; disregard requirements that seem in or to the developer)

4. Logical design errors

Logical design is the cause of the fatal error of manufacturing or software development.

This error occurs because some of the omissions committed by the programmer or developer in the early initiation of the program.

Such errors can be errors in determining the program algorithm which includes error theory is used.

For example, errors in the calculation formula, execution process, erroneous algorithms that represent software requirements, omission of required software system states and erroneous definition of boundary conditions.

5. Coding errors

No matter what language is used when coding, there are common errors that can lead to big problems, if they're not dealt with early.

These coding errors can result in severe breaches in security and unstable platforms, costing time, effort and money to repair.

Recently, even big companies like Sony have suffered from programming errors requiring extensive fixes to protect their customer's private information online. One small error in the coding required the removal of part of the Sony PlayStation Network.

6. Size of Program

- Other programs must be able to run in environment.
- **Coding:** Data Elements and Codes: AFM 300-4;
- **Development Procedures:** Required documentation manuals and operating instructions; AFSDCM 300-8, etc.
- **SQA Team:** testing not only execution software but coding standards; manuals, messages displayed; resources needed; resources named (file names, program names,...) Shortcomings of the testing process

Q4. What are software quality challenges? Explain.

Ans :

Challenge

1. Incomplete or Inaccurate Requirements

Some clients fail to communicate their requirements clearly to the stakeholders during the requirement-freezing stage. This can lead to misunderstanding of the functionality by the BA and they draft the RSD document based on their understanding. This can lead to functionality mismatch and affect the quality of the application thus lowering the credibility of the organization.

2. Communication Breakdowns

Poor interaction between the project members can also lead to so much of ambiguities affecting the output quality.

3. Limited Resources

An inadequate Quality Assurance team can also fail to meet the desired quality due to a lack of manpower. The pressure goes on the limited resources which can be exhausting and lead to improper testing of the functionalities. The final

result is quality cannot be assured to the end users.

4. Changing Requirements

Some clients are so choosy that they keep on changing requirements after seeing the module deliveries. New changes added would need more time for implementation and testing thereby affecting the delivery timeline. Also, sometimes changes could be confusing as developers could develop something completely different from what is expected. It is best to approach changes as enhancements and do it after delivering the project to avoid any confusion.

5. Technical Challenges

Covering all possible scenarios in the test document could be tedious when there is a short time frame. So, skilled resources with automation knowledge are required to overcome this problem and cover wide scenarios to deliver a minimal-error product.

6. Testing in Production Environments

Every QA person needs to ensure the application is working fine in the production URL given to the client. This is achieved either by accessing the client machine remotely or checking the URL given to the client. The application could be down on the client machine or some major functionalities breakdown would be there. The client needs to provide maximum cooperation to overcome problems in a production environment. It is because some functionalities can work fine on production URLs but not on the client machine, Testers need to be smart in talking to the clients and finding out the problem and resolving it for them.

7. Test Automation Challenges

There are advanced automation tools that can do testing in a fast-paced way efficiently. Finding a talented automation tester for the project who will perform the testing smartly is another big challenge to reaching maximum customer satisfaction. Finding the right tester who shares knowledge about automation tools and programming language is equally important for the betterment of the product.

8. Balancing Speed and Quality

Speed and quality go hand in hand in hasty projects as the product quality cannot be

compromised. Assigning a big team with enough resources to dedicatedly run the test cases and check all possible scenarios can lead to a fast process and promise quality outputs. Do it hastily but without missing any important scenarios to avoid quality lags in the deliverable.

Q5. Write about the strategies to overcome quality assurance testing challenges.

Ans : (Imp.)

Strategies for Overcoming Quality Assurance Testing Challenges

There are certain strategies to be followed by the Quality Assurance testing team to reach the project goals in spite of discrepancies. Following them can really help and allow the QA team to deliver high-quality products to the end users.

1. Establishing Clear Requirements and Priorities

QA team needs to be sharp in analyzing the requirements and clear their doubts with the clients to avoid any confusion. They can confirm with the client once on the expectation of the functionality to deliver it with perfection. Raise a concern immediately to the clients when there is an unclear document and get it resolved on time. Also, set your priorities straight about which module needs to be delivered first and work closely towards delivering them.

2. Improving Communication and Collaboration

QA members need to communicate effectively with all project members and circulate the information among the team for better project focus. Also, Quality Assurance testing is the end process in stamping the product quality. So, they should collaborate with all stakeholders in understanding their expectations on the project and work in alignment to live up to their expectations.

3. Allocating Resources Efficiently

Resource management is very crucial in testing as every tester needs to be independently responsible for assigned modules. They have to take sole ownership and ensure the module is error free and one tester needs to do integration testing to ensure all functionalities are working fine together. Assigning testers for regression,

retesting, smoke, usability, and compatibility can collectively contribute to the quality of the product altogether.

4. Implementing Continuous Testing

Back-to-back rigorous testing after every error fix can contribute towards a quality deliverable as thorough testing will be done. Testers need to be proactive and gather the no of bugs fixed and separate them among themselves and work on them. Every tester should ensure a single bug fix is not affecting other modules and that all functionalities are working fine.

1.2

INTRODUCTION SOFTWARE QUALITY FACTORS

Q6. What is software quality? Explain the factors affecting software quality factors.

Ans :

Meaning of Software Quality Factors

Software quality factors, also known as software quality characteristics or software quality attributes, are the key aspects of software that are used to evaluate its overall Quality. There are many different factors that can contribute to software quality, but some of the most commonly used factors include:

1. Functionality

This refers to the degree to which the software meets its specified requirements and performs the functions that it is intended to perform.

2. Reliability

This refers to the ability of the software to perform its functions consistently and reliably over time, without errors or crashes.

3. Usability

This refers to the degree to which the software is easy to use and understand, and how well it meets the needs of its users.

4. Efficiency

This refers to how efficiently the software uses system resources, such as CPU time, memory, and disk space.

5. Maintainability

This refers to the ease with which the software

can be modified, maintained, and updated over time.

6. Portability

This refers to how easily the software can be moved from one system or platform to another.

7. Compatibility

This refers to how well the software works with other systems, software, and hardware.

8. Security

This refers to the ability of the software to protect data and systems from unauthorized access or attacks.

Overall, these software quality factors are critical to ensuring that software is of high quality and meets the needs of its users. Evaluating these factors can help software developers and testers to identify areas for improvement and ensure that the software is of high quality.

Q7. What is McCall's Software Quality Model?

Ans :

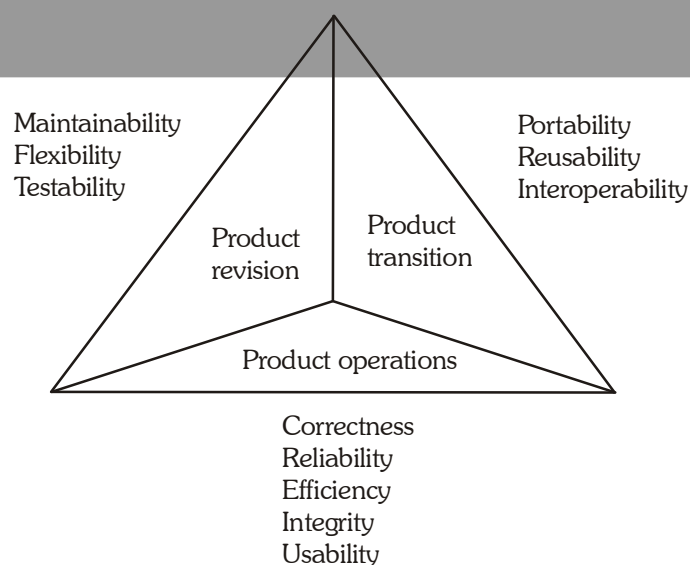
(Imp.)

McCall's Software Quality Model was introduced in 1977. This model is incorporated with many attributes, termed software factors, which influence software. The model distinguishes between two levels of quality attributes:

- Quality Factors
- Quality Criteria

1. **Quality Factors:** The higher-level quality attributes that can be accessed directly are called quality factors. These attributes are external. The attributes at this level are given more importance by the users and managers.
2. **Quality Criteria:** The lower or second-level quality attributes that can be accessed either subjectively or objectively are called Quality Criteria. These attributes are internal. Each quality factor has many second-level quality attributes or quality criteria.

McCall's Software Quality Model Triangle



Q8. What are the factors for product quality?*Ans :***(Imp.)****Factors of Product Quality**

Below are the factors of Product Quality, that are discussed in detail.

- Product Operation
- Product Revision
- Product Transition

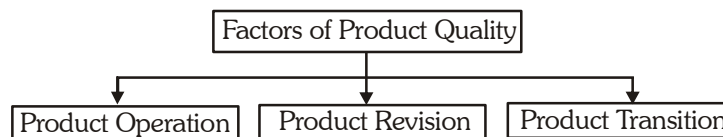


Fig.: Factors of product quality

Factors of Product Quality**1. Product Operation**

Product Operation includes five software quality factors, which are related to the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage, and correctness. These factors help in providing a better user experience.

- **Correctness:** The extent to which software meets its requirements specification.
- **Efficiency:** The number of hardware resources and code the software, needs to perform a function.
- **Integrity:** The extent to which the software can control an unauthorized person from accessing the data or software.
- **Reliability:** The extent to which software performs its intended functions without failure.
- **Usability:** The extent of effort required to learn, operate, and understand the functions of the software.

2. Product Revision

Product Revision includes three software quality factors, which are required for testing and maintenance of the software. They provide ease of maintenance, flexibility, and testing efforts to support the software to be functional according to the needs and requirements of the user in the future.

- **Maintainability:** The effort required to detect and correct an error during maintenance.
- **Flexibility:** The effort needed to improve an operational software program.
- **Testability:** The effort required to verify software to ensure that it meets the specified requirements.

3. Product Transition

Product Transition includes three software quality factors, that allow the software to adapt to the change of environments in the new platform or technology from the previous.

- **Portability:** The effort required to transfer a program from one platform to another.
- **Re-usability:** The extent to which the program's code can be reused in other applications.
- **Interoperability:** The effort required to integrate two systems.

1.3

THE COMPONENTS OF THE SOFTWARE QUALITY ASSURANCE

Q9. What is Quality Assurance (QA)?

Ans :

1. Quality Assurance is also known as QA Testing. QA is defined as an activity to ensure that an organization is providing the best product or service to the customers.
2. Software Quality Assurance seems it is all about evaluation of software based on functionality, performance, and adaptability; however software quality assurance goes beyond the quality of the software, it also includes the quality of the process used to develop, test and release the software.
3. Software Quality assurance is all about the Software Development lifecycle that includes requirements management, software design, coding, testing, and release management.
4. Quality Assurance is the set of activities that defines the procedures and standards to develop the product.
5. Quality Assurance is a systematic way of creating an environment to ensure that the software product being developed meets the quality requirements. This process is controlled and determined at the managerial level.
6. It is a preventive process whose aim is to establish the correct methodology and standard to provide a quality environment to the product being developed.
7. Quality Assurance focuses on process standard, projects audit, and procedures for development.
8. QA is also known as a set of activities designed to evaluate the process by which products are manufactured.
9. QA focused on improving the processes to deliver Quality Products.

Q10. What is the Quality Attribute of a software?

Ans :

The following six characteristics can define the quality of the software:

1. Functionality

Quality of software is defined as how effectively

the software interacts with other components of the system. The software must provide appropriate functions as per requirement, and these functions must be implemented correctly.

2. Reliability

It is defined as the capability of the software to perform under specific conditions for a specified duration.

3. Usability

Usability of software is defined as its ease of use. Quality of the software is also identified as how easily a user can understand the functions of the software and how much efforts are required to follow the features.

4. Efficiency

The efficiency of the software is dependent on the architecture and coding practice followed during development.

5. Maintainability

Maintainability is also one of the significant factors to define the quality of the software. It refers to identify the fault and fix in the software. It should be stable when the changes are made.

6. Portability

Portability of the software, defined as how easily a system adapts to changes in the specifications. Quality of the software is also determined by the portability of the system how easy it is to install the software and how easy it is to replace a component of the order in a given environment.

To ensure about a software score well on these quality attribute, we need the following software Quality Assurance.

Q11. What are the components of Software Quality assurance?

Ans :

(Imp.)

Components

1. Pre-Project Components

Before starting any project, it is important to establish (Mobility standards, select development methodologies, define the scope and requirements, and identify risks and challenges. For example, a company that is planning to develop a new software application may conduct

a feasibility study, define the scope of the project, identify stakeholders, and establish quality standards and requirements for the application.

2. **Components of Project Life Cycle Activities Assessment**

During, the project life cycle, it is important to assess each activity to ensure that it adheres to established Quality standards. For example, during the coding phase of a software development project, a team lead may review the code written by a developer to ensure that it meets the established coding standards and is free of errors.

3. **Components of Infrastructure Error Prevention and Improvement**

To prevent errors and improve efficiency, it is important to identify and improve the infrastructure, tools, and technologies used in software development. For example, a company may invest in new software testing tools that automate testing processes and improve the accuracy of test results.

4. **Components of Software Quality Management**

Software quality management involves the planning, monitoring, and control of software quality throughout the project life cycle. For example, a software development company may establish a quality assurance team that is responsible for testing software products, reporting defects, and verifying that defects have been fixed.

5. **Components of Standardization, Certification, and SQA System Assessment**

Standardization, certification, and SQA system assessment involve the development and implementation of standards and best practices for software development for example, an organization may choose to follow a specific software development methodology, such as Agile or Waterfall, to ensure consistency and quality in their software development processes.

6. **Organizing for SQA-The Human Components**

The human component of SQA focuses on the people involved in the software development

process. For example, a company may invest in training and development programs to improve the skills and competencies of their development team, or establish communication protocols to ensure that stakeholders are informed and involved throughout the development process.

Q12. Write about some software quality assurance standards.

Ans : (Imp.)

Software Quality Assurance Standards

Several organizations, national and international institutes are involved in the development of SQA standards. The below mentioned are the main organizations and institutes involved in it :

1. IEEE
2. DOT
3. ISO
4. ANSI
5. EIA
6. IEC

SQA Standards are basically divided into Two Categories:

1. Software Quality Assurance Standard, which is known as Quality Management Standards.

Example: ISO 9000-3, CMM (Capability Maturity Model).

They focus on the organization's infrastructure, SQA system, requirements leaving the choice of tools and methods of testing to an organization. Their standard objective is "what" to achieve. It assures that organizations achieve an acceptable quality of software.

2. Software Project Development Process standards which are known as Project Process Standards.

Example: ISO/IEC 12207 IEEE Std 1012-1998.

They focus on methodologies that must be implemented in software development and maintenance. It focuses on "how" to perform. It includes design documentation requirements, steps to be taken, software testing to be performed, and design review and review issues.

Q13. What are various techniques of SQA?*Ans :***SQA Techniques**

There are several SQA techniques. Some of them are mentioned below:

1. Reviewing

In reviewing, a meeting is held by both internal and external stakeholders to review the whole project who analyses the whole software and if finds an issue, distinguishes whether it is testing, development, requirement or a design. The main objective is to measure the quality of software and ensure that whether it meets the customer expectations or not.

2. Auditing

In Auditing, the whole work product and all the data are inspected by stakeholders to check whether it follows the standard processes or not.

3. Functional Testing

In functional testing, the functionality of the whole software is tested whether it is functioning as expected or not. It checks “what the system works” without knowing “how the system works”. It is like the black box testing of an application in which the user knows the expected output without knowing how it is produced.

4. Standardization

It assures that everything in the software should be standardized, i.e. it follows all the standards either the standards in documentation, development, quality control. It reduces ambiguity and hence improves the quality of software.

5. Code Inspection

Code Inspection is one of the most formal kinds of review with the main objective of finding defects in the code and highlighting any issues in the Code Inspection is led by a trained Moderator rather than the Author of the code. Meeting has proper entry and exit criteria. Users must need complete preparation before the meeting in order to have complete knowledge of documents and all before raising their points.

6. Walkthroughs

Software walkthrough is a kind of informal process and usually, it is initiated by the Author to read the document or code and the peer members write down their suggestions or errors in it and submits them. It is not formally documented like Inspection and moderator is not necessary for the meeting. Its main objective is to know the status of code completed to date and collecting suggestions from peers for a better quality of software.

7. Stress Testing

Stress testing is done to check how the system works under heavy load. This testing plays an important role in software quality as in e-commerce applications, stress and load testing are done properly in order to test the capacity of the software (how many maximum numbers of users can access an application at a time).

8. Design Inspection

Design Inspection is done to check the various areas of software using the checklist like functional and interface design, conventions, general requirements and design, requirement traceability, logic, coupling, and cohesion.

1.4**SYSTEM OVERVIEW****Q14. Write the overview of a Software Quality Assurance.***Ans :***(Imp.)**

Software Quality Assurance (SQA) is a system that ensures the quality of software products by identifying and fixing issues throughout the software development process. SQA is a critical part of the software development process and helps companies build high-quality products that meet customer expectations.

Here are some key aspects of SQA:

- **Phases:** SQA includes all phases of the software development process, from initial design to deployment.
- **Testing:** SQA involves testing every part of the software, including unit testing, integration testing, system testing, and acceptance testing.
- **Quality objectives:** SQA establishes quality objectives and documentation to ensure consistency.

- **Process monitoring:** SQA monitors the development processes and how to improve them.
- **Design control:** SQA ensures that the design of the software meets quality standards.
- **Code reviews:** SQA finds bugs and ensures adherence to standards.
- **Configuration management:** SQA controls changes to preserve software integrity.
- **Release management:** SQA ensures the quality of the release.
- **User feedback:** SQA welcomes user feedback to improve performance.
- **Security management:** SQA ensures that appropriate process and technology are used to achieve software security.
- **Risk management:** SQA ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

1.5**DEVELOPMENT AND QUALITY PLANS****Q15. Explain about development and quality plans in SQA.***Ans :*

Development and quality plans are two important components of SQA that are used to ensure that software products are developed in accordance with established quality standards and requirements.

A development plan outlines the activities and milestones involved in the software development process, while a quality plan outlines the procedures and techniques that will be used to ensure that the software product meets the established quality standards. Both plans are critical for achieving a successful software product.

Development Plan

A development plan is a document that outlines the activities involved in the software development process. This plan provides of roadmap for the entire development team including project managers, developers, and testers, to follow throughout the project life cycle. A development plan typically includes:

1. Project objectives and scope
2. Milestones and timelines
3. Tasks and activities
4. Resource allocation
5. Communication plan

Quality Plan

A quality plan is a document that outlines the procedures and techniques that will be used to ensure that the software product meets the established quality standards. The plan is developed based on the quality standards, specifications, and requirements identified during the planning phase of the project. A quality plan typically includes:

1. Quality objectives and scope
2. Quality standards and requirement:
3. Quality control and assurance procedures
4. Testing and inspection procedures
5. Defect reporting and management procedures

By developing a comprehensive development plan and quality plan, organizations can ensure that their software products are developed in a structured and controlled manner, and that they meet the established quality standards and requirements.

Q16. Describe briefly the elements of the quality plan?*Ans :*

A quality plan is a document that outlines the approach, activities, and resources necessary to ensure that a software product or project meets its quality objectives. The elements of a quality plan may vary depending on the specific needs of the project, but some of the common elements of a quality plan include:

1. Introduction

This section provides an overview of the quality plan, including the scope, purpose, and objectives of the plan.

2. Quality Objectives

This section defines the quality objectives of the software product or project, including the criteria that will be used to measure quality.

3. Quality Standards

This section outlines the quality standards that the software product or project will adhere to, including any industry-specific standards or regulations.

4. Quality Processes

This section describes the quality processes that will be used to ensure that the software product or project meets its quality objectives. This may include processes for requirements management, design reviews, code reviews, testing, and defect management.

5. Quality Metrics

This section defines the quality metrics that will be used to measure the quality of the software product or project, including the criteria for success and the tools that will be used to collect and analyze data.

6. Quality Resources

This section outlines the resources necessary to carry out the quality processes and activities, including personnel, equipment, and software tools.

7. Quality Assurance and Quality Control Activities

This section describes the activities that will be carried out to ensure that the quality objectives are met, including audits, inspections, testing, and reviews.

8. Risk Management

This section outlines the risks associated with the software product or project and describes the measures that will be taken to mitigate these risks.

9. Documentation and Reporting

This section describes the documentation and reporting requirements for the quality plan, including the format and frequency of reports, and the tools that will be used to generate them documentation, and training materials. It also defines the milestones that must be reached in order to complete the project successfully.

Overall, the elements of a development plan are designed to ensure that the software project is completed on time, within budget, and to the

required quality standards. The development plan serves as a roadmap for the development team, providing a framework for continuous improvement throughout the software development lifecycle.

Q17. Explain the objectives of a development plan and a quality plan?

Ans :

The development plan and quality plan are two important documents used in software development projects. While both plans are critical for project success, they have different objectives.

The objectives of a development plan are:

1. To define the scope of the project

The development plan outlines the boundaries of the project and defines what will and will not be included in the project.

2. To identify the resources required

The development plan identifies the resources needed to carry out the project, including personnel, equipment, and software tools.

3. To establish the project schedule

The development plan defines the project schedule, including start and end dates, as well as intermediate milestones.

4. To allocate tasks and responsibilities

The development plan assigns tasks and responsibilities to team members, ensuring that everyone knows what is expected of them.

5. To manage risk

The development plan outlines the risks associated with the project and describes the measures that will be taken to mitigate these risks.

The objectives of a quality plan are:

1. To define the quality objectives of the project

The quality plan defines the quality objectives of the software product or project, including the criteria that will be used to measure quality.

2. To identify the quality processes and activities:

The quality plan describes the quality- processes

and activities that will be used to ensure that the software product or project meets its quality objectives

3. To establish quality metrics

The quality plan defines the quality metrics that will be used to measure the quality of the software product or project, including the criteria for success and the tools that will be used to collect and analyze data.

4. To allocate quality responsibilities

The quality plan assigns responsibilities for quality activities to team members, ensure that everyone knows what is expected of them.

5. To manage quality risks

The quality plan outlines the risks associated with quality and describes the measures that will be taken to mitigate these risks.

In summary, the development plan focuses on the project's scope, resources, schedule, tasks, and risks, while the quality plan focuses on ensuring that the software product or project meets its quality objectives through the use of quality processes, activities, metrics, and risk management.

Q18. Discuss the importance of development and quality plans for small projects.

Ans :

(Imp.)

Development and quality plans are essential for small software projects just as they are for larger ones. Here are some reasons why:

1. Clear roadmap

A development plan provides a clear roadmap for the project, outlining the goals, objectives, and timelines. This helps the development team to stay on track and ensures that all stakeholders are on the same page.

2. Efficient resource allocation

A development plan helps to identify the resources required to complete the project. This allows for efficient allocation of resources, ensuring that the project is completed on time and within budget.

3. Risk management

A development plan helps to identify and manage risks associated with the project. By planning ahead, the development team can take steps to mitigate risks and avoid potential roadblocks.

4. Quality assurance

A quality plan helps to ensure that the software product or project meets the desired quality standards. This includes defining quality metrics, processes, and responsibilities to ensure that the final product meets the user's needs.

5. Better communication

Both development and quality plans help to facilitate communication among team members and stakeholders. By clearly outlining the project scope, goals, and objectives, the team can work together more effectively.

6. Improved customer satisfaction

By having a well-defined development and quality plan, small projects can better meet the needs and expectations of their customers. This can help to improve customer satisfaction and lead to more business in the future.

In conclusion, even small software projects benefit greatly from the use of development and quality plans. They help to ensure that the project is completed on time, within budget, and to the desired quality standards. They also facilitate communication and collaboration among team members and stakeholders, leading to improved customer satisfaction and potential business opportunities in the future.

Q19. Discuss the importance of development and quality plans for internal projects?

Ans :

Internal projects are those that are undertaken within an organization and are not meant for external customers. Development and quality plans are just as important for internal projects as they are for customer-facing ones, for the following reasons:

1. Clarity of purpose

Development and quality plans help to define the purpose of the project and ensure that all stakeholders are on the same page. This clarity is especially important for internal projects, where there may be multiple departments or teams involved.

2. Resource management

A development plan helps to identify the resources required to complete the project, including personnel, equipment; and software tools. This allows for efficient allocation of resources, ensuring that the project is completed on time and within budget.

3. Risk management

A development plan helps to identify and manage risks associated with the project. By planning ahead, the development team can take steps to mitigate risks and avoid potential roadblocks.

4. Quality assurance

A quality plan helps to ensure that the software product or project meets the desired quality standards. This includes defining quality metrics, processes, and responsibilities to ensure that the final product meets the organization's needs.

5. Efficient communication

Both development and quality plans help to facilitate communication among team members and stakeholders. By clearly outlining the project scope, goals, and objectives, the team can work together more effectively.

6. Improved productivity

By having a well-defined development and quality plan, internal projects can be completed more efficiently and effectively. This can lead to improved productivity and cost savings for the organization.

7. Knowledge transfer

A development plan helps to document the project details, which can be useful for knowledge transfer when team members change or when the project is revisited in the future.

UNIT II

Integrating Quality Activities in the Project Life Cycle, Assuring the Quality of Software Maintenance Components, CASE Tools and their effect on Software Quality, Procedure and Work Instructions, Supporting Quality Devices, Configuration Management, Documentation Control, Project Progress Control.

2.1

INTEGRATING QUALITY ACTIVITIES IN THE PROJECT LIFE CYCLE

Q1. Explain different types of SDLC models

Ans :

(Imp)

Classic and other software development methodologies

At the end of each phase, outputs are reviewed and evaluated by the developer as well as, in many cases, by the customer. The outcomes range from approval of the phase results and continuation to the next phase, to demands to correct, redo or alter parts of the respective phase.

1. The software development life cycle (SDLC) model

The Software Development Life Cycle model (the SDLC model) is the classic model (still applicable today); it provides the most comprehensive description of the process available. The model displays the major building blocks for the entire development process, described as a linear sequence that begins with requirements definition and ends with regular system operation and maintenance.

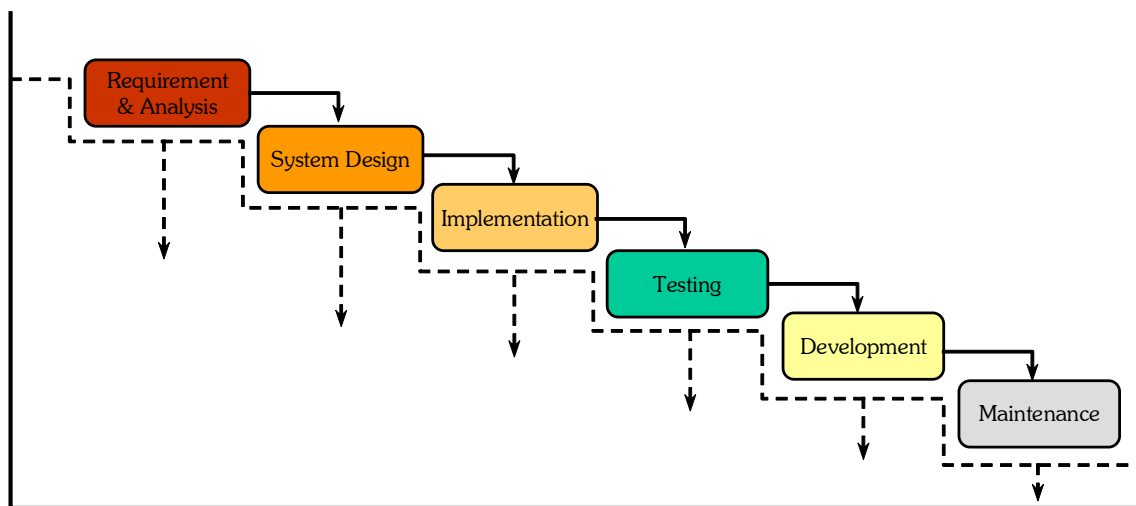


Fig.: The Waterfall Model

- **System Design:** This stage involves the detailed definition of the outputs, inputs and processing procedures, including data structures and databases, software structure, etc.
- **Implementation (Coding):** In this phase, the design is translated into a code. Coding involves quality assurance activities such as inspection, unit tests and integration tests.

- **Testing (System tests):** System tests are performed once the coding phase is completed. The main goal of testing is to uncover as many software errors as possible so as to achieve an acceptable level of software quality once corrections have been completed. System tests are carried out by the software developer before the software is supplied to the customer.
- **Development (Installation and conversion):** After the software system is approved, the system is installed to serve as firmware, that is, as part of the information system that represents a major component of the expanded system.
- **Maintenance (Regular operation):** Regular software operation begins once installation and conversion have been completed. Throughout the regular operation period, which usually lasts for several years or until a new software generation appears on the scene, maintenance is needed.

2. The prototyping model

The prototyping model is based on replacement of one or more SDLC model phases by an evolutionary process, where software prototypes are used for communication between the developer and the users and customers. Prototypes are submitted to user representatives for evaluation.

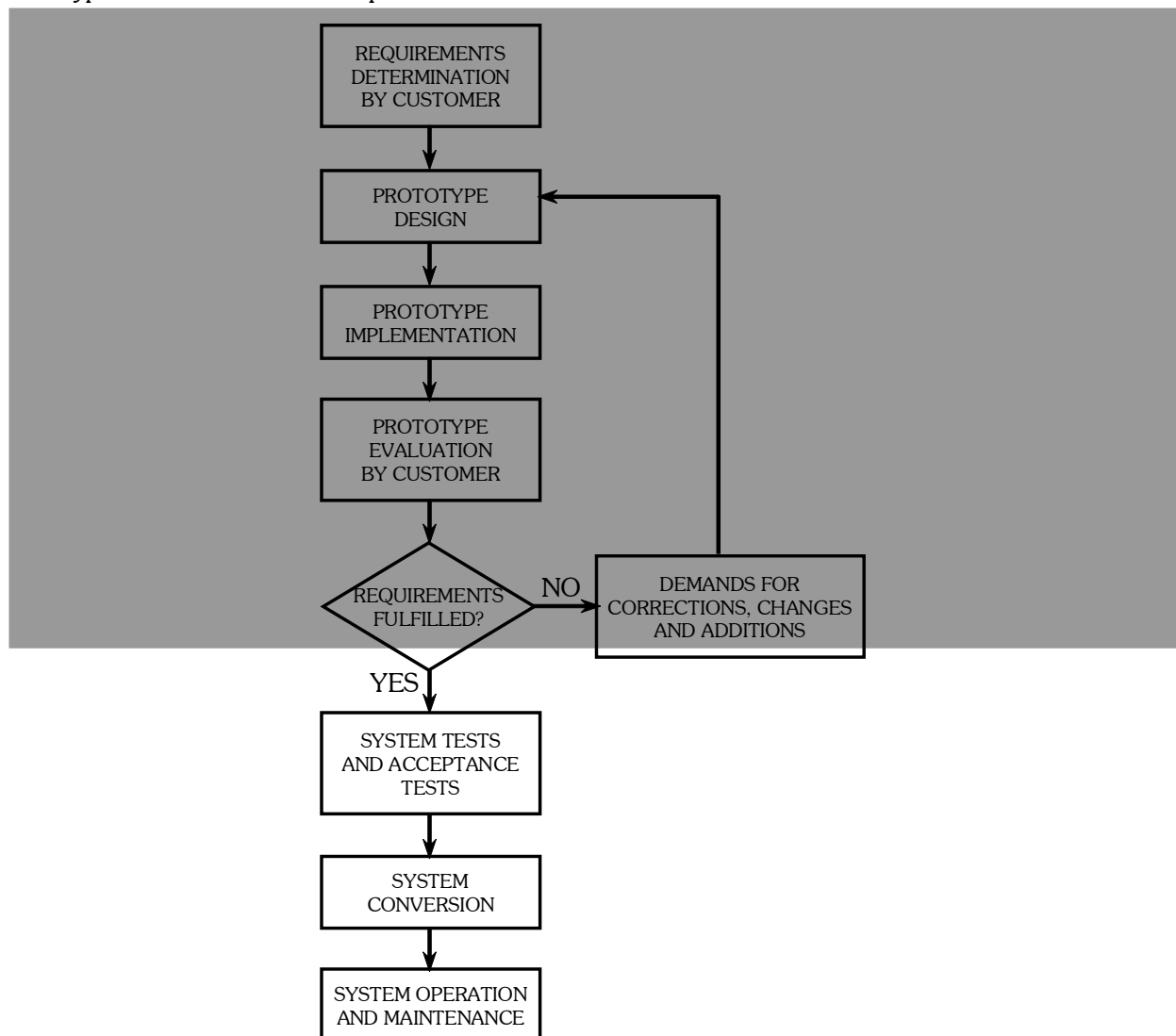


Fig.: Prototyping model

Prototyping as a software development methodology has been found to be efficient and effective mainly for small-to medium-sized software development projects.

Advantages of prototyping

- Shorter development process.
- Substantial savings of development resources (man-days).
- Better fit to customer requirements and reduced risk of project failure.
- Easier and faster user comprehension of the new system.

Disadvantages of prototyping

- Diminished flexibility and adaptability to changes and additions.
- Reduced preparation for unexpected instances of failure.

3. The spiral model

The spiral model, as revised by Boehm (1988, 1998), offers an improved methodology for overseeing large and more complex development projects displaying higher prospects for failure, it combines an iterative model that introduces and emphasizes risk analysis and customer participation into the major elements of SDLC and prototyping methodologies.

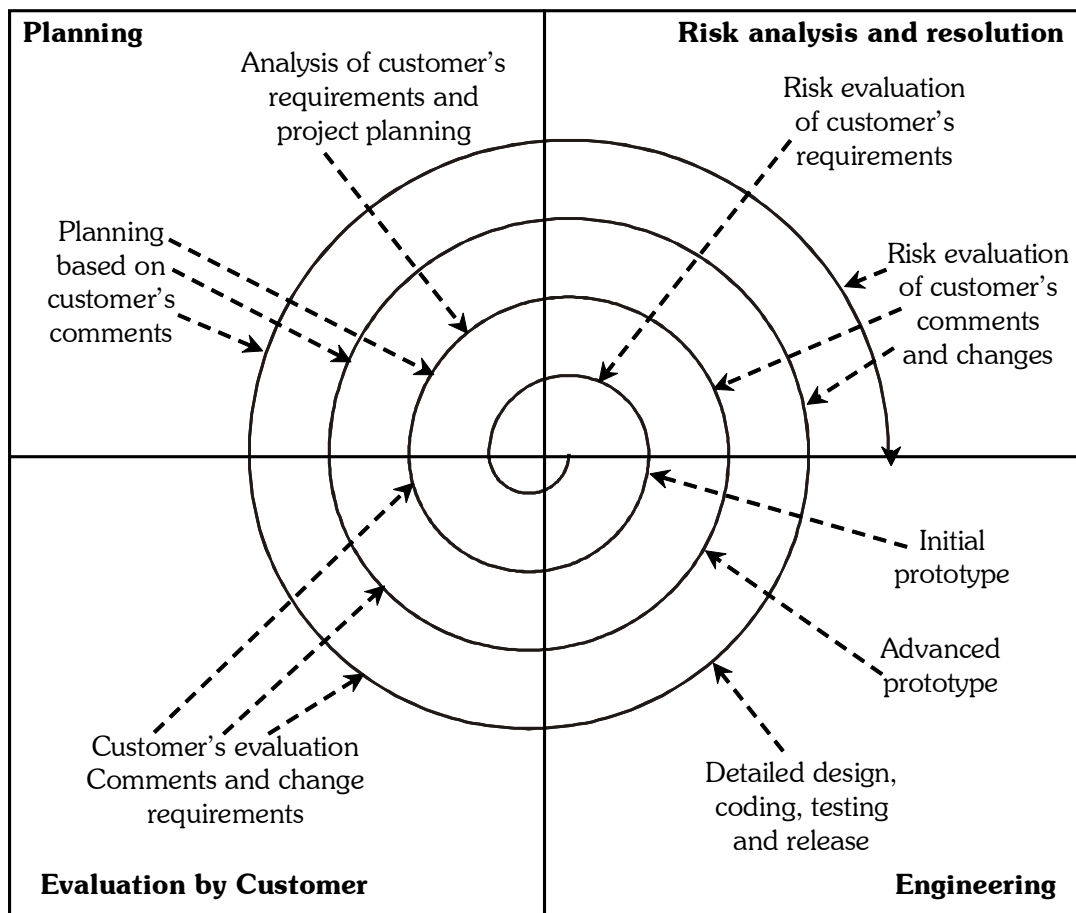


Fig. : The spiral model (Boehm, 1988)

An advanced spiral model, the Win-Win Spiral model (Boehm, 1998), enhances the Spiral model (Boehm, 1988) still further. The advanced spiral model (the Win-Win model) places extra emphasis on communication and negotiation between customer and developer. The customer wins by improving chances to receive a system that satisfies most of his needs while the developer wins by improving chances of completing the project within budgetary and timetable constraints.

