BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 5

DCA3103

SOFTWARE ENGINEERING

Unit 9

Software Testing Assurance

Table of Contents

1. INTRODUCTION

Software testing assurance is an important phase in the software engineering process that guarantees the dependability and quality of software programmes. It includes employing a systematic and organised strategy to check software against predetermined requirements, find errors, and make ensuring the product lives up to user expectations. Organisations may produce high-quality software solutions that satisfy customer expectations and promote commercial success by integrating software testing assurance practises into the development process.

1.1 Learning Objectives:

After studying this unit, students should be able to:

- *Identify the key components and activities involved in conducting a quality audit.*
- *Explain the importance of performance monitoring in ensuring the quality and reliability of software applications.*
- *Explain the importance of Verification and Validation (V&V) in software testing assurance.*

2. QUALITY AND CONFIGURATION AUDITS

Software testing methodologies depend mainly on quality and configuration audits to make sure that the programme follows established procedures, is up to code quality requirements, and is configured correctly. They support the enhancement of software products' overall quality and dependability by highlighting potential areas for improvement, lowering risks, and improving safety.

A ***Quality Audit,*** commonly referred to as a quality assurance (QA) audit, is a systematic examination of software assets and procedures to evaluate the fulfilment of established quality standards, best practices, and quality requirements. A quality audit's goal is to find any holes or flaws in the software development process and confirm that the programme is of the intended quality.

Key aspects of a quality audit include:

i. *Process Compliance:* The audit determines whether the techniques, standards, and development procedures established for the software project are being followed correctly. Examining the implementation of coding conventions, documentation standards, testing processes, and other relevant regulations are all part of this process.

ii. *Quality Standards:* The audit looks at how well the software conforms with the established quality standards, which may include performance, dependability, security, maintainability, and usability standards. It evaluates whether the programme performs as anticipated and satisfies user needs.

iii. *Issues & Defects:* The audit identifies any problems with the programme and evaluates their impact and severity. It assists in identifying and fixing issues early in the development lifecycle to stop them from getting worse and impairing the quality of the finished product.

iv. *Process Improvement:* The audit offers comments and suggestions for enhancing processes to increase software quality. It helps in pinpointing areas that need improvement, streamlining development procedures, and putting corrective measures in place to prevent similar problems in the future.

A ***Configuration Audit,*** also known as a configuration management audit, focuses on verifying the integrity and consistency of software configuration items (SCIs) throughout the software development lifecycle. It ensures that the software is developed and delivered in a controlled and traceable manner.

Key aspects of a configuration audit include:

i. *Configuration Management Plan:* The audit assesses whether the project has a well-defined configuration management plan in place. This plan outlines the processes, tools, and procedures for managing and controlling the software configuration items, including version control, change management, and release management.

ii. *Configuration Identification:* The audit confirms that the software configuration items are correctly recognised, labelled, and controlled. It makes sure that every item has a distinctive identity and that the right versions of the objects are being utilised during development and testing.

iii. *Configuration Control:* The audit examines whether changes to the software and its components are properly managed and controlled. It ensures that changes are requested, reviewed, approved, and implemented following established change management processes.

iv. *Configuration Baselines:* The audit guarantees that configuration baselines are established and kept up to date at relevant points in the software development lifecycle. It confirms that the baselines accurately represent the programme and its approved versions so that future updates and comparisons can use them as a starting point.

v. *Traceability and Auditability:* The audit assesses whether there is proper traceability and auditability of changes made to the software configuration items. It ensures that changes are documented, recorded, and tracked, allowing for easy identification and resolution of issues.

The Key aspects of Quality Audit and Configuration Audit are shown in Figure 1.