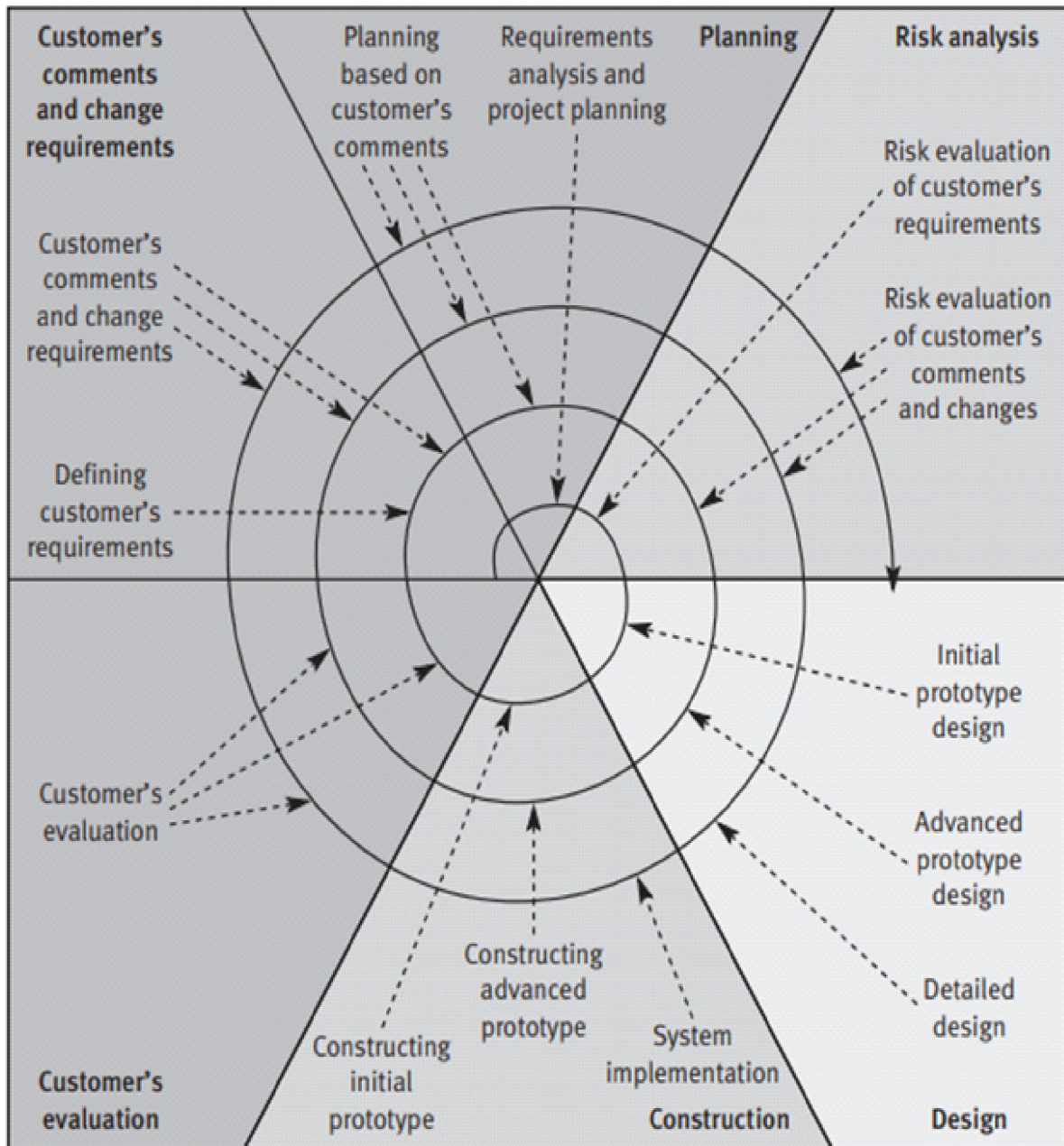


Fig. : The advanced spiral model (Boehm, 1998)

4. The object-oriented model

According to this model, the development process begins with a sequence of object-oriented analysis and design activities. The design phase is followed by acquisition of a reusable software library together with "regular" development of the unavailable software components.

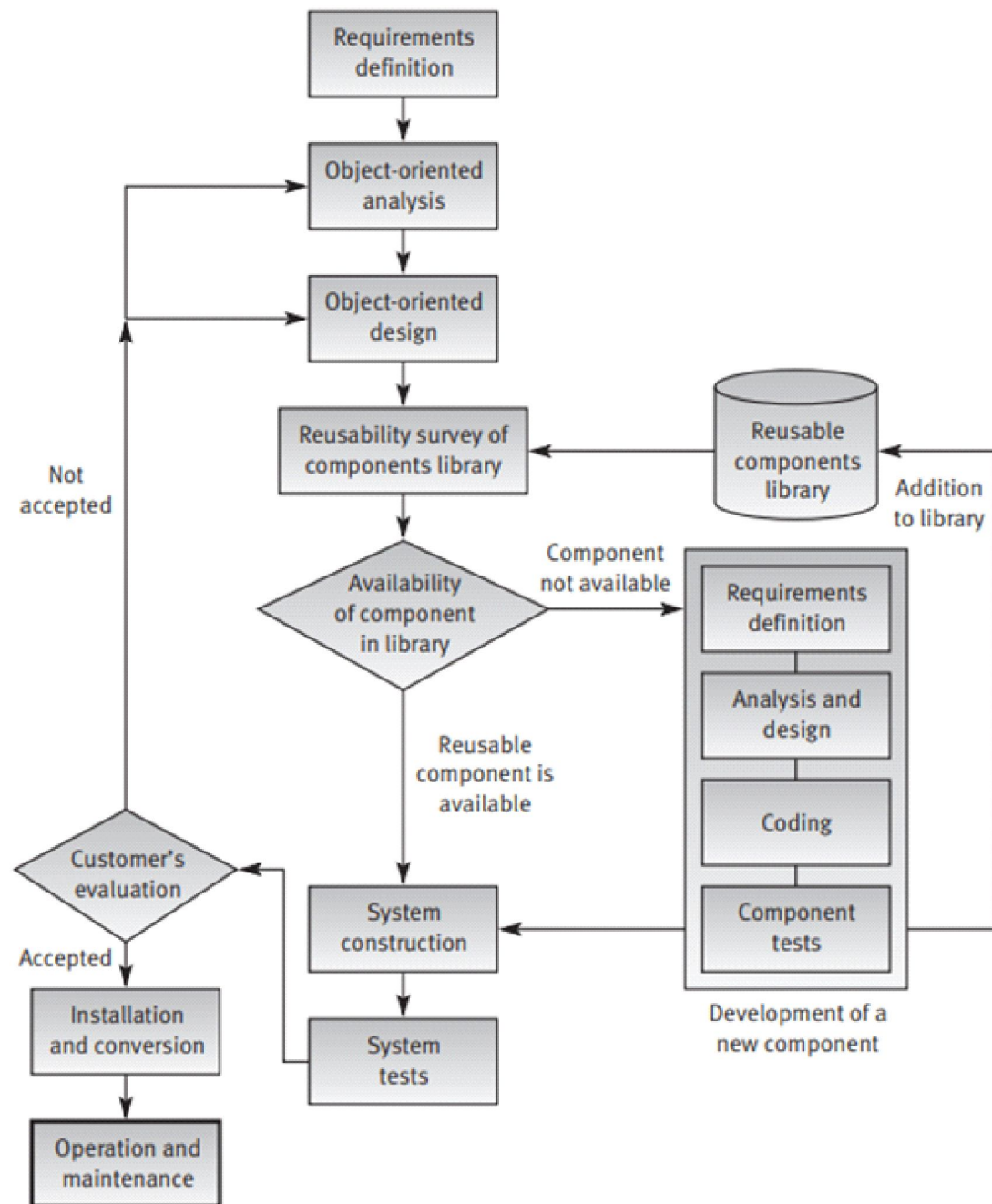


Fig. : The object-oriented model

Q2. What are the Factors affecting intensity of quality assurance activities in the development process?

Ans :

Project life cycle quality assurance activities are process oriented, in other words, linked to completion of a project phase, accomplishment of a project milestone, and so forth. The quality assurance activities will be integrated into the development plan that implements one or more software development models - the waterfall, prototyping, spiral, object-oriented or other models.

The list of quality assurance activities needed for a project.

For each quality assurance activity:

- Timing
- Type of quality assurance activity to be applied
- Who performs the activity and the resources required? It should be noted that various bodies may participate in the performance of quality assurance activities: development team and department staff members together with independent bodies such as external quality assurance team members or consultants

- Resources required for removal of defects and introduction of change

Factors affecting intensity of quality assurance activities in the development process:

Project Factors

- Magnitude of the project
- Technical complexity and difficulty
- Extent of reusable software components
- Severity of failure outcomes, if the project fails

Team Factors

- Professional qualification of the team members
- Team acquaintance with the project and its experience in the area
- Availability of staff members who can professionally support the team
- Familiarity with the team members, in other words the percentage of new staff members in the team

Example 2

The real-time software development unit of a hospital's information systems department has been assigned to develop an advanced patient monitoring system. The new monitoring unit is to combine of patient's room unit with a control unit. The patient's room unit is meant to interface with several types of medical equipment, supplied by different manufacturers, which measure various indicators of the patient's condition. A sophisticated control unit will be placed at the nurses' station, with data to be communicated to cellular units carried by doctors.

The project leader estimates that 14 months will be required to complete the system; a team of five will be needed, with an investment of a total of 40 man-months. She estimates that only 15% of the components can be obtained from the reusable component library.

The SDLC methodology was chosen to integrate application of two prototypes of the patient's room unit and two prototypes of the control unit for the purpose of improving communication with the users and enhancing feedback of comments at the analysis and design phases.

The main considerations affecting this plan are:

High complexity and difficulty of the system

- Low percentage of reusable software available
- Large size of the project
- High severity of failure outcomes if the project fails.
- The quality assurance activities and their duration, as defined by the project leader, are listed in Table.

Q3. List various quality assurance activities.

Ans :

Quality assurance activity

- i) Design review of requirements definition
- ii) Design review of analysis of patient's room unit
- iii) Design review of analysis of control unit
- iv) Design review of preliminary design
- v) Inspection of design of patient's room unit
- vi) Inspection of design of control unit
- vii) Design review of prototype of patient's room unit
- viii) Design review of prototype of control unit
- ix) Inspection of detailed design for each software interface component
- x) Design review of test plans for patient's room unit and control unit
- xi) Unit tests of software code for each interface module of patient's room unit
- xii) Integration test of software code of patient's room unit
- xiii) Integration test of software code of control unit
- xiv) System test of completed software system
- xv) Design review of user's manual

2.2

**ASSURING THE QUALITY OF SOFTWARE
MAINTENANCE COMPONENTS**

Q4. What are the elements to assure the quality of software maintenance components.

Ans :

Software maintenance is the process of modifying a software system or application after it has been deployed. There are several components of software maintenance, and each one plays a distinct role in ensuring the continued functionality and usability of the software. Here are some of the main components of software maintenance and their distinctions:

1. Corrective Maintenance

This component involves fixing defects or errors in the software. Corrective maintenance is performed to ensure that the software operates as expected and to address any issues that may arise after deployment.

2. Adaptive Maintenance

This component involves modifying the software to adapt to changes in the environment or to new requirements. Adaptive maintenance may be necessary if the software needs to support new hardware or operating systems or if it needs to comply with new regulations.

3. Perfective Maintenance

This component involves improving the software's performance or adding new functionality. Perfective maintenance is performed to enhance the software's usability, efficiency, and effectiveness.

4. Preventive Maintenance

This component involves making changes to the software to prevent future problems or to address potential issues. Preventive maintenance may involve code refactoring, adding new documentation, or improving testing procedures.

Q5. What are the various requirements of software quality.

Ans :

1. Maintainability Requirements

The degree of effort needed to identify reasons (find the problem) for software failure and to correct

failures and to verify the success of the corrections. It deals with the modular structure of the software, internal program documentation, programmer manual, architectural and detail design and corresponding documentation.

Example

Typical maintainability requirements:

- (a) The size of a software module will not exceed 30 statements.
- (b) The programming will adhere to the company coding standards and guidelines.

2. Flexibility Requirements

It deals with resources to change (adopt) software to different types of customers.

Example

TSS (teacher support software) deals with the documentation of pupil achievements, the calculation of final grades, the printing of term grade documents, and the automatic printing of warning letters to parents of failing pupils. The software specifications included the following flexibility requirements:

- (a) The software should be suitable for teachers of all subjects and all school levels (elementary, junior and high schools).

3. Testability Requirements

Testability requirements deal with the testing of an information system as well as with its operation. Testability requirements related to software operation include automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the software system are in working order and to obtain a report about the detected faults.

Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

Example

An industrial computerized control unit is programmed to calculate various measures of production status, report the performance level of the machinery, and operate a warning signal in predefined situations. One testability requirement demanded was to develop a set of standard test data with known system expected correct reactions in each stage. This standard test data is to be run every morning, before production begins, to check whether the computerized unit reacts properly.

Product Transition Software Quality Factors

1. Portability Requirements

If the software must be ported to different environments (different hardware, operating systems ...) and still maintain an existing environment, then portability is a must.

Example

A software package designed and programmed to operate in a Windows 2000 environment is required to allow low-cost transfer to Linux and Windows NT environments.

2. Reusability Requirements

A software product has good reusability, if different modules of the product can easily be reused to develop new products.

Example

A software development unit has been required to develop a software system for the operation and control of a hotel swimming pool that serves hotel guests and members of a pool club.

Although the management did not define any reusability requirements, the unit's team leader, after analyzing the information processing requirements of the hotel's spa, decided to add the reusability requirement that some of the software modules for the pool should be designed and programmed in a way that will allow its reuse in the spa's future software system, which is planned to be developed next year.

These modules will allow:

- Entrance validity checks of membership cards and visit recording.
- Restaurant billing.
- Processing of membership renewal letters.

3. Interoperability Requirements

The application need to interface with other existing systems. It will be known ahead of time and plans can be made to provide for this requirement during design time.

Q6. Explain briefly about software maintenance policy.

Ans :

Maintenance policy

The main maintenance policy components that affect the success of software maintenance are the version development and change policies to be applied during the software's life cycle.

Version development policy

This policy relates mainly to the question of how many versions of the software should be operative simultaneously. The version development policy can take a "sequential" or "tree" form. When adopting a sequential version policy, only one version is made available to the entire customer population. This version includes a profusion of applications that exhibit high redundancy, an attribute that enables the software to serve the needs of all customers. The software must be revised periodically but once a new version is completed, it replaces the version currently used by the entire user population.

When adopting a tree version policy, the software maintenance team supports marketing efforts by developing a specialized, targeted version for groups of customers or a major customer once it is requested.

Example

After a few years of application, Inventory Perfect, an inventory management package developed according to the tree policy, has evolved into a seven-version software package with these main branches: Pharmacies, Electronics, Hospitals, Bookstores, Supermarkets, Garages

Auto Repairs, and Chemical Plants. Each of the branches includes four or five secondary branches that vary by number of software modules, level of implementation or specific customer oriented applications. For example, the Bookstores version has the following five secondary branches (versions): bookstore chains, single bookstores, advanced management bookstores, and special versions for the LP bookstore chain and for CUCB (City University Campus Bookstores).

The software maintenance team tends to a total of 30 different versions of the software package simultaneously, with each version revised periodically according to customer requests and the team's technical innovations.

Pre-maintenance software quality components

- Maintenance contract review
- Maintenance plan construction.
- Maintenance contract review

When considering the maintenance contract, a broad perspective should be embraced. More than anything else, decisions are required about the categories of services to be contracted.

These decisions depend on the type of customers served: customers for whom a custom-made package has been developed, customers who purchased a COTS software package, and internal customers. So, before commencing to supply software maintenance services to any of these customers, an adequate maintenance contract should be finalized that sets down the total range of maintenance obligations according to the relevant conditions.

Objectives

The major objectives of software maintenance contract reviews.

1. Customer requirements clarification

The following issues deserve special attention:

- Type of corrective maintenance services required: list of remote services and on-site services to be provided, hours of service, response time, etc.
- Size of the user population and the types of applications to be used.
- Location of users, especially of long-distance (or overseas) sites and the types of applications installed in each.
- Adaptive and functionality improvement maintenance to be provided and procedures for submission of requests for service as well as proposing and approving performance of these services.

2. Review of alternative approaches to maintenance provision

The following options deserve special consideration:

Subcontracting for sites or type of service
Performance of some services by the customer himself with support from supplier's maintenance team.

3. Review of estimates of required maintenance resources

First, these estimates should be examined on the basis of the required maintenance services, clarified by the proposal team. Then, the company's capacity to meet its commitments with respect to professional competence as well as availability of maintenance teams should be analyzed.

4. Review of maintenance services to be provided by subcontractors and/or the customer

This review refers to the definition of the services provided by each participant, payments to subcontractors, quality assurance and follow-up procedures to be applied.

5. Review of maintenance costs estimates

These estimates should be reviewed on the basis of required resources.

Maintenance plan

Maintenance plans should be prepared for all customers, external and internal. The plan should provide the framework within which maintenance provision is organized. The plan includes the following:

1. A list of the contracted maintenance services

The internal and external customers, the number of users, the locations of each customer site.

2. A description of the maintenance team's organization

The maintenance team organization plan focuses on manpower requirements, which should be carefully considered according to these criteria.

The number of required team members. If services are to be provided from several.

The required facilities, the team requirement for each facility. Including acquaintance with the software package(s) to be maintained. Locations for team members according to the maintenance tasks,

Organizational structure of the maintenance teams, including names of team leaders.

Definition of tasks (responsibility for customers, types of applications, etc.) for each team and Training needs.

A list of maintenance facilities

The maintenance support center with its installed hardware and communication equipment to provide user support and software correction services.

A documentation center containing a complete set of documents (in printed or electronic format):

The software documentation, including the development documentation

The service contracts

The software configurations for each customer and versions of the software packages installed at each site, provided by configuration management

The maintenance history records for each user and customer.

2.3

CASE TOOLS AND THEIR EFFECT ON SOFTWARE QUALITY

Q7. How case tools can effect on software quality.

Ans :

CASE (Computer-Aided Software Engineering) tools are software applications that support various activities in the software development process, such as requirements management, design, coding, testing, and maintenance. The use of CASE tools can have a significant impact on software quality (SQ) by improving the efficiency and effectiveness of various quality assurance activities. Here are some of the main ways in which CASE tools can affect SQ:

1. **Automation:** CASE tools can automate various software development activities, such as testing, code review, and defect tracking, which can reduce the likelihood of human errors and increase the reliability and consistency of these activities.
2. **Standardization:** CASE tools can enforce the use of standards and best practices in software development, such as coding conventions, testing methodologies, and documentation standards. This can help ensure that the software is developed to a high level of quality and is easier to maintain and modify.
3. **Collaboration:** CASE tools can support collaboration among team members, such as developers, testers, and quality assurance professionals, which can help ensure that everyone is working toward the same goals and following the same processes and procedures.

4. **Traceability:** CASE tools can provide traceability of requirements, design decisions, test cases, and defects, which can help ensure that the software meets its intended purpose and that changes are properly documented and tested.
5. **Analysis:** CASE tools can provide various forms of analysis and metrics, such as code coverage, complexity, and defect trends, which can help identify potential issues and areas for improvement in the software.

In summary, CASE tools can have a significant impact on SQ by automating tasks, enforcing standards and best practices, supporting collaboration, providing traceability, and enabling analysis. The use of CASE tools can help ensure that the software is developed and maintained to a high level of quality, which can improve its reliability, performance, and usability, and reduce the likelihood of defects and failures.

Q8. What are the various performance controls for software maintenance.

Ans :

1. Performance controls for software maintenance services:

- Software correction
- Increased resources utilization
- Decreased rate of remote failure repairs (low cost repairs) versus customer's on-site repairs
- Increased rate of on-site repairs at long-distance locations and over-seas services
- Increased percentage of failures to meet repair schedule requirements
- Increased rate of faulty repairs, and list of specific "model" cases of extreme failure situations
- Lower customer satisfaction based on customer satisfaction surveys.
- User support
- Increased rates of requests for service for a specific software system, for service type, etc.
- Increased resource utilization in user support services
- Increased rate of failures to provide requested consulting services
- Increased rate of faulty consulting, and specific cases of "outstanding" failures
- Customer satisfaction information based on customer satisfaction surveys.

2 Costs of software maintenance quality

The quality costs of corrective maintenance are classified into six classes. Following are definitions for each class and examples.

i) Costs of prevention

Costs of error prevention, i.e. costs of instruction and training of maintenance team, costs of preventative and corrective actions.

ii) Costs of appraisal

Costs of error detection, i.e. costs of review of maintenance services carried out by SQA teams, external teams and customer satisfaction surveys.

iii) Costs of managerial preparation and control

Costs of managerial activities carried out to prevent errors, i.e. costs of preparation of maintenance plans, maintenance team recruitment and follow-up of maintenance performance.

iv) Costs of internal failure

Costs of software failure corrections initiated by the maintenance team (prior to receiving customer complaints).

v) Costs of external failure

Costs of software failure corrections initiated by customer complaints.

vi) Costs of managerial failure

Costs of software failures caused by managerial actions or inaction, i.e. costs of damages resulting from shortage of maintenance staff and/or inadequate maintenance task organization.

3. Costs of external failure of software corrective maintenance activities. In order to define external failure costs, the two maintenance periods must be considered separately. These are:

- (a) The warranty period (usually 3–12 months after the software is installed) and
- (b) The contracted maintenance services period, which begins at the end of the warranty period.

For software corrections

All costs of software correction initiated by users during the warranty period are external quality costs because they are considered to result directly from software development failures.

For user support services

During the warranty period, user support services are considered to be an inherent part of the instruction effort, and therefore should not be considered external failure costs.

During the contracted maintenance period, all types of user support services, whether dealing with an identified software failure or consultations about application options, are all part of regular service, and their costs are not considered external failure costs.

During both maintenance periods, an external failure is defined as a case where a second consultation is required after the initial consultation proves to be inadequate. The costs of furnishing the second and further consultations for the same case are considered external failure costs.

2.4

PROCEDURE AND WORK INSTRUCTIONS

Q9. What are procedures and work instructions? Why do we use them.

Ans :

- i) A procedure is, as transmitted in documents, are the detailed activities or processes to be performed according to a given method for the purpose of accomplishing a task.
- ii) The procedures adopted by an organization are considered to be binding on that organization's employees, meaning that each employee is to perform his or her tasks according to the steps appearing in the relevant procedure document, often bearing the name of the designated task.
- iii) Procedures also tend to be universal within the organization, meaning that they are applied whenever the task is performed, irrespective of the person performing the task or the organizational context.
- iv) Work instructions are used mainly in cases where a uniform method of performing the task throughout the organization is either impossible

or undesirable. As a result, work instructions are specific to a team or department; they supplement procedures by providing explicit details that are suitable solely to the needs of one team, department, or unit.

- v) The software quality assurance procedures and work instructions of special interest to us are those that affect the quality of a software product, software maintenance or project management.
- vi) The need for procedures and work instructions “Why should we use SQA procedures and work instructions?”

SQA procedures and work instructions aim at:

Performance of tasks, processes or activities in the most effective and efficient way without deviating from quality requirements.

Effective and efficient communication between the separate staffs involved in the development and maintenance of software systems. Uniformity in performance, achieved by conformity with procedures and work instructions, reduces the misunderstandings that lead to software errors.

Simplified coordination between tasks and activities performed by the various bodies of the organization. Better coordination means fewer errors.

Q10. Explain briefly about procedures and procedures manuals

Ans :

i) Procedures

Procedures supply all the details needed to carry out a task according to the prescribed method for fulfilling that task's function. These details can be viewed as responding to five issues, known as the Five W's, listed here: The Five W's: issues resolved by procedures

- What activities have to be performed?
- How should each activity be performed?
- When should the activity be performed?
- Where the activity should be performed?
- Who should perform the activity?

Fixed table of contents for procedures

- Introduction
- Purpose
- Terms and abbreviations
- Applicable documents
- Method
- Quality records and documentation
- Reporting and follow-up
- Responsibility for implementation
- List of appendices

ii) SQA procedures manual

The collection of all SQA procedures is usually referred to as the SQA procedures manual.

The content of any one organization's procedures manual varies according to:

The types of software development and maintenance activities carried out by the organization
The range of activities belonging to each activity type.

The range of customers (e.g., internal, customers of custom-made software, COTS software customers) and suppliers (e.g., self-development and maintenance, subcontractors, suppliers of COTS software and reused software modules)

The conceptions governing the choice of method applied by the organization to achieve desired SQA objectives.

Q11. Explain about work instructions and work instructions manuals.

Ans :

Work instructions and work instruction manuals

Work instructions deal with the application of procedures, adapted to the requirements of a specific project team, customer, or other relevant party. While general methodology is defined in a procedure, the precise details that allow its application to a specific project or unit are often laid out in a work procedure. One can add, change or cancel work instructions without altering the respective procedure.

Examples

Departmental work instructions

- Audit process for new software development subcontractors (supplier candidates).
- Priorities for handling corrective maintenance tasks.
- Annual evaluation of software development subcontractors.
- On-the-job instructions and follow-up for new team members.
- Design documentation templates and their application.
- C++ (or other language) programming instructions

Project management work instructions

- Coordination and cooperation with the customer
- Weekly progress reporting by team leaders
- Special design report templates and their application in the project
- Follow-up of beta site reporting
- Monthly progress reporting to the customer
- Coordination of installation and customer's team instructions.

Q12. Write, how to prepare and update of new procedures,

Ans :

Preparation of new procedures

The initial steps taken in development of a new SQA procedures manual should deal with the conceptual and organizational frameworks that determine the menu of the proposed procedures and who will be responsible for their preparation, updating and approval.

This framework is usually also formulated as a procedure (frequently called the procedure of procedures). The subsequent steps will, naturally, deal with specific procedures. A common approach to preparation of procedures is the appointment of an adhoc committee of professionals working in the units involved, SQA unit members and experts in the respective topics to be dealt with.

The committee pours over the proposed drafts until a satisfactory version is reached, and ceases its work only after the procedure is approved by the authorized person(s).

An alternative approach to procedure manual preparation is dependence on consulting, where an outside expert is assigned the responsibility of preparing procedure or the complete manual.

The main advantages of employing a consultant are found in the added value of his or her expertise and experience in other organizations, the reduced burden on the organization's senior professionals as well as the shortened task completion timetable.

The main disadvantage experienced with this approach is reduced applicability due to the organization's unique characteristics.

Updating procedures

The motivation to update existing procedures is based, among other things, on the following:

- Technological changes in development tools, hardware, communication equipment, etc.
- Changes in the organization's areas of activity
- User proposals for improvement
- Analysis of failures as well as successes
- Proposals for improvements initiated by internal audit reports
- Learning from the experience of other organizations
- Experiences of the SQA team.

Q13. Explain the difference between procedures and work instructions?

Ans :

Procedures and work instructions are two types of documents that are commonly used in organizations to provide guidance on how to perform various tasks. While both documents provide instructions on how to complete a task, there are some key differences between procedures and work instructions:

1. Scope

Procedures are typically wider in scope than work instructions. They provide an overview of the steps involved, in completing a process or activity, while work instructions provide detailed, step-by-step instructions on how to perform a specific task within that process or activity.

2. **Level of detail:** Work instructions are more detailed and specific than procedures. They provide specific guidance on how to perform a task, including any tools, equipment, or materials that may be needed, as well as any safety precautions or quality checks that must be followed. Procedures, on the other hand, provide a higher-level overview of the process and may not include as much detail.
3. **Frequency of use:** Work instructions are typically used more frequently than procedures. They are often used by employees on a daily basis to perform routine tasks, while procedures may be used less frequently for tasks that are performed less frequently.
4. **Authorship:** Procedures are typically developed and maintained by management or other subject matter experts, while work instructions may be developed and maintained by front-line employees who have direct experience performing the task.

In summary, procedures provide a high-level overview of a process or activity, while work instructions provide detailed, step-by-step instructions for performing specific tasks within that process or activity. Work instructions are typically more detailed and specific than procedures are used more frequently, and may be developed and maintained by front-line employees.

2.5

SUPPORTING QUALITY DEVICES

Q14. What are templates? How to use templates in software quality

Ans :

Supporting quality devices, such as checklists and templates, are tools that can help software quality assurance (SQA) meet its goals.

Templates

Template is “a gauge, pattern or mold (as a thin plate or board) used as a guide to the form of a piece being made” (Webster’s New College Dictionary).

When applied to software engineering, the term template refers to a format (especially tables of contents) created by units or organizations, to be applied when compiling a report or some other type of document. Application of templates may be obligatory for some documents and elective for others; in some cases, only part of a template (e.g., specific chapters or general structure) is demanded.

Three examples of templates are:

- Software test plan (STP)
- Software test description (STD)
- Software test report (STR).

Template use is quite advantageous to development teams and to review teams. For development teams, template use:

Facilitates the process of preparing documents by saving the time and energy required to elaborate the report’s structure. Most organizations allow templates to be copied from a SQA public file or downloaded from the organization’s intranet files, which even saves keying the table of contents to the new document.

Ensures that documents prepared by the developer are more complete as all the subjects to be included in the document have already been defined and repeatedly reviewed by numerous professionals over the course of the template’s use. Common errors, such as overlooking a topic, are less likely to occur.

Provides for easier integration of new team members through familiarity. The document’s standard structure, prepared according to templates that may be known to the new member from previous work in another of the organization’s units or teams, makes finding information much easier. It also smoothes ongoing document preparation in cases where parts of the document have been prepared by another team member who may or may not have Left.

Facilitates review of documents by eliminating the need to study a document’s structure and confirm its completeness, if the document is based on the appropriate template. It also simplifies review of the completed document as its structure is standard and reviewers are familiar with its expected contents (chapters, sections and appendices). As a result of this consistency, the review is expected to be more thorough yet less time-consuming.

For software maintenance teams, template use:

Enables easier location of the information required for performing maintenance tasks.

The organizational framework for preparing, implementing and updating templates Organizations tend to save their internal resources, which often mean employing successful reports prepared for one department or purpose as models for the entire organization. The SQA unit is usually responsible for preparing professional templates of the more common types of reports and documents required of the organization’s staff.

Preparation of new templates

Development of a template infrastructure naturally centers on the work of a group of professionals devoted to the task. This group (or committee) should include senior staff that represents the various software development lines, the department's chief software engineer and SQA unit members. Informal developers of "template services" should likewise be encouraged to join the group.

One of the group's first tasks is to compile a target list of templates to be developed.

Once the list is accepted, priorities must be set. Higher priority should be given to templates of the most commonly prepared documents.

Subcommittees are then assigned the task of preparing the first drafts. An SQA unit member can be anticipated to undertake the task of leading the group. Irrespective of whom the group's head may be, he or she must see to the distribution of template drafts among members, the organization of meetings and the follow-up of progress made by template preparation subcommittees.

Distribution of template drafts among team leaders for their comments can yield important improvements and at the same time promote the templates' future use.

The most common information sources used in preparing a template are as follows:

Informal templates already in use in the organization
Template examples found in professional publications
Templates used by similar organizations

Application of templates

Several fundamental decisions are involved in the implementation of new or updated templates:

- What channels should be used for advertising the templates?
- How should the templates be made available to the organization's internal "consumers"?
- Which templates will be compulsory and how can their application be enforced?

All professional internal means of communication can be used for advertising templates internally within the organization: leaflets, e-mail, intranet as well as short presentations at meetings. One of the most efficient methods of making templates available to the organization is the internal net (intranet).

Updating templates

The decision to update an existing template may be considered a reactive measure, stemming from any of the following:

- User proposals and suggestions
- Changes in the organization's areas of activity
- Proposals initiated by design review and inspection teams based on their review of documents prepared according to the templates
- Analysis of failures as well as successes
- Other organizations' experience
- SQA team initiatives.

Example

Test Cases : Template 1.0							
Project Code			Created By :		Verified By :		
Test ID			Use Cases References				
Test Types		<input type="checkbox"/> Unit Test	<input type="checkbox"/> Integration		<input type="checkbox"/> System		
Test Environment							
Quality Covered							
Pre-Conditions							
Post-Conditions							
Notes							
Testing Steps							
No.	Description	Test Data	Expected Results	Actual Results	Severity	Test Status	Remarks
				Test Date:			
				Total Test Time :			
				Test By:			

Ans :

Checklists are planned to be comprehensive if not complete. Usually, checklist use tends to be considered an optional infrastructure tool, depending mainly on the list's professional attributes, user acquaintance with the list and availability.

Like templates, checklists provide many benefits to development teams, software maintenance teams and document quality.

- i) Helps developers carrying out self-checks of documents or software code prior to document or software code completion and formal design reviews or inspections. Checklists are expected to help the developer discover incomplete sections as well as detect overlooked lapses.
- ii) Checklists are also expected to contribute to the quality of documents or software code submitted for review as the quality issues to be surveyed, by the review team are already listed in the checklist.
- iii) Assists developers in their preparations for tasks such as installation of software at customer sites, performance of quality audits at subcontractors' sites or signing contracts with suppliers of reused software modules. Checklists are expected to help the developers be better equipped for task performance.

The advantages to review teams are

- i) Assures completeness of document reviews by review team members as all the relevant review items appear on the list.
- ii) Facilitates improves efficiency of review sessions as the subjects and order of discussion are defined and well known in advance.
- iii) The organizational framework for preparing, implementing and updating checklists A “checklist group”, headed by a SQA unit member, can undertake the task of maintaining a collection of updated lists. The participation of other staff interested in promoting the use of checklists in the group is also voluntary; in some cases, however, the assistance of an SQA consultant is recommended.

Preparation of new checklists

One of the first tasks awaiting the “checklist group” is compilation of a list of checklists targeted for development, followed by definition of a common format for all the checklists released by the group.

The first checklists approved by the group are usually informal check-lists already in use by some development team members and reviewers. In most cases, a few changes and adaptations of these checklists are sufficient to satisfy the format and contents defined by the group. Preparation of new checklists as well as improvement of informal checklists is supported by the following sources of information:

Informal checklists already in use in the organization

Checklist examples found in books and other professional publications

Checklists used by similar organizations.

The process of preparing a new checklist is similar to that for templates.

Promotion of checklist use

As the use of checklists is rarely mandatory, promotion of their use is based on advertising and guaranteed availability. All internal channels of communication can be used for publicizing the checklists: leaflets, e-mail, SQA intranet as well as professional meetings. The internal net remains, however, the preferred and most efficient method for making checklists available to the organization’s internal “consumers”.

Updating checklists

Like templates and procedures, initiatives to update an existing checklist generally flow from the following sources:

- User proposals and suggestions
- Changes in technology, areas of activity and clientele
- Proposals initiated by design review and inspection teams emanating from document
- reviews
- Analysis of failures as well as successes
- Other organizations’ experience

SQA team initiatives.

The process of updating checklists is quite similar to their preparation.

Q16. Explain the importance of staff training and certification to improve the software quality.

Ans :

Developing - Staff training and certification

The objectives of training and certification

To develop the knowledge and skills new staff need to perform software development and maintenance tasks at an adequate level of efficiency and effectiveness. Such training facilitates integration of new team members.

To assure conformity to the organization’s standards for software products (documents and code) by transmitting style and structure procedures together with work instructions.

To update the knowledge and skills of veteran staff in response to developments in the organization, and to assure efficient and effective performance of tasks as well as conformity to the organization’s style and structure procedures and work instructions.

To transmit knowledge of SQA procedures.

The training and certification process

The operation of a successful training and certification system demands that the following activities be regularly performed:

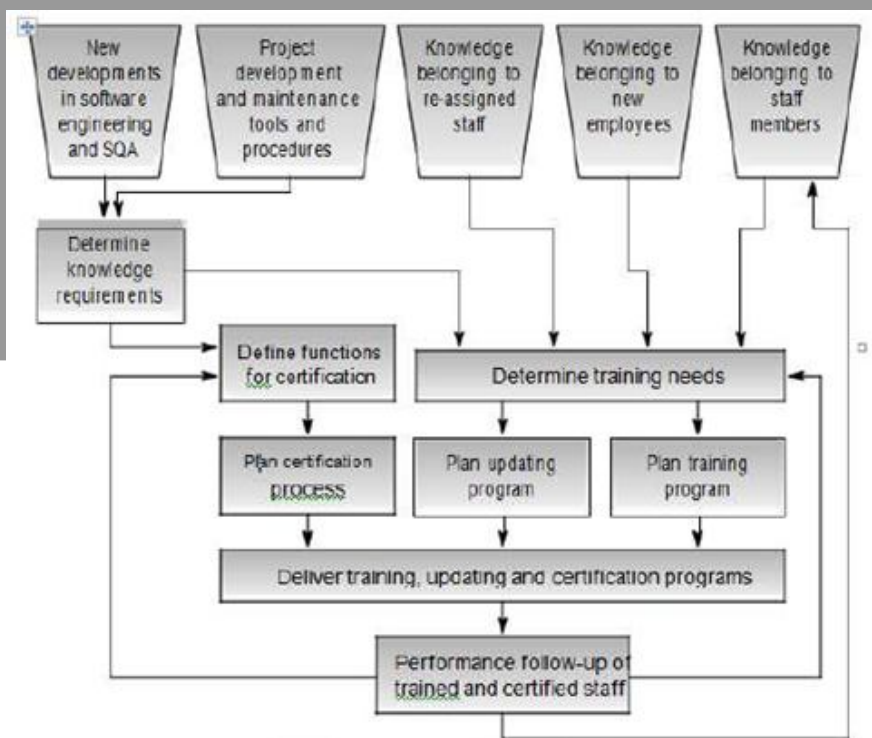
- Determine the professional knowledge requirements for each position
- Determine the professional training and updating needs

- Plan the professional training program
- Plan the professional updating program
- Define positions requiring certification
- Plan certification processes
- Deliver training, updating and certification programs
- Perform follow-up of trained and certified staff.

All these activities converge into an integrated process in which feedback from past activities and information about professional developments stimulates a cycle of continuous training certification and adaptation to changing quality requirements.

Training and certification activities are meant to fill the needs of veteran staff and new employees. Comprehensive follow-up of the outcomes of current programs as well as keeping track of developments in the profession are required to make sure that programs are adequately up-to-date.

The training and certification process is displayed in Figure:



Determining professional knowledge requirements

The most common positions in a software development and maintenance organization are those of systems analyst, programmer, software development team leader, programming team leader, software maintenance technician, software tester, and software testing team leader.

Most organizations set education and experience requirements for each of these positions. Staff members who fulfill education requirements still need additional “local” or “internal” knowledge and skills, related to specific development and maintenance procedures. This specialized knowledge can be grouped into two categories: Knowledge and skills of software engineering topics, such as software development tools, programming language versions, and CASE tool versions applied by the specific organization or unit. The relevant procedures and work instructions that were compiled for their implementation also belong to this category.

Knowledge of SQA topics, such as the procedures pertaining to the various development and maintenance activities, assigned to be performed by the individual occupying a specific position.

Determining training and updating needs

Training and updating needs are determined by comparison of the staff's current knowledge with the updated knowledge requirements. The type of training is adapted to the needs of three distinct groups of staff:

- Training for new employees, according to their designated assignment.
- Retraining for employees assigned to new positions or receiving new assignments.
- Updating for staff members as demanded by their position.

The need to update staff should be assessed regularly to facilitate planning of the required programs. Finally, follow-up of staff performance in the wake of training and updating provides major input to be used in redefining training needs.

Planning training and updating programs

Planning training and updating programs for software engineering topics:

The timing of many training and retraining activities cannot be determined in advance because new personnel are recruited and veteran staff are shifted often after relatively short notice. However, updating activities can be scheduled well ahead (the audience is known), with contents finalized close to the date of their implementation. Irrespective of whether the programs are carried out in-house or by an outsourcing organization, high-level staffs, such as the chief software engineer, usually participate in their preparation.

Planning training and updating programs for SQA topics

Training programs for SQA topics include training for new employees as well as updating for veteran staff members. The general characteristics of SQA training programs allow them to be organized periodically, every one or two months, and delivered to all new staff recruited in the interim.

Typical SQA updating programs are carried out once a year or once every six months, depending on the pace of change. The SQA unit or others responsible for SQA issues in the organization usually prepare these training and updating programs.

Defining positions requiring certification

It is commonly accepted that assignment of personnel to key positions in software development and maintenance organizations requires extreme care. One of the procedures used to guarantee the suitability of candidates is certification.

Planning the certification processes

Certification is intended to provide a framework for the thorough investigation of a candidate's qualifications and a demonstration of his or her professional knowledge and skills.

The details of the certification process are unique to the organization; they reflect its special characteristics, areas of specialization, software development and maintenance tools, customers and so on.

Because the process is geared toward the needs and decisions of specific organizations, internal certification cannot be automatically substituted by the general certification that is granted by professional societies and leading suppliers of development tools and network communication software or their equivalents.

The certification process, in every detail and for every position, requires approval as defined in the certification procedure.

Typical certification requirements

For the individual undergoing certification, a typical certification process entails meeting some or even all of the following requirements:

- **Professional education:** academic or technical degrees and in some cases certification by a professional organization or by a leading commercial software producer
- Professional experience in the organization (may be partially or completely replaced by experience in other organizations)
- Assessment of achievements and ability as noted in periodic performance appraisals
- Evaluation by the candidate's direct superior (often by completion of a special questionnaire)
- Demonstration of knowledge and skills by means of a test or a project
- Mentor's supervision for a specified period of time.

Functions of the certification committee

Similar to the pattern recommended for training and retraining programs, the person or committee members responsible for certification are usually senior software development and maintenance staff. The responsibilities of the certifying body include:

- To perform the certification process on the basis of requests made by individual applicants or units and grant certification to those who qualify
- To follow up certification activities (such as mentoring) carried out by others.
- To update certification requirements in response to developments in the organization as well as the profession
- To revise the list of positions requiring certification.

Delivery of training and certification programs

Training and updating can cover topics such as software engineering, software quality assurance and management skills (within the framework of certification or for general information), all of which are coordinated with the organization's or firm's needs. How training and updating are carried out varies accordingly.

Courses can be transmitted in formats that range from short lectures and demonstrations, often lasting only half a day, to lengthy courses held over several weeks or months. These may be conducted in-house, by the organization's training unit, or externally, by vocational or academic institutions that prepare programs attuned to the organization's requirements.

More about organizing and delivering training and certification programs can be found in the human resources management literature.

Follow-up subsequent to training and certification

Managers and software professionals often express doubts about the effectiveness of training and certification in general or of one of the associated activities. They question whether the substantial resources and efforts invested in training are really worthwhile. To assuage these doubts, systematic follow-up is necessary to provide feedback to the professional units.

Such feedback indicates whether the training efforts were justified at the same time that it assures continuous improvement of training and certification activities. The information provided by follow-up relates to:

All training activities and certification procedures conducted – records of the performance of the participants in the program.

Information about special cases of training activities that proved to be either highly successful or clearly unsuccessful in improving staff performance.

Information about proven cases of failures of certified staff in the performance that point to clearly inadequate certification requirements.

Analysis of the data accumulated following a training course provides the information necessary to revise programs by guiding the modification, addition and deletion of identified activities and materials. Meaningful follow-up of training requires performance information collected prior as well as sub-sequent to training.

The Corrective Action Board (CAB), takes action based on follow-up subsequent to training and certification and other sources of information.

2.6

CORRECTIVE AND PREVENTIVE ACTIONS

Q17. Explain about various corrective and preventive actions to improve the software quality.

Ans :

Corrective actions

A regularly applied feedback process that includes collection of information on quality non-conformities, identification and analysis of sources of irregularities as well as development and assimilation of improved practices and procedures, together with control of their implementation and measurement of their outcomes.

Preventive actions

A regularly applied feedback process that includes collection of information on potential quality problems, identification and analysis of departures from quality standards, development and assimilation of improved practices and procedures, together with control of their implementation and measurement of their outcomes.

The corrective and preventive actions process. Successful operation of a CAPA process includes the following activities:

1. Information collection
2. Analysis of information

3. Development of solutions and improved methods
4. Implementation of improved methods
5. Follow-up.

The process is regularly fed by the flow of information from a variety of sources. In order to estimate the success of the process, a closed feedback loop is applied to control the flow of information, implementation of the resulting changes in practices and procedures together with measurement of the outcomes.

1. Information collection

The variety of information sources, internal and external, that serve the CAPA process is quite remarkable, the four main internal sources of information are the (1) Software development process, (2) Software maintenance, (3) SQA infrastructure and (4) Software quality management procedures. External sources of information are mainly customers' application statistics and customer complaints.

Internal information sources are categorized as follows:

i) Software development process

- Software risk management reports
- Design review reports
- Walkthrough reports
- Experts' opinion reports
- Test reviews
- Special reports on development failures and successes
- Proposals suggested by staff members.

ii) Software maintenance

- Customer applications statistics
- Software change requests initiated by customer applications
- Software change requests initiated by maintenance staff
- Special reports on maintenance failures and successes
- Proposals suggested by staff members.

iii) SQA infrastructure class of sources

- Internal quality audit reports
- External quality audit reports
- Performance follow-up of trained and certified staff
- Proposals suggested by staff members.

iv) Software quality management procedures class of sources

- Project progress reports
- Software quality metrics reports
- Software quality cost reports
- Proposals of staff members.

v) **External information sources**

- Customer complaints
- Customer service statistics
- Customer-suggested proposals.

2. Analysis of collected information

Regular operation of the CAPA process is expected to create a massive flow of documents related to a wide range of information.

i) **Analysis involves**

Screening the information and identifying potential improvements. Documents received from the various sources of information are reviewed by professionals in order to identify potential opportunities for CAPA.

This stage includes comparison of documents of the same type received from various units as well as comparison of documents of different types related to the same case.

Analysis of potential improvements.

Efforts are directed to determine:

Typical causes are non-compliance with work instructions and procedures, insufficient technical knowledge, extreme time and/or budget pressures due to unrealistic estimates, and lack of experience with new development tools.

Estimates of the extent of organization-wide potential faults of each type. This information is needed to estimate the total damage expected and to determine the priority of each fault case.

3. Development of solutions

Solutions to identified causes of recurrent software systems faults are required to:

Eliminate recurrence of the types of faults detected

Contribute to improved efficiency by enabling higher productivity and shorter schedules.

4. Implementation of improved methods

Implementation of CAPA solutions relies on proper instructions and often training but most of all on the cooperation of the relevant units and individuals. Therefore, successful implementation requires that targeted staff members.

5. Follow-up of activities

Three main follow-up tasks are necessary for the proper functioning of a corrective and preventive action process in any organization:

i) Follow-up of the flow of development and maintenance CAPA records from the various sources of information. This enables feedback that reveals cases of no reporting as well as low quality reporting, where important details are missing or inaccurate. This type of follow-up is conducted mainly through analysis of long-term activity information, which generates feedback to the CAPA information sources.

ii) Follow-up of implementation. This activity is intended to indicate whether the designated actions – training activities, replacement of development tools, and procedural changes (after approval) – have been performed in practice. Adequate feedback is delivered to the bodies responsible for implementation of the corrective and preventive actions.

iii) Follow-up of outcomes. Follow-up of the improved methods' actual out-comes, as observed by project teams and organizational units, enables assessment of the degree to which corrective or preventive actions have achieved the expected results. Feedback on the outcomes is delivered to the improved methods' developers. In cases of low performance, formulation of a revised or new corrective action is needed, a task undertaken by the CAPA team.

2.7

CONFIGURATION MANAGEMENT

Q18. Explain briefly about configuration management.

Ans :

Configuration management is the process of identifying, organizing, controlling, and tracking changes to a software product or system throughout its entire life cycle. It is a key aspect of software engineering that involves managing and maintaining the various artifacts that make up a software product, including source code, documentation, and other project artifacts.

Configuration management helps to ensure that changes to the software product are properly managed and tracked, so that developers can maintain a clear understanding of the product's status at all times. The primary goals of configuration management include:

1. **Establishing a baseline:** A baseline is a snapshot of the software product at a specific point in time. Configuration management ensures that a baseline is established for each version of the software product, so that developers can easily track changes and ensure that the product is developed consistently.
2. **Managing changes:** Configuration management ensures that all changes to the software product are properly managed and tracked. This helps to ensure that changes are properly documented, reviewed, and approved before they are incorporated into the product.
3. **Controlling versions:** Configuration management ensures that multiple versions of the software product can be maintained and tracked. This helps to ensure that developers can easily access and manage previous versions of the product as needed.
4. **Facilitating collaboration:** Configuration management facilitates collaboration among developers by ensuring that everyone is working with the same version of the software product. This helps to prevent conflicts and reduces the risk of errors and defects.

Configuration management involves using various tools and techniques to manage and track changes to the software product. These may include version control systems, issue tracking systems, and build automation tools. By using configuration management, software developers can improve the quality of the software product and ensure that it is developed consistently and efficiently over time.

Discuss the concept of Software configuration management

An SQA component responsible for applying (computerized and non-computerized) technical tools and administrative procedures that enable completion of the tasks required to maintain SCIs and software configuration versions.

The tasks of software configuration management may be classified into four groups:

1. Software Change Control
2. Quality assurance of software changes.
3. Release of software

1. Software change control

Software change management controls the process of introducing changes mainly by doing the following:

Examining change requests and approving implementation of appropriate requests. Assuring the quality of each new version of software configuration before it becomes operational.

(i) Approval to carry out proposed changes

Baseline software configuration versions are planned early, during a system's development or operating stage. As part of the process, they are reviewed, tested and approved, as are their SCIs. Baseline versions serve as milestones in the software system's life cycle, and represent the foundations for further system development.

The factors affecting the decision whether to implement a proposed change include:

- Expected contribution of the proposed change
- Urgency of the change
- Effect of the proposed change on project timetables, level of service, etc.
- Efforts required in making the change operational
- Required software quality assurance efforts
- Estimated required professional resources and cost of performing the change.

The information items required before any decision about a change proposal can be made are reflected in the contents of a typical software change request (SCR) form. It consist of

ii) Change principles

- The initiator
- The date the SCR was presented
- The character of the change
- The expected contribution to the project/system
- The urgency of performance

iii) Change details

- Description of the proposed change
- A list of the SCIs to be changed
- Expected effect on other SCIs
- Expected effect on interfaces with other software systems and hardware firmware
- Expected delays in development completion schedules and expected disturbances to services to customers

iv) Change timetable and resources estimates

- Timetable for implementation
- Estimated required professional resources
- Other resources required
- Estimated total cost of the requested change

2. Quality assurance of software changes

- Quality assurance efforts are required at two levels:
- Quality assurance of each of the changed SCIs
- Quality assurance of the entire new software system version (that includes changed SCIs).

(i) Quality assurance of the changed SCIs

This requires preparation of a reviews and testing plan at a magnitude appropriate to the character of the change. It is most important that reviews and testing be carried out by professional testers and not by the SCI's developer. The process of reviews and testing, corrections and retesting (regression testing) the changed SCIs is expected to conclude with their approval.

(ii) Quality assurance of the entire new software system version

A new version of the software is considered to have been completed once the changed SCIs replace the former SCIs. Although one might expect the new version of the software system to function perfectly and certainly better than the old original version. The system failures generally occur as a result of damage done to interfaces between the changed SCIs and other SCIs left unchanged and not retested. The entire new version, or at least all the whole software parts that might be affected, is tested to identify unexpected interface defects.

3. Release of software configuration versions

The need to release a new software configuration version usually stems from one or more of the following conditions:

Defective SCIs

Special features demanded by new customers

The team's initiatives to introduce SCI improvements.

Types of software configuration releases

Among software configuration releases, baseline versions, intermediate versions and revisions are considered to be the three main types of release.

Baseline versions

Baseline software configuration versions are planned early, during a system's development or operating stage. As part of the process, they are reviewed, tested and approved, as are their SCIs. Baseline versions serve as milestones in the software system's life cycle, and represent the foundations for further system development.

Intermediate versions

When problems arise that require immediate attention (such as the need to correct defects identified in an important SCI) or perform immediate adaptations as defined in a contract with a new customer: an intermediate version of the software is often prepared.

Usually, intermediate versions serve only a portion of a firm's customers, and then for a limited period, until replaced by a new baseline version. Naturally, we can expect that these versions will not receive the attention and investment of efforts usually devoted to the release of baseline versions. An intermediate software configuration version can thus serve as a "pilot" or springboard to the next baseline version.

Revisions

Revisions introduce minor changes and corrections to a given software configuration version.

Q19. Explain about software configuration management plans.

Ans :

Software configuration management plans (SCMPs)

The main objective of a software configuration management plan (SCMP) is to plan ahead the schedule

of baseline version releases and the required resources to carry out all the activities required for the software configuration releases. An additional objective of the SCMP is to enable one to follow up the progress of activities involved in software version release. SCMPs are required during the development stage as well as the operation (maintenance) stage.

Accordingly, an SCMP usually includes:

- An overview of the software development project or existing software system.
- A list of scheduled baseline version releases.
- A list of SCIs (documents, code, etc.) to be included in each version.
- A table identifying the relationship of software development project plans and
- maintenance plans to scheduled releases of new SCIs or SCI versions.
- A list of assumptions about the resources required to perform the various activities required by the SCMP.
- Estimates of the human resources and budget needed to perform the SCMP.

SCMP for the development stage

Based on the project plan, the SCMP sets the release dates of baseline versions, which usually coincide with the conclusion of one or more of the following three events: the design stage, the coding stage and the system test stage.

Quite commonly, these plans represent a segment of the entire system's development plans, prepared at a project's initiation. External participants in the project are required to comply with the SCMP or to suggest an alternative SCMP that is appropriate for their part of the project, contingent on its acceptance by the project manager.

All the instructions and procedures necessary for performing SCM tasks at this stage are documented in the SCMP. The project manager is usually the person responsible for carrying out these tasks.

All the instructions and procedures for performing SCM tasks during the operation (maintenance) stage are likewise documented in the respective SCMP.

Q20. Explain briefly about Software configuration evolution models Software configuration management audits.

Ans :

Successive development or evolution of a software system's configuration versions should be undertaken according to a route that is planned in advance by the system's developer.

The choice of routes depends on the system's characteristics, the customer population and the firm's intentions regarding the system's market. Two fundamental software configuration evolution models – the linear model and the tree model are generally applied.

The linear evolution model

According to the linear model, only one unique software system's configuration version serves all customers at any given time. Each new configuration version then replaces the prior version. This model is the natural choice for software systems developed to serve a single organization. The model is also applied to popular software packages, which tend to be uniform in structure, where the need to meet a wide range of maintenance demands for a single version is a great advantage.

The tree evolution model

According to this model, several parallel versions of the software are developed to serve the needs of different customers simultaneously throughout the system's life cycle.

Tree models are typically applied in firmware configuration versions, where each branch serves a different product or product line.

Software configuration management audits

SCM audits are performed by the SCM authority and the CCB in order to control compliance with SCM procedures. SCM audits may be combined with internal quality issues and are expected to initiate updates and changes of SCM procedures and instructions.

SCM audits may be also performed for a sample of planned releases, as specified in the SCMP. The following is a list of typical bits of control information that SCM audits are meant to discover and transmit to management:

- Percentage of unapproved changes introduced in the system during development or operation.

- Percentage of SCOs not carried out according to instructions and not fully complying with procedures.
- Percentage of design reviews and software tests of changed SCIs that have not been performed according to the relevant procedures.
- Percentage of SCOs that have been completed on schedule.
- Percentages of cases where SCIs affected by changes have not been checked, with some necessary changes not implemented.
- Percentages of properly documented new SCIs and software configuration versions.
- Percentage of properly documented installations of new software configuration versions.
- Percentage of cases of failure to transmit all version-related information to the customer.

2.8**DOCUMENTATION CONTROL**

Q21. What is document control? Explain.

Ans :

Documentation control is a process of managing the creation, review, approval, distribution, and storage of documents related to a software project. This process ensures that all documents are up-to-date, accurate, and easily accessible to the project team. Effective documentation control helps to ensure that the software product is developed consistently, efficiently, and to the required quality standards.

Documentation control involves several steps, including:

- 1. Document Creation:** Documents are created by the appropriate team members, following a defined format and style guide. The documents may include technical specifications, design documents, user manuals, test plans, and other related documents.
- 2. Review and Approval:** Documents are reviewed and approved by the relevant stakeholders, including technical leads, project managers, and customers. This ensures that the documents are accurate, complete, and meet the required quality standards.
- 3. Distribution:** Once approved, the documents are distributed to the relevant team members.

This may include developers, testers, project managers, and other stakeholders who require access to the documents.

- 4. Storage and Maintenance:** The documents are stored in a central repository that is accessible to all team members. The repository should have appropriate access controls and versioning features to ensure that only authorized team members can access and modify the documents. The documents are also regularly updated and maintained as needed to ensure that they remain accurate and up-to-date.

Effective documentation control is essential for the success of a software project. It helps to ensure that the project team has access to the most current and accurate information, which enables them to work more efficiently and effectively. By managing the creation, review, approval, distribution, and storage of project documents, documentation control helps to ensure that the project is completed on time, within budget, and to the required quality standards.

2.9**PROJECT PROGRESS CONTROL**

Q22. Explain the components of project progress control?

Ans :

Project progress control is the process of monitoring and tracking a project's progress to ensure that it is meeting its goals and objectives within the planned timeline and budget. It involves several components that work together to provide visibility into the project's status and allow project managers to make informed decisions. Here are the main components of project progress control:

- 1. Project plan**

The project plan is the foundation of project progress control. It outlines the project's objectives, scope, timeline, budget, resources, and other key details. The project plan serves as a reference point for measuring progress and identifying any deviations from the original plan.

- 2. Project schedule**

The project schedule is a timeline of activities and milestones that need to be completed to achieve the project objectives. The schedule provides a roadmap for the project team and helps to track progress against the plan.

3. Performance metrics

Performance metrics are measurable indicators of progress toward project goals. They help project managers to assess the health of the project and identify areas where corrective action may be needed. Examples of performance metrics include budget variance, schedule variance, resource utilization, and quality metrics.

4. Status reports

Status reports are regular updates on the project's progress. They provide information on key performance metrics, issues, risks, and other relevant information. Status reports can be used to communicate project progress to stakeholders and to identify areas where corrective action may be needed.

5. Change control

Change control is the process of managing changes to the project scope, schedule, or budget. It involves evaluating the impact of changes on the project plan and making adjustments as needed. Change control helps to ensure that changes are properly managed and that they do not negatively impact the project's progress.

6. Risk management

Risk management is the process of identifying and mitigating risks that may impact the project's progress. It involves identifying potential risks, assessing their likelihood and impact, and developing strategies to mitigate them. Risk management helps to ensure that the project stays on track and that potential risks are identified and managed proactively.

In summary, project progress control involves several components, including the project plan, project schedule, performance metrics, status reports, change control, and risk management. By effectively managing these components, project managers can ensure that the project stays on track and achieves its goals within the planned timeline and budget.

Implementation issues involved in project progress control

Project progress control is an important part of project management, but implementing it successfully can be challenging. Here are some implementation issues that can arise when implementing project progress control:

1. Data collection

One of the biggest challenges in project progress control is collecting accurate and timely data. This can be a challenge if project team members are not diligent in reporting progress or if there is no system in place for collecting and consolidating data.

2. Resource allocation

Another challenge is allocating resources to implement project progress control. This includes not only the tools and technology needed but also the people responsible for collecting and analyzing data.

3. Stakeholder engagement

Project progress control requires engagement and buy-in from all stakeholders involved in the project. This can be challenging if stakeholders have competing priorities or are not fully committed to the project's success.

4. Training and communication

Implementing project progress control requires training team members on the tools and processes involved. It also requires effective communication to ensure that everyone understands their roles and responsibilities and the importance of project progress control.

5. Resistance to change

Resistance to change is a common issue when implementing project progress control. Team members may be resistant to new tools and processes, or they may be hesitant to report progress for fear of being held accountable for delays or issues.

6. Continuous improvement

Finally, implementing project progress control is an ongoing process that requires continuous improvement. This includes evaluating the effectiveness of the tools and processes used and making adjustments as needed to ensure that project progress control remains effective and relevant.

In summary, implementing project progress control requires overcoming several challenges, including data collection, resource allocation, stakeholder engagement, training and communication, resistance to

change, and continuous improvement. By addressing these issues proactively, project managers can improve the chances of successfully implementing project progress control and ensuring the success of their projects.

Integrating quality activities in the project life cycle

Integrating quality activities into the project life cycle involves incorporating quality management practices into each phase of The software development process. This ensures that quality is built into the software product from the beginning and that defects are caught and corrected as early as possible. Here are some ways that quality activities can be integrated into the project life cycle:

1. Planning Phase

In this phase, the project team should establish quality objectives and define the quality requirements for the software product. Quality planning activities may include identifying quality standards, establishing quality metrics, and creating a quality management plan.

2. Requirements Gathering Phase

In this phase, the project team should ensure that quality is built into the requirements gathering process. This may involve defining acceptance criteria, conducting stakeholder interviews to gather requirements, and ensuring that requirements are clear, complete, and testable.

3. Design Phase

In this phase, the project team should ensure that the software design meets quality standards and is appropriate for the intended use of the product. This may involve conducting design reviews, evaluating design alternatives, and ensuring that the design is modular, maintainable, and extensible.

4. Implementation Phase

In this phase, the project team should ensure that the code meets quality standards and that defects are caught and corrected as early as possible. Quality activities in this phase may include code reviews, unit testing, and integration testing.

5. Testing Phase

In this phase, the project team should ensure that the software product meets quality standards and that defects are identified and corrected. Quality activities in this phase may include functional testing, performance testing, and security testing.

6. Deployment Phase

In this phase, the project team should ensure that the software product is deployed correctly and that end-users are satisfied with the quality of the product. Quality activities in this phase may include user acceptance testing, customer feedback, and post-deployment support.

By integrating quality activities into each phase of the software development process, the project team can ensure that quality is built into the software product from the beginning. This helps to reduce the overall cost of quality and improve customer satisfaction by delivering a high-quality software product that meets the needs of end-users.

UNIT III

Software Quality Metrics, Costs of Software Quality, Quality Management Standards - ISO 9000 and Companion ISO Standards, CMM, CMMI, PCMM, Malcom Balridge, 3 Sigma, 6 Sigma, SQA Project Process Standards – IEEE Software Engineering Standards.

3.1

SOFTWARE QUALITY METRICS

Q1. Explain briefly about software quality Metrics.

Ans :

(Imp.)

Software quality metrics are quantitative measures used to evaluate and assess the quality of software products or processes. These metrics provide a way to objectively measure and track the performance of software development, testing, and maintenance activities.

Here are some common software quality metrics:

1. **Defect density:** The number of defects identified in the software per unit of size (e.g., per line of code).
2. **Code coverage:** The percentage of code that has been executed during testing.
3. **Test case coverage:** The percentage of test cases that have been executed.
4. **Code complexity:** The complexity of the software code, as measured by metrics such as cyclomatic complexity or the number of code paths.
5. **Response time:** The time taken by the system to respond to user input.
6. **Mean time between failures (MTBF):** The average time between failures in the software.
7. **Mean time to repair (MTTR):** The average time taken to repair the software after a failure.
8. **User satisfaction:** User satisfaction with the software, as measured by surveys or other feedback mechanisms.
9. **Time to market:** The time taken to release the software to the market.

10. Resource utilization: The resources (e.g., CPU, memory) used by the software during operation.

These metrics can be used to track the progress of software development, identify areas for improvement, and ensure that the software meets the required quality standards. However, it's important to note that metrics should be used in conjunction with other quality assurance techniques, such as code reviews and testing, to ensure that software quality is maintained at all stages of the development process.

3.1.1 Costs of Software Quality

Q2. Write about the costs of software quality

Ans :

(Imp.)

The major objectives of software quality metrics includes the management to achieve economic control over SQA activities and outcomes.

The specific objectives are:

- Control organization-initiated costs to prevent and detect software errors.
- Evaluation of the economic damages of software failures as a basis for revising the SQA budget.
- Evaluation of plans to increase or decrease SQA activities or to invest in SQA infrastructure on the basis of past economic performance.

The model further subdivides these classes into subclasses:

Costs of Control

Prevention costs: Investments in quality infrastructure and quality activities that are not directed to a specific project or system. Appraisal costs: The costs of activities performed for a specific project or software system for the purpose of detecting software errors.

Costs of Failure of Control

Internal failure costs: Costs of correcting errors that have been detected by design reviews, software tests and acceptance tests and completed before the software is installed at customer sites. External failure costs: All costs of correcting failures detected by customers or the maintenance team after the software system has been installed.

Prevention costs include investments in establishing a software quality infrastructure, updating and improving that infrastructure as well as performing the regular activities required for its operation.

- (a) Investments in development of SQA infrastructure components

- Procedures and work instructions
- Support devices: templates, checklists etc
- Software configuration management system
- Software quality metrics

- (b) Regular implementation of SQA preventive activities

- Instruction of new employees in SQA subjects
- Certification of employees

- (c) Control of the SQA system through performance of

- Internal quality reviews
- External quality audits
- Management quality reviews

Appraisal costs are devoted to detection of software errors in specific projects or software systems.

Typical appraisal costs cover:

- (a) Costs of reviews
- Formal design reviews (DRs)
 - Peer reviews (inspections and walkthroughs)
 - Expert reviews
- (b) Costs of software testing
- Unit, integration and software system tests
 - Acceptance tests (carried out by customers)
- (c) Costs of assuring quality of external participants by means of design reviews and software testing. These activities are applied to the activities performed by

- Subcontractors
- Suppliers of COTS software systems and reusable software modules
- The customer as a participant in performing the project.

Internal Failure Costs consists of the Costs of redesign or design corrections subsequent to design review and test findings, Costs of re-programming or correcting programs in response to test findings, Costs of repeated design review and re-testing (regression tests). External failure costs entail the costs of correcting failures detected by customers or maintenance teams after the software system has been installed at customer sites.

Typical external failure costs cover: Resolution of customer complaints during the warranty period. Correction of software bugs detected during regular operation. Correction of software failures after the warranty period is over even if the correction is not covered by the warranty.

In the extended model for cost of the software quality it analyses the software quality costs defined by the classic model which reveals that several costs of substantial magnitude are excluded. Typical software quality failure costs include:

- Damages paid to customers as compensation for late completion of the project due to unrealistic scheduling.
- Damages paid to customers in compensation for late completion of the project as a result of failure to recruit sufficient staff.

The element common to these two failures is that they result not from any particular action of the development team or any lack of professionalism; they are actually outcomes of Managerial failure.

Managerial preparation and control costs are associated with activities performed to prevent managerial failures or reduce prospects of their occurrence. Managerial failure costs can be incurred throughout the entire course of software development, beginning in the pre-project stage.

In order to apply a cost of software quality system in an organization, the following are required

- Definition of a cost of software quality model and specification of cost items: the organization should select the classic or extended model.

- Definition of the method of data collection for each cost item: use of Management Information Systems (MIS) in place.
- Application of a cost of software quality system, including thorough follow up.
- Actions taken in response to the model's findings.

At a preliminary stage in a project, the organization has to choose its type of cost model – the classic or the extended model. Effectiveness of the selected model is determined to a great degree by its suitability for the organization or project of the cost items designed to be measured for the model.

Each item should belong to one of the subclasses comprising the cost model. Once the list of software quality cost items is finalized, a method for collecting the relevant data must be determined.

3.2 QUALITY MANAGEMENT STANDARDS - ISO 9000 AND COMPANION ISO STANDARDS

Q3. Write ISO 9000 standards for quality management

Ans : (Imp.)

ISO 9000 is defined as a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements needed to maintain an efficient quality system. They are not specific to any one industry and can be applied to organizations of any size.

ISO 9000 can help a company satisfy its customers, meet regulatory requirements, and achieve continual improvement. It should be considered to be a first step or the base level of a quality system.

ISO 9000 series of Standards

The ISO 9000 family contains these standards:

- **ISO 9001:2015:** Quality Management Systems - Requirements
- **ISO 9000:2015:** Quality Management Systems - Fundamentals and Vocabulary (definitions)
- **ISO 9004:2018:** Quality Management - Quality of an Organization - Guidance to Achieve Sustained Success (continuous improvement)

- **ISO 19011:2018:** Guidelines for Auditing Management Systems

ASQ is the only place where organizations can obtain the American National Standard Institute (ANSI) versions of these standards in the ISO 9000 family.

ISO 9000 history and revisions: ISO 9000:2000, 2008, and 2015

ISO 9000 was first published in 1987 by the International Organization for Standardization (ISO), a specialized international agency for standardization composed of the national standards bodies of more than 160 countries. The standards underwent revisions in 2000 and 2008. The most recent versions of the standard, ISO 9000:2015 and ISO 9001:2015, were published in September 2015.

ASQ administers the U.S. Technical Advisory Groups and subcommittees that are responsible for developing the ISO 9000 family of standards. In its standards development work, ASQ is accredited by ANSI.

ISO 9000:2000

ISO 9000:2000 refers to the ISO 9000 update released in the year 2000.

The ISO 9000:2000 revision had five goals:

- Meet stakeholder needs
- Be usable by all sizes of organizations
- Be usable by all sectors
- Be simple and clearly understood

Connect quality management system to business processes

ISO 9000:2000 was again updated in 2008 and 2015. ISO 9000:2015 is the most current version.

ISO 9000:2015 principles of Quality Management

The ISO 9000:2015 and ISO 9001:2015 standards are based on seven quality management principles that senior management can apply to promote organizational improvement.

Customer Focus

Understand the needs of existing and future customers

Align organizational objectives with customer needs and expectations

- Meet customer requirements

- Measure customer satisfaction
- Manage customer relationships
- Aim to exceed customer expectations

Leadership

Establish a vision and direction for the organization

- Set challenging goals
- Model organizational values
- Establish trust
- Equip and empower employees
- Recognize employee contributions

Engagement of People

Ensure that people's abilities are used and valued

- Make people accountable
- Enable participation in continual improvement
- Evaluate individual performance
- Enable learning and knowledge sharing
- Enable open discussion of problems and constraints

Process Approach

- Manage activities as processes
- Measure the capability of activities
- Identify linkages between activities
- Prioritize improvement opportunities
- Deploy resources effectively

Improvement

Improve organizational performance and capabilities

- Align improvement activities
- Empower people to make improvements
- Measure improvement consistently
- Celebrate improvements

Evidence-based decision making

Ensure the accessibility of accurate and reliable data

- Use appropriate methods to analyze data
- Make decisions based on analysis
- Balance data analysis with practical experience

Relationship Management

- Identify and select suppliers to manage costs, optimize resources, and create value
- Establish relationships considering both the short and long term
- Share expertise, resources, information, and plans with partners
- Collaborate on improvement and development activities
- Recognize supplier successes

3.3

CMM

Q4. What is the Capability Maturity Model (CMM)? Explain.

Ans :

(Imp.)

Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987. It is not a software process model.

It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products. It also provides guidelines to enhance further the maturity of the process used to develop those software products.

It is based on profound feedback and development practices adopted by the most successful organizations worldwide. This model describes a strategy for software process improvement that should be followed by moving through 5 different levels. Each level of maturity shows a process capability level. All the levels except level 1 are further described by Key Process Areas (KPA).

Importance of Capability Maturity Model

- **Optimization of Resources:** CMM helps businesses make the best use of all of their resources, including money, labor, and time. Organizations can improve the effectiveness of resource allocation by recognizing and getting rid of unproductive practices.

- **Comparing and Evaluating:** A formal framework for benchmarking and self-evaluation is offered by CMM. Businesses can assess their maturity levels, pinpoint their advantages and disadvantages, and compare their performance to industry best practices.
- **Management of Quality:** CMM emphasizes quality management heavily. The framework helps businesses apply best practices for quality assurance and control, which raises the quality of their goods and services.
- **Enhancement of Process:** CMM gives businesses a methodical approach to evaluate and enhance their operations. It provides a road map for gradually improving processes, which raises productivity and usefulness.
- **Increased Output:** CMM seeks to boost productivity by simplifying and optimizing processes. Organizations can increase output and efficiency without compromising quality as they go through the CMM levels.

Principles of Capability Maturity Model (CMM)

- People's capability is a competitive issue. Competition arises when different organizations are performing the same task (such as software development). In such a case, the people of an organization are sources of strategy and skills, which in turn results in better performance of the organization.
- The people's capability should be defined by the business objectives of the organization.
- An organization should invest in improving the capabilities and skills of the people as they are important for its success.
- The management should be responsible for enhancing the capability of the people in the organization.
- The improvement in the capability of people should be done as a process. This process should incorporate appropriate practices and procedures.
- The organization should be responsible for providing improvement opportunities so that people can take advantage of them.
- Since new technologies and organizational practices emerge rapidly, organizations should continually improve their practices and develop the abilities of people.

Shortcomings of the Capability Maturity Model (CMM)

- It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.
- It only helps if it is put into place early in the software development process.
- It has no formal theoretical basis and in fact, is based on the experience of very knowledgeable people.
- It does not have good empirical support and this same empirical support could also be constructed to support other models.
- Difficulty in measuring process improvement: The SEI/CMM model may not provide an accurate measure of process improvement, as it relies on self-assessment by the organization and may not capture all aspects of the development process.
- Focus on documentation rather than outcomes: The SEI/CMM model may focus too much on documentation and adherence to procedures, rather than on actual outcomes such as software quality and customer satisfaction.
- May not be suitable for all types of organizations: The SEI/CMM model may not be suitable for all kinds of organizations, particularly those with smaller development teams or those with less structured development processes.
- May not keep up with rapidly evolving technologies: The SEI/CMM model may not be able to keep up with rapidly evolving technologies and development methodologies, which could limit its usefulness in certain contexts.
- Lack of agility: The SEI/CMM model may not be agile enough to respond quickly to changing business needs or customer requirements, which could limit its usefulness in dynamic and rapidly changing environments.

Key Process Areas (KPA)

Each of these KPA (Key Process Areas) defines the basic requirements that should be met by a software process to satisfy the KPA and achieve that level of maturity.

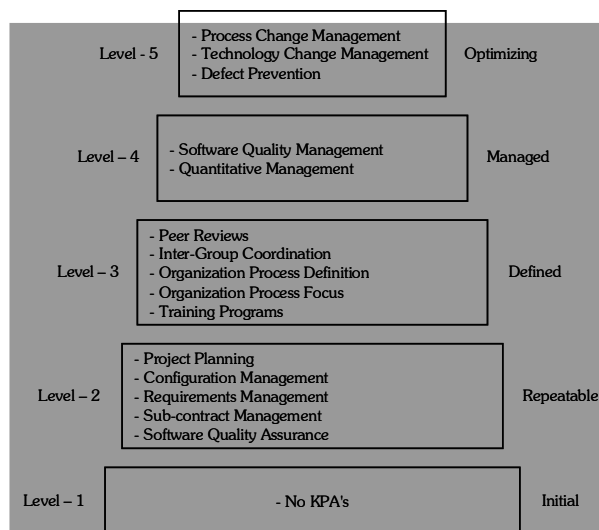
Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

Q5. Explain the levels of CMM.

Ans :

Levels of Capability Maturity Model (CMM)

There are 5 levels of Capability Maturity Models. We will discuss each one of them in detail.



CMM

Level-1: Initial

- No KPIs defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.
- Limited project management capabilities, such as no systematic tracking of schedules, budgets, or progress.
- We have limited communication and coordination among team members and stakeholders.
- No formal training or orientation for new team members.

- Little or no use of software development tools or automation.
- Highly dependent on individual skills and knowledge rather than standardized processes.
- High risk of project failure or delays due to a lack of process control and stability.

Level-2: Repeatable

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar-natured projects.
- Project Planning- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good-quality software.
- Configuration Management- The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- Requirements Management- It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- Subcontract Management- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software developed by third parties.
- Software Quality Assurance- It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- Peer Reviews: In this method, defects are removed by using several review methods like walkthroughs, inspections, buddy checks, etc.
- Intergroup Coordination: It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.

- Organization Process Definition: Its key focus is on the development and maintenance of standard development processes.
- Organization Process Focus: It includes activities and practices that should be followed to improve the process capabilities of an organization.
- Training Programs: It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- Software Quality Management: It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- Quantitative Management: It focuses on controlling the project performance quantitatively.

Level-5: Optimizing

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- The use of new tools, techniques, and evaluation of software processes is done to prevent the recurrence of known defects.
- Process Change Management: Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- Technology Change Management: It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- Defect Prevention It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

3.4

CMMI

Q6. What is Capability Maturity Model Integration (CMMI)?

Ans :

(Imp.)

Capability Maturity Model Integration (CMMI) is a successor of CMM and is a more evolved model that incorporates best components of individual disciplines of CMM like Software CMM, Systems Engineering CMM, People CMM, etc.

Since CMM is a reference model of matured practices in a specific discipline, so it becomes difficult to integrate these disciplines as per the requirements. This is why CMMI is used as it allows the integration of multiple disciplines as and when needed.

Objectives of CMMI

1. Fulfilling customer needs and expectations.
2. Value creation for investors/stockholders.
3. Market growth is increased.
4. Improved quality of products and services.
5. Enhanced reputation in Industry.

CMMI Representation – Staged and Continuous

A representation allows an organization to pursue a different set of improvement objectives. There are two representations for CMMI :

Staged Representation

- uses a pre-defined set of process areas to define improvement path.
- provides a sequence of improvements, where each part in the sequence serves as a foundation for the next.
- an improved path is defined by maturity level.
- maturity level describes the maturity of processes in organization.
- Staged CMMI representation allows comparison between different organizations for multiple maturity levels.

Continuous Representation

- allows selection of specific process areas.
- uses capability levels that measures improvement of an individual process area.

- Continuous CMMI representation allows comparison between different organizations on a process-area-by-process-area basis.
- allows organizations to select processes which require more improvement.
- In this representation, order of improvement of various processes can be selected which allows the organizations to meet their objectives and eliminate risks.

Q7. Discuss the various levels of CMMI.

Ans :

CMMI Model – Maturity Levels

In CMMI with staged representation, there are five maturity levels described as follows :

1. Maturity level 1 : Initial

- processes are poorly managed or controlled.
- unpredictable outcomes of processes involved.
- ad hoc and chaotic approach used.
- No KPAs (Key Process Areas) defined.
- Lowest quality and highest risk.

2. Maturity level 2 : Managed

- requirements are managed.
- processes are planned and controlled.
- projects are managed and implemented according to their documented plans.
- This risk involved is lower than Initial level, but still exists.
- Quality is better than Initial level.

3. Maturity level 3 : Defined

- processes are well characterized and described using standards, proper procedures, and methods, tools, etc.
- Medium quality and medium risk involved.
- Focus is process standardization.

4. Maturity level 4 : Quantitatively managed

- quantitative objectives for process performance and quality are set.

- quantitative objectives are based on customer requirements, organization needs, etc.
- process performance measures are analyzed quantitatively.
- higher quality of processes is achieved.
- lower risk

5. Maturity level 5 : Optimizing

- continuous improvement in processes and their performance.
- improvement has to be both incremental and innovative.

- highest quality of processes.
- lowest risk in processes and their performance.

Q8. Explain briefly about CMMI model.

Ans :

(Imp.)

CMMI Model – Capability Levels

A capability level includes relevant specific and generic practices for a specific process area that can improve the organization's processes associated with that process area. For CMMI models with continuous representation, there are six capability levels as described below :

1. Capability level 0 : Incomplete

- incomplete process – partially or not performed.
- one or more specific goals of process area are not met.
- No generic goals are specified for this level.
- this capability level is same as maturity level 1.