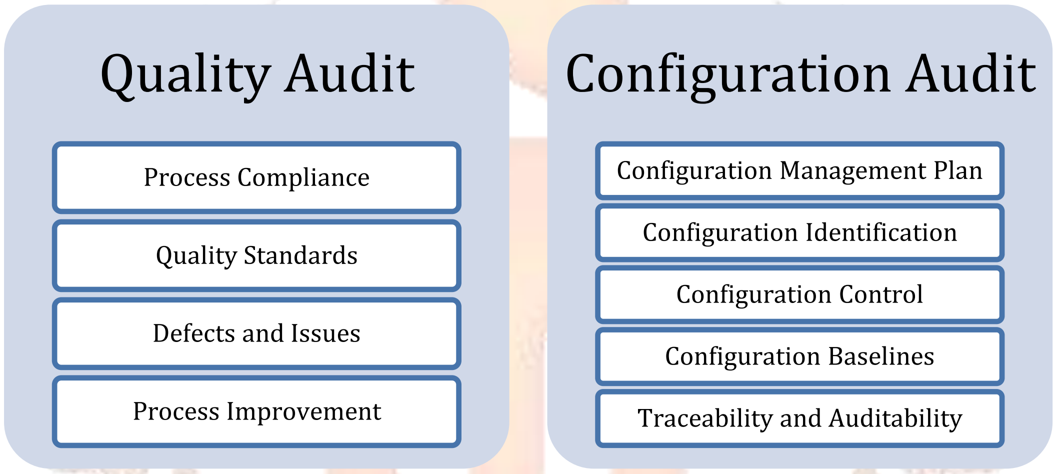

Figure 1: Key Aspects of Quality Audit and Configuration Audit

3. PERFORMANCE MONITORING

Performance monitoring in software testing assurance involves the ongoing assessment, evaluation, and improvement of a software application's performance attributes. Software engineers can use it to find performance problems, improve software parts, and make sure the programme satisfies performance standards. Organisations can provide high-performing and dependable software solutions that match user expectations and business objectives by tracking performance and optimising it.

Important Performance Monitoring Elements in Assurance of Software Testing are:

i. ***Performance Metrics:*** Performance metrics are essential elements in performance monitoring. They provide quantitative measurements of various performance aspects of the software application.

The important performance metrics are shown in Figure 2 and are:

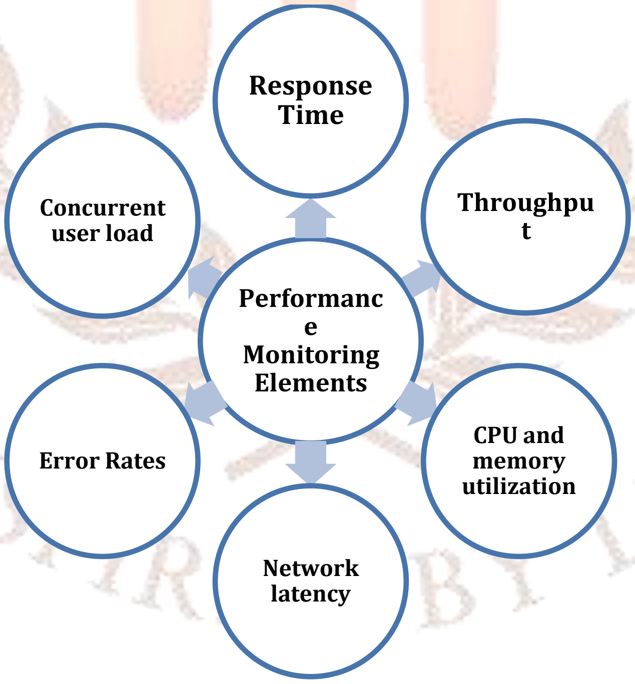

Figure 2: Performance Monitoring elements

a. ***Response time:*** The time taken for the system to respond to a user request.

b. ***Throughput:*** The number of transactions or operations performed per unit of time.

c. ***CPU and memory utilization:*** The extent to which the software utilizes system resources.

d. ***Network latency:*** The time data travels between network endpoints.

e. ***Error rates:*** The frequency of errors or failures encountered during operation.

f. ***Concurrent user load:*** The number of users accessing the software simultaneously.

ii. ***Performance Baselines:*** Performance monitoring requires the creation of performance baselines. Performance baselines serve as benchmarks for the software's expected performance features under typical operating locations. Departures and abnormalities can be found by comparing current performance data to baselines, enabling targeted analysis and optimisation.

iii. ***Performance Profiling:*** To identify performance bottlenecks, performance profiling involves examining the software code and system components. It assists in identifying specific components of the software, such as ineffective algorithms, high resource utilisation, or database queries, that cause poor performance. Software engineers can concentrate their optimisation efforts on key regions by means of performance profiling.

iv. ***Scalability Testing:*** A key component of performance monitoring is scalability testing. It involves assessing how effectively the software can manage increasing workloads and user loads. The software's scalability can be ascertained through scalability testing, which also helps to discover performance constraints and make sure the software can continue to operate effectively under changing demand conditions.

v. ***Load Testing:*** Another crucial component of performance monitoring is load testing. It involves modelling a workload that corresponds to expected user behaviour and usage patterns and placing the software application under it. The performance and behaviour of the programme are assessed through load testing, which enables the detection of capacity and performance bottlenecks.

vi. ***Performance Analysis and Tuning:*** Performance analysis and tuning involve analysing performance metrics, profiling results, and optimizing the software to improve its performance. This process may include optimizing algorithms, improving database queries, fine-tuning system configurations, or utilizing caching mechanisms.