2. Capability level 1 : Performed

- process performance may not be stable.
- objectives of quality, cost and schedule may not be met.
- a capability level 1 process is expected to perform all specific and generic practices for this level.
- only a start-step for process improvement.

3. Capability level 2 : Managed

- process is planned, monitored and controlled.
- managing the process by ensuring that objectives are achieved.
- objectives are both model and other including cost, quality, schedule.
- actively managing processing with the help of metrics.

4. Capability level 3 : Defined

- a defined process is managed and meets the organization's set of guidelines and standards.
- focus is process standardization.

5. Capability level 4 : Quantitatively Managed

- process is controlled using statistical and quantitative techniques.
- process performance and quality is understood in statistical terms and metrics.
- quantitative objectives for process quality and performance are established.

6. Capability level 5 : Optimizing

- focuses on continually improving process performance.
- performance is improved in both ways – incremental and innovation.
- emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

3.5

PCMM

Q9. What is PCMM? Explain various methods of PCMM.

Ans :

(Imp.)

- The PCMM is a working framework used for the organization in defining the maturity structure for improving and developing the skill set of people who work within the organization.
- The PCMM framework defines the complete path from starting an initial phase, which includes

inconsistent business activities, to a complete, mature phase that includes proper development of skills and working practices to benefit the organization.

- Using the PCMM framework, the organization can benefit from their business activities and find the critical people issues which they face in the organization while doing their working activities.
- The framework guides the organization to develop the people's skill set, knowledge set and solve all the [people issues for improving the organisation's business activities.
- The PCMM framework includes the five stages for achieving the goal: the initial level, managed level, defined level, predictable level, and optimizing level.
- All these levels help the organization achieve its desired goals and objectives and compete with other organizations in the market.

Methods of PCMM

In the PCMM framework, five methods help in continuous improvement of the knowledge set, skill set, development of effective methods, and improve the people mindset for the organisation's benefit. These five methods are 5 maturity level which has its own importance which is used for defining the capability and develop the capability within the organization.

The 5 PCMM maturity level are defined below:

1. Initial Level

This phase deals with inconsistent management. In this stage, there are no defined process areas. At this level, the organization process is ritualistic and inconsistent. The process is disorganized because they are not properly defined and documented, and the organization growth is dependent on individual efforts.

The process is not also in the repeatable phase because of not properly defined. The people of an organization have the skill set, but they don't know how to use them properly for the organization, and they also have a less emotional attachment towards the organization. That's the main reason for inconsistency in the process that occurred in the working organization.

2. Managed Level

This phase deals with the actual process of people management. In this stage, the managers play a crucial role in developing the workforce practice in the people.

They follow different practices like operating performance, staffing, adjusting the compensation to maintain the discipline and acquire the managed level. All these activities are performed repeatedly by the manager to be done, which could be beneficial for the organization.

The organization set certain goals, objectives that every person who works for the organization needs to meet and work with full efficiency. All these practices are done at the unit level, and the organization handles the performance and the skill set for developing this capability in the people.

All the process areas which are included in the managed level of the PCMM framework are training development, work environment, performance management, staffing, compensation, and communication coordination within the organization.

3. Defined Level

The defined level is the third phase of the PCMM framework, and this phase deals with the process named competency management. The aim of this stage is to develop the competency skills in the organization so that it can work well so that the organization can compete with other organizations for better business activities.

The organization has a role in maintaining a proper workforce so that it can achieve competency and do proper business activities within the organization.

The workforce competency includes the strategic workforce competency and improved workforce competency. In strategic workforce competency, it includes the activity which can benefit both present and future business activities.

The improved workforce competency is beneficial for the organization as it can be important for the betterment of the skill set and practices involved in the organisation's business activities. And in this level, the organization of the proper documentation defines the standards and integrates the process for achieving its business objectives.

4. Predictable Level

The predictable level is the fourth phase of the PCMM framework. At this level, the organization handles the capability, which is developed in an earlier phase so that it can achieve its working requirements.

At this level, the organization develops enough skill set to handle the performance of the business activity and manage the working capacity within the organization.

As the capability is developed, the organization can predict the capability and working capacity as the competency methods have been done in an earlier phase. The prediction of capacity will help the organization and help to better business activities within the organization.

5. Optimizing Level

The optimizing level is the last phase of the PCMM framework. In this stage, the whole organization is more focused on the continuous improvement of business activity in the organization.

Continuous improvement, it can benefit both the working groups and individuals and maintain the working efficiency within the organization. In this stage, the organization looks at the results of all other stages so that improvement can be done in that business activities, which can be beneficial for the organization.

3.6

MALCOLM BALDRIDGE

Q10. What is Malcom Balridge? Explain.

Ans :

The Malcolm Baldrige National Quality Award (MBNQA) is a national award in the United States that is given annually to businesses, education institutions, healthcare organizations, and nonprofit organizations that have demonstrated outstanding performance in the areas of leadership, strategic planning, customer focus, measurement, analysis and knowledge management, workforce focus, and operations focus.

The award was created by the US Congress in 1987 and is named after Malcolm Baldrige, who served as the US Secretary of Commerce from 1981 to 1987. The award is managed by the National Institute of Standards and Technology (NIST), an agency of the US Department of Commerce.

The MBNQA criteria are based on a set of seven categories, which are further divided into a total of 18 subcategories. The categories are:

1. **Leadership:** This category assesses how senior leaders guide the organization and how they create a vision for the future.
2. **Strategic Planning:** This category assesses how the organization sets strategic objectives and goals and how it aligns its resources to achieve them.
3. **Customer Focus:** This category assesses how the organization listens to and understands its customers' needs and expectations.
4. **Measurement, Analysis, and Knowledge Management:** This category assesses how the organization collects and analyzes data to support performance improvement and how it manages its knowledge assets.
5. **Workforce Focus:** This category assesses how the organization engages and develops its workforce and how it creates a positive work environment.
6. **Operations Focus:** This category assesses how the organization designs, manages, and improves its key operational processes.
7. **Results:** This category assesses the organization's performance results, including customer satisfaction, financial performance, and employee satisfaction.

The MBNQA criteria provide a framework for organizations to assess and improve their performance. Organizations can use the criteria to identify areas for improvement, set goals, and measure their progress over time. By applying the criteria, organizations can become more customer-focused, efficient, and effective, leading to increased competitiveness and long-term success.

3.7**3 SIGMA****Q11. Write about 3 sigma?**

Ans :

3 sigma is a statistical calculation that measures the predictability of outcomes by showing data that is three standard deviations from the mean. It's used in software engineering and other commercial applications to indicate that processes are running smoothly and producing high-quality products:

Quality standard

3 sigma yields a quality standard of 93.319%.

Control limits

3 sigma limits are used to set the upper and lower control limits in statistical quality control charts.

Error rate

For 3 sigma, the error rate is three parts per million, or 66,807 defective parts.

Performance

3 sigma is considered a good level of performance for most processes and products.

Deviation causes

3 sigma allows organizations to examine the causes of deviation and determine if they can be known or unknown.

While 3 sigma is an effective quality assurance method, some operations may require a higher level of accuracy. For example, in the medical sector, a process must attain at least Six Sigma level to be considered acceptable.

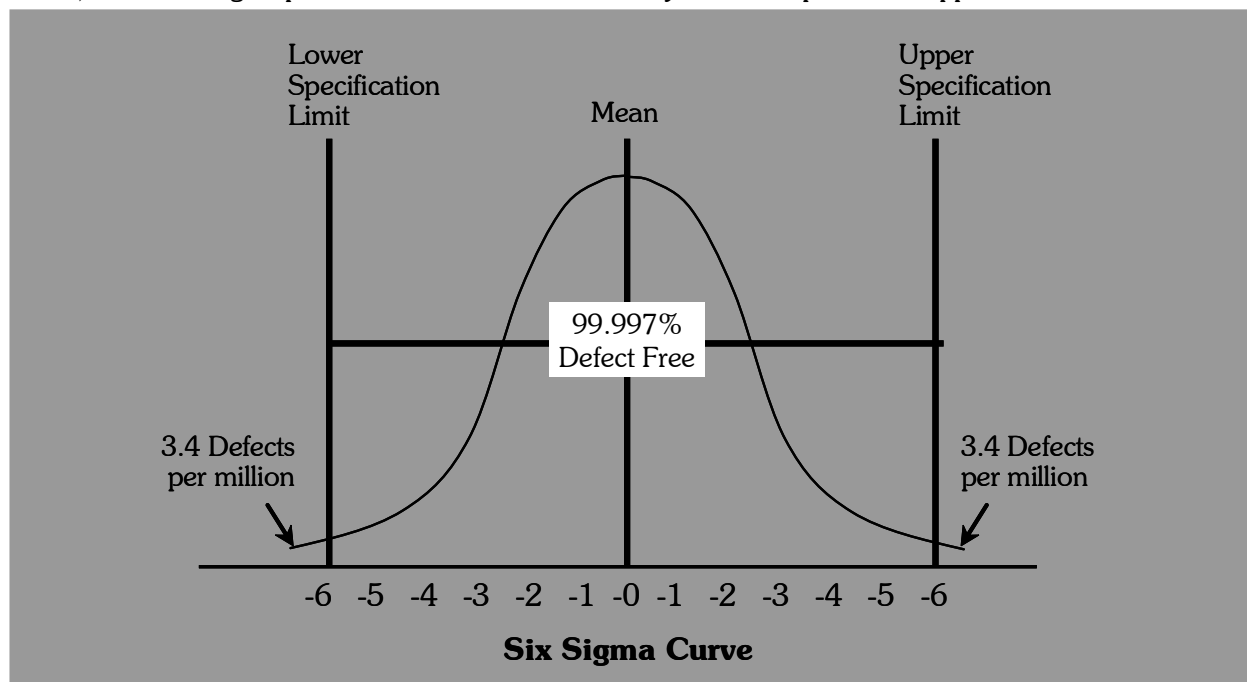
3.8

6 SIGMA

Q12. What is Six Sigma? Explain the Characteristics of Six Sigma.

Ans :

Six Sigma is a methodology used by most organizations for process improvement, and It is a statistical concept that aims to define the variation found in any process. Six Sigma is a process of producing high and improved quality output. This can be done in two phases – identification and elimination. The cause of defects is identified and appropriate elimination is done, which reduces variation in whole processes. Six Sigma processes have a failure rate of only 3.4 per million opportunities i.e. 99.99966 percent of Six Sigma products are free from defect, while Five Sigma processes have a failure rate of only 233 errors per million opportunities.

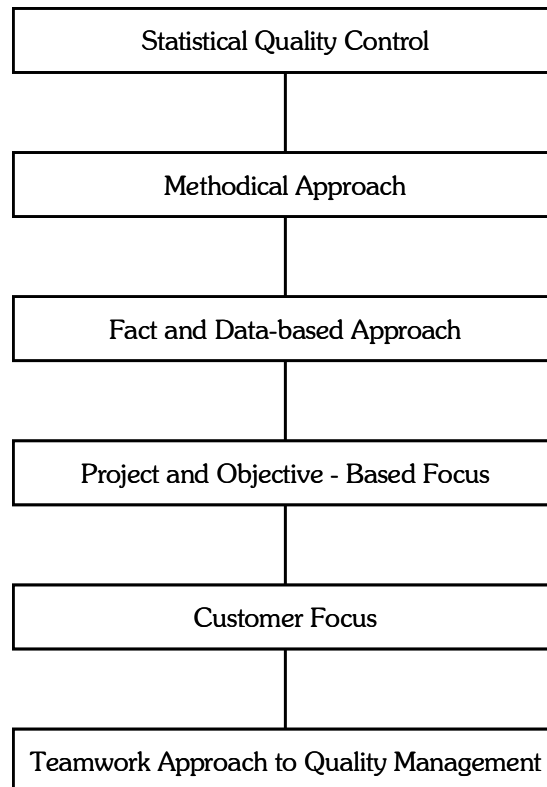


Characteristics of Six Sigma

The Characteristics of Six Sigma are as follows:

1. **Statistical Quality Control:** Six Sigma is derived from the Greek Letter σ which denote Standard Deviation in statistics. Standard Deviation is used for measuring the quality of output.
2. **Methodical Approach:** The Six Sigma is a systematic approach of application in DMAIC and DMADV which can be used to improve the quality of production. DMAIC means for Design-Measure- Analyze- Improve-Control. While DMADV stands for Design-Measure-Analyze-Design-Verify.
3. **Fact and Data-Based Approach:** The statistical and methodical method shows the scientific basis of the technique.
4. **Project and Objective-Based Focus:** The Six Sigma process is implemented to focus on the requirements and conditions.
5. **Customer Focus:** The customer focus is fundamental to the Six Sigma approach. The quality improvement and control standards are based on specific customer requirements.

6. **Teamwork Approach to Quality Management:** The Six Sigma process requires organizations to get organized for improving quality.



Characteristics of Six Sigma

Q13. Explain the 6 Key Principals of Six Sigma.

Ans :

(Imp.)

Organizations can enhance their sigma level by integrating Six Sigma principles into leadership, process management, and improvement efforts. Some Common Six Sigma Principals are:

1. Customer Centric Improvement

The primary principal of six sigma methodology is to focus on customer. Voice of the Customer (VoC) and methods for determining what the customer truly want from a product or process. Organizations can boost customer happiness by combining that knowledge with measurements, analytics, and process improvement approaches, resulting in higher profits, client retention, and loyalty.

2. Continuous Process Improvement

The Six Sigma approach requires constant process improvement. An organization that fully implements the Six Sigma technique never stops improving. It continuously discovers and priorities opportunities. Once one area has been improved, the organization will move on to another. The organization continuously find ways to increase the sigma level because the goal is to achieve the level of 99.99966 accuracy for all processes inside an organization while also making sure other essentials like financial stability.

3. Reduce Variation

A method to continuously improve a process is to reduce its variation. Every process has an inherent variation. Variation in processes can lead to errors, these errors can lead to product defect and product defect can lead to poor customer satisfaction. By reducing variation and errors six sigma can reduce process cost and increase customer satisfaction.

Suppose there are some developers developing web application, variation will exist as every developer has different coding styles, expertise levels, environment factors and project requirements. By adopting strategies like coding standards and guidelines, code reviews, automated testing and documentation variation can be reduced to some extent.

4. Eliminating Waste

Waste is a major problem in the six sigma methodology. Eliminating waste means removing items, procedures or people that are not required for the process's outcome or removing anything that does not add value to the customer. Eliminating waste can reduce processing time, errors in process and lower overall costs.

5. Empowering Employees

Until organizations provide employees with the tools they need to monitor and sustain improvements, implementing improved processes is only a temporary solution. Process improvement usually involves two approaches in most organizations. An improvement is first defined, planned, and carried out by a process improvement team consisting of project managers, methodology specialists, and subject matter experts. The employees that deal with the process on a daily basis are then equipped by that team to supervise and handle it in its improved condition.

6. Controlling the Process

Six Sigma improvements are frequently used to handle uncontrolled processes. Out-of-control processes meet certain statistical conditions. The purpose of improvement is to bring a process back under statistical control. Then, after the improvements are implemented, measurements, statistics, and other Six Sigma tools are utilized to keep the process under control. Implementing controls and training people on how to apply them is a key component of continuous improvement.

Q14. Explain various methodologies of six sigma

Ans :

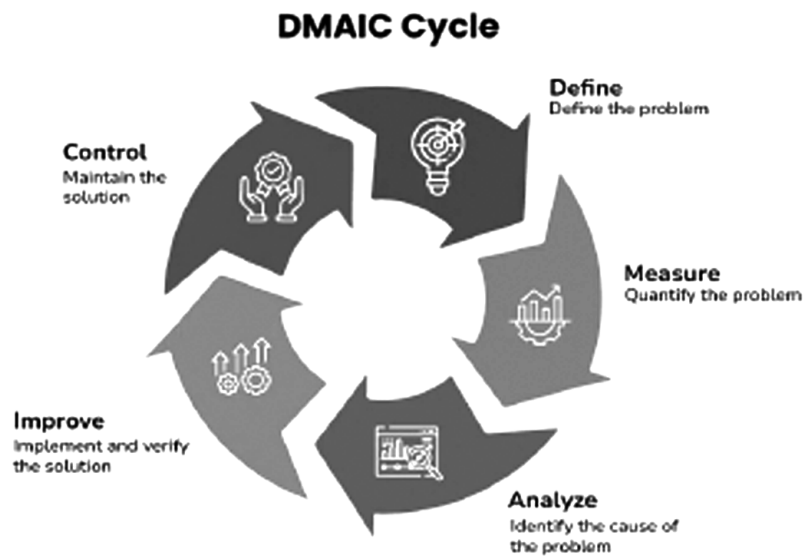
(Imp.)

The Six Sigma Methodology

The two Six Sigma methodologies used in the Six Sigma projects are DMAIC and DMADV. Six Sigma teams usually use DMAIC or DMADV approaches to achieve process improvements and establish process control.

DMAIC Six Sigma Methodology

DMAIC is used to enhance an existing business process. A DMAIC project involves identifying important problems that are creating the problem, verifying those problems, brainstorming solutions, implementing them, and designing a control plan to maintain the improved state. The DMAIC methodology is designed for the team who are responsible for improving a project. DMAIC phases allow flexibility, that helps the team to fit their activities. If a process needs to be completely replaced or redesigned for user experience in such a case, the team can use the DMADV method.



DMAIC Six Sigma Methodology

The DMAIC project methodology has five phases:

1. Define
2. Measure
3. Analyze
4. Improve
5. Control

Let's see the explanation of each phase:

1. Define

The Define phase of a DMAIC project involves identifying problems, establishing project requirements, and setting success goals. Six Sigma leaders can use tools inside the phase to create flexibility for different project types, depending on factors such as leadership advice and budgets.

2. Measure

During the DMAIC Measure phase, teams use data to validate assumptions about the process and problem. Validation of assumptions also makes it into the Analysis step. The measurement phase focuses on collecting and arranging data for analysis. Measuring in a Six Sigma project might be challenging without proper data collection. To gather data, teams may need to build tools, create queries, filter through large amounts of information, or use manual processes.

3. Analyze

Analyze phase is a critical stage where the root causes of problems or inefficiencies within a process are identified and understood. During the Analyze phase of a DMAIC project, teams develop predictions about relationships between inputs and outputs, use statistical analysis and data to validate the prediction and assumptions they've made thus far. In a DMAIC project, the Analyze phase leads to the Improve phase, where hypothesis testing can confirm assumptions and potential solutions.

4. Improve

During the Improve phase of a project, Six Sigma teams begin developing the concepts that came from the Analyze phase. They employ statistics and real-world observations to test assumptions and solutions.

As teams select and start implementing solutions, hypothesis testing keeps going throughout the enhance phase. It starts in the analyse phase.

5. Control

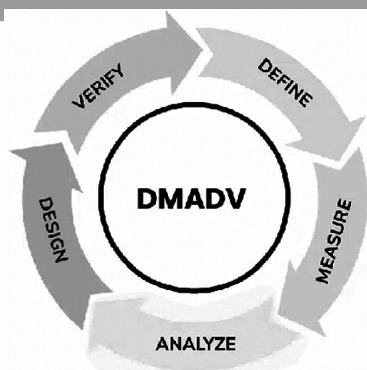
In DMAIC Phase Controls and standards are established so that improvements can be maintained, but the responsibility for those improvements is transitioned to the process owner.

DMADV Six Sigma Methodology

DMADV is used to create new product designs or process designs. Six Sigma teams use DMADV in the following scenario:

- The organization wants to launch a new service or product.
- Business leaders decide to replace a process to meet upgrade requirements or to align business processes, machinery, or workers with future goals.
- A Six Sigma team learns that upgrading a process is unlikely to result in the expected project outcomes.

DMADV Methodology



DMADV Six Sigma Methodology

The DMADV project methodology also has five phases:

1. Define
2. Measure
3. Analyze
4. Design
5. Verify

1. Define

In a DMADV project, the Define stage is slightly more strict. Teams must identify problems and define requirements within a change management environment. When an organization has a change management program in place, Six Sigma teams must include all program needs in the DMADV stages.

2. Measure

During the DMADV Measure phase, teams use data to validate assumptions about the process and problem. Validation of assumptions also makes it into the analysis step. The measurement phase focuses on collecting and arranging data for analysis.

3. Analyze

Analyze phase is a critical stage where the root causes of problems or inefficiencies within a process are identified and understood. They priorities identifying best practices and standards for measuring and designing new processes.

4. Design

The fourth phase is when DMADV projects start to vary significantly from DMAIC projects. The team designs a new process that includes solution testing, mapping, workflow principles, and infrastructure development.

5. Verify Phase

The Verify phase in DMADV checks if the designed solutions work as intended, measuring their success against initial goals, ensuring improvements are effective and sustainable.

The primary difference between DMAIC and DMADV in terms of team goals and project outcomes. Both methodology aims to deliver better quality, better efficiency, more production, more profits and provide excellent customer satisfaction.

Six Sigma Certification

A Six Sigma certification demonstrates practical knowledge and execution of the methodology. Some organizations provide internal certification methods. Most Six Sigma certifications are obtained through online or onsite training courses.

3.9**SQA PROJECT PROCESS STANDARDS****Q15. Explain about SQA project protocol standards.**

Ans :

(Imp.)

SQA project protocol standards are a set of guidelines and best practices that are used to ensure the quality of software development projects. These standards provide a framework 'for project management, development processes, and quality assurance activities.

The SQA project protocol standards cover various aspects of the software development life cycle, from planning and requirements analysis to design, development, testing, and maintenance. Some of the key areas covered by these standards include:

1. Project Planning

This includes defining project objectives, requirements, and scope, as well as identifying stakeholders, resources, and timelines.

2. Requirements Analysis

This involves eliciting, documenting, and verifying the functional and non-functional requirements of the software system.

3. Design and Development

This includes creating a detailed design specification, implementing the software code, and conducting peer reviews to ensure code quality.

4. Testing

This involves developing and executing test plans to ensure that the software meets the specified requirements and is free from defects.

5. Maintenance

This includes managing change requests, tracking defects and issues, and implementing updates and patches to the software.

Adhering to SQA project protocol standards can help ensure that software development projects are well-planned, well-managed, and well-executed. These standards can also help improve the quality of the software being developed, reduce the risk of project failure, and increase customer satisfaction. Ultimately, SQA project protocol standards can help organizations develop software that meets the needs of their users and delivers value to their business.

3.10**IEEE SOFTWARE ENGINEERING STANDARDS****Q16. Explain about IEEE Software engineering standards.**

Ans :

(Imp.)

IEEE (Institute of Electrical and Electronics Engineers) Software Engineering Standards are a set of guidelines and best practices that are used to ensure the quality of software development processes and products. These standards cover various aspects of the software development life cycle, from requirements analysis to testing and maintenance.

Some of the key IEEE Software Engineering Standards include:

1. **IEEE 830-1998:** This standard defines the format and content of software requirements specifications.
2. **IEEE 1012-2012:** This standard provides guidelines for software verification and validation processes.
3. **IEEE 12207-2008:** This standard provides a framework for software life cycle processes, including planning, implementation, and maintenance.
4. **IEEE 1471-2000:** This standard provides a framework for describing the architecture of a software system.
5. **IEEE 1-5288-2015:** This standard provides a framework for system and software engineering life cycle processes.

Adhering to IEEE Software Engineering Standards can help ensure that software development projects are well-planned, well-managed, and well-executed.

These standards can also help improve the quality of the software being developed, reduce the risk of project failure, and increase customer satisfaction. Ultimately, IEEE Software Engineering Standards can help organizations develop software that meets the needs of their users and delivers value to their business.

UNIT IV

Building a Software Testing Strategy, Establishing a Software Testing Methodology, Determining Your Software Testing Techniques, Eleven – Step Software Testing Process Overview, Assess Project Management Development Estimate and Status, Develop Test Plan, Requirements Phase Testing, Design Phase Testing, Program Phase Testing, Execute Test and Record Results, Acceptance Test, Report Test Results, Test Software Changes, Evaluate Test Effectiveness.

4.1

BUILDING A SOFTWARE TESTING STRATEGY

Q1. What is a Test Strategy? Explain the benefits of test strategy.

Ans :

Meaning of Test Strategy

A test strategy outlines the approach, goals, and standards for testing activities within a project. It provides a structured framework for defining testing scope, objectives, methodologies, and tools, ensuring consistency and alignment with project requirements.

Benefits

Following are the benefits of test strategy:

i) Clarity and Consistency

A test strategy provides a clear structure and standardized approach for testing activities, ensuring that every team member understands the objectives, scope, and expectations. This clarity reduces confusion, aligns team efforts, and maintains consistency across all testing phases, from initial planning to final execution.

ii) Risk Mitigation

A well-defined test strategy helps prevent issues that could compromise the quality or delay the release of the software by identifying potential risks and setting up contingency plans early. Proactively addressing high-risk areas enables the team to focus on critical functionalities, thus minimizing the impact of potential setbacks on the project.

iii) Resource Efficiency

A test strategy allocates resources such as tools, personnel, and time more effectively. The strategy helps teams prioritize testing activities based on project needs, avoiding redundant efforts and optimizing team capacity. Efficient resource management helps cost savings and prevents unnecessary delays in the testing lifecycle.

iv) Stakeholder Alignment

A test strategy is a communication tool for aligning the testing approach with stakeholder expectations. Defining the goals, methods, and scope of testing keeps project managers, developers, and stakeholders informed, ensuring everyone is on the same page about testing objectives and outcomes.

v) Enhanced Quality Assurance

A strategic approach to testing ensures comprehensive coverage of functionalities, reducing the likelihood of undetected issues in production. This structured methodology enhances the overall quality of the software by focusing on thorough validation and verification that is aligned with user requirements.

Q2. What is to Include in a Test Strategy Document? Explain.

Ans :

(Imp.)

A comprehensive test strategy document guides the testing process and ensures alignment among team members and stakeholders. Key elements to include are:

Scope of Testing

- Define the areas of the application that will undergo testing, outlining both included and excluded features.
- Specify the testing boundaries, including any limitations affecting testing, like time constraints or restricted access to certain system parts.

Testing Objectives

- Clearly state the goals for the testing phase, such as ensuring compatibility, performance, functionality, or security.
- Include specific success criteria or benchmarks to indicate the objectives met.

Testing Approach and Methodology

- Describe the approach for different types of testing (e.g., functional, performance, regression, security) and how each type aligns with the objectives.
- Outline whether tests will be manual, automated, or combined, and provide reasoning for the selected methods.
- Include details on the testing phases (unit, integration, system, and acceptance testing) and how they will be executed.

Test Environment Setup

- Specify the environments where testing will occur (e.g., development, staging, production-like) and detail any setup requirements.
- List hardware and software configurations, operating systems, network specifications, and any dependencies required for consistent test results.

Resource Allocation and Roles

- Identify key roles and responsibilities, assigning specific tasks to team members involved in testing.
- Include any additional resources like third-party vendors, consultants, or external testing teams.

Q3. Explain the Software Testing Strategy Document.*Ans :*

A software testing strategy document is a comprehensive plan that outlines the testing approach, goals, scope, resources, and timelines for a software project. Acting as a reference for all stakeholders involved in testing, this document ensures a unified understanding of the overall testing strategy. It clarifies the testing scope, types of testing to be conducted, test deliverables, and criteria for test completion.

Types of Software Testing Strategies

An appropriate test strategy in software testing is vital for effective quality assurance. Here are some common types:

1. **Analytical Approach:** Analyze requirements, risks, and critical functionalities to determine the testing scope and prioritize test activities accordingly.

2. **Model-Based Approach:** Utilize modeling techniques to create test models that represent system behavior, interactions, and expected outcomes, facilitating strategic test planning and execution.
3. **Regression-Based Approach:** Focus on ensuring that modifications or enhancements to the software do not introduce new defects or impact existing functionalities. Regression test strategy in software testing validates system stability after changes.
4. **Risk-Based Approach:** Direct efforts to the test strategy in software testing to areas of the software deemed high-risk based on factors like impact, probability, and criticality. This approach optimizes resource allocation and prioritization of testing activities.
5. **Agile Testing Strategy:** This approach aligns with the Agile methodology, emphasizing iterative and incremental testing throughout development. It involves close collaboration between testers, developers, and stakeholders, enabling continuous feedback and frequent testing cycles.
6. **Exploratory Testing Strategy:** Exploratory test strategy in software testing is an approach where testers actively explore the application under test, focusing on learning, investigation, and discovery. This strategy involves less pre-planning and more real-time test design, allowing flexibility and adaptability to uncover unexpected defects.
7. **User-Centric Testing Strategy:** This strategy places the end-user at the center of the testing process. Testers simulate real-world user scenarios and interactions to validate the application's usability, accessibility, and overall user experience. Usability testing, user acceptance testing (UAT), and beta testing are standard techniques employed in this strategy.
8. **Compliance-Based Testing Strategy:** Compliance-based test strategy in software testing are crucial in the industries such as healthcare or finance. These strategies ensure that the software meets regulatory standards, legal requirements, and industry-specific guidelines. Compliance testing often involves validating data privacy, security measures, and adherence to industry-specific regulations.

9. **Continuous Testing Strategy:** Continuous test strategy in software testing is a DevOps and Agile-oriented approach that emphasizes integrating testing throughout the software development lifecycle. It involves running automated tests continuously, incorporating them into the CI/CD pipeline, and providing quick feedback on software quality. Continuous testing enables faster releases, early bug detection, and improved collaboration between development and testing teams.
10. **Crowdtesting Strategy:** Crowdtesting involves harnessing a community of external testers who test the software across various devices, platforms, and real-world conditions. This strategy leverages testers' diversity and unique perspectives to uncover defects that may go unnoticed with traditional in-house testing teams. Crowdtesting can provide valuable insights from a broader user perspective.

Q4. Explain briefly about building a software testing strategy?

Ans :

Building a software testing strategy is essential for ensuring the quality of software products. A testing strategy outlines the testing approach, test methodologies, and test processes that 'will be used to evaluate software quality. Here are some steps to consider when building a software testing strategy:

1. **Define testing objectives:** The first step in building a testing strategy is to define the objectives of the testing effort. The objectives should be specific and measurable and should be aligned with the overall goals of the software project.
2. **Identify testing scope:** Once the objectives are defined, you need to identify the scope of the testing effort. This can include the types of testing to be performed, the testing environments to be used, and the testing resources needed.
3. **Determine testing methodologies:** The next step is to determine the testing methodologies to be used. This can include automated testing, manual testing, exploratory testing, regression testing, and performance testing.
4. **Define test processes:** Once the testing methodologies are identified, you need to define the test processes that will be used. This can include test planning, test design, test execution, and test reporting.
5. **Define testing metrics:** Testing metrics are used to measure the effectiveness of the testing effort. Metrics can include defect density, test coverage, test case effectiveness, and test cycle time.
6. **Define testing tools:** Testing tools can help streamline the testing process and improve the accuracy of test results. Tools can include test management software, automated testing tools, defect tracking systems, and performance testing tools.
7. **Develop a testing schedule:** A testing schedule outlines the timeline for testing activities and ensures that testing is completed within the project timeline.
8. **Obtain stakeholder buy-in:** Once the testing strategy is developed, it's important to obtain stakeholder buy-in to ensure that everyone understands the testing approach and supports the testing effort.
9. **Continuously evaluate and improve:** Finally, it's essential to continuously evaluate and improve the testing strategy. This can include gathering feedback from stakeholders, reviewing testing metrics, and making changes to the testing approach as needed.

Q5. Differentiate between test strategy and test plan.

Ans :

(Imp)

In software testing, Test Strategy and Test Plan are often used interchangeably, but they serve different purposes within a project.

- **Test Strategy:** This high-level document defines the approach and goals of testing for an organization or a product. It focuses on the big picture, addressing questions such as what types of testing will be conducted, the testing objectives, and how quality will be ensured across the development lifecycle. The test strategy is usually a static document that does not change frequently.
- **Test Plan:** A test plan, on the other hand, is more detailed and project-specific. It outlines the specific testing activities for a project or feature, including the scope, schedule, resources, environment, and tasks required. A test plan is more dynamic and may evolve as the project progresses.

S.No.	Nature	Test Strategy	Test Plan
1.	Purpose	Defines a high-level testing approach	Details specific testing activities, scope, and schedule
2.	Scope	Broad and organizational or product-level	Narrow, specific to a project or feature
3.	Change Frequency	Static and rarely updated	Dynamic and may evolve as the project changes
4.	Content	Types of testing, risk management approach, tools, quality goals	Specific test cases, timelines, resources, environments, and responsibilities
5.	Ownership	Generally crafted by QA leads or test architects	Created by the test manager or QA team specific to a project

Q6. Explain how a test strategy document enhances project outcomes ?

Ans :

(Imp.)

Creating a test strategy document offers several benefits for software development projects. Consider the following advantages:

- i) **Clear Direction:** The test strategy document provides a clear direction and roadmap for the testing effort, ensuring all team members are aligned and working towards common objectives.
- ii) **Efficient Resource Utilization:** By outlining the required resources, such as personnel, tools, and environments, the test strategy document helps allocate resources effectively, optimizing the testing process.
- iii) **Risk Mitigation:** The document identifies potential risks and mitigation strategies, enabling proactive risk management and reducing the impact of potential issues on the project.
- iv) **Test Coverage and Quality:** A well-defined test strategy document ensures comprehensive test coverage, addressing the software's functional and non-functional aspects. This leads to higher software quality and reduces the likelihood of critical issues escaping production.
- v) **Enhanced Communication:** The test strategy document serves as a communication tool, providing stakeholders with a common understanding of the testing approach, timelines, and expectations. It promotes effective collaboration and transparency among team members.

Q7. What are the key components of a software testing strategy document template.

Ans :

To create an effective software testing strategy document, consider incorporating the following components:

- i) **Introduction:** Provide an overview of the document, including its purpose, intended audience, and project background.

- ii) **Test Objectives:** Clearly define the goals and objectives of the testing effort, aligning them with the project's overall objectives.
- iii) **Testing Scope:** Specify the testing boundaries, including the features, functionalities, platforms, and environments to be covered.
- iv) **Test Approach:** Outline the high-level strategy and techniques to be employed during testing, such as manual testing, automated testing, or both.
- v) **Test Deliverables:** Enumerate the artifacts and documentation to be produced during the testing process, such as test plans, test cases, and test reports.
- vi) **Test Schedule and Timeline:** Provide a timeline for testing activities, including milestones, dependencies, and estimated effort for each testing phase.
- vii) **Resource Allocation:** Specify the resources required for testing, encompassing personnel, hardware, software, and tools.
- viii) **Risk Assessment:** Identify potential risks and mitigation strategies associated with testing, ensuring proactive risk management.
- ix) **Test Environment:** Describe the necessary test environment setup, including hardware, software, network configurations, and data requirements.
- x) **Exit Criteria:** These are the predefined conditions that must be met to conclude the testing process, such as a specific defect density threshold or completion of a predefined set of test cases.

Q8. What are the steps to develop a test strategy document.

Ans :

The Steps to Develop a Software Testing Strategy Document Now that we understand the significance of a test strategy document, let's explore the step-by-step process of creating one:

Step 1: Define the Testing Goals and Objectives

- Clearly define the goals as well as the objectives of the testing effort.
- Understand the project requirements, business goals, and user expectations to align testing activities accordingly.

Step 2: Identify the Testing Scope

- Define the testing boundaries, including the modules or functionalities to be tested, the test levels, and the types of testing required.
- Determine any exclusions or limitations in the testing scope.

Step 3: Determine the Testing Approach

- Choose the appropriate testing approaches and techniques based on the project requirements and objectives.
- Consider risk, complexity, and the software development lifecycle model being used.

Step 4: Specify the Test Environment and Infrastructure

- Identify the hardware, software, and network configurations needed for testing.
- Determine if any specialized tools or resources are required and plan their procurement or setup.

Step 5: Define the Test Data Management Strategy

- Establish guidelines for test data creation, identification of data dependencies, and data management practices.
- Ensure data security, privacy, and compliance with relevant regulations.

Step 6: Outline the Test Execution and Reporting Process

- Detail the test execution methodology, including creating test cases, test execution cycles, and defect tracking.
- Specify the reporting formats, metrics, and KPIs to measure the testing progress and quality.

Step 7: Develop the Test Automation Strategy Document

- Identify the areas suitable for test automation and outline the implementation strategy.
- Choose appropriate tools, define the framework, and allocate resources for test automation.

Step 8: Address Risks and Mitigation Strategies

- Identify potential risks and challenges in the testing process.
- Devise mitigation strategies to minimize the impact of risks on project timelines and deliverables.

Step 9: Define Test Deliverables and Exit Criteria

- Specify the expected test deliverables, including test plans, cases, reports, and other artifacts.
- Establish the criteria for deciding when testing can be considered complete.

4.2**ESTABLISHING A SOFTWARE TESTING METHODOLOGY****Q9. What are Testing Methodologies? Explain.***Ans :***(Imp.)**

Testing methodologies are the strategies and approaches used to evaluate a particular product to ensure it performs as expected and is easy to use. Testing methodologies usually involve testing that the product works in accordance with its specification, has no undesirable side effects when used in ways outside of its design parameters, and will fail safely in the worst-case scenario.

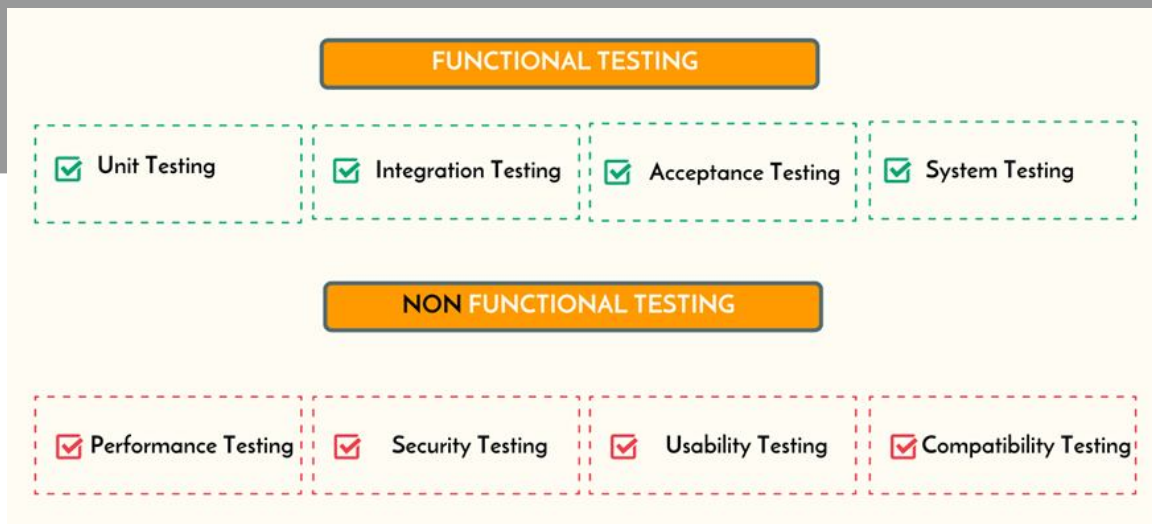
Software Testing Methodologies

Software testing methodologies, then, are the different tactics to ensure that a software application is fully dependable, secure, and vetted. They encompass everything from unit testing individual modules, integration testing an entire system, or specialized forms of testing such as security and performance.

Importance of Software Testing Methodologies

As software applications get ever more complex (accelerated even more by AI innovations in recent years) and intertwined with the large number of different platforms and devices required to test, it is more important than ever to have a robust and efficient testing methodology. This can ensure that software products and systems being developed have been fully assessed, meet their specified requirements, and can successfully operate in all the anticipated environments (with the required usability and security of course).

Without the proper development and testing methodologies for modern software, projects will inevitably go over budget, take longer than necessary, and not meet stakeholder expectations.

Functional & Non-Functional Testing

When it comes to software testing, there are two primary categories that sub-processes of every testing methodology fall into. These buckets are functional and non-functional tests:

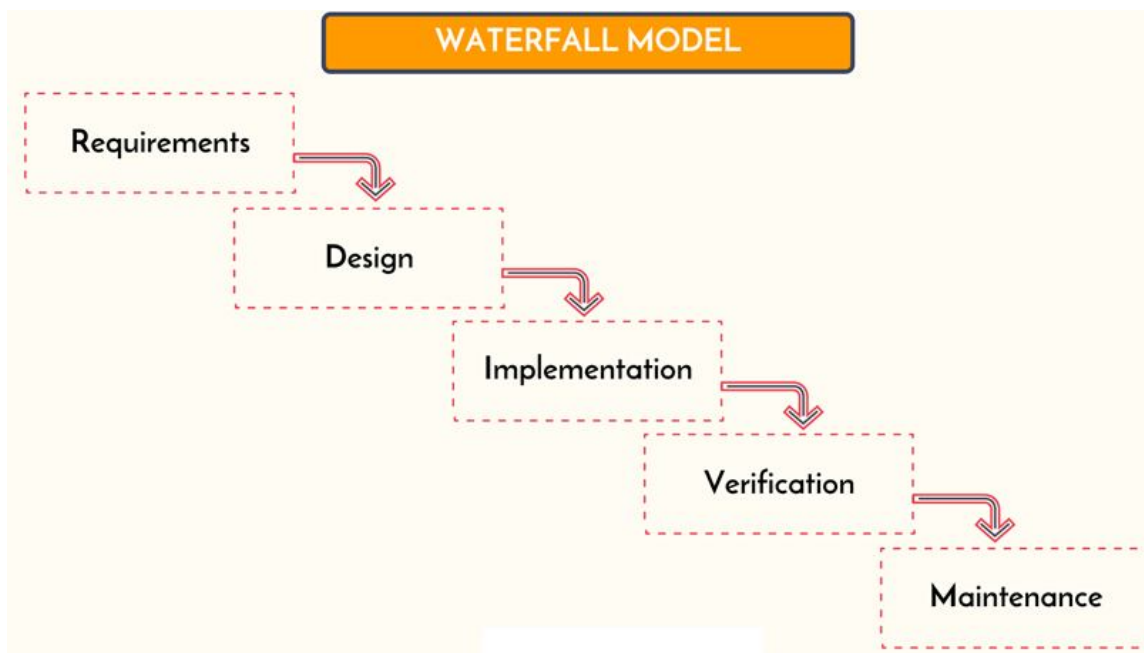
- **Functional Testing:** Typically broken down into four components (unit testing, integration testing, system testing, and acceptance testing), this verifies that the functions and features of the software work as intended.

- **Non-Functional Testing:** On the other hand, non-functional testing involves testing basically everything else like performance and customer expectations (also known as the ‘-ilities’ because they all end in ‘-ility’ — i.e. vulnerability, scalability, usability).

Testing Methodologies & Models

Now that we have a foundation for why choosing the right testing methodology for your project is so critical and the broad categories they can fall under, let’s discuss each major model in more detail:

Waterfall Methodology

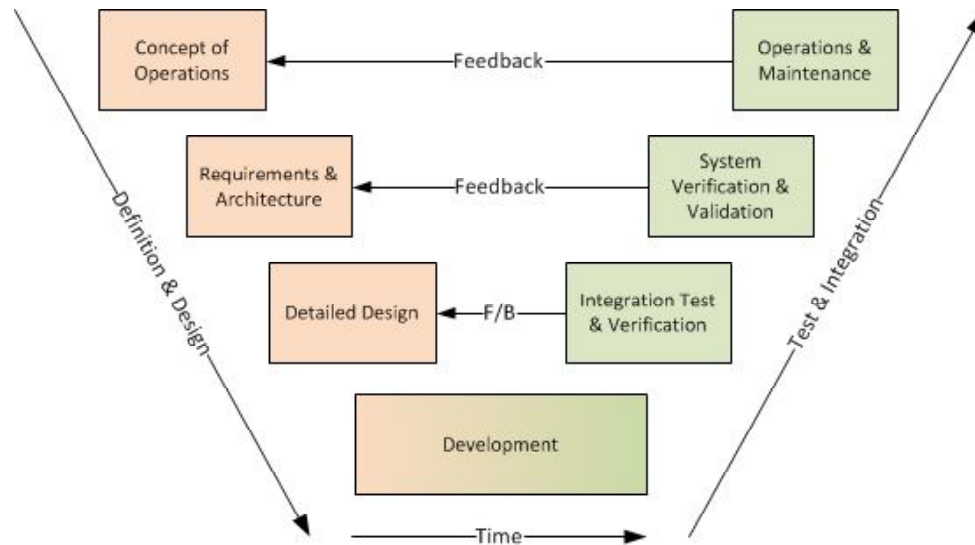


The most rigid of the methodologies we’ll discuss today, waterfall development and testing uses sequential steps to plan, execute, and maintain software. The next step can only be started once its predecessor has been completed, with the five predefined stages being:

- Requirements
- Design
- Implementation
- Verification
- Maintenance

Beginning with the first step, requirements like testing objectives, draft documents, testing strategy, and all other pieces are gathered and defined. A design is then selected and approved by someone tasked with making final decisions. The development team will then implement the design plan that was selected. After this, QA testing and stakeholders will verify the application’s functionality and performance, followed by continual maintenance post-launch.

Because of its rigid documentation and planning, this methodology is best suited for small applications rather than larger, more complex projects that might undergo extensive or frequent changes.

V-Model

The Verification and Validation Method is also known as the V-Model due to the shape of the diagram it's typically depicted by. Considered an extension or variant of the waterfall model, it follows a V-shape that's broken into two sections (verification and validation), or the "legs" of the V.

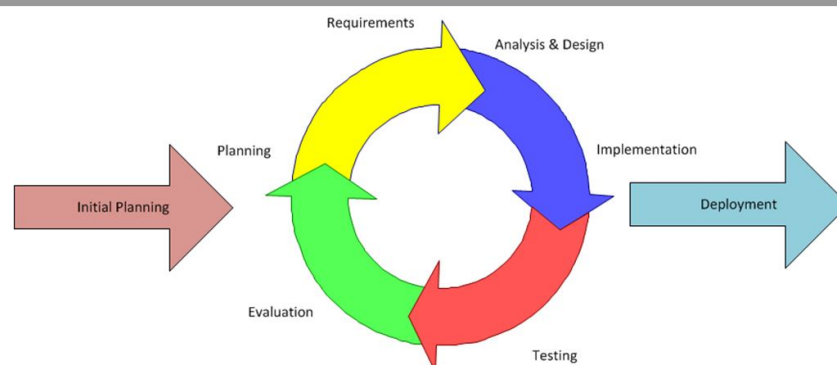
In the first phase, the model starts with a static verification process that covers analysis and planning on business requirements, system design, architecture design, and module design. From there, these pieces are used in the coding phase to develop the application. Once the coding phase is complete, the validation phase evaluates and tests the pieces from the verification phase.

Benefits

- Errors can be caught early in development
- Rigidity is effective for smaller projects and applications

Drawbacks

- While errors can be identified early on, there isn't a defined way to resolve them
- Not suitable for larger, more complex projects
- Can't overlap steps (as with waterfall development)
- Once a module has entered the testing phase, you can't go back

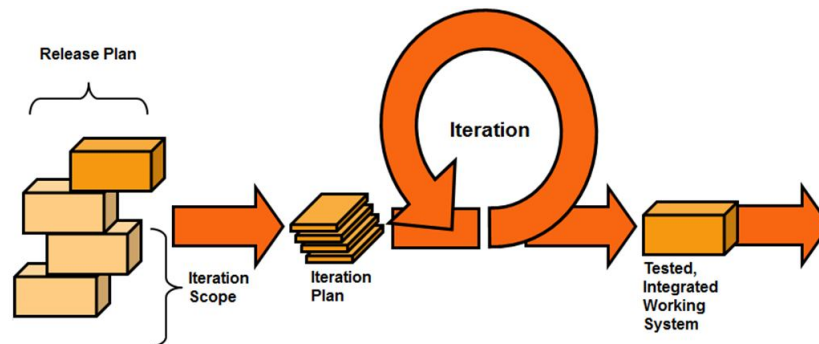
Iterative Methodology

Unlike the waterfall methods, iterative development centers around breaking down a project into its smaller components, where each is then iterated and tested before merging into a final product. The cycles for each component mimic a full development cycle, from planning and design to testing and deployment.

Iterative development is very data-driven and uses the results of each test cycle for the next iteration created. It's ideal for flexible applications where requirements are loosely defined and scalability is an important factor.

Agile Testing Methodology

Based on the idea of iterative development, agile methodologies (such as Scrum or Kanban) use rapid, incremental cycles, referred to as sprints. Rather than the only completed work being delivered at the end of the process like iterative development, agile delivers these incremental deliverables throughout the process.



Additionally, agile allows stages to overlap and includes frequent communication with stakeholders to continually refine requirements and other important factors. Each cycle (or sprint) is tested thoroughly to create a deliverable that can be presented to stakeholders and adapted with feedback.

Q10. Explain, how to establish software testing methodologies

Ans :

Establishing software testing methodologies involves creating a structured framework to ensure that a software application meets its requirements, is free of defects, and functions as intended.

Here's a step-by-step guide to setting up software testing methodologies:

1. Understand Project Requirements

- Collaborate with stakeholders to gather and analyze functional and non-functional requirements.
- Create use cases, user stories, and acceptance criteria for clarity on the expected behavior of the software.

2. Define Testing Objectives

- **Purpose:** Identify what testing should achieve (e.g., defect identification, performance validation, compliance testing).
- **Scope:** Determine the aspects of the application to be tested (e.g., functionality, security, usability).

3. Choose a Testing Methodology

There are several methodologies to consider based on project needs:

- **Waterfall Testing:** Sequential testing aligned with the development lifecycle.
- **Agile Testing:** Continuous testing integrated into Agile development sprints.
- **DevOps/CI-CD Testing:** Automated and continuous testing within CI/CD pipelines.
- **V-Model Testing:** Verification and validation occur in parallel with development.

4. Establish Testing Types

Define and incorporate various types of testing:

- **Unit Testing:** Verifies individual components or modules.
- **Integration Testing:** Tests interactions between integrated modules.
- **System Testing:** Examines the entire application for functionality and performance.
- **Acceptance Testing:** Validates the software against user requirements.
- **Regression Testing:** Ensures new changes do not break existing functionality.
- **Non-Functional Testing:** Includes performance, security, usability, and scalability testing.

5. Create a Testing Strategy

Develop a strategy that outlines:

- **Tools:** Selection of tools for manual and automated testing (e.g., Selenium, JIRA, TestNG).
- **Environment:** Define test environments and configurations.
- **Schedule:** Align testing phases with the development timeline.
- **Metrics:** Establish key performance indicators (KPIs) like defect density, test coverage, and pass rates.

6. Test Plan Development

Draft a detailed test plan that includes:

- Objectives, scope, and assumptions.
- Testing deliverables (test cases, test scripts, test reports).
- Responsibilities of team members.
- Risks and mitigation strategies.

7. Test Case Design

- Develop detailed test cases covering all scenarios (positive, negative, edge cases).
- Use techniques like boundary value analysis, equivalence partitioning, and decision tables.

8. Implement Automation

- Identify repetitive and critical test cases for automation.
- Select an automation framework (e.g., Data-Driven, Keyword-Driven, or Hybrid).
- Develop and execute automated test scripts.

9. Execute Tests

- Run tests in defined environments.
- Log and track defects using bug-tracking tools
- Retest fixed defects and conduct regression testing.

10. Monitor and Evaluate

- Collect and analyze test metrics.
- Identify patterns and areas for improvement in testing processes.

11. Continuous Feedback and Improvement

- Conduct retrospectives to assess testing methodologies.
- Incorporate feedback from stakeholders and testers.
- Update methodologies to adapt to evolving project requirements or technologies.

12. Documentation

Maintain comprehensive documentation for:

- Test cases and scripts.
- Test execution logs and reports.
- Defect tracking and resolutions.

4.3**DETERMINING YOUR SOFTWARE TESTING TECHNIQUES**

Q11. What are Software Testing Techniques? Explain.

Ans:

(Imp.)

Software testing techniques are methods to check if a software program works properly, meets its goals, and assesses the quality of software.

These techniques include different methods, like manual testing, where testers check the software, and automa he software's internal design, back-end architecture, components, and business/technical requirements.

The internet of black-box testing is to find performance errors/deficiencies, entire functions missing (if any), initialization bugs, and glitches that may show up when accessing any external database.

i) White Box Testing

In White Box Testing, testers verify systems they are deeply acquainted with, sometimes even ones they have created themselves. No wonder white box testing has alternate names like open box testing, clear box testing, and transparent box testing.

White box testing is used to analyze systems, especially when running unit, integration, and system tests.

ii) Functional Testing

Functional tests are designed and run to verify every function of a website or app. It checks that each function works in line with expectations set out in corresponding requirements documentation.

Example

Tests are created to check the following scenario – when a user clicks “Buy Now”, does the UI take them directly to the next required page?

There are multiple sub-sets of function testing, the most prominent of which are:

iii) Unit Testing

Each individual component is tested before the developer pushes it for merging. Unit tests are created and run by the devs themselves.

iv) Integration Testing

Units are integrated and tested to check that they work together seamlessly.

v) System Testing

All systems elements, hardware and software, are tested for overall functioning to check that it works according to the system’s specific requirements. Regression Testing is a type of System Testing that is performed before every release.

vi) Acceptance Testing

Often called User Acceptance Testing, this subset of testing puts the software in the hands of a control group of potential users, and note their feedback. It’s the app’s first test in a truly real-world scenario.

vii) Non Functional Testing

It’s in the name. Non functional tests check the non functional attributes of any software – performance, usability, reliability, security, quality, responsiveness, etc. These tests establish software quality and performance in real user conditions.

Example

Tests are created to simulate high user traffic so as to check if a site or app can handle peak traffic hours/days/occasions.

Q12. Explain how to determining your software testing techniques.

Ans :

Determining the appropriate software testing techniques in software engineering depends on several factors, including the project requirements, system complexity, timelines, and risk assessment. Here’s a structured approach to selecting the right testing techniques:

1. Analyze Requirements

- **Understand Objectives:** Clarify what needs to be tested (e.g., functionality, performance, security).
- **Prioritize Features:** Identify high-priority and high-risk areas that require thorough testing.

2. Assess System Characteristics

- **Complexity:** Highly complex systems may require advanced techniques like model-based testing.
- **Integration:** Systems with multiple interacting components benefit from integration testing techniques.
- **Criticality:** High-stakes systems (e.g., medical software) require exhaustive techniques like formal verification and stress testing.

3. Identify Test Levels

- **Unit Testing:** For individual components or modules, use techniques like white-box testing (statement and branch coverage).
- **Integration Testing:** Test interactions between modules using incremental (top-down, bottom-up) or big-bang approaches.

<ul style="list-style-type: none"> ➤ System Testing: Ensure the entire system works as intended, combining functional and non-functional testing techniques. ➤ Acceptance Testing: Use user-centered techniques like exploratory testing or scenario-based testing. <p>4. Match Testing Types with Goals</p> <ul style="list-style-type: none"> ➤ Functional Testing: Focus on correctness by testing features against requirements. ➤ Techniques: Equivalence partitioning, boundary value analysis, decision table testing. 	<p>DevOps/CI-CD</p> <ul style="list-style-type: none"> ➤ Techniques: Automated testing, API testing, and end-to-end testing within CI/CD pipelines. <p>6. Use Risk-Based Testing</p> <ul style="list-style-type: none"> ➤ Identify areas with the highest business or operational risk. ➤ Apply techniques like fault-tree analysis, error guessing, and risk-based prioritization. <p>7. Leverage Automation</p> <ul style="list-style-type: none"> ➤ Identify repetitive, high-volume, or critical test cases.
<ul style="list-style-type: none"> ➤ Performance Testing: Evaluate system performance under load. ➤ Techniques: Load testing, stress testing, and soak testing. ➤ Security Testing: Ensure the system is protected against threats. ➤ Techniques: Penetration testing, vulnerability scanning. ➤ Usability Testing: Check for user-friendliness. ➤ Techniques: A/B testing, heuristic evaluation. ➤ Regression Testing: Ensure changes don't break existing functionality. ➤ Techniques: Test suite automation, selective retesting. 	<ul style="list-style-type: none"> ➤ Use automation techniques such as: ➤ Data-driven testing for multiple input scenarios. ➤ Keyword-driven testing for modularity. ➤ Framework-based automation for scalability. <p>8. Evaluate Constraints</p> <ul style="list-style-type: none"> ➤ Time: Limited time may favor exploratory testing or risk-based prioritization. ➤ Budget: Budget constraints may limit automation or exhaustive testing. ➤ Resources: The skill level of testers may influence technique selection. <p>9. Combine Techniques</p> <p>For comprehensive coverage, employ a mix of techniques:</p>
<ul style="list-style-type: none"> ➤ Exploratory Testing: Find defects using tester intuition in an unstructured manner. <p>5. Choose Based on Development Methodology</p> <p>Agile Development</p> <ul style="list-style-type: none"> ➤ Techniques: Continuous testing, behavior-driven development (BDD), test-driven development (TDD). <p>Waterfall Development</p> <ul style="list-style-type: none"> ➤ Techniques: Sequential testing methods like requirements-based testing and traceability matrix. 	<ul style="list-style-type: none"> ➤ Functional techniques (e.g., equivalence partitioning) for functionality. ➤ Structural techniques (e.g., code coverage) for code-level testing. ➤ Error-based techniques (e.g., error guessing) for fault-prone areas. <p>10. Monitor and Adjust</p> <ul style="list-style-type: none"> ➤ Track test effectiveness and defect detection rates. ➤ Optimize techniques based on feedback and test outcomes.

4.4

**ELEVEN STEP SOFTWARE TESTING
PROCESS OVERVIEW**

Q13. Explain to briefly about Eleven step software testing process.

Ans :

(Imp.)

The Eleven-Step Software Testing Process provides a structured approach to ensure systematic and thorough testing of software applications. Each step is critical in identifying, addressing, and preventing defects. Here's an overview of the process:

1. Requirement Analysis

Objective: Understand and analyze requirements to determine testable features.

Activities

- Collaborate with stakeholders to gather functional and non-functional requirements.
- Identify ambiguities or unclear specifications.
- Create a requirements traceability matrix (RTM).

2. Test Planning

Objective: Develop a comprehensive plan outlining the testing strategy, scope, and schedule.

Activities

- Define testing objectives, scope, and deliverables.
- Allocate resources, roles, and responsibilities.
- Select tools and technologies for testing.
- Identify risks and create mitigation strategies.

3. Test Design

Objective: Design test cases and scenarios based on the requirements.

Activities

- Use techniques like boundary value analysis, equivalence partitioning, and decision tables.
- Create test data and define expected results.
- Prioritize test cases based on risk and criticality.

4. Test Environment Setup

Objective: Prepare the environment where testing will be conducted.

Activities

- Configure hardware, software, and network settings.
- Ensure access to required tools, test data, and servers.
- Validate the environment with a sanity check.

5. Test Case Development

Objective: Develop detailed and executable test cases and scripts.

Activities

- Write test cases for all identified scenarios.
- Develop automated test scripts if applicable.
- Review test cases for coverage and correctness.

6. Test Data Preparation

Objective: Create or acquire the data needed for executing test cases.

Activities

- Generate test data for positive, negative, and edge cases.
- Anonymize or mask production data if used.
- Ensure the data aligns with test scenarios.

7. Test Execution

Objective: Execute test cases and record the results.

Activities

- Run manual and automated test cases.
- Log defects in a tracking system for failed tests.
- Retest fixed defects and conduct regression testing.

8. Defect Reporting and Tracking

Objective: Log, prioritize, and monitor defects until resolution.

Activities

- Document defects with details (e.g., steps to reproduce, environment, severity).

- Collaborate with developers for defect resolution.
- Retest after fixes and verify the closure of defects.

9. Test Result Analysis

Objective: Evaluate test outcomes to assess software quality.

Activities

- Analyze pass/fail rates and defect trends.
- Measure metrics like test coverage, defect density, and time-to-fix.
- Identify areas needing additional testing.

10. Test Closure

Objective: Finalize and document the testing process.

Activities

- Prepare a test summary report with findings and recommendations.
- Archive test cases, data, and scripts for future use.
- Conduct a retrospective to gather lessons learned.

11. Continuous Improvement

Objective: Enhance the testing process for future projects.

Activities

- Analyze feedback from stakeholders and testers.
- Identify inefficiencies and optimize processes.
- Update templates, tools, and practices based on new insights.

Benefits of the Eleven-Step Process

- Ensures thorough test coverage and defect detection.
- Provides clear accountability and traceability.
- Aligns testing with project goals and timelines.
- Facilitates continuous improvement for long-term quality.

4.5

ASSESS PROJECT MANAGEMENT DEVELOPMENT ESTIMATE AND STATUS

Q14. Discuss the Assess Project Management Development Estimate and Status.

Ans :

(Imp.)

Assessing project management development estimates and status involves evaluating the accuracy of time, cost, and resource predictions and monitoring the progress of a project against its planned objectives. Here's how you can approach this systematically:

1. Assess Initial Development Estimates

Objective: Verify the accuracy and feasibility of original project estimates.

Activities

(i) Review Project Scope

- Validate the alignment of scope with estimates.
- Identify any assumptions or constraints in the estimation process.

(ii) Examine Estimation Techniques

- Determine whether techniques like analogous estimation, parametric modeling, or expert judgment were used.
- Check if historical data or industry bench-marks were applied.

(iii) Breakdown of Tasks

- Ensure the work breakdown structure (WBS) is granular enough for accurate estimates.

(iv) Resource Allocation

- Evaluate the availability and capability of assigned resources.
- **Output:** Documented analysis of estimation accuracy and recommendations for adjustments.

2. Monitor Current Status

Objective: Measure progress against baseline plans.

- Activities**
- (i) **Track Milestones**
Compare achieved milestones to the project schedule.
Identify delays and their causes.
- (ii) **Evaluate Resource Usage**
Check resource utilization rates against planned allocation.
Identify overuse or underuse of resources.
- 3. Cost Performance**
- Measure actual costs against budgeted costs using earned value management (EVM) metrics like:
 - Cost Performance Index (CPI): $CPI = \frac{EV}{AC}$
 - Schedule Performance Index (SPI): $SPI = \frac{EV}{PV}$
 - Determine if the project is under or over budget.
- 4. Quality Checks**
- Ensure deliverables meet quality standards.
 - Identify defects or rework needs.
 - **Output:** Status reports highlighting deviations, risks, and corrective actions.
- 3. Identify Risks and Issues**
- Objective:** Proactively manage potential threats to project success.
- Activities**
- (i) **Risk Assessment**
Update the risk register with new risks.
Assess the impact of identified risks on the timeline and budget.
- (ii) **Issue Resolution**
Track issues using an issue log.
Prioritize and resolve issues in alignment with project objectives.
Output: Updated risk and issue management plan.
- 4. Reassess and Reforecast**
- Objective:** Adjust plans to reflect current project realities.

- Activities**
- (i) **Reforecast Schedule**
Use actual progress data to revise future activity durations.
- (ii) **Reestimate Costs:**
Update budget forecasts based on current spending trends.
- (ii) **Optimize Resources**
Reassign resources as needed to critical tasks.
Output: Revised project schedule and budget forecasts.
- 5. Communicate Status and Recommendations**
- Objective:** Keep stakeholders informed and aligned.
- Activities**
- (i) **Prepare Reports**
Use visual aids like Gantt charts, burndown charts, and dashboards to illustrate progress.
Highlight critical path activities and their status.
- (ii) **Stakeholder Meetings**
Conduct regular updates to discuss progress, risks, and required changes.
- (iii) **Seek Approvals**
Present revised estimates and get stakeholder buy-in for necessary adjustments.
Output: Stakeholder-approved updated project plan.
- 6. Evaluate Performance Post-Completion**
- Objective:** Learn from project outcomes to improve future estimates.
- Activities**
- i) **Analyze Variances**
Compare initial estimates to actual results.
Determine reasons for variances.
- ii) **Document Lessons Learned:**
Capture insights into estimation accuracy and process inefficiencies.

iii) Update Methodologies:

Refine estimation techniques and tools for future projects.

Output: Comprehensive lessons-learned document and improved estimation guidelines.

Tools to Assist in Assessment

- **Project Management Tools:** Jira, Microsoft Project, Asana, or Trello.
- **Estimation Tools:** Function Point Analysis, COCOMO (Constructive Cost Model).
- **Performance Metrics:** Earned Value Management (EVM), Key Performance Indicators (KPIs).
- **Communication Tools:** Slack, Zoom, or MS Teams for stakeholder updates.

4.6

DEVELOP TEST PLAN

Q15. Explain how to develop a test plan.

Ans :

(Imp.)

Developing a test plan is a critical aspect of the software testing process as it outlines the objectives, scope, approach, and schedule of testing. A test plan typically includes the following components:

1. **Test Objectives:** The test objectives define the purpose and goals of the testing effort. The objectives should be specific, measurable, and achievable, and they should align with the overall project goals.
2. **Test Scope:** The test scope defines the boundaries of the testing effort. It includes the features, functionalities, and components of the software that will be tested. The scope should be well-defined and clearly communicated to all stakeholders.
3. **Test Approach:** The test approach defines the strategies and methods that will be used to test the software. It includes the types of tests that will be conducted, such as functional, performance, security, and usability testing. The approach should be based on industry best practices and should be tailored to the specific needs of the project.
4. **Test Schedule:** The test schedule defines the timeline for the testing effort. It includes the start and end dates of testing, as well as the milestones

and deliverables that need to be achieved during the testing process. The schedule should be realistic and take into account any dependencies and constraints.

5. **Test Environment:** The test environment defines the hardware, software, and network configurations that will be used for testing. It includes the tools and technologies that will be used to support the testing effort.
6. **Test Resources:** The test resources include the people, tools, and equipment that will be required for testing. It includes the roles and responsibilities of the testing team members, as well as any external resources that may be needed.
7. **Risk Assessment:** The risk assessment identifies the potential risks and issues that may arise during testing. It includes the probability and impact of each risk, as well as the mitigation strategies that will be used to minimize the risk.

Once the test plan has been developed, it should be reviewed and approved by all stakeholders, including the project manager, development team, and testing team. The test plan should be a living document that is updated as needed throughout the testing process to ensure that it remains relevant and effective.

4.7

REQUIREMENTS PHASE TESTING

Q16. Define requirement phase testing. State the objectives, activities of requirements Phase Testing.

Ans :

Meanin of requirement phase testing

Requirements Phase Testing is the process of verifying and validating requirements during the early stages of software development. The goal is to ensure that the requirements are clear, complete, consistent, and testable before proceeding to design and development. This proactive testing minimizes defects introduced later in the lifecycle.

Objectives of Requirements Phase Testing

1. **Defect Prevention:** Identify inconsistencies, ambiguities, or missing details early.
2. **Improved Communication:** Ensure a common understanding among stakeholders.
3. **Testability:** Confirm that all requirements are measurable and testable.

- 4. Risk Mitigation:** Reduce the likelihood of costly changes or rework.

Key Activities in Requirements Phase Testing

1. Requirement Review

- Conduct formal or informal reviews (e.g., walkthroughs or inspections).
- Ensure the requirements adhere to quality attributes like correctness, completeness, and feasibility.

2. Ambiguity Analysis

- Identify vague terms or unclear statements.
- Use tools like natural language parsers to spot ambiguous words (e.g., “fast,” “secure,” or “user-friendly”).

3. Validation

- Verify that the requirements align with business objectives and stakeholder needs.
- Check for feasibility given time, budget, and technical constraints.

4. Traceability Analysis

- Establish a Requirements Traceability Matrix (RTM) to ensure every requirement can be traced to a test case, feature, or business objective.

5. Prototyping and Simulation

- Use prototypes or mockups to clarify requirements and uncover hidden expectations.

6. Conflict Resolution

- Resolve contradictions between requirements through stakeholder discussions or prioritization.

7. Test Case Preparation

- Design high-level test cases or acceptance criteria based on requirements.
- Validate that requirements are testable by linking them to these test cases.

Q17. Discuss the techniques and challenges of requirement phasing testing?

Ans :

Techniques for Requirements Phase Testing

1. Peer Reviews

- Team members review each requirement document for clarity, correctness, and completeness.

2. Checklist-Based Testing

- Use a predefined checklist to systematically assess requirements.
- Examples of checklist items:
- Are all functional and non-functional requirements specified?
- Are dependencies and constraints documented?

3. Model-Based Testing

- Create models (e.g., use case diagrams, data flow diagrams) to validate workflows and scenarios.

4. Use Case Analysis

- Break down requirements into use cases and validate them for completeness and testability.

5. Formal Methods

- Apply mathematical models to validate critical systems, especially in high-risk domains like healthcare or aviation.

Challenges

- 1. Ambiguity:** Vague or incomplete requirements make validation difficult.
- 2. Stakeholder Misalignment:** Conflicting priorities or interpretations among stakeholders.
- 3. Unrealistic Requirements:** Overambitious goals may not be technically or economically feasible.
- 4. Lack of Testability:** Some requirements may lack measurable criteria.

4.8

DESIGN PHASE TESTING

Q18. Define design phase testing. State the objectives and key activities of design phase testing?

Ans :

(Imp.)

Design Phase Testing is the process of verifying and validating the design documents and artifacts created during the software design phase. The goal is to ensure that the system's architecture, modules, interfaces, and database designs align with the requirements and are technically sound, feasible, and testable.

Objectives

1. **Defect Detection:** Identify design errors, omissions, or inconsistencies before coding begins.
2. **Verification of Requirements Mapping:** Ensure that the design aligns with the validated requirements.
3. **Improve Quality:** Enhance the robustness, maintainability, and scalability of the design.
4. **Facilitate Test Planning:** Develop test cases and strategies based on design artifacts.

Key Activities in Design Phase Testing**1. Review Design Artifacts:**

- **High-Level Design (HLD):** Verify the system architecture, data flow, and module relationships.
- **Low-Level Design (LLD):** Validate detailed module designs, algorithms, and logic.
- **Interface Design:** Ensure all external and internal interfaces are well-defined and testable.

2. Validation Against Requirements

- Confirm that all functional and non-functional requirements are covered in the design.
- Update the Requirements Traceability Matrix (RTM) to include design-level mappings.

3. Static Testing Techniques

- Conduct **walkthroughs** and **inspections** of design documents to identify defects.
- Use **checklist-based reviews** for systematic evaluation of design quality.

4. Scenario Validation

- Test design against real-world scenarios to ensure completeness and feasibility.
- Validate workflows using UML diagrams like sequence diagrams and activity diagrams.

5. Database Design Validation

- Check the database schema for normalization, relationships, and indexing.
- Validate data integrity, constraints, and scalability.

6. Risk Analysis

- Assess potential risks in the design, such as performance bottlenecks or scalability issues.
- Document and address identified risks early.

7. Test Case Design

- Begin creating detailed test cases based on the design artifacts.
- Develop test scenarios for integration testing and system testing.

Q19. State the techniques and challenges of design phase testing?*Ans :***Techniques for Design Phase Testing****1. Peer Reviews and Inspections**

- Conduct team reviews of HLD and LLD documents for accuracy and completeness.

2. Prototyping

- Build prototypes for complex or ambiguous parts of the design to validate feasibility.

3. Model-Based Testing

- Use UML or ER diagrams to verify that the system's logic and workflows meet requirements.

4. Checklist-Based Testing:

- Examples of checklist items:
- Are all modules and interfaces defined clearly?
- Are security requirements incorporated into the design?
- Are error-handling mechanisms included?

5. Static Analysis Tools:

- Use tools like SonarQube or Checkstyle to analyze the design for maintainability and scalability.

6. Simulation and Modeling:

- Simulate workflows, data processing, or load handling using modeling tools.

Common Challenges in Design Phase Testing

1. **Ambiguity in Design:** Lack of detail or unclear definitions in design documents.
2. **Incomplete Mapping:** Missing or incorrectly mapped requirements.
3. **Overlooking Non-Functional Requirements:** Ignoring performance, security, or scalability aspects.
4. **Resource Constraints:** Insufficient time or expertise for detailed design reviews.

4.9**PROGRAM PHASE TESTING**

Q20. Define program phase testing. State its activities and objectives?

Ans :

Meaning of Program Phase Testing

Program Phase Testing refers to testing activities performed during the coding or implementation phase of the software development lifecycle. The primary objective is to ensure that individual components, modules, or units of the software are developed correctly and meet their functional and non-functional requirements before integration.

Objectives

1. **Defect Identification:** Find and fix coding errors at the earliest possible stage.
2. **Code Verification:** Ensure the code adheres to design specifications and coding standards.
3. **Unit Functionality Validation:** Test each module independently for expected functionality.
4. **Facilitate Integration:** Prepare tested and error-free units for smooth integration.

Key Activities in Program Phase Testing

1. **Unit Testing**
 - Focus on testing individual units (functions, methods, or classes).
 - Validate that each unit works as intended in isolation.
 - Use tools like JUnit (for Java), pytest (for Python), or NUnit (for .NET).

2. Static Code Analysis

- Identify issues like code smells, inefficiencies, or non-adherence to coding standards.
- Use tools like SonarQube, Checkstyle, or ESLint.

3. Code Reviews and Inspections

- Conduct peer reviews to examine the quality, correctness, and maintainability of the code.
- Verify adherence to design documents and coding standards.

4. Test Case Development

- Write test cases for all code paths, including edge cases and negative scenarios.
- Create automated tests wherever possible to streamline regression testing.

5. Debugging

- Use debugging tools to locate and fix issues.
- Ensure proper handling of exceptions and errors.

6. Test Data Preparation

- Create or use realistic test data to validate program behavior.
- Ensure test data covers all scenarios, including edge cases.

7. Code Coverage Analysis

- Measure the percentage of code executed during tests.
- Aim for high coverage (e.g., 80% or more) to ensure most paths are tested.

Q21. Discuss the techniques and challenges of program phase testing?

Ans :

Techniques**1. White-Box Testing**

- Test internal structures or workings of the program.
- Techniques include statement coverage, branch coverage, path coverage, and decision coverage.

2. Boundary Value Analysis (BVA)

- Test edge conditions to ensure the program handles them correctly.

3. Equivalence Partitioning

- Divide input data into valid and invalid partitions and test one value from each partition.

4. Error Guessing

- Leverage experience to predict where bugs might occur and test those areas.

5. Automated Testing

- Use frameworks like Selenium, TestNG, or Robot Framework to automate repetitive tests.

Common Challenges in Program Phase Testing**1. Incomplete Requirements**

- Ambiguity in requirements can lead to misaligned code and tests.

2. Unrealistic Deadlines

- Pressure to deliver may reduce the time available for thorough testing.

3. Complex Logic

- Highly complex algorithms can be difficult to test comprehensively.

4. Dependency Issues

- Units may depend on other modules that are not yet developed or available.

4.10**EXECUTE TEST AND RECORD RESULTS****Q22. What is Test Execution? Explain the importance test execution?***Ans :***Meaning of Test Execution**

Test Execution is the process of executing the tests written by the tester to check whether the developed code or functions or modules are providing the expected result as per the client requirement or business requirement. Test Execution comes under one of the phases of the Software Testing Life Cycle (STLC).

In the test execution process, the tester will usually write or execute a certain number of test cases, and test scripts or do automated testing. If it creates any errors then it will be informed to the respective development team to correct the issues in the code. If the test execution process shows successful results then it will be ready for the deployment phase after the proper setup for the deployment environment.

Importance of Test Execution

- **The project runs efficiently:** Test execution ensures that the project runs smoothly and efficiently.
- **Application competency:** It also helps to make sure the application's competency in the global market.
- **Requirements are correctly collected:** Test executions make sure that the requirements are collected correctly and incorporated correctly in design and architecture.
- **Application built in accordance with requirements:** It also checks whether the software application is built in accordance with the requirements or not.

Q23. Discuss the Activities for Test Execution.*Ans :*

The following are the 5 main activities that should be carried out during the test execution.

1. Defect Finding and Reporting

Defect finding is the process of identifying the bugs or errors raised while executing the test cases on the developed code or modules. If any error appears or any of the test cases failed then it will be recorded and the same will be reported to the respective development team. Sometimes, during the user acceptance testing also end users may find the error and report it to the team. All the recorded details will be reported to the respective team and they will work on the recorded errors or bugs.

2. Defect Mapping

After the error has been detected and reported to the development team, the development team will work on those errors and fix them as per the requirement. Once the development team has done its job, the tester team will again map the test cases or test scripts to that developed module or code to run the entire tests to ensure the correct output.

3. Re-Testing

From the name itself, we can easily understand that Re-Testing is the process of testing the modules or entire product again to ensure the smooth release of the module or product. In some cases, the new module or functionality will be developed after the product release. In this case, all the modules will be re-tested for a smooth release. So that it cannot cause any other defects after the release of the product or application.

4. Regression Testing

Regression Testing is software testing that ensures that the newly made changes to the code or newly developed modules or functions should not affect the normal processing of the application or product.

5. System Integration Testing

System Integration Testing is a type of testing technique that will be used to check the entire component or modules of the system in a single run. It ensures that the whole system will be checked in a single test environment instead of checking each module or function separately.

Q24. Discuss the process of test execution.

Ans : (Imp.)

Test Execution Process

The test Execution technique consists of three different phases which will be carried out to process the test result and ensure the correctness of the required results. In each phase, various activities or work will be carried out by various team members. The three main phases of test execution are the creation of test cases, test case execution, and validation of test results.