Performance analysis and tuning aim to address identified performance bottlenecks and optimize the software's overall performance.

vii. ***Reporting and Alerting:*** Mechanisms for reporting and alerting are essential for efficient performance monitoring. Reports, dashboards, and alerts are frequently provided by performance monitoring technologies to stakeholders to share performance metrics, trends, and anomalies. The development and operations teams can work together more effectively and take prompt action thanks to these reports and alerts.

4. VERIFICATION AND VALIDATION

Verification refers to the set of tasks that ensure that software correctly implements a specific function.

Boehm [Boe81] states this another way:

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

Verification activities aim to ensure that the software is built correctly and according to its specifications. It involves conducting various tasks to verify that the software correctly implements the intended functionality. Verification typically includes activities such as code reviews, inspections, unit testing, and system testing. The focus is on confirming that the software meets the defined requirements, adheres to coding standards, and operates as expected.

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

Validation: "Are we building the right product?"

Validation activities focus on determining if the software being developed aligns with the customer's requirements and expectations. It involves validating that the software satisfies the intended purpose and delivers the desired value to the end-users or stakeholders. Validation activities often include user acceptance testing (UAT) and other forms of customer or end-user feedback. The objective is to ensure that the software meets the customer's needs, addresses the intended problem, and provides the desired benefits.

Verification and validation include a wide array of SQA (Software Quality Assurance) activities:

- *Technical Reviews:* Conduct formal or informal reviews of software products, such as requirements, design documents, and code, to identify defects, inconsistencies, and improvement opportunities.

- *Quality And Configuration Audits:* Performing systematic examinations of software products and processes to assess compliance with quality standards, best practices, and configuration management guidelines.
- *Performance Monitoring:* Collecting and analysing performance metrics to evaluate the software's responsiveness, resource utilization, scalability, and efficiency.
- *Simulation:* By simulating real-world situations, simulation techniques can be used to evaluate the software's functionality and behaviour under various circumstances.
- *Feasibility Study*: Evaluating a software project's technical, financial, and operational feasibility before allocating resources to its creation, and documentation review.
- *Database Review:* Assessing the design, structure, and performance of the software's underlying database to ensure data integrity, efficiency, and reliability.
- *Algorithm Analysis:* assessing the effectiveness, accuracy, and performance of algorithms used in the software, particularly those with computationally demanding or critical components.
- *Development Testing:* To find and fix bugs and make sure the programme works as intended, various testing activities, such as unit testing, integration testing, and system testing, are carried out during the software development process.
- *Usability Testing:* Evaluating the software's user interface, interaction design, and overall user experience to ensure it is intuitive, efficient, and meets user needs.
- *Qualification Testing:* Conducting tests to verify that the software meets specified standards, regulations, or industry-specific requirements.
- *Acceptance Testing:* Executing tests to ensure that the programme conforms with the necessary standards, laws, or regulations.
- *Installation Testing:* Testing the software installation process to ensure it can be installed correctly and functions properly in different environments.

Figure 3 shows that verification will be done followed by validation.