1. Creation of Test Cases

The first phase is to create suitable test cases for each module or function. Here, the tester with good domain knowledge must be required to create suitable test cases. It is always preferable to create simple test cases and the creation of test cases should not be delayed else it will cause excess time to release the product. The created test cases should not be repeated again. It should cover all the possible scenarios raised in the application.

2. Test Cases Execution

After test cases have been created, execution of test cases will take place. Here, the Quality Analyst team will either do automated or manual testing depending upon the test case scenario. It is always preferable to do both automated as well as manual testing to have 100% assurance of correctness. The selection of testing tools is also important to execute the test cases.

3. Validating Test Results

After executing the test cases, note down the results of each test case in a separate file or report. Check whether the executed test cases achieved the expected result and record the time required to complete each test case i.e., measure the performance of each test case. If any of the test cases is failed or not satisfied the condition then report it to the development team for validating the code.

Q25. Discuss the various Ways to Perform Test Execution?

Ans : (Imp.)

Testers can choose from the below list of preferred methods to carry out test execution:

1. Run test cases

It is a simple and easiest approach to run test cases on the local machine and it can be coupled with other artifacts like test plans, test suites, test environments, etc.

2. Run test suites

A test suite is a collection of manual and automated test cases and the test cases can be executed sequentially or in parallel. Sequential execution is useful in cases where the result of the last test case depends on the success of the current test case.

3. Run test case execution and test suite execution records

Recording test case execution and test suite execution is a key activity in the test process and helps to reduce errors, making the testing process more efficient.

4. Generate test results without execution

Generating test results from non-executed test cases can be helpful in achieving comprehensive test coverage.

5. Modify execution variables

Execution variables can be modified in the test scripts for particular test runs.

6. Run automated and manual tests

Test execution can be done manually or can be automated.

7. Schedule test artifacts

Test artifacts include video, screenshots, data reports, etc. These are very helpful as they document the results of the past test execution and provide information about what needs to be done in future test execution.

8. Defect tracking

Without defect tracking test execution is not possible, as during testing one should be able to track the defects and identify what when wrong and where.

Q26. Explain briefly about test execution priorities.

Ans : (Imp.)

Test Execution Priorities

Test Execution Priorities are nothing but prioritizing the test cases depending upon several factors. It means that it executes the test cases with high efficient first than the other test cases. It depends upon various factors. Let us discuss some of the factors to be considered while prioritizing the test cases.

- i) **Complexity:** The complexity of the test cases can be determined by including several factors such as boundary values of test cases, features or components of test cases, data entry of test cases, and how much the test cases cover the given business problem.
- ii) **Risk Covered:** How much risk that a certain test case may undergo to achieve the result. Risk in the form of time required to complete the test case process, space complexity whether it is executed in the given memory space, etc.,

iii) **Platforms Covered:** It simply tells that in which platform or operating system the test cases have been executed i.e., test cases executed in the Windows OS, Mac OS, Mobile OS, etc.,

iv) **Depth:** It covers how depth the given test cases cover each functionality or module in the application i.e., how much a given test procedure covers all the possible conditions in a single functionality or module.

➤ **Breadth:** It covers how the breadth of the given test cases covers the entire functionality or modules in the application i.e., how much a given test procedure covers all the possible conditions in the entire functionality or modules in the product or application.

Q27. Explain the concept of test execution cycle.

Ans : (Imp.)

Test Execution Cycle

A test execution cycle is an iterative approach that will be helpful in detecting errors. The test execution cycle includes various processes. These are:

1. Requirement Analysis

In which, the QA team will gather all the necessary requirements needed for test execution. For example, how many testers are needed, what automation test tools are needed, what testing covers under the given budget, etc., the QA team will also plan depending upon the client or business requirement.

2. Test Planning

In this phase, the QA team will plan when to start and complete the testing. Choosing of correct automation test tool, and testers needed for executing the test plan. They further plan who should develop the test cases for which module/function, who should execute the test cases, how many test cases needed to be executed, etc.,

3. Test Cases Development

This is the phase in which the QA team assigned a group of testers to write or generate the test cases for each module. A tester with good domain knowledge will easily write the best test cases or test scripts. Prioritizing the developed test cases is also the main factor.

4. Test Environment Setup

Test Environment Setup usually differs from project to project. In some cases, it is created by the team itself and it is also created by clients or customers. Test Environment Setup is nothing but testing the entire developed product with suitable software or hardware components or with both by executing all the tests on it. It is essential and it is sometimes carried out along with the test case development process.

5. Test Execution

This stage involves test execution by the team and all the detected bugs are recorded and reported for remediation and rectification.

6. Test Closure

This is the final stage and here it records the entire details of the test execution process. It also contains the end-users testing details. It again modifies the testing process if any defects are found during the testing. Hence, it is a repetitive process.

Q28. State the guidelines for test execution.

Ans :

Guidelines for Test Execution

- i) Write the suitable test cases for each module of the function.
- ii) Assign suitable test cases to respective modules or functions.
- iii) Execute both manual testing as well as automated testing for successful results.
- iv) Choose a suitable automated tool for testing the application.
- v) Choose the correct test environment setup.
- vi) Note down the execution status of each test case and note down the time taken by the system to complete the test cases.
- vii) Report all the success status and the failure status to the development team or to the respective team regularly.
- viii) Track the test status again for the already failed test cases and report it to the team.
- ix) Highly Skilled Testers are required to perform the testing with less or zero failures/defects.
- x) Continuous testing is required until success test report is achieved.

4.11**ACCEPTANCE TEST****Q29. What is Acceptance Testing? Explain various types of acceptance testing?**

Ans :

(Imp)

Meaning of Acceptance Testing

It is 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.

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

Types of Acceptance Testing

Here are the Types of Acceptance Testing

1. User Acceptance Testing (UAT)

- User acceptance testing is used to determine whether the product is working for the user correctly.
- Specific requirements which are quite often used by the customers are primarily picked for testing purposes. This is also termed as End-User Testing.

2. Business Acceptance Testing (BAT)

- BAT is used to determine whether the product meets the business goals and purposes or not.
- BAT mainly focuses on business profits which are quite challenging due to the changing market conditions and new technologies, so the current implementation may have to be changed which results in extra budgets.

3. Contract Acceptance Testing (CAT)

- CAT is a contract that specifies that once the product goes live, within a predetermined period, the acceptance test must be performed, and it should pass all the acceptance use cases.
- Here is a contract termed a Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled.

- Sometimes, this contract happens before the product goes live.
- There should be a well-defined contract in terms of the period of testing, areas of testing, conditions on issues encountered at later stages, payments, etc.

4. Regulations Acceptance Testing (RAT)

- RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released.
- This may be unintentional but will impact negatively on the business. Generally, the product or application that is to be released in the market, has to go under RAT, as different countries or regions have different rules and regulations defined by its governing bodies.
- If any rules and regulations are violated for any country then that country or the specific region then the product will not be released in that country or region.
- If the product is released even though there is a violation then only the vendors of the product will be directly responsible.

5. Operational Acceptance Testing (OAT)

- OAT is used to determine the operational readiness of the product and is non-functional testing.
- It mainly includes testing of recovery, compatibility, maintainability, reliability, etc. OAT assures the stability of the product before it is released to production.

6. Alpha Testing

- Alpha testing is used to determine the product in the development testing environment by a specialized testers team usually called alpha testers.

7. Beta Testing

- Beta testing is used to assess the product by exposing it to the real end-users, typically called beta testers in their environment.

- Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

Q30. State the uses, advantages and disadvantages of acceptance testing.

Ans :

Use

1. To find the defects missed during the functional testing phase.
2. How well the product is developed.
3. A product is what actually the customers need.
4. Feedback help in improving the product performance and user experience.
5. Minimize or eliminate the issues arising from the production.

Advantages

1. This testing helps the project team to know the further requirements from the users directly as it involves the users for testing.
2. Automated test execution.
3. It brings confidence and satisfaction to the clients as they are directly involved in the testing process.
4. It is easier for the user to describe their requirement.
5. It covers only the Black-Box testing process and hence the entire functionality of the product will be tested.

Disadvantages

4. Users should have basic knowledge about the product or application.
5. Sometimes, users don't want to participate in the testing process.
6. The feedback for the testing takes a long time as it involves many users and the opinions may differ from one user to another user.
7. Development team is not participated in this testing process.

4.12

REPORT TEST RESULTS

Q31. Explain briefly about report test results?

Ans :

Reporting test results is an essential part of the software testing process. It involves collecting and documenting information on the test execution, including the results, defects found, and any relevant observations. The test results report provides valuable information to stakeholders, such as the development team, project manager, and customers, about the quality of the software.

The following are the key components of a test results report:

1. **Test summary:** This provides an overview of the testing activities, including the number of tests executed, passed, failed, and blocked.
2. **Defect summary:** This section provides information on the defects found during testing, including their severity, priority, and status. It may also include any recommendations or suggestions for resolving the defects.
3. **Test coverage:** This section describes the test coverage achieved, including the areas of the application that were tested, and the percentage of requirements or functionality that were covered.
4. **Test environment:** This section provides information on the testing environment, including the hardware and software configurations, tools used, and any environmental issues encountered during testing.
5. **Test logs:** This section includes the detailed logs of the test execution, including any error messages, stack traces, or other relevant information.
6. **Conclusion:** This section provides a summary of the overall test results and any recommendations or conclusions based on the testing activities.

4.13

TEST SOFTWARE CHANGES

Q32. Discuss the concept of test software changes?

Ans :

(Imp.)

Testing software changes is a crucial part of software development. It ensures that the changes work as expected, do not introduce new bugs, and do not break existing functionality. However, testing software

changes can also be challenging, time-consuming, and complex. How do you test software changes effectively and efficiently? Here are some strategies and tips to help you.

Plan your tests

Before you start testing software changes, you need to have a clear plan of what to test, how to test, and when to test. A test plan should define the scope, objectives, criteria, methods, tools, and resources of your testing activities. It should also align with your project requirements, specifications, and standards. A test plan can help you organize your testing process, communicate your expectations, and track your progress and results.

Use different types of tests

Depending on the nature and scale of your software changes, you may need to use different types of tests to verify their quality and functionality. Unit tests, for example, are small tests that check the logic and behavior of individual components or functions of your software. These are usually automated and run frequently to detect errors early in the development cycle. Integration tests, on the other hand, check how different components or modules of your software interact and work together. System tests, meanwhile, verify the overall functionality and performance of your software as a whole. Lastly, regression tests check whether your software changes have affected or broken any existing functionality or features. These are usually automated or semi-automated and run after each change to ensure that the software remains stable and reliable.

You need to catch bugs or unintended behavior as early as you can. To achieve this from testing point of view you can use shift-left testing approach.

Shift-left testing is an approach to software testing and system testing in which testing is performed earlier in the development lifecycle. This way your product would be less prone to errors/bugs.

Apply testing best practices

To test software changes effectively and efficiently, you should follow some testing best practices, such as testing early and often, in different environments, with real data, and with different perspectives. Doing so can help you identify and fix errors quickly, reduce rework, improve quality, ensure your software works consistently and correctly in various conditions and scenarios, simulate actual user behavior and expectations, and evaluate your software from multiple angles. Ultimately, this will help you meet the needs and expectations of different audiences.

Review and document your tests

Testing software changes is an essential part of software development. To ensure quality, you should review and document your tests. Reviewing your test results allows you to evaluate the outcomes and impacts of your software changes, detect any errors or defects, and prioritize any fixes or improvements. Documenting your test cases records and organizes your test scenarios, inputs, outputs, expected results, and actual results - helping you reuse and update your test cases for future changes or testing cycles. Additionally, documenting your test reports helps summarize and communicate your test findings, conclusions, and recommendations. It also tracks and measures your test coverage, quality, and performance. By following these strategies and tips, you can test software changes more effectively and efficiently to deliver high-quality software products.

4.14**EVALUATE TEST EFFECTIVENESS****Q33. How do you evaluate the test effectiveness?**

Ans :

‘Evaluating test effectiveness is an essential step in the software testing process. It involves measuring how well the testing activities have achieved their intended objectives and determining the value of the testing effort. Test effectiveness evaluation can help identify areas for improvement and make the testing process more efficient and effective.

The following are some of the key steps involved in evaluating test effectiveness:

1. Define the evaluation criteria

To evaluate the test effectiveness, you need to define the criteria that you will use to measure the testing results. The evaluation criteria can include factors such as the number of defects found, the percentage of test cases passed, the test coverage achieved, and the time and resources expended.

2. Collect and analyze data

Once you have defined the evaluation criteria, you need to collect and analyze the relevant data. This can involve reviewing the test results, defect reports, and other documentation to determine how well the testing objectives were met.

3. Interpret the results

After collecting and analyzing the data, you need to interpret the results to determine the effectiveness of the testing effort. This can involve comparing the actual results to the expected results, identifying any trends or patterns in the data, and determining the root cause of any issues or problems.

4. Identify areas for improvement

Based on the evaluation results, you can identify areas for improvement in the testing process. This can involve making changes to the testing approach, tools, or methodologies, or providing additional training or support to the testing team.

5. Implement improvements

Once you have identified areas for improvement, you can implement the necessary changes to improve the effectiveness of the testing process. This can involve updating the testing plan, modifying the test cases, or changing the testing approach.

UNIT V

Testing Client / Server Systems, Testing the Adequacy of System Documentation, Testing Web-based Systems, Testing Off — the — Shelf Software, Testing in a Multiplatform Environment, Testing Security, Testing a Data Warehouse, Creating Test Documentation, Software Testing Tools, Taxonomy of Testing Tools, Methodology to Evaluate Automated Testing Tools, Load Runner, Win Runner and Rational Testing Tools, Java Testing Tools, JMelra, JUNIT and Cactus.

5.1

TESTING CLIENT / SERVER SYSTEMS

Q1. What is Client Server Testing? Discuss the various scenarios of client server testing?

Ans :

Client-server testing is a testing approach designed to verify the accurate and secure exchange of data between the client and server, guaranteeing that requests and responses are synchronized correctly.

This testing also involves assessing the system's performance, scalability, and resource utilization to confirm its ability to handle various loads and user interactions without compromising performance. Moreover, client-server testing includes functional testing to ensure that the application's features and functionalities operate as expected on both the client and server sides.

Example Of Client Server Testing

Web applications often utilize client-server architecture, with the user's web browser (client) sending requests to a web server for data or services. In this scenario, client-server testing involves ensuring that web pages load correctly and that user interactions, such as form submissions, result in proper data transmission and server responses. Let's discuss a few examples and test scenarios of client-server testing in real life.

Test Scenario 1

Simulate a user registration process on a web application. Verify that user data entered via the client-side form is correctly transmitted to the server, stored in the database, and retrievable upon subsequent logins.

Components Tested: Client, Web Server, Database.

Test Scenario 2

Test the effectiveness of caching by repeatedly accessing a frequently used page on the web application.

Measure the response time, and assess whether caching (e.g., Redis) reduces server load and speeds up page rendering.

Components Tested: Client, Web Server, Caching.

Test Scenario 3

Test a chat feature within the web application. Send messages between users and verify that they are delivered in real-time using a message queue (e.g., RabbitMQ, Kafka). Confirm that messages are processed asynchronously and that users receive them promptly.

Components Tested: Client, Web Server, Message Queue.

Test Scenario 4

Test email synchronization between the email client and the email server (e.g., IMAP). Ensure that newly received emails are correctly displayed in the client's inbox and that read / unread statuses are synchronized with the server.

Components Tested: Email Client, Email Server, Database.

Test Scenario 5

Conduct a load test by simulating a surge in user activity, increasing the number of concurrent users significantly. Evaluate how the server scales to handle the increased load and whether it maintains low latency and responsiveness.

Components Tested: Web Users Client Interface(Multiple), Database or Caching Server, Load Balancer.

Q2. What are the Objectives of client server testing.

Ans :

(Imp.)

The main goal of client-server testing is to ensure the robustness, availability, and reliability of software applications or systems that are built upon a client-server architecture.

Objectives

Key objectives of client-server testing include:

1. Functionality Validation

Confirm that the client and server components work together to deliver the intended features and functionalities without errors or inconsistencies.

2. Data Integrity and Security

Ensuring that data exchanged between the client and server is accurate, secure, and protected from unauthorized access or manipulation.

3. Performance Assessment

Evaluating the responsiveness, scalability, and resource utilization of the system to guarantee that it can handle various loads and user interactions while maintaining acceptable performance levels.

4. Fault Tolerance and Reliability

Testing the system's ability to handle adverse conditions, such as network failures or server crashes, and recover gracefully without data loss or service disruption.

5. Compatibility

Verifying that the client software is compatible with different server configurations, versions, and environments, ensuring a seamless user experience.

6. Scalability and Load Handling

Determining how well the system scales to accommodate a growing number of clients and transactions while maintaining performance and stability.

7. Security

Identifying vulnerabilities and weaknesses in data transmission, authentication, and access control mechanisms to enhance security measures and protect user data.

Q3. Discuss the Basic Characteristics of Client Server Testing Architecture

Ans :

Client-server testing architecture is characterized by several fundamental attributes that distinguish it from other software testing methodologies. These basic characteristics include:

(i) Distributed Components

Client-server architecture consists of two primary components the client, which runs on the user's device, and the server, hosted on remote hardware. Testing involves evaluating the interaction between these distributed components.

(i) Communication Over a Network

Clients and servers communicate over a network, typically using protocols like HTTP, TCP/IP, or custom communication protocols. Testing ensures reliable and efficient data exchange between them.

(ii) Service Independence

Each microservice is responsible for a specific function or feature, making it crucial to verify that each service operates independently and integrates seamlessly with other services. Testing assesses how well these services work together to deliver end-to-end functionality.

(iii) Data Integrity

Ensuring the accuracy and integrity of data transmission between the client and server is a key concern. Testing validates that the data sent and received is correct, complete, and secure.

(iv) Caching and Messaging

Depending on the architecture, client-server systems may involve caching and message queues. Testing these components ensures they function correctly and enhance system performance.

(v) Asynchronous Processing

In some cases, client-server systems handle asynchronous operations. Testing verifies that asynchronous tasks, such as background processing, are executed reliably and efficiently.

(vi) Continuous Integration/Continuous Deployment (CI/CD)

Typical client-server architectures are part of a CI/CD pipeline these days due to automation. Testing ensures that the testing and deployment processes are automated, consistent, and reliable.

(vii) Monitoring and Observability

Effective monitoring and observability solutions (e.g., Prometheus, Grafana) are crucial in any distributed system. Testing these covers the integration and functionality of these tools for real-time system insights.

Q4. Discuss various Types Client-Server Test.*Ans :*

Performing a comprehensive range of testing types is essential in client-server testing to ensure the reliability, performance, and security of systems. Here are the

I. Primary types of testing to consider**1. Functional Testing****(i) Unit Testing**

Evaluate individual client and server components to verify that they perform their specific functions correctly.

(ii) Integration Testing

Assess how well client and server components integrate and work together, ensuring that data is exchanged accurately.

(iii) System Testing

Conduct end-to-end testing to validate that the entire client-server system functions as expected in real-world scenarios.

2. Performance Testing**(i) Load Testing**

Measure the system's response and behavior under varying levels of user load to identify performance bottlenecks.

(ii) Stress Testing

Push the system beyond its intended capacity to determine breaking points and assess its ability to recover.

(iii) Scalability Testing

Evaluate how well the system scales to accommodate growing numbers of clients or data transactions while maintaining performance.

3. Security Testing**(i) Authentication Testing**

Verify that the authentication mechanisms (e.g., username/password, tokens) work correctly and securely.

(ii) Authorization Testing

Ensure that users can only access resources and functionalities they are permitted to, and unauthorized access is prevented.

(iii) Encryption Testing

Confirm that data transmission between the client and server is properly encrypted to protect sensitive information.

4. Data Integrity Testing**(i) Data Validation Testing**

Check that data sent and received between the client and server is accurate, complete, and follows validation rules.

(ii) Data Corruption Testing

Assess how the system handles data corruption or loss scenarios, ensuring data integrity is maintained.

5. Compatibility Testing**(i) Cross-Browser and Cross-Device Testing**

Ensure that the client application functions correctly on various browsers and devices.

(ii) Cross-Platform Testing

Confirm that the client application is compatible with different server configurations, versions, and operating systems.

II. Caching and Performance Optimization Testing**(i) Cache Testing**

Assess the effectiveness of caching mechanisms (e.g., Redis) to speed up data retrieval and reduce server load.

(ii) Performance Optimization Testing

Identify areas for optimization to enhance the system's overall performance and responsiveness.

III. Fault Tolerance and Recovery Testing**(i) Failover Testing**

Simulate server failures and network disruptions to evaluate how well the system can recover without data loss.

(ii) Redundancy Testing

Verify that redundant server setups work as intended to ensure system availability.

IV. Usability and User Experience Testing**(i) Usability Testing**

Evaluate the client application's user interface and overall user experience to ensure it is user-friendly.

(ii) Accessibility Testing

Confirm that the application is accessible to users with disabilities, complying with accessibility standards.

V. Regression Testing

- (i)** Continuously run regression tests to ensure that new updates or changes do not introduce unexpected issues or regressions in existing functionality.

VI. Load Balancing and Network Testing

- (i)** Test load balancers to ensure they distribute client requests effectively and maintain high availability.
- (ii)** Assess network configurations to confirm they support secure and efficient client-server communication.

VII. Message Queue Testing

- (i)** Validate the reliability and efficiency of message queues (e.g., RabbitMQ) in handling asynchronous communication between client and server components.

VIII. Containerization and Orchestration Testing

- (i)** Verify the functionality and compatibility of containerized applications (e.g., Docker) and their orchestration configurations (e.g., Kubernetes).

Manual Testing and Its Types

Manual testing is a fundamental testing approach where human testers execute test cases without the use of automation tools or scripts. It relies on human intuition and expertise to evaluate an application's functionality, user interface, and overall quality. Manual testing encompasses several types:

(i) Functional Testing

This involves testers verifying that an application's features and functionalities work as expected. For example, in a client-server context, a functional test may involve manually validating that a user can log in to a web application and access their account information without encountering errors.

(ii) Usability Testing

Usability testing assesses the user-friendliness of an application. Testers, acting as end-users,

interact with the client-side interface and provide feedback on the ease of use, navigation, and overall user experience. For instance, in a client-server environment, testers might evaluate the intuitiveness of a web application's menu structure.

(iii) Exploratory Testing

Exploratory testing is an unscripted approach where testers explore the application to uncover defects, usability issues, or unexpected behavior. Testers use their creativity and domain knowledge to simulate real-world user interactions and identify potential defects.

(iv) Automated Testing

Automated testing involves the use of specialized tools and scripts to perform testing tasks automatically. It is especially valuable for repetitive or complex test scenarios. Automated testing offers various advantages, such as consistency, repeatability, and the ability to perform tests quickly and efficiently.

Example: In a client-server application, automated functional testing could involve using a tool like Testsigma to create and execute test scripts that validate the registration process. These scripts can simulate user actions such as filling out a registration form, submitting it and verifying that the user's data is correctly stored on the server.

(v) Black-Box Testing

Black-box testing focuses on evaluating an application's functionality without knowledge of its internal code or structure. Testers interact with the application's interface and assess how it responds to different inputs and conditions. This approach ensures that testing is conducted from a user's perspective, emphasizing expected outcomes and behaviors.

Example: In a client-server context, black-box testing might involve validating that a file-sharing application (client) can successfully upload files to a server and that the server correctly stores and retrieves these files. Testers would not need to examine the server's code but instead, assess whether the system functions as expected.

(vi) White-Box Testing

White-box testing is an approach where testers have access to the internal code and structure of the application. They design tests to evaluate the correctness of the code, its logic, and the execution paths within the application. White-box testing aims to uncover defects in the code's implementation.

Example: In a client-server application, white-box testing might involve code reviews and static code analysis of the server-side components to identify potential vulnerabilities or code quality issues. Testers may inspect the server's code to ensure that it handles user authentication securely and adheres to coding standards.

(vii) Mocking and Simulation

Mocking and simulation involve creating simulated or mock components to mimic the behavior of real components or services that an application relies on. This is useful for testing when the actual components or services are not readily available or should not be used during testing.

Example: In a client-server application, testers can create a mock payment gateway that simulates responses if the server depends on an external payment gateway service that should not be invoked during testing. This allows the testing of payment-related scenarios without using the actual payment service.

(viii) Network Testing

Network testing assesses how an application performs under various network conditions, such as latency, packet loss, or limited bandwidth. It ensures that the client and server components can maintain functionality and responsiveness in real-world network environments.

Example: In a client-server setup, network testing might involve simulating high-latency conditions to evaluate how well the application handles delayed responses. Testers can use network emulation tools to introduce latency and assess the impact on client-server communication.

(ix) Concurrency Testing

Concurrency testing evaluates how an application behaves when multiple users or processes access it simultaneously. This type of testing helps identify synchronization issues, race conditions, and potential conflicts that can occur in a multi-user environment.

Example: In a client-server system, concurrency testing could involve simulating concurrent logins from multiple clients to assess whether the server accurately handles authentication requests without conflicts or unexpected behavior.

Q5. Advantages and Disadvantages Of Client Server Testing.

Ans :

Here are a few pros and cons of performing client-server testing to help you identify whether to adopt it for your application.

Advantages

(i) Comprehensive

Allows for comprehensive testing of both client and server components, ensuring that they work seamlessly together.

(ii) Realistic Scenarios

Supports testing in realistic, production-like environments, helping identify issues that may occur in live settings.

(iii) Scalability

Enables scalability testing to determine how the system performs under varying user loads and concurrent connections.

(iv) Security Assessment

Allows for thorough security testing, including authentication, authorization, and data encryption, to identify and address vulnerabilities.

(v) Fault Tolerance

Facilitates fault tolerance and recovery testing to ensure the system can recover gracefully from failures without data loss.

Disadvantages

(i) Complexity

Testing client-server applications can be complex due to the need to manage two distinct components and their interactions.

(ii) Resource Intensive

Setting up and maintaining client-server testing environments can be resource-intensive, requiring additional hardware and infrastructure.

(iii) Network Dependencies

Testing may be influenced by network conditions, which can be challenging to simulate accurately.

(iv) Cost

Establishing and maintaining client-server testing configurations can be costly, particularly in terms of hardware and licensing.

(v) Time-Consuming

Setting up and configuring client-server environments can be time-consuming, potentially delaying testing efforts.

Q6. Explain various challenges of client server testing.

Ans :

Client-server testing, while essential for ensuring the functionality and reliability of applications built on this architecture, comes with its share of challenges. Here are some common client-server testing challenges:

(i) Complexity of Dual Components

Testing client-server applications involves evaluating both the client-side and server-side components. This dual-component nature introduces complexity, as it requires synchronized testing of interactions between the two, potentially leading to challenges in test design and coordination.

(ii) Data Management

Testing data management can be complex in client-server environments. Ensuring data consistency between client and server, especially during concurrent access, requires careful planning and synchronization.

(iii) Compatibility Challenges

Client applications may run on a variety of devices and platforms, each with its own configurations and specifications. Ensuring compatibility across different client environments adds complexity to testing.

(iv) Versioning and Updates

Maintaining consistency between client and server versions can be challenging, particularly when clients need to be updated to work with new server features or vice versa. Testing across different versions adds an extra layer of complexity.

(v) Test Data Management

Managing test data, including creating realistic test scenarios and ensuring data consistency, can be challenging, especially when dealing with large datasets and complex data structures.

(vi) Time Constraints

Setting up and configuring client-server environments can be time-consuming. Testers may face tight deadlines, making it challenging to conduct thorough testing within limited time frames.

(vii) Performance Bottlenecks

Identifying performance bottlenecks and optimizing server-side code can be challenging, as server-side components may interact with multiple clients simultaneously, leading to complex performance tuning requirements.

5.2

TESTING THE ADEQUACY OF SYSTEM DOCUMENTATION

Q7. Discuss the Adequacy of System Documentation

Ans :

Testing the adequacy of system documentation involves:

(i) Measuring project documentation needs

The level of detail, formality, and extent of documentation needed depends on the project's complexity, risk, and size, as well as the organization's management practices.

(ii) Determining what documents are needed

Determine which documents need to be produced.

(iii) Checking the completeness of individual documents

Ensure that each document is complete.

(iv) Assessing the currency of project documents

Determine the currency of the project's documents.

It's important to ensure that the right documentation is prepared, as there's little value in making sure that unneeded documentation is adequate. Too much documentation can also be wasteful.

Proper documentation can help find defects and save project costs. It can also help improve the efficiency of the system testing process.

Here are some other tips for system documentation:

(i) Document test results

Document the lessons learned and test results from the system testing process. This can help communicate the findings, issues, and recommendations.

(ii) Include test data and environment information

Thorough test plans can help improve quality and reduce development and maintenance expenses.

(iii) Use clear and concise language:

Avoid jargon, acronyms, slang, or abbreviations that may confuse readers.

5.3**TESTING WEB-BASED SYSTEMS**

Q8. Define webtesting. State various types of web testing?

Ans : (Imp.)

Meaning

Web testing is a software testing technique to test web applications or websites for finding errors and bugs. A web application must be tested properly before it goes to the end-users. Also, testing a web application does not only mean finding common bugs or errors but also testing the quality-related risks associated with the application. Software Testing should be done with proper tools and resources and should be done effectively. We should know the architecture and key areas of a web application to effectively plan and execute the testing.

Testing a web application is very common while testing any other application like testing functionality, configuration, or compatibility, etc. Testing a web application includes the analysis of the web fault compared to the general software faults. Web applications are required to be tested on different browsers and platforms so that we can identify the areas that need special focus while testing a web application.

Types

Basically, there are 4 types of web-based testing that are available and all four of them are discussed below:

1. Static Website Testing

A static website is a type of website in which the content shown or displayed is exactly the same as it is stored in the server. This type of website has great UI but does not have any dynamic feature that a user or visitor can use. In static testing, we generally focus on testing things like UI as it is the most important part of a static website. We check things font size, color, spacing, etc. testing also includes checking the contact us form, verifying URLs or links that are used in the website, etc.

(ii) Dynamic Website Testing

A dynamic website is a type of website that consists of both a frontend i.e, UI, and the backend of the website like a database, etc. This

type of website gets updated or change regularly as per the user's requirements. In this website, there are a lot of functionalities involved like what a button will do if it is pressed, are error messages are shown properly at their defined time, etc. We check if the backend is working properly or not, like does enter the data or information in the GUI or frontend gets updated in the databases or not.

(iii) E-Commerce Website Testing

An e-commerce website is very difficult in maintaining as it consists of different pages and functionalities, etc. In this testing, the tester or developer has to check various things like checking if the shopping cart is working as per the requirements or not, are user registration or login functionality is also working properly or not, etc. The most important thing in this testing is that does a user can successfully do payment or not and if the website is secured. And there are a lot of things that a tester needs to test apart from the given things.

(iv) Mobile-Based Web Testing

In this testing, the developer or tester basically checks the website compatibility on different devices and generally on mobile devices because many of the users open the website on their mobile devices. So, keeping that thing in mind, we must check that the site is responsive on all devices or platforms.

Q9. Discuss various Points to be Considered While Testing a Website?

Ans : (Imp.)

As the website consists of a frontend, backend, and servers, so things like HTML pages, internet protocols, firewalls, and other applications running on the servers should be considered while testing a website. There are various examples of considerations that need to be checked while testing a web application. Some of them are:

- Do all pages are having valid internal and external links or URLs?
- Whether the website is working as per the system compatibility?
- As per the user interface-Does the size of displays are the optimal and the best fit for the website?

- What type of security does the website need (if unsecured)?
- What are the requirements for getting the website analytics, and also controlling graphics, URLs, etc.?
- The contact us or customer assistance feature should be added or not on the page, etc.?

Q10. Discuss the Objectives of Web Based Testing.

Ans :

Following are the Objectives of Web Based Testing

1. Testing for functionality

Make that the web application performs as expected for all features and functions. Check that user interface elements like form submissions and navigation work as intended.

2. Testing for Compatibility

To make sure it is compatible, test the web application across a variety of devices, operating systems, and browsers. Verify that the program operates consistently in a range of settings.

3. Evaluation of Performance

Analyze the online application's overall performance, speed, and responsiveness. Any performance bottlenecks, such as slow page loads or delayed server response times, should be located and fixed.

4. Testing for load

Examine how well the web application can manage a particular load or multiple user connections at once. Determine and fix performance problems when there is a lot of traffic.

5. Testing for accessibility

Make sure the online application complies with applicable accessibility standards (e.g., WCAG) and is usable by people with disabilities. Make sure the program can communicate with assistive technologies efficiently.

6. Testing Across Browsers

Make sure the operation and appearance of the web application are consistent by testing it in various web browsers. Determine and fix any problems that might develop with a particular browser.

Q11. Discuss various Steps in Software Testing.

Ans :

Following are the various steps in software testing are :

1. App Functionality

In web-based testing, we have to check the specified functionality, features, and operational behavior of a web application to ensure they correspond to its specifications. For example, Testing all the mandatory fields, Testing the asterisk sign should display for all the mandatory fields, Testing the system should not display the error message for optional fields, and also links like external linking, internal linking, anchor links, and mailing links should be checked properly and checked if there's any damaged link, so that should be removed. We can do testing with the help of Functional Testing in which we test the app's functional requirements and specifications.

2. Usability

While testing usability, the developers face issues with scalability and interactivity. As different numbers of users will be using the website, it is the responsibility of developers to make a group for testing the application across different browsers by using different hardware. For example, Whenever the user browses an online shopping website, several questions may come to his/her mind like, checking the credibility of the website, testing whether the shipping charges are applicable, etc.

3. Browser Compatibility

For checking the compatibility of the website to work the same in different browsers we test the web application to check whether the content that is on the website is being displayed correctly across all the browsers or not.

4. Security

Security plays an important role in every website that is available on the internet. As a part of security, the testers check things like testing the unauthorized access to secure pages should not be permitted, files that are confined to the users should not be downloadable without the proper access.

5. Load Issues

We perform this testing to check the behavior of the system under a specific load so that we can measure some important transactions and the load on the database, the application server, etc. are also monitored.

6. Storage and Database

Testing the storage or the database of any web application is also an important component and we must sure that the database is properly tested. We test things like finding errors while executing any DB queries, checking the response time of a query, testing whether the data retrieved from the database is correctly shown on the website or not.

5.3.1 Testing Off the Shelf Software**Q12. What is off the shelf software? State its disadvantages?**

Ans : (Imp.)

Off-the-shelf software is a product that is entirely readymade and has expanded functionalities that satisfy a large number of users. They aren't one-of-a-kind, but they are universal and designed for mass commercial use, so they can usually be easily integrated with existing systems without requiring complicated configurations.

A diverse set of software modules allows you to meet various requirements, but more than standard features are required for complex specific tasks. You can customize them, but most off-the-shelf products can only partially meet particular needs.

Off-The-Shelf Software Examples

ERP software like Oracle's SAP ERP, Microsoft Dynamics, and Sage Intact.

Insightly, HubSpot CRM, and Zoho CRM are ready-made, user-friendly CRM suite solutions.

Gmail and Microsoft are basic and generic software products that provide standard features such as emailing and file sharing. These custom off the shelf software solutions are secure solutions that enable businesses to create private accounts.

Custom software vs off-the-shelf software

Let's start with their essences to see how they differ. Off-the-shelf software is a ready-to-use solution. These tools are designed for a wide range of audiences and business sectors. They are based on best practices that have been proven effective by multiple users.

However, off-the-shelf apps only occasionally meet all of the requirements of a specific task or area.

Well-known examples include Google and Microsoft solutions and CRM systems such as Salesforce, HubSpot, Oracle, and Zoho.

On the other hand, custom software demonstrates the most personalized approach, as tailor-made software can specifically consider business demands. The entrepreneur should assemble or hire a dedicated team to develop an application for specific requirements to obtain it.

The disadvantages of off-the-shelf software**1. Cost-cutting measures**

Off the shelf computer software is initially less expensive. Still, its underlying costs may accumulate over time due to the need for ongoing service payments, licensing, and the connection of additional users. Expansion of features and migration to newer software versions will undoubtedly raise the cost of your initial tariff plan.

2. A compelled trade-off

Because the available features may or may not fully meet your needs, you lose some flexibility. Workflows must be adapted or drastically altered to fit the software, not vice versa.

3. Complete dependency on the provider

The solution will be updated and changed at any moment its vendor decides to, without the consent of your enterprise modernization plans. Some updates may adversely affect your system. There is also a risk that the software developer may stop supporting the application, and you will have to look for an alternative vendor.

4. Closing Thoughts

Both off the shelf software and custom software provide users with various solutions and benefits. When you need to implement software quickly, need an all-inclusive solution, and want a readily available and extensive support system, off-the-shelf software is helpful.

Testing off-the-shelf software (COTS) is important to ensure that the software meets the needs of your business and users. Here are some things to consider when testing COTS software:

5. Validation

Test the software in real-world environments and cases to determine if it's fit for use.

6. Risk tolerance

Consider the system's risk tolerance and how to mitigate potential damage.

7. Detailed records

Keep detailed records of test plans, cases, and results to help manage feedback loops.

5.4**TESTING IN A MULTIPLATFORM ENVIRONMENT****Q13. What is Cross-Platform Testing? State its importance?**

Ans :

Cross-platform testing involves validating the functionality and performance of an application across different platforms, including various operating systems (such as Windows, iOS, Android, macOS, and Linux), browsers (Chrome, Firefox, Safari, Edge), and devices (such as smartphones, tablets, and desktops from diverse brands). The goal is to ensure that the application provides a consistent user experience, regardless of the platform being used.

This type of testing focuses on several key aspects of an application, including functionality, usability, and user interface, across different environments. By doing so, it helps ensure that users will have a seamless experience with the application, no matter which platform they are using.

Importance

- With the global expansion of internet usage, users access applications from a wide range of devices, browsers, and operating systems. To ensure that your application functions flawlessly on different platforms, it must be rigorously tested across these various environments. Starting with the most popular and widely used platforms is a strategic approach to reach your target audience effectively.
- Cross-platform testing is a crucial aspect of software quality assurance, encompassing both cross-platform browser testing and cross-platform device testing for mobile and desktop applications.

- It aims to identify and resolve issues related to usability, consistency, user interface, and performance across different devices, browser versions, and operating systems.

Without proper cross-platform testing, an application that performs well on one platform may malfunction on another, leading to a loss of users, decreased web traffic, lower revenues, and negative reviews.

Q14. Cross-Platform Testing Best Practices.

Ans :

(Imp.)

1. Varied User Interfaces Across Platforms

Ensure consistency in design and usability across platforms using UI test cases.

2. Diverse Screen Sizes and Resolutions

Test responsive designs to ensure the app displays correctly on various screen sizes and resolutions.

3. Platform-Specific Security Concerns

Include security testing in your cross-platform testing strategy, focusing on platform-specific vulnerabilities.

4. Integration Problems

Perform integration testing to ensure the application interacts correctly with external systems on all platforms.

5. Platform-Specific Features

Test platform-specific functionality only on the relevant platforms.

6. Different OS Version Support

Prioritize OS versions based on user demographics and perform compatibility testing across these versions.

7. Network Condition Variations

Test the application under different network conditions to ensure consistent performance.

8. Multiple Language Support

Implement localization testing to ensure the application works well in different languages.

9. Handling Updates

Create an update plan that aligns with platform update cycles and perform regression testing after each update.

Q15. Discuss the Tips and Techniques for Efficient Cross-Platform Testing.*Ans :***1. Identify and Prioritize Target Platforms**

Determine the operating systems, web browsers, and devices on which your software will run, and prioritize testing based on market and user data.

2. Use Emulators or Real Devices

Balance the use of emulators and real devices based on your testing needs. Emulators can save costs, while real devices provide more accurate results.

3. Conduct Functional and Non-Functional Testing

Ensure that the application's features work correctly and meet performance, security, and usability requirements across all platforms.

4. Automate Where Possible

Automation can greatly enhance cross-platform testing by allowing tests to be executed consistently across multiple platforms. Tools like Selenium or Appium are valuable for this purpose, but not every test is suitable for automation.

Q16. Discuss the Recommended Cross-Platform Testing Tools.*Ans :***1. Selenium**

An open-source automation testing tool that supports multiple operating systems and browsers.

2. Appium

An open-source tool for automating native, mobile web, and hybrid applications on iOS, Android, and Windows platforms.

3. TestComplete

A versatile automation testing tool for desktop, mobile, and web applications.

4. BrowserStack

A cloud-based testing platform for testing websites and mobile applications across various browsers, operating systems, and real devices.

5. Sauce Labs

A cloud-based platform offering continuous testing for mobile and web applications across various environments.

6. CrossBrowserTesting

A cloud-based testing platform by SmartBear for automated and manual testing across a wide range of browsers and devices.

7. Ranorex

A commercial GUI test automation framework for testing desktop, web, and mobile applications, offering both codeless and advanced scripting capabilities.

5.5**TESTING SECURITY****Q17. Define security testing. Explain the goals of security testing?***Ans :***(Imp.)**

Security testing is a type of software testing that focuses on evaluating the security of a system or application. The goal of security testing is to identify vulnerabilities and potential threats and to ensure that the system is protected against unauthorized access, data breaches, and other security-related issues.

The goal of Security Testing

The goal of security testing is to:

- To identify the threats in the system.
- To measure the potential vulnerabilities of the system.
- To help in detecting every possible security risk in the system.
- To help developers fix security problems through coding.
- The goal of security testing is to identify vulnerabilities and potential threats in a system or application and to ensure that the system is protected against unauthorized access, data breaches, and other security-related issues. The main objectives of security testing are to:
- **Identify vulnerabilities**

Security testing helps identify vulnerabilities in the system, such as weak passwords, unpatched software, and misconfigured systems, that could be exploited by attackers.

➤ **Evaluate the system's ability to withstand an attack**

Security testing evaluates the system's ability to withstand different types of attacks, such as network attacks, social engineering attacks, and application-level attacks.

➤ **Ensure compliance**

Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

➤ **Provide a comprehensive security assessment**

Security testing provides a comprehensive assessment of the system's security posture, including the identification of vulnerabilities, the evaluation of the system's ability to withstand an attack, and compliance with relevant security standards.

➤ **Help organizations prepare for potential security incidents**

Security testing helps organizations understand the potential risks and vulnerabilities that they face, enabling them to prepare for and respond to potential security incidents.

➤ **Identify and fix potential security issues before deployment to production**

Security testing helps identify and fix security issues before the system is deployed to production. This helps reduce the risk of a security incident occurring in a production environment.

3. Penetration Testing

Penetration testing is the simulation of the attack from a malicious hacker. It includes an analysis of a particular system to examine for potential vulnerabilities from a malicious hacker who attempts to hack the system.

4. Risk Assessment

In risk assessment testing security risks observed in the organization are analyzed. Risks are classified into three categories i.e., low, medium, and high. This testing endorses controls and measures to minimize the risk.

5. Security Auditing

Security auditing is an internal inspection of applications and operating systems for security defects. An audit can also be carried out via line-by-line checking of code.

6. Ethical Hacking

Ethical hacking is different from malicious hacking. The purpose of ethical hacking is to expose security flaws in the organization's system.

7. Posture Assessment

It combines security scanning, ethical hacking, and risk assessments to provide an overall security posture of an

8. Application security testing

Application security testing is a type of testing that focuses on identifying vulnerabilities in the application itself. It includes testing the application's code, configuration, and dependencies to identify any potential vulnerabilities.

9. Network security testing

Network security testing is a type of testing that focuses on identifying vulnerabilities in the network infrastructure. It includes testing firewalls, routers, and other network devices to identify potential vulnerabilities.

10. Social engineering testing

Social engineering testing is a type of testing that simulates phishing, baiting, and other types of social engineering attacks to identify vulnerabilities in the system's human element.

Q18. Discuss various Types of Security Testing.

Ans :

1. Vulnerability Scanning

Vulnerability scanning is performed with the help of automated software to scan a system to detect known vulnerability patterns.

2. Security Scanning

Security scanning is the identification of network and system weaknesses. Later on, it provides solutions for reducing these defects or risks. Security scanning can be carried out in both manual and automated ways.

11. Tools such as Nessus, OpenVAS, and Metasploit can be used to automate and simplify the process of security testing. It's important to ensure that security testing is done regularly and that any vulnerabilities or threats identified during testing are fixed immediately to protect the system from potential attacks. Organization.

Q19. State the advantages and disadvantages of security testing?

Ans : (Imp.)

Advantages of Security Testing

1. Identifying vulnerabilities

Security testing helps identify vulnerabilities in the system that could be exploited by attackers, such as weak passwords, unpatched software, and misconfigured systems.

2. Improving system security

Security testing helps improve the overall security of the system by identifying and fixing vulnerabilities and potential threats.

3. Ensuring compliance

Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

4. Reducing risk

By identifying and fixing vulnerabilities and potential threats before the system is deployed to production, security testing helps reduce the risk of a security incident occurring in a production environment.

5. Improving incident response

Security testing helps organizations understand the potential risks and vulnerabilities that they face, enabling them to prepare for and respond to potential security incidents.

Disadvantages of Security Testing

6. Resource-intensive

Security testing can be resource-intensive, requiring significant hardware and software resources to simulate different types of attacks.

7. Complexity

Security testing can be complex, requiring specialized knowledge and expertise to set up and execute effectively.

8. Limited testing scope

Security testing may not be able to identify all types of vulnerabilities and threats.

9. False positives and negatives

Security testing may produce false positives or false negatives, which can lead to confusion and wasted effort.

10. Time-consuming

Security testing can be time-consuming, especially if the system is large and complex.

11. Difficulty in simulating real-world attacks

It's difficult to simulate real-world attacks, and it's hard to predict how attackers will interact with the system.

Q20. What is Data Warehouse testing? Explain various tools of Data warehouse testing?

Ans :

Data Warehouse testing also known as dwh testing is a process of building and executing the data test case strategies to ensure that all comprehensive data in the warehouse has integrity and is reliable, accurate, and consistent within the organization's data framework. Primarily used to validate the reliability of analytical data within an organization, ensuring the trustworthiness of its overall business insights.

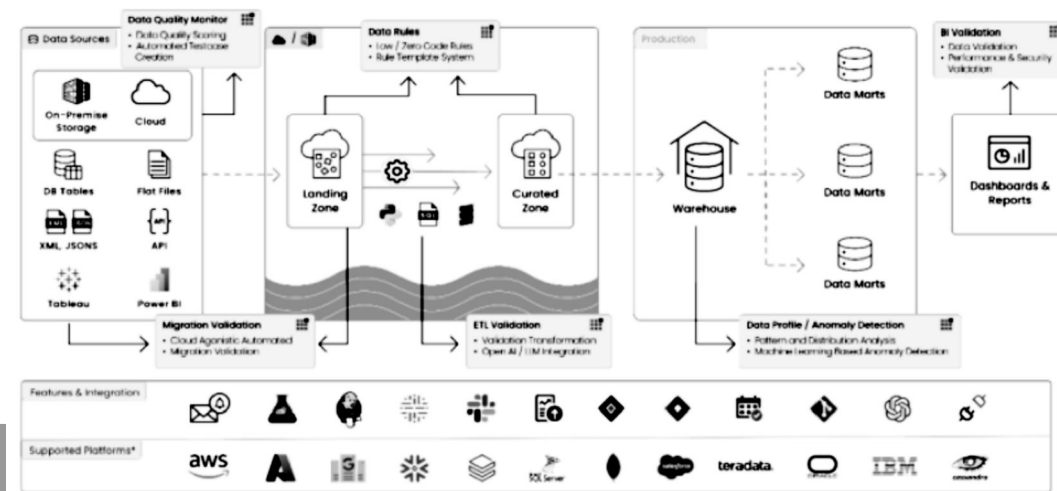
DWH testing is not a simple testing process, it completely checks the data pipelines, when the data is in ETL process operations. ETL testing and data validation process are intermediate stages, it will process the data time issues and resolve them quickly. In data warehouse testing, the scope extends to BI reports and dashboards. It encompasses the ETL testing stages from the data warehouse to data marts, with data validation ensuring the quality of data post-completion of ETL testing operations. In this data warehouse testing strategies encompass both ETL testing and BI Testing.

Tools

Data Warehouse Testing tools (DWH testing tools) are specialized for data-centric systems, aiming to automate testing and certification processes in data warehousing. These tools play a crucial role during the development phase of the data warehouse.

Datagaps DataOps Suite: Datagaps DataOps Suite is a comprehensive solution for data warehousing testing. It proficiently tests both data processes and data within a Data Warehouse. Now, let's delve into the Modern Data Warehouse Testing Scenario.

End to End Data Testing Automation



Q21. Discuss the benefits and challenges to data warehouse testing?

Ans :

(Imp.)

Benefits of data warehouse testing

Although the primary benefit of data warehouse testing is the ability to test data integrity and consistency, there are many advantages to instating a reliable process. For example, data warehouse testing is an extension of the rigorous testing mindset that IT teams apply to aid development and deployment activities.

1. High confidence in data quality, the key building block of a superlative analytical model
2. Data acquisition errors are spotted early on rather than by data analysts or domain experts at a later point, when the cost of fixing issues may be high
3. The financial impact due to subpar data quality can lead to losses in millions — a pegs the number at an average of \$15 million
4. Reputational loss due to bad data can prove to be costly. For example, systemic weaknesses in its reporting systems cost its reputation along with fines to the tune of 5.9 million pounds.
5. Businesses spend huge resources in setting up a data warehouse, and testing data veracity is integral to justifying this dollar spend
6. Non-compliance of regulatory acts can incur heavy penalties. The average cost of a data breach, a possible consequence of not testing data thoroughly, is estimated to be around \$8.19 million in the US, as per a recent

Challenges to data warehouse testing

While there is no doubt that data warehouse testing is important to a successful business, the execution of the process is not often straightforward due to the involved complexities and challenges. Let's discuss the common obstacles:

(i) Heterogeneity of data

As we need to validate data from heterogeneous sources, the testing scenarios can get unwieldy. For example, the source could be an application, OLTP database such as Oracle or DB2, or even a flat file (different formats). It could reside on premises or in the cloud. The destination again could be an in-house data warehouse or a cloud data warehouse such as Amazon Redshift or Microsoft Azure.

The testing framework needs to have the ability to plug in to these diverse systems. Additionally, the testing team members need to have the expertise to deftly switch working between different platforms.

(ii) High volumes and scalability

The number of datasets can get overwhelming as organizations aim to assimilate data into a common repository. The testing framework needs to be able to run through large volumes of data executing stress and regression tests. In addition, it needs to be able to scale as new sources come into the picture or as data load increases.

The pressure of having to deliver test results quickly sometimes leads to shortcut methods such as sampling, in which datasets are tested partially. Although it helps in avoiding the handling of large data volumes, sampling increases the probability of errors in data.

(iii) Data mapping and transformations

Data is often represented differently in systems and data types, and column names tend to vary. Hence, one of the preliminary requirements to activate data warehouse tests is to clearly map data between source and destination databases. This is a difficult task and prone to errors when done manually.

Q22. Explain the process of data ware house testing.

Ans :

Testing a data warehouse is a multi-step process that involves activities such as identifying business requirements, designing test cases, setting up a test framework, executing the test cases, and validating data.

1. Identify the various entry points

As loading data into a warehouse involves multiple stages, it's essential to find out the various entry points to test data at each of those stages. If testing is done only at the destination, it can be confusing when errors are found as it becomes more difficult to determine the root cause.

Examples of entry points: sources, various points between ETL such as before and after transformation, staging database which is often used as a temporary store before loading into the warehouse, and BI engine where reports read from warehouse data.

2. Prepare the required collaterals

Two fundamental collaterals required for the testing process are database schema representation and a mapping document.

The mapping document is usually a spreadsheet which maps each column in the source database to the destination database. It also includes complex SQL queries (might need multiple table joins) that compare the two columns to evaluate if the data has landed correctly inside the destination.

A data integration solution can help generate the mapping document, which is then used as an input to design test cases.

3. Design an elastic, automated, and integrated testing framework

ETL is not a one-time activity. While some data is loaded all at once and some through batches, new updates may trickle in through streaming queues. A testing framework design has to be generic and architecturally flexible to accommodate new and diverse data sources and types, more volumes, and the ability to work seamlessly with cloud and on-premises databases.

Also, integrating the test framework with an automated data solution (that contains features as discussed in the previous section) increases the efficiency of the testing process.

4. Adopt a comprehensive testing approach

The testing framework needs to aim for 100% coverage of the data warehousing process. For instance, although the primary focus here is on the data itself, application components such as ETL tools, reporting engines, or GUI applications need to be included in the testing framework. Also, it's important to design multiple testing approaches such as unit, integration, functional, and performance testing.

The data itself has to be scrutinized for many checks that includes looking for duplicates, matching record counts, completeness, accuracy, loss of data, and correctness of transformation. Additionally, data on BI reports needs to be evaluated for adherence to business rules along with quality checks. Even the metadata and mapping document needs to be validated for accuracy.

5.6**CREATING TEST DOCUMENTATION**

Q23. What is a Test Strategy? Explain the benefits of test strategy?

Ans :

(Imp.)

A test strategy outlines the approach, goals, and standards for testing activities within a project. It provides a structured framework for defining testing scope, objectives, methodologies, and tools, ensuring consistency and alignment with project requirements.

Benefits of Test Strategy

(i) Clarity and Consistency

A test strategy provides a clear structure and standardized approach for testing activities, ensuring that every team member understands the objectives, scope, and expectations. This clarity reduces confusion, aligns team efforts, and maintains consistency across all testing phases, from initial planning to final execution.

(ii) Risk Mitigation

A well-defined test strategy helps prevent issues that could compromise the quality or delay the release of the software by identifying potential risks and setting up contingency plans early. Proactively addressing high-risk areas enables the team to focus on critical functionalities, thus minimizing the impact of potential setbacks on the project.

(iii) Resource Efficiency

A test strategy allocates resources such as tools, personnel, and time more effectively. The strategy helps teams prioritize testing activities based on project needs, avoiding redundant efforts and optimizing team capacity. Efficient resource management helps cost savings and prevents unnecessary delays in the testing lifecycle.

(iv) Stakeholder Alignment

A test strategy is a communication tool for aligning the testing approach with stakeholder expectations. Defining the goals, methods, and scope of testing keeps project managers, developers, and stakeholders informed, ensuring everyone is on the same page about testing objectives and outcomes.

(v) Enhanced Quality Assurance

A strategic approach to testing ensures comprehensive coverage of functionalities, reducing the likelihood of undetected issues in production. This structured methodology enhances the overall quality of the software by focusing on thorough validation and verification that is aligned with user requirements.

Q24. What to Include in a Test Strategy Document?

Ans :

A comprehensive test strategy document guides the testing process and ensures alignment among team members and stakeholders. Key elements to include are:

Scope of Testing

- (i) Define the areas of the application that will undergo testing, outlining both included and excluded features.
- (ii) Specify the testing boundaries, including any limitations affecting testing, like time constraints or restricted access to certain system parts.

Testing Objectives

- (i) Clearly state the goals for the testing phase, such as ensuring compatibility, performance, functionality, or security.
- (ii) Include specific success criteria or benchmarks to indicate the objectives met.

Testing Approach and Methodology

- (i) Describe the approach for different types of testing (e.g., functional, performance, regression, security) and how each type aligns with the objectives.
- (ii) Outline whether tests will be manual, automated, or combined, and provide reasoning for the selected methods.
- (iii) Include details on the testing phases (unit, integration, system, and acceptance testing) and how they will be executed.

Test Environment Setup

- (i) Specify the environments where testing will occur (e.g., development, staging, production-like) and detail any setup requirements.
- (ii) List hardware and software configurations, operating systems, network specifications, and any dependencies required for consistent test results.

Resource Allocation and Roles

- (i) Identify key roles and responsibilities, assigning specific tasks to team members involved in testing.
- (ii) Include any additional resources like third-party vendors, consultants, or external testing teams.

Types of test cases

Test cases can be categorized based on the purpose they serve in testing. As a quality assurance professional, knowing the difference between them helps focus your efforts and choose the right test format.

(i) Functionality test cases

These are the most basic and obvious test cases to write. They ensure that each feature of your system works correctly.

(ii) Performance test case

This test ensures that the system runs fast enough. It makes sure that all system requirements work as expected regarding speed, scalability, or stability.

(iii) Unit test cases

Software developers usually write unit tests for their code to check individual units, for example, modules, procedures, or functions, to determine if they work as expected.

(iv) User interface (UI) test cases

It's important to remember that the user interface is part of the overall system and not just a shell where functionality appears. UI test cases check that each UI element works correctly, displays, and is easy to use.

(v) Security test cases

Security test cases help ensure that a product or system functions properly under all conditions, including when malicious users attempt to gain unauthorized access or damage the system. These test cases safeguard the security, privacy, and confidentiality of data.

(vi) Integration test cases

These ensure that the application components work together as expected. These test cases check whether modules or components integrate seamlessly to form a complete product.

(vii) Database test cases

These test cases ensure that the database meets its functional and non-functional requirements. They make sure database management systems (DBMS) support all business requirements.

(viii) Usability test cases

Usability test cases help check if users can use the application successfully. These determine whether users can easily use the system without difficulty or confusion. They also verify if users can navigate the system using common procedures and functions.

(ix) User acceptance test cases

User acceptance test cases verify that an application satisfies its business requirements before users accept it. These determine whether users accept or reject the output produced by a particular system before release to the live environment.

(x) Regression testing

Regression test cases verify that changes made during development don't cause any existing functionality to stop working. Regression testing happens after changes have been made to existing code to test that all existing or legacy functionality continues to work as expected after making the changes.

Types of Software Testing Strategies

An appropriate test strategy in software testing is vital for effective quality assurance. Here are some common types:

1. Analytical Approach

Analyze requirements, risks, and critical functionalities to determine the testing scope and prioritize test activities accordingly.

2. Model-Based Approach

Utilize modeling techniques to create test models that represent system behavior, interactions, and expected outcomes, facilitating strategic test planning and execution.

3. Regression-Based Approach

Focus on ensuring that modifications or

enhancements to the software do not introduce new defects or impact existing functionalities. Regression test strategy in software testing validates system stability after changes.

4. Risk-Based Approach

Direct efforts to the test strategy in software testing to areas of the software deemed high-risk based on factors like impact, probability, and criticality. This approach optimizes resource allocation and prioritization of testing activities.

5. Agile Testing Strategy

This approach aligns with the Agile methodology, emphasizing iterative and incremental testing throughout development. It involves close collaboration between testers, developers, and stakeholders, enabling continuous feedback and frequent testing cycles.

6. Exploratory Testing Strategy

Exploratory test strategy in software testing is an approach where testers actively explore the application under test, focusing on learning, investigation, and discovery. This strategy involves less pre-planning and more real-time test design, allowing flexibility and adaptability to uncover unexpected defects.

7. User-Centric Testing Strategy

This strategy places the end-user at the center of the testing process. Testers simulate real-world user scenarios and interactions to validate the application's usability, accessibility, and overall user experience. Usability testing, user acceptance testing (UAT), and beta testing are standard techniques employed in this strategy.

8. Compliance-Based Testing Strategy

Compliance-based test strategy in software testing are crucial in the industries such as healthcare or finance. These strategies ensure that the software meets regulatory standards, legal requirements, and industry-specific guidelines. Compliance testing often involves validating data privacy, security measures, and adherence to industry-specific regulations.

9. Continuous Testing Strategy

Continuous test strategy in software testing is a DevOps and Agile-oriented approach that

emphasizes integrating testing throughout the software development lifecycle. It involves running automated tests continuously, incorporating them into the CI/CD pipeline, and providing quick feedback on software quality. Continuous testing enables faster releases, early bug detection, and improved collaboration between development and testing teams.

10. Crowdfunding Strategy

Crowdfunding involves harnessing a community of external testers who test the software across various devices, platforms, and real-world conditions. This strategy leverages testers' diversity and unique perspectives to uncover defects that may go unnoticed with traditional in-house testing teams. Crowdfunding can provide valuable insights from a broader user perspective.

Q25. Discuss the Key Components of a Software Testing Strategy Document Template.

Ans : (Imp.)

To create an effective software testing strategy document, consider incorporating the following components:

➤ Introduction

Provide an overview of the document, including its purpose, intended audience, and project background.

➤ Test Objectives

Clearly define the goals and objectives of the testing effort, aligning them with the project's overall objectives.

➤ Testing Scope

Specify the testing boundaries, including the features, functionalities, platforms, and environments to be covered.

➤ Test Approach

Outline the high-level strategy and techniques to be employed during testing, such as manual testing, automated testing, or both.

➤ Test Deliverables

Enumerate the artifacts and documentation to be produced during the testing process, such as test plans, test cases, and test reports.

➤ **Test Schedule and Timeline**

Provide a timeline for testing activities, including milestones, dependencies, and estimated effort for each testing phase.

➤ **Resource Allocation**

Specify the resources required for testing, encompassing personnel, hardware, software, and tools.

➤ **Risk Assessment**

Identify potential risks and mitigation strategies associated with testing, ensuring proactive risk management.

➤ **Test Environment**

Describe the necessary test environment setup, including hardware, software, network configurations, and data requirements.

➤ **Exit Criteria**

These are the predefined conditions that must be met to conclude the testing process, such as a specific defect density threshold or completion of a predefined set of test cases.

Q26. Discuss the process of testing strategy document.

Ans :

Now that we understand the significance of a test strategy document, let's explore the step-by-step process of creating one:

Step 1: Define the Testing Goals and Objectives

- Clearly define the goals as well as the objectives of the testing effort.
- Understand the project requirements, business goals, and user expectations to align testing activities accordingly.

Step 2: Identify the Testing Scope

- Define the testing boundaries, including the modules or functionalities to be tested, the test levels, and the types of testing required.
- Determine any exclusions or limitations in the testing scope.

Step 3: Determine the Testing Approach

- Choose the appropriate testing approaches and techniques based on the project requirements and objectives.
- Consider risk, complexity, and the software development lifecycle model being used.

Step 4: Specify the Test Environment and Infrastructure

- Identify the hardware, software, and network configurations needed for testing.
- Determine if any specialized tools or resources are required and plan their procurement or setup.

Step 5: Define the Test Data Management Strategy

- Establish guidelines for test data creation, identification of data dependencies, and data management practices.
- Ensure data security, privacy, and compliance with relevant regulations.

Step 6: Outline the Test Execution and Reporting Process

- Detail the test execution methodology, including creating test cases, test execution cycles, and defect tracking.
- Specify the reporting formats, metrics, and KPIs to measure the testing progress and quality.

Step 7: Develop the Test Automation Strategy Document

- Identify the areas suitable for test automation and outline the implementation strategy.
- Choose appropriate tools, define the framework, and allocate resources for test automation.

Step 8: Address Risks and Mitigation Strategies

- Identify potential risks and challenges in the testing process.
- Devise mitigation strategies to minimize the impact of risks on project timelines and deliverables.

Step 9: Define Test Deliverables and Exit Criteria

- Specify the expected test deliverables, including test plans, cases, reports, and other artifacts.
- Establish the criteria for deciding when testing can be considered complete.

Q27. How HeadSpin Empowers Organizations to Enhance Their Testing Strategy

Ans :

HeadSpin, a leading mobile, web, and IoT testing Platform, offers a comprehensive solution that optimizes software testing strategies and empowers organizations

to deliver exceptional user experiences. Let's explore four key capabilities of the HeadSpin Platform that transform software testing.

1. Global Device Infrastructure

HeadSpin provides access to an extensive global device infrastructure, enabling testing on real devices in various locations worldwide. With this capability, organizations can ensure that their applications are thoroughly tested across different devices, operating systems, network conditions, and geographical regions. Testing on real devices eliminates the pitfalls of relying solely on emulators or simulators, providing accurate results and insights into how the application performs in real-world scenarios.

2. Test Automation

HeadSpin Platform offers robust test automation capabilities, empowering organizations to streamline and scale their testing efforts. Through integration with popular test automation frameworks and tools, such as Appium, Selenium, and Espresso, HeadSpin enables the creation and execution of automated test scripts across various devices and platforms. This significantly reduces manual effort, accelerates test cycles, and enhances overall test coverage, leading to faster time-to-market and improved software quality.

3. Performance Testing at Scale

Performance is a crucial aspect of any software application, and HeadSpin excels in providing performance testing at scale. By leveraging its distributed infrastructure and network virtualization capabilities, organizations can simulate a high volume of user traffic, replicate real-world network conditions, and accurately measure application performance under different loads. This empowers teams to proactively identify and address performance bottlenecks, scalability issues, and response time constraints, ensuring optimal user experiences even under demanding conditions.

4. AI-driven Insights and Analytics

HeadSpin leverages the power of artificial intelligence (AI) and machine learning (ML) to provide actionable insights and advanced analytics for software testing. By analyzing vast

amounts of data collected during test executions, the HeadSpin Platform uncovers hidden patterns, trends, and anomalies, allowing organizations to make data-driven decisions to optimize their testing strategies. These AI-driven insights enable teams to prioritize testing efforts, identify improvement areas, and enhance their testing practices' overall effectiveness.

5. Collaborative Platform

HeadSpin provides a centralized Platform that facilitates effective collaboration among team members involved in the testing process. The platform allows testers, developers, and other stakeholders to easily share test artifacts, results, and insights. This promotes efficient communication, knowledge sharing, and collaboration, leading to faster issue resolution and improved overall testing outcomes.

5.7

SOFTWARE TESTING TOOLS

Q28. Discuss briefly about various Software Testing Tools.

Ans :

(Imp.)

Software Testing tools are the tools that are used for the testing of software. Software testing tools are often used to assure firmness, thoroughness, and performance in testing software products. Unit testing and subsequent integration testing can be performed by software testing tools. These tools are used to fulfill all the requirements of planned testing activities. These tools also work as commercial software testing tools. The quality of the software is evaluated by software testers with the help of various testing tools.

Top 10 Software Testing Tools

1. BrowserStack Test Management

BrowserStack Test management is the latest software test management platform that offers a centralized test case repository with the best-in-class UI/UX. Integrates with other BrowserStack software testing tools such as Live, Test Observability, Automate & App Automate.

Features :

- Facilitates two-way integration with Jira, enhancing traceability for test cases and runs.

- Provides a rich dashboard for real-time reports & insights.
- Users can import data from existing tools using APIs or CSVs, with smart parsing for CSV fields.
- Test results can be uploaded from Test Observability or report formats like JUnit-XML/BDD-JSON.
- Supports test automation frameworks such as TestNG, WebdriverIO, Nightwatch.js, Appium, Playwright, etc.
- Integrates with CI/CD tools such as Jenkins, Azure Pipelines, Bamboo & CircleCI.

2. Lambda Test

Lambda Test is an AI-powered test orchestration and execution platform that allows developers and testers to perform manual and automated software testing at scale across different permutations of real browsers, devices, and operating systems.

Features:

- Run your test scripts on a cloud grid using popular test automation frameworks like Selenium, Playwright, Cypress, Appium, and more.
- Accelerate your software release cycles by multiple fold folds with parallel test execution.
- Test locally hosted projects with LambdaTest Tunnel and UnderPass before going live with your websites.
- Leverage the HyperExecute platform to perform end-to-end test orchestration and get high-test execution speed up to 70% faster than traditional cloud grids.
- Integrate LambdaTest with third-party tools like Jira, Asana, Jenkins, GitHub Actions, and more as per your project requirements.

3. Test Grid

Test Grid is a leading cloud-based end-to-end testing and test infrastructure platform designed to streamline and enhance the automated testing of web and mobile applications. The platform integrates seamlessly with leading test automation frameworks like Selenium, Appium, and Cypress, allowing for the automated execution of test scripts and enhancing testing efficiency and reliability.

Features:

- It supports integration with popular CI/CD tools such as Jenkins, CircleCI, and GitLab.
- Offers true scriptless testing for test case generation & execution
- It allows remote access to testers and developers for manual testing and debugging.
- It offers detailed reporting and analytics features for testing results.
- It enables cross-browser and cross-device testing.
- Both private and on-premise browser and mobile cloud infrastructure are available

4. QA Wolf

QA Wolf revolutionizes software testing by offering a powerful fusion of automated testing tools and professional QA services, enabling web application teams to attain 80% end-to-end test coverage at an unprecedented pace.

Features

- Accelerates end-to-end user workflow test coverage to 80% in a matter of weeks
- Runs complete test suites in minutes via parallel execution.
- Completely eliminates flaky tests through proactive maintenance
- Charges are based on achieved test coverage, not hourly
- Frees up developers to focus on feature development
- Speeds up releases and minimizes production bugs
- Tests end-to-end web and native mobile applications including performance, accessibility, Salesforce, SAP, and more.

5. Aqua cloud

aqua cloud is an AI-powered test management solution designed to streamline and enhance the efficiency of QA teams. Offering 100% traceability, aqua helps you manage all testing activities with unparalleled ease and precision. The platform seamlessly integrates with leading test automation frameworks like Selenium, JMeter, Ranorex, and SoapUI, as well as any other tool via REST API, empowering teams without limiting their options.

6. Zephyr Scale

Zephyr Scale is a test management provides a smarter and more structured way to plan, manage, and measure tests inside Jira.

Features

- It offers cross-project integration, traceability, and a structured designed useful in large environments.
- It helps to scale tests in Jira.
- It helps to improve visibility, data analysis, and collaboration.
- It provides detailed changed history, test case versioning, and end-to-end traceability with Jira issues and challenges.

7. Selenium

Selenium provides a playback tool for authoring tests across most web browsers without the need to learn a test scripting language.

Features

- It provides multi-browser support.
- It makes it easy to identify web elements on the web apps with the help of its several locators.
- It is able to execute test cases quicker than the other tools.

8. Ranorex

Ranorex Studio is a GUI test automation framework used for testing web-based, desktop, and mobile applications. It does not have its own scripting language to automate application.

Features

- It helps to automate tests on Windows desktop, then execute locally or remotely on real or virtual machines.
- It runs tests in parallel to accelerate cross-browser testing for Chrome, Firefox, Safari, etc.
- It tests on real iOS or Android devices, simulators, emulators, etc.

9. Test Project

Test Project is a test automation tool that allows users to create automated tests for mobile and web applications. It is built on top of popular frameworks like Selenium and Appium.

Features

- It is a free end-to-end test automation platform for web, mobile, and API testing.
- Tests are saved as local files directly on your machine with no cloud-footprint to get a complete offline experience.
- It helps to create reliable codeless tests powered by self-healing, adaptive wait, and community add-ons.
- It provides insights about release quality, step-by-step detailed report with screenshots and logs.

10. Katalon Platform

Katalon Platform is a comprehensive quality management platform that enables team to easily and efficiently test, launch, and optimize the best digital experiences.

Features

- It is designed to create and reuse automated test scripts for UI without coding.
- It allows running automated tests of UI elements including pop-ups, iFrames, and wait-time.
- It eases deployment and allows wider set of integrations compared to Selenium.

5.8**TAXONOMY OF TESTING TOOLS****Q29. Discuss the concept of Taxonomy Testing Tools.**

Ans :

The taxonomy of testing tools organizes testing tools into categories based on their functionality, purpose, and usage in the software development lifecycle. Below is an outline of this taxonomy:

1. Test Management Tools

- **Purpose:** To manage the testing process, plan tests, and track test progress.

Examples:

- **Test case management:** TestRail, Zephyr
- **Bug tracking:** Jira, Bugzilla
- **Requirements management:** HP ALM, ReQtest

2. Functional Testing Tools

- **Purpose:** To verify the application's functionality against requirements.

- **Types**
- **Manual Testing Tools:** Excel sheets, test case templates
- **Automated Functional Testing Tools:** Selenium, QTP/UFT, TestComplete
- 3. Non-Functional Testing Tools**
- **Purpose:** To evaluate the application's performance, security, usability, etc.
- **Types**
- **Performance Testing Tools:**
- **Load testing:** JMeter, LoadRunner
- **Stress testing:** BlazeMeter, Gatling
- **Security Testing Tools:**
- **Vulnerability scanning:** OWASP ZAP, Nessus
- **Penetration testing:** Metasploit, Burp Suite
- **Usability Testing Tools:**
- **Heatmaps and user interaction:** Hotjar, Crazy Egg
- 4. Test Automation Tools**
- **Purpose:** To automate repetitive testing tasks.
- **Examples:**
- Selenium, Appium, Katalon Studio, Robot Framework
- 5. API Testing Tools**
- **Purpose:** To test APIs for functionality, performance, and security.
- **Examples:**
- Postman, SoapUI, REST Assured
- 6. Unit Testing Tools**
- **Purpose:** To test individual components or modules of code.
- **Examples:**
- JUnit (Java), NUnit (.NET), PyTest (Python), Jasmine (JavaScript)
- 7. Mobile Testing Tools**
- **Purpose:** To test mobile applications for functionality, compatibility, and performance.
- **Examples:**
- Appium, Espresso, TestProject

8. Continuous Integration/Continuous Deployment (CI/CD) Tools

- **Purpose:** To integrate and automate the testing process in the CI/CD pipeline.
- **Examples**
- Jenkins, CircleCI, TravisCI

9. Code Quality Analysis Tools

- **Purpose:** To analyze code for potential defects or adherence to coding standards.
- **Examples**
- SonarQube, Checkmarx, Coverity

10. Specialized Testing Tools

- **Purpose:** For niche testing areas.
- **Types:**
- **Compatibility Testing:** BrowserStack, Sauce Labs
- **Data Testing:** QuerySurge, Talend
- **Accessibility Testing:** AXE, WAVE

11. Simulation and Emulation Tools

- **Purpose:** To mimic real-world scenarios and systems for testing.
- **Examples**
- **Network simulators:** Packet Tracer, GNS3
- **Device emulators:** Android Emulator, iOS Simulator

12. Collaboration and Reporting Tools

- **Purpose:** To enhance communication and report sharing among the testing team.
- **Examples:**
- Slack, Microsoft Teams, Confluence

5.9

METHODOLOGY TO EVALUATE AUTOMATED TESTING TOOLS

Q30. Discuss the Methodology to Evaluate Automated Testing Tools.

Ans :

Test automation framework is an application that allows you to write a series of tests without worrying about the constraints or limitations of the underlying

test tools. Regardless of the automation tool, you would be able to achieve your automation desires and objectives. Our automation expert building framework to include the following:

- Common codebase and database
- Exception and error handling
- Modularized reusable, and maintainable code and data
- Standardized test idioms
- Configurable test suites
- Global test properties

One of the main advantages of our proprietary framework is

- Optimization of testing efforts through unattended execution
- Customized reporting and Test evidence recording
- Multi-environment run with minimal configuration changes
- Metrics data collation and analysis
- Comprehensive documentation facilitating quick learning and easy maintenance

Test automation has undergone several stages of evolution, both in the development of marketable test tool technologies and in the development of test automation processes and frameworks within individual QA organizations. The typical path followed is described below:

➤ **Record and Playback**

Monitoring of an active user session, recording user inputs related to objects encountered in the user interface, and storing all steps and input data in a procedural script. This method is useful in learning how to use a test tool, but the scripts produced are difficult to maintain after the application under test changes and do not produce reliable, consistent results.

➤ **Test Script Modularity**

Creating small, independent scripts that represent modules, sections, and functions of the application-under-test; then combining them in a hierarchical fashion to construct larger tests. This represents the first step toward creating reusable test assets.

➤ **Test Library Architecture**

Dividing the application under test into procedures and functions – also known as objects and methods depending on your implementation language – instead of a series of unique scripts. This requires the creation of library files that represent modules, sections, and functions of the application under test. These files, often referred to as function libraries, are then called directly from within test case scripts. Thus, as elements of the application change, only the common library components which reference them must be changed, not multiple test scripts with hard-coded references which might be difficult to locate and validate.

➤ **Data-Driven Testing**

Reading input and output values from data files or tables into the variables used in recorded or manually coded test scripts. These scripts include navigation through the application and logging of test status. This abstraction of data from the test script logic allows testers with limited knowledge of the test tool to focus on developing, executing and maintaining larger and more complex sets of test data. This increase in organizational efficiency fosters enhanced test coverage with shorter test cycles.

➤ **Keyword-Driven Testing**

Including test step functionality in the data driven process by using data tables and keywords to trigger test events. Test steps are expressed as Object '! Action '!Expected Result. The difference between data-driven and keyword-driven testing is that each line of data in a keyword script includes a reference that tells the framework what to do with the test data on that line. The keyword attached to the test step generally maps to a call to a library function using parameters read in from the data file or table. One major benefit is the improved maintainability of the test scripts: by fully modularizing automation of each step, it's easier to accommodate any user interface changes in the application under test.

Automation Framework Describes

How test data will be handled

Script naming conventions

Test script creation standards

Script organization

As noted earlier, one of the challenges facing test automation is to speed up testing processes while increasing the accuracy and completeness of tests. The evolution of test automation frameworks has been driven by accepting this challenge.

Data-Driven Frameworks

This type of functional test automation framework abstracts the data layer from the test script logic. Ideally, only data used as inputs to test objects and outputs from test events would need to change from one iteration to the next. The types of test scripts used in this architecture are described below.

Driver Script

- Performs initialization of the test environment (as required)
- Calls each Test Case Script in the order specified by the Test Plan
- Controls the flow of test set execution

Test Case Script

- Executes application test case logic using Business Component Function scripts
- Loads test data inputs (function parameters) from data files and tables
- Evaluates actual result based on the expected result loaded from data files and tables

Business Component Function Script

- Exercises specific business process functions within an application
- Issues a return code to indicate result or exception
- Uses parameter (input) data derived from data files and tables

Common Subroutine Function Script

- Performs application specific tasks required by two or more business component functions
- Issues a return code to indicate result or exception
- Uses parameter (input) data derived from data files and tables

User-Defined Function Script

- Contains logic for generic, application-specific, and screen-access functions
- Can include code for test environment initialization, debugging and results logging

In this architectural model, the “Business Component” and “Common Subroutine” function scripts invoke “User Defined Functions” to perform navigation. The “Test Case” script would call these two scripts, and the “Driver” script would call this “Test Case” script the number of times required to execute Test Cases of this kind. In each case, the only change between iterations is in the data contained in the files that are read and processed by the “Business Function” and “Subroutine” scripts.

Keyword-driven Frameworks

This type of framework builds on the data-driven framework by including business component functionality in the data tables. Keywords are used within each test step to trigger specific actions performed on application objects.

Driver Script

- Governs text execution workflow
- Performs initialization of the test environment (as required)
- Calls the application-specific Action (“Controller”) Script, passing to it the names of the business process test cases. These test cases can be stored in spreadsheets, delimited text files, or database records.

Action Script

- Acts as the “controller” for test case execution
- Reads and processes the business process test case name received from the Driver Script
- Matches on “key words” contained in the input dataset
- Builds a list of parameters from values included with the test data record
- Calls “Utility” scripts associated with the “key words”, passing the created list of parameters

Utility Scripts

- Process the list of input parameter received from the Action Script
- Perform specific tasks (e.g. press a key or button, enter data, verify data, etc.), calling “User Defined Functions” as required
- Record any errors encountered during test case execution to a Test Report (e.g. data sheet, table, test tool UI, etc.)

- Return to the Action Script, passing a result code for processing status (e.g. pass, fail, incomplete, error)

User-Defined Function Libraries

- Contain code for general and application-specific functions
 - May be called by any of the above script-types in order to perform specific tasks
 - Can contain business rules
- Mercury Quality Center with TestDirector and QuickTest Professional
- Business Process Testing uses a role-based model, allowing collaboration between non-technical Subject Matter Experts and QA Engineers versed in QuickTest Pro.

5.10

LOAD RUNNER

Q31. What is LoadRunner? How does Load Runner work?

Ans :

Load Runner is one of the oldest performance testing tools on the market. It is used to test the behavior and performance of applications under load. You can create your performance test scripts with its logger or you can do some coding from scratch using your own development language. Then, of course, analyze the test results with reporting tools. Since Load Runner has been on the market for a long time, it supports many different technologies.

LoadRunner works on the concept of recording and replaying user activities and generating the desired load on the server. It simply simulates the actions of the user in the real world and creates a virtual load, helping to determine the performance of the software application or system. The main steps include:

1. Recording/Scripting

To record user action in a script.

2. Test Execution

Replaying the script along with the virtual payload to simulate the real world situation in the test environment.

3. Result Analysis

To provide accurate result in terms of load carrying capacity and responsiveness of the application.

LoadRunner simulates real user activities in the form of scripts (program) and runs these scripts by creating virtual users (threads/processes). These virtual users are known as 'Vusers'. During performance test execution, Vusers runs synchronously and generates the traffic on the server. Upon completion of the test, LoadRunner compiles the results and saves them to a file (called RAW results). This file can be opened through the Microfocus Analysis tool and further analysis can be performed on the test result. Finally, the Analysis tool generates the report (in pdf, HTML, excel, etc. format) that concludes the test result.

Pros of LoadRunner

Here are some of the most prominent benefits for LoadRunner

- LoadRunner can accurately detect system-level, end-user, and code-based bottlenecks in code and intent fixtures
- Helps minimize the cost of application downtime by identifying the root cause of application performance issues
- Allows testing of legacy software with updated technologies
- Allows easy mobile app testing
- LoadRunner helps reduce software and hardware costs by informing its users about the full capacity and scalability of its software.
- Enables teams to develop smart service-level deals before their products are released
- It offers its users short test cycles that help accelerate application delivery across global test resources.
- It makes optimum use of load generators and gives better load test results.
- Optimum use of load builder farm

Cons of LoadRunner

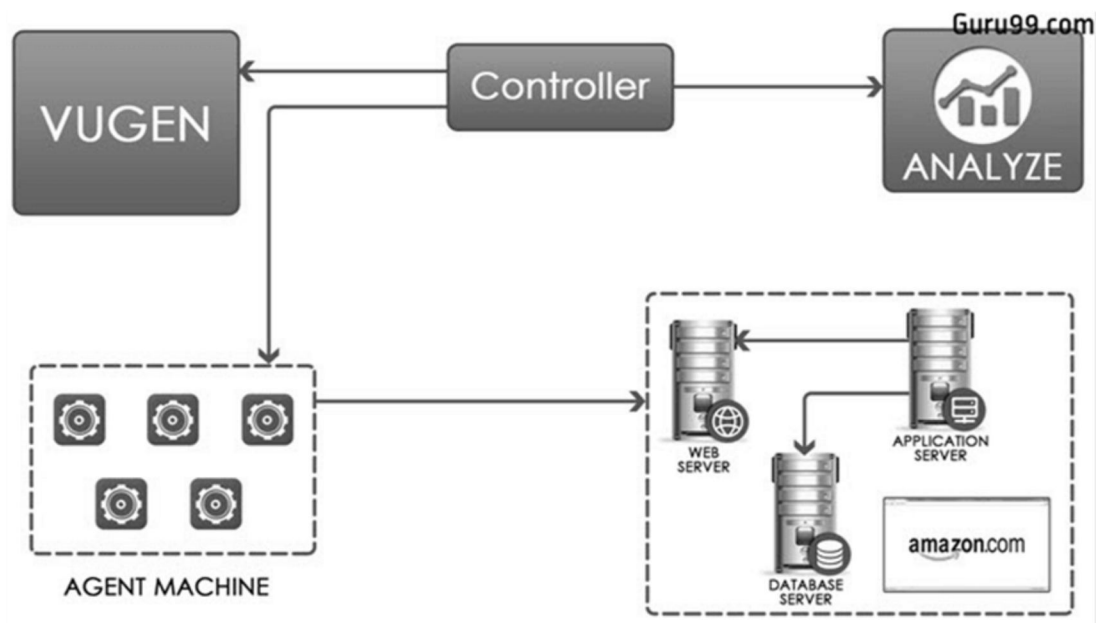
- Much more expensive compared to other test tools available on the market
- Debugging features need fundamental improvements for better results.
- LoadRunner crashes a lot when the system cannot meet the computing requirements
- It uses a lot of memory space and has major compatibility issues with other frameworks.
- It has a limited load generation capacity.

Some features of LoadRunner are

- Reduce hardware and software costs by accurately estimating system capacity
- Quickly and accurately identifies root cause of application performance issues
- Has effective team usage tracking
- Browser-based access to global testing resources and optimal use of load generator farm.

What is LoadRunner Architecture?

Broadly speaking, the architecture of HP LoadRunner is complex, yet easy to understand.

**LoadRunner Architecture Diagram**

Suppose you are assigned to check the performance of Amazon.com for 5000 users

In a real-life situation, these all these 5000 users will not be at homepage but in a different section of the websites. How can we simulate differently.

VUGen

VUGen or Virtual User Generator is an IDE (Integrated Development Environment) or a rich coding editor. VUGen is used to replicate System Under Load (SUL) behavior. VUGen provides a “recording” feature which records communication to and from client and Server in form of a coded script – also called VUser script.

So considering the above example, VUGen can record to simulate following business processes:

1. Surfing the Products Page of Amazon.com
2. Checkout
3. Payment Processing
4. Checking MyAccount Page

Controller

Once a VUser script is finalized, Controller is one of the main LoadRunner components which controls the Load simulation by managing, for example:

- How many VUsers to simulate against each business process or VUser Group
- Behavior of VUsers (ramp up, ramp down, simultaneous or concurrent nature etc.)
- Nature of Load scenario e.g. Real Life or Goal Oriented or verifying SLA
- Which injectors to use, how many VUsers against each injector
- Collate results periodically
- IP Spoofing
- Error reporting
- Transaction reporting etc.

Taking an analogy from our example controller will add the following parameter to the VUGen Script

1. 3500 Users are Surfing the Products Page of Amazon.com
2. 750 Users are in Checkout
3. 500 Users are performing Payment Processing
4. 250 Users are Checking MyAccount Page ONLY after 500 users have done Payment Processing

Even more complex scenarios are possible

1. Initiate 5 VUsers every 2 seconds till a load of 3500 VUsers (surfing Amazon product page) is achieved.
2. Iterate for 30 minutes
3. Suspend iteration for 25 VUsers
4. Re-start 20 VUsers

5. Initiate 2 users (in Checkout, Payment Processing, MyAccounts Page) every second.
6. 2500 VUsers will be generated at Machine A
7. 2500 VUsers will be generated at Machine B

Agents Machine/Load Generators/Injectors

HP LoadRunner Controller is responsible to simulate thousands of VUsers – these VUsers consume hardware resources for example processor and memory – hence putting a limit on the machine which is simulating them. Besides, Controller simulates these VUsers from the same machine (where Controller resides) & hence the results may not be precise. To address this concern, all VUsers are spread across various machines, called Load Generators or Load Injectors.

As a general practice, Controller resides on a different machine and load is simulated from other machines. Depending upon the protocol of VUser scripts and machine specifications, a number of Load Injectors may be required for full simulation. For example, VUsers for an HTTP script will require 2-4MB per VUser for simulation, hence 4 machines with 4 GB RAM each will be required to simulate a load of 10,000 VUsers.

Taking Analogy from our Amazon Example, the output of this component will be Analysis

Once Load scenarios have been executed, the role of “Analysis” components of LoadRunner comes in.

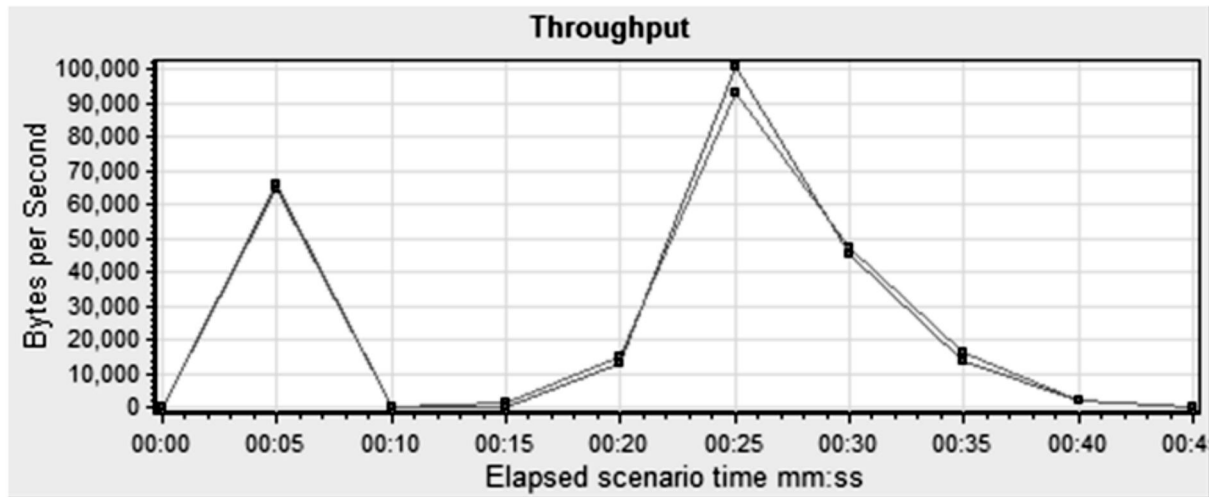
During the execution, Controller creates a dump of results in raw form & contains information like, which version of LoadRunner created this results dump and what were configurations.

All the errors and exceptions are logged in a Microsoft access database, named, output.mdb. The “Analysis” component reads this database file to perform various types of analysis and generates graphs.

These graphs show various trends to understand the reasoning behind errors and failure under load; thus help to figure whether optimization is required in SUL, Server (e.g. JBoss, Oracle) or infrastructure.

Below is an example where bandwidth could be creating a bottleneck. Let’s say Web server has 1GBps capacity whereas the data traffic exceeds this capacity

causing subsequent users to suffer. To determine system caters to such needs, Performance Engineer needs to analyze application behavior with an abnormal load. Below is a graph LoadRunner generates to elicit bandwidth.

**5.11****WIN RUNNER**

Q32. Explain the concept of Win Runner?

Ans :

Win Runner is the most used Automated Software Testing Tool.

Main Features of Win Runner are

- Developed by Mercury Interactive
- Functionality testing tool
- Supports C/s and web technologies such as (VB, VC++, D2K, Java, HTML, Power Builder, Delphi, Cibell (ERP))
- To Support .net, xml, SAP, Peoplesoft, Oracle applications, Multimedia we can use QTP.
- Winrunner run on Windows only.
- Xrunner run only UNIX and linux.
- Tool developed in C on VC++ environment.
- To automate our manual test win runner used TSL (Test Script language like c)

The main Testing Process in Win Runner is

1. Learning

Recognition of objects and windows in our application by winrunner is called learning. Winrunner 7.0 follows Auto learning.

2. Recording

Winrunner records over manual business operation in TSL.

3. Edit Script

Depends on corresponding manual test, test engineer inserts check points in to that record script.

4. Run Script

During test script execution, winrunner compare tester given expected values and application actual values and returns results.

5. Analyze Results

Tester analyzes the tool given results to concentrate on defect tracking if required.

5.12**RATIONAL TESTING TOOLS****Q33. Describe the concept Rational Testing Tools**

Ans :

Rational Testing Tools are a suite of software testing tools developed by IBM under the Rational Software brand. These tools are designed to support various stages of the software development and testing lifecycle, including requirements management, functional testing, performance testing, and test management. Below is an overview of key Rational Testing Tools and their functions:

1. Rational Functional Tester (RFT)

- **Purpose:** Automated functional and regression testing.
- **Key Features**
- Scriptless testing using visual record-and-playback.
- Supports testing of web, desktop, and mobile applications.
- Integration with other Rational tools like Rational Quality Manager.
- Supports Java and VBScript for scripting.

2. Rational Quality Manager (RQM)

- **Purpose:** Test planning, test execution, and test lifecycle management.
- **Key Features**
- Centralized test management system.
- Integration with Rational Functional Tester and Rational Performance Tester.
- Role-based access and customizable workflows.
- Comprehensive reporting and analytics.

3. Rational Performance Tester (RPT)

- **Purpose:** Performance and load testing of web and server-based applications.

- **Key Features**
- Simulates multiple virtual users to measure system performance under load.
- Generates detailed performance reports and bottleneck analysis.
- Integration with other Rational tools for end-to-end testing.

4. Rational Test Workbench

- **Purpose:** Provides a comprehensive set of tools for test automation and service virtualization.

- **Key Features**
- Includes capabilities for API testing, performance testing, and mobile testing.
- Service virtualization to simulate system dependencies.
- Enables continuous integration and delivery workflows.

5. Rational Test Virtualization Server

- **Purpose:** Service virtualization for simulating unavailable systems or components.
- **Key Features:**
- Allows testing in incomplete environments by virtualizing dependent services.
- Reduces costs and delays in test environment setup.

6. Rational Test Automation Server

- **Purpose:** Centralized management and execution of automated tests.

- **Key Features**
- Enables scheduling and monitoring of automated test suites.
- Supports DevOps and continuous testing strategies.
- Integration with Rational Quality Manager and Rational Functional Tester.

7. Rational ClearQuest

- **Purpose:** Defect and change tracking system.
- **Key Features:**
- Tracks defects, enhancements, and change requests throughout the lifecycle.

- Customizable workflows and reporting.
- Integration with Rational tools and third-party systems.

8. Rational ClearCase

- **Purpose:** Configuration and version control.
- Key Features
- Manages source code and software artifacts across teams.
- Integration with Rational testing tools for traceability.

9. Rational Requirements Composer (Now part of IBM Engineering Requirements Management DOORS Next)

- **Purpose:** Requirements management and traceability.
- Key Features:
- Captures, manages, and tracks requirements.
- Links requirements to test cases and other artifacts for traceability.

10. Rational Software Architect

- Purpose: Design and modeling tool.
- Key Features:
- Supports UML modeling for software architecture and design.
- Integrates with Rational testing tools to align design and testing efforts.

Integration with DevOps

IBM Rational Testing Tools are designed to integrate seamlessly into DevOps workflows. They support continuous integration (CI) and continuous delivery (CD) pipelines by enabling automated testing, version control, and performance monitoring.

5.13

JAVA TESTING TOOLS

Q34. Describe the concept of Java testing tools?

Ans : (Imp.)

In Java, the code can be smaller or larger that depends on the functionality. If a user requires small functionality, the code will be smaller in length and easy to do testing. But if a user requires more functionality in the application, the code will be larger in length and not

so easy to do testing. There are various testing tools like JUnit, Mockito, and Selenium for testing Java codes.

Below are the top 10 testing tools or framework which are best and essential to test the Java code.

1. JUnit
2. Mockito
3. Selenium
4. TestNG
5. Spock Framework
6. Cucumber
7. FitNesse
8. Arquillian
9. DBUnit
10. Rest Assured

Let's understand each tool one by one and get the difference between these tools.

1. JUnit

JUnit promotes the idea of "first testing then coding", which emphasizes setting test data for a piece of code that can be tested first and then implemented. JUnit increases the stability of the code. It also increases the productivity of the programmer.

Features

These are the following features of JUnit:

- An open-source framework used to write and run tests.
- For testing the expected result, the JUnit provides assertions.
- To identify the test methods, it provides annotation.
- We can write the code faster for increasing quality using JUnit.
- For running tests, it provides test runners.
- It is very simple, not so complex and requires less time.

2. Mockito

Mockito is a Java-based library or mocking framework that is mainly used to perform unit testing of Java applications. Mockito allows us to add mock data or dummy functionality to the mock interface to perform unit testing.

In order to create a dummy object for a given interface, Mockito uses Java reflection. The mock objects are the proxy of the actual implementations. Testing the functionality of a class without requiring a database connection is referred to as Mocking. For performing the Mocking of the real service, mock objects are used.

These are the following benefits of using the Mockito for testing:

3. Selenium

Selenium is another important testing suite. Selenium is an open-source Web User Interface for performing automation testing of the Java application. Selenium can be easily deployed on platforms like Linux, Windows etc. Selenium also supports multiple browsers like Firefox, Chrome, and Microsoft Edge etc. In Java, Selenium is mostly used for automated functional tests.

Selenium is very easy to integrate with tools such as Jenkins, Maven and Docker to achieve a consistent testing approach. Tools like TestNG and JUnit help in structuring selenium tests for easy maintenance and report generation.

Below are some features of Selenium which make it useful for automation testing.

- For controlling the speed of test cases, Selenium provides help to the user.
- It allows us to execute the entire Test Suite.
- Selenium helps to run the currently selected test.
- It helps step into each specific command in the test script.
- It helps group all the Selenese Commands together and make them execute as a single operation.

4. Test NG

TestNG is a special testing framework that is derived from JUnit and NUnit. It defines some more functionality compared to JUnit and NUnit that make TestNG more powerful and easier to use.

TestNG is also used for performing automation testing. In the name of this framework, NG represents NextGeneration. TestNG is similar to the JUnit but an extension of it. TestNG removed the limitations of JUnit and NUnit like frameworks.

These are the following features of TestNG:

- Just like JUnit, TestNG also supports the annotations.
- TestNG supports the testing of integrated classes.
- TestNG has a flexible runtime configuration.
- TestNG supports several features like load testing, dependent test methods, partial failure and parallel testing.
- TestNG separates compile-time test code from runtime configuration/data info.

5. Spoke Framework

Spoke Framework is another important testing framework for the Java application. The Mockito framework, which we discussed above, is not a complete testing framework for Java applications. Unlike Mockito, the Spoke framework is a complete testing framework for both the Java and Groovy code. The Spoke framework function on its own, which means Mockito works in addition to JUnit.

These are the following features of the Spoke framework that make it more useful in comparison to JUnit:

- Spoke has more readability in comparison to JUnit.
- It has a clear, documented code.
- There is no need to use a third party.
- Easy and fast to perform API testing.
- It is inspired by several frameworks and tools like Scala, Groovy, JUnit and jMock etc. So, it has features of all these frameworks.

6. Cucumber

Cucumber is another special tool for performing testing of Java applications. A Behavior Driven Development tool for developing test cases is referred to as Cucumber. It is one of the essential tools for testing the behavior of the Java application. It is mainly used to design test cases, but however, it also plays a supporting role in automation testing.

For observing the functionality of the application, the Cucumber tool follows the BDD(Behavior Driven Development) framework. In Cucumber testing, we write the test cases in the Gherkin language, which is very easy and simple to understand.

7. FitNesse

FitNesse is another important testing tool. It is a new tool that is mainly focused on requirements and acceptance testing. It is used to specify and verify the application acceptance criteria. It is a bridge between different disciplines in a software delivery process.

Its test-execution capabilities allow us to verify documentation against the software to ensure that the document remains up-to-date and does not experience regression.

These are the following features of the FitNesse tool:

- It is easy to use the wiki web server.
- It is very easy to setup. We just need to download the application, i.e., the Java jar file.
- It uses the specifications and requirements as test input.
- It supports all the major languages like Java, Python, and C# etc.

8. Arquillian

Arquillian is another testing platform design for JVM. It is a highly innovative and extendible tool that allows us to create automated integration, functional and acceptance tests for Java. We don't need to manage the runtime from the test because it allows us to run the test in the runtime.

9. DBUnit

DBUnit is another testing tool to test Java applications. It is an extension of JUnit, which is mainly used to test those applications which are highly dependent on the database.

Many times, we need to run all our test cases multiple times, and the data comes from the database. In such cases, there is a high risk of corrupting the database. The DBUnit helps us to get rid of the risk of corrupting the database. DBUnit fetches the data from the database and test that data.

10. Rest Assured

Rest Assured is also a Java-based library that is mainly used to test Restful Web Services. For accessing Restful Web Service, Rest Assured library work as a headless client. By using the Rest Assured library, we can create a highly customizable HTTP request to send to the Restful server.

These are the following features of Rest Assured:

- It allows us to validate JSON responses according to the schema.
- It provides timeouts in tests.
- It allows us to use Groovy closures or Java 8 lambdas to validate the response.
- It can test different authentication methods and SSL.
- It allows us to work with HTTP headers.
- It works with cookies.
- It allows us to work with XML data instead of JSON.
- It allows us to write custom de-serializers.

5.14**JMETRA**

Q35. Define JMetra. Explain the features of J Metra.

Ans :

Apache JMeter is a pure Java-based open-source software application used for testing load testing functional behavior and measuring performance. It is used for testing a variety of services, i.e. web applications, databases, etc.

Features

Features of JMeter that make it so powerful are mentioned below:

1. Performance Testing

Performance is one of the most important features of any application, so performance testing plays a vital role in testing any application. JMeter tool has this feature of Performance Testing.

2. Load Testing

Load Testing is where we assess how the application will perform when it faces a load of users and activity, and JMeter is good with this type of testing both with web applications and databases.

3. Functional Testing

Functional Testing checks if the functionality of the application is up to the mark. It ensures that the application behaves as expected and delivers the desired features and functionalities to the user.

4. Protocol Support

Protocol Support is the ability of an application to exchange data with other systems using specific communication protocols. Like protocols supported by JMeter are mentioned below:

- Web: HTTP, HTTPS, FTP
- Web Services: SOAP/XML-RPC
- Network Devices: TCP/IP, Ethernet
- API: REST, GraphQL

5. Scripting

Scripting allows us to create custom test scenarios which can't be created by pre-built components, because of this feature JMeter is able to test wide range scenarios and corner cases.

6. Distributed Testing

Distributed Testing is technique which allows to distribute the load for testing into multiple machines rather than a single machine. These multiple machines are controlled by a single parent machine. It is powerful technique used JMeter check the real time consequences of load.

7. Graphical Reporting

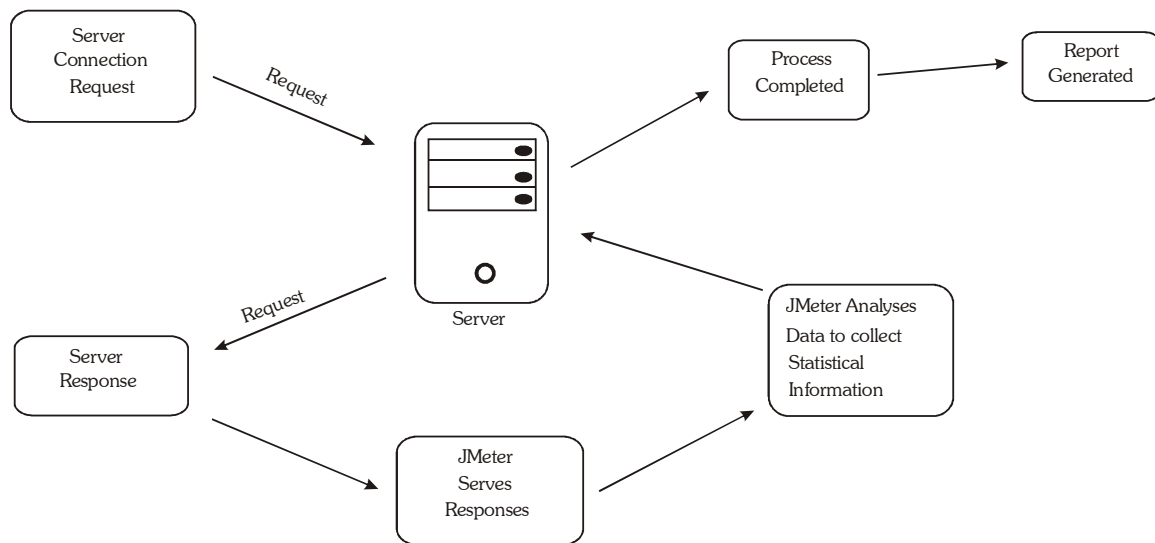
Graphical Reporting is where we use Interactive visual to represent the report to understand performance trends, identify bottlenecks, and optimize your application's performance.

Q36. Explain the Applications of Apache JMeter

Ans :

Applications of Apache JMeter are not limited as it covers the wide range of systems mentioned below:

1. **Web Application Testing:** Web Application covers many frameworks web testing can be divided into Load Testing, Stress Testing, Functional Testing, API Testing, etc.
2. **Mobile Application Testing:** Wide range features of JMeter supports various protocols, scripting which effectively helps in Mobile Testing. Some uses of JMeter testing in Mobile Application are Performance Testing, Battery Consumption Testing, etc.
3. **Database Testing:** It is the process in we evaluate how a database responds under different workloads and conditions. It is very beneficial for improving performance and for enhancing the user experience. And, JMeter is one of the tool used for doing so, as it offers various plugins and customization operation to improve the database testing.
4. **Other Application:** Apart from Web , Mobile and Database JMeter can be used for other applications like Desktop Applications , Testing Network Protocols, etc.

**5.15****JUNIT AND CACTUS**

Q37. What is JUnit? Discuss the features of Junit?

Ans :

JUnit is a unit testing open-source framework for the Java programming language. Java Developers use this framework to write and execute automated tests. In Java, there are test cases that have to be re-executed every time a new code is added. This is done to make sure that nothing in the code is broken.

JUnit

JUnit has several graphs that represent the progress of a test. When the test runs smoothly, the graph displays a green color, and it turns red if the test fails. JUnit Testing enables developers to develop highly reliable and bug-free code.

Unit testing, as the name suggests, refers to the testing of small segments of code. Here, a unit indicates the smallest bit of code that can be fetched out of the system. This small bit can be a line of the code, a method, or a class. The smaller the chunk of code, the better it is, as smaller chunks will tend to run faster. And this provides a better insight into the code and its performance.

- This makes the code more readable, reliable, and bug-free
- It boosts the confidence of the developer and motivates them immensely

Features of JUnit

There are several features of JUnit that make it so popular. Some of them are as follows:

Open Source Network: JUnit is an open-source network that enables developers to write codes fast and with better quality.

- **Provides Annotations:** It provides several annotations to identify test methods.
- **Provides Assertions:** There are assertions to test expected results.
- **Provides Test Runners:** JUnit has test runners to run tests.

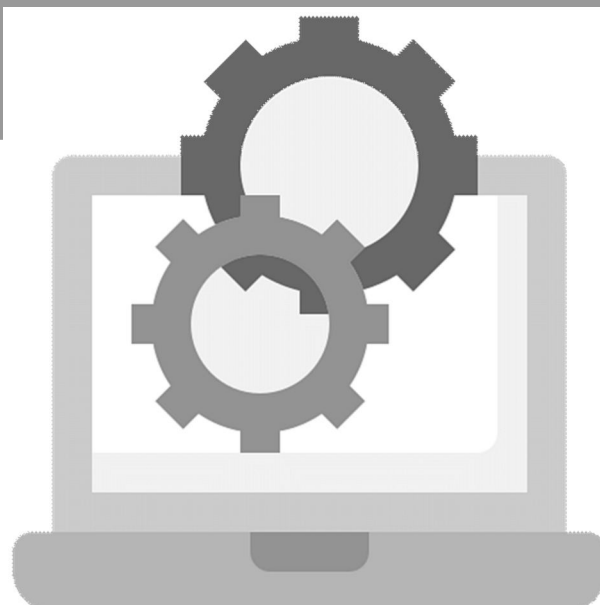


➤ **Improves Code Quality**

JUnit is the most popular testing framework for efficient testing. It allows faster code writing, which results in an increase in the code's quality.

➤ **Automated Test Running**

The test results do not require manual checking. All the tests run automatically on JUnit, the results obtained again automatically checked, and it provides feedback.



➤ **Easily interpretable results**

The test results are represented interactively by showing test progress in a bar, thus making them easily interpretable.

Moving on, let's have a look at JUnit Annotations.

JUnit Annotations

- JUnit Annotations refer to the syntactic meta-data added to the Java source code for better structure and readability. Here, syntactic meta-data refers to the type of data representing the structure of a file with references to bytes, data types, and data structures.
- The main point of difference between JUnit4 and JUnit3 is the introduction of JUnit Annotations.

@ Test	Tells JUnit which public void method can be run as a test case
@ Before	To execute some statement before each test case
@ After	To execute some statement after each test case
@ Ignore	Used to ignore some statements during test execution

The Cactus Testing Tool is a testing framework that was specifically designed for testing server-side Java components, such as Java Servlets, Enterprise JavaBeans (EJBs), and JSPs. It was part of the Apache Jakarta Project and focused on unit testing in a container-based environment.

Introduction to Cactus Testing Tool

- **Purpose:** To perform in-container testing of server-side Java code.
- **Developer:** Apache Software Foundation (as part of the Jakarta Project).
- **Type:** Unit testing framework with container support.
- **Core Idea:** Tests are executed in the actual environment where the application is deployed, ensuring realistic testing of server-side components.

Key Features

1. In-Container Testing

- Executes tests inside the application server or servlet container to replicate the actual runtime environment.
- Provides higher fidelity testing compared to mocks or standalone tests.

2. Servlet Redirection Mechanism

- Routes test requests through a proxy servlet to control and execute test logic inside the container.

3. Support for Java Components

- Java Servlets
- JSPs
- Enterprise JavaBeans (EJBs)
- Custom server-side Java code

4. Integration with JUnit

- Built on top of JUnit, it leverages JUnit's features while extending it for server-side testing.

5. Test Isolation

- Tests are isolated in the context of the container, ensuring no interference between tests.

Workflow of Cactus

1. Test Runner

- Starts the test execution process.
- Sends requests to the server-side container.

2. Redirector

- Acts as a middle layer between the test runner and the container.
- Executes the test logic within the server-side container.

3. Response Validation

- The framework validates responses and behaviors against expected results.

Advantages

- Realistic testing in a production-like environment.
- Simplifies testing of complex server-side components.
- Reduces reliance on mocks or stubs for testing.

Disadvantages

- Configuration complexity compared to modern frameworks.
- Performance overhead due to in-container testing.
- Deprecation: As of now, Cactus is no longer actively maintained, and modern frameworks like JUnit, TestNG, and Spring Boot Testing have largely replaced it.

Alternatives

Given that Cactus is outdated, consider using:

- JUnit or TestNG for standalone and unit testing.
- Mockito for mocking server-side dependencies.
- Spring Boot Test for comprehensive testing in Spring-based applications.
- Arquillian for in-container testing in modern Java EE applications.

FACULTY OF INFORMATICS
M.C.A. II Year III Semester Examination
Model Paper - I
SOFTWARE QUALITY AND TESTING

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice

ANSWERS

- | | | |
|----|---|-------------------|
| 1. | (a) What are software quality challenges? Explain. | (Unit-I, Q.No. 4) |
| | (b) What is software ? Explain about the types of software. | (Unit-I, Q.No. 1) |

OR

- | | | |
|----|---|--------------------|
| 2. | (a) Write about the strategies to over come quality assurance testing challenges. | (Unit-I, Q.No. 5) |
| | (b) What is the Quality Attribute of a software? | (Unit-I, Q.No. 10) |
| 3. | (a) What are the various requirements of software quality. | (Unit-II, Q.No. 5) |
| | (b) What are procedures and work instructions? Why do we use them. | (Unit-II, Q.No. 9) |

OR

- | | | |
|----|---|----------------------|
| 4. | (a) Explain the importance of staff training and certification to improve the software quality. | (Unit-II, Q.No. 16) |
| | (b) Explain briefly about configuration management. | (Unit-II, Q.No. 18) |
| 5. | (a) What is Malcom Balridge? Explain. | (Unit-III, Q.No. 10) |
| | (b) Write ISO 9000 standards for quality management. | (Unit-III, Q.No. 3) |

OR

- | | | |
|----|---|----------------------|
| 6. | (a) What is Six Sigma? Explain the Characteristics of Six Sigma. | (Unit-III, Q.No. 12) |
| | (b) Explain about SQA project protocol standards. | (Unit-III, Q.No. 15) |
| 7. | (a) What are Testing Methodologies? Explain. | (Unit-IV, Q.No. 9) |
| | (b) What is Test Execution? Explain the impor-tance test execution? | (Unit-IV, Q.No. 22) |

OR

- | | | |
|----|--|---------------------|
| 8. | (a) Explain the concept of test execution cycle. | (Unit-IV, Q.No. 27) |
| | (b) How do you evaluate the test effectiveness? | (Unit-IV, Q.No. 33) |

9. (a) Discuss various Types Client-Server Test. (Unit-V, Q.No. 4)
- (b) What is Data Warehouse testing? Explain various tools of Data warehouse testing? (Unit-V, Q.No. 20)

OR

10. (a) What is Client Server Testing? Discuss the various scenarios of client serves testing? (Unit-V, Q.No. 1)
- (b) What is off the shelf software? State its disadvantages? (Unit-V, Q.No. 12)

FACULTY OF INFORMATICS**M.C.A. II Year III Semester Examination****Model Paper - II****SOFTWARE QUALITY AND TESTING**

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)**Note :** Answer all the question according to the internal choice**ANSWERS**

1. (a) Discuss the causes of software error. (Unit-I, Q.No. 3)
(b) What are the factors for product quality? (Unit-I, Q.No. 8)

OR

2. (a) What is software quality? Explain the factors affecting software quality factors. (Unit-I, Q.No. 6)
(b) Write about the classification of software errors. (Unit-I, Q.No. 2)
3. (a) Explain different types of SDLC models. (Unit-II, Q.No. 1)
(b) Explain the difference between procedures and work instructions? (Unit-II, Q.No. 13)

OR

4. (a) What are the uses of checklists in software quality Checklists. (Unit-II, Q.No. 15)
(b) Explain the components of project progress control? (Unit-II, Q.No. 22)
5. (a) What is Capability Maturity Model Integration (CMMI)? (Unit-III, Q.No. 6)
(b) What is PCMM? Explain various methods of PCMM. (Unit-III, Q.No. 9)

OR

6. (a) Explain various methodologies of six sigma (Unit-III, Q.No. 14)
(b) Discuss the various levels of CMMI. (Unit-III, Q.No. 7)
7. (a) Explain how to determining your software testing techniques. (Unit-IV, Q.No. 12)
(b) Explain the Software Testing Strategy Document. (Unit-IV, Q.No. 3)

OR

8. (a) What is Acceptance Testing? Explain various types of acceptance testing? (Unit-IV, Q.No. 29)
(b) Discuss the techniques and challenges of requirement phasing testing? (Unit-IV, Q.No. 17)

9. (a) Define webtesting. State various types of web testing? (Unit-V, Q.No. 8)
(b) Discuss the Recommended Cross-Platform Testing Tools. (Unit-V, Q.No. 16)

OR

10. (a) Discuss the Basic Characteristics of Client Server Testing Architecture (Unit-V, Q.No. 3)
(b) Discuss various Types of Security Testing. (Unit-V, Q.No. 18)

FACULTY OF INFORMATICS**M.C.A. II Year III Semester Examination****Model Paper - III****SOFTWARE QUALITY AND TESTING**

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)**Note :** Answer all the question according to the internal choice**ANSWERS**

1. (a) What is Quality Assurance (QA)? (Unit-I, Q.No. 9)
 (b) What are various techniques of SQA? (Unit-I, Q.No. 13)

OR

2. (a) Write the overview of a Software Quality Assurance. (Unit-I, Q.No. 14)
 (b) What is McCall's Software Quality Model? (Unit-I, Q.No. 7)

3. (a) What are the Factors affecting intensity of quality assurance activities in the development process? (Unit-II, Q.No. 2)
 (b) Explain about work instructions and work instructions manuals. (Unit-II, Q.No. 11)

OR

4. (a) Explain about various corrective and preventive actions to improve the software quality (Unit-II, Q.No. 17)
 (b) What is document control? Explain. (Unit-II, Q.No. 21)
 5. (a) Explain briefly about software quality Metrics. (Unit-III, Q.No. 1)
 (b) What is Malcom Balridge? Explain. (Unit-III, Q.No. 10)

OR

6. (a) Explain about IEEE Software engineering standards. (Unit-III, Q.No. 16)
 (b) What is PCMM? Explain various methods of PCMM. (Unit-III, Q.No. 9)
 7. (a) Define requirement phase testing. State the objectives, activities of requirements Phase Testing. (Unit-IV, Q.No. 16)
 (b) What are the steps to develop a test strategy document. (Unit-IV, Q.No. 8)

OR

8. (a) State the uses, advantages and disadvantages of acceptance testing. (Unit-IV, Q.No. 30)
 (b) Explain briefly about test execution priorities. (Unit-IV, Q.No. 26)

9. (a) Explain various challenges of client server testing. (Unit-V, Q.No. 6)
(b) Discuss the Tips and Techniques for Efficient Cross-Platform Testing. (Unit-V, Q.No. 15)

OR

10. (a) Discuss various Points to be Considered While Testing a Website? (Unit-V, Q.No. 9)
(b) Discuss the process of testing strategy document. (Unit-V, Q.No. 26)