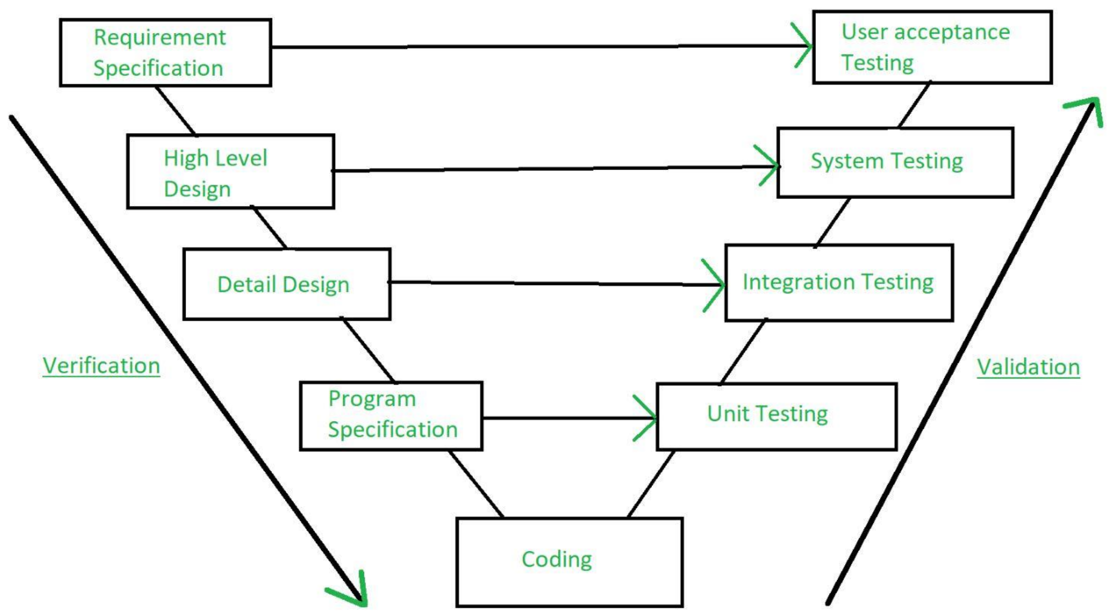

Figure 3: Verification followed by Validation.

5. TEST PLAN

Test Plan is a detailed document describing the test strategy, goals, schedule, estimation, deliverables, and resources needed to test a software product. The test plan helps in estimating the amount of work required to verify the application's quality. The test manager carefully monitors and controls every aspect of the test plan to ensure that software testing activities are carried out according to a defined methodology.

According to the ISTQB (International Software Testing Qualifications Board) definition, "*A Test Plan is a document describing the scope, approach, resources, and schedule of intended test activities*."

5.1 Importance of Test Plan

The software testing process can be improved in numerous ways by the creation of a test plan document.

Here are a few significant benefits:

i. ***Communication and Understanding:*** The Test Plan document serves as a communication tool, enabling stakeholders outside the test team, such as developers, business managers, and customers, to understand the testing process in detail. It provides clarity on the objectives, scope, and approach of testing, promoting better collaboration and alignment among project teams.

ii. ***Guiding Testing Efforts:*** The Test Plan document acts as a rule book or a roadmap for testing activities. It guides the thinking and actions of the testing team by providing a structured and documented plan to follow. It helps ensure consistency, adherence to best practices, and proper sequencing of testing phases, leading to more effective and efficient testing efforts.

iii. ***Management Review and Reusability:*** The Test Plan document includes important aspects such as test estimation, test scope, and Test Strategy. These elements can be reviewed by the management team, enabling them to have visibility into the testing process and make informed decisions. Moreover, a well-documented Test Plan can be reused as a template or reference for future projects, saving time and effort in creating new plans from scratch.

iv. ***Risk Identification and Mitigation:*** The Test Plan document provides a platform to identify and assess risks associated with the testing process. By documenting potential risks, their impact, and mitigation strategies, the testing team can proactively address these risks and take necessary precautions. This helps in minimizing potential issues, delays, and failures during testing.

v. ***Test Coverage and Traceability:*** The Test Plan document outlines the test coverage, including the features, functionalities, and areas of the software that will be tested. It ensures that the testing effort covers all the necessary components, requirements, and scenarios. Additionally, the Test Plan establishes traceability between the requirements and the corresponding test cases, enabling clear visibility into the testing coverage and ensuring that all requirements are tested.

vi. ***Consistency and Standardization:*** By documenting the Test Plan, organizations can establish consistent testing practices and standards across projects. This promotes uniformity in testing approaches, methodologies, and documentation, facilitating knowledge sharing, and ensuring that best practices are followed consistently.

vii. ***Documentation of Test Results and Progress:*** The Test Plan document can be used as a reference for documenting and tracking test results, defects, and progress during the testing process. It helps in monitoring the status of testing activities, providing insights into the completion of test milestones, and facilitating reporting to stakeholders.

5.2 Writing a Test Plan

Test Plan is the most important task of the Test Management Process. Users can create a comprehensive test plan as per IEEE 829, ensuring that the testing process is well-defined, structured, and aligned with industry standards as shown in Figure 4.

The steps to create a test plan are as shown below:

i. Analyse the product.

ii. Design the Test Strategy

iii. Define the Test Objectives

iv. Define Test Criteria

v. Resource Planning

vi. Plan Test Environment

vii. Schedule & Estimation

viii. Determine Test Deliverables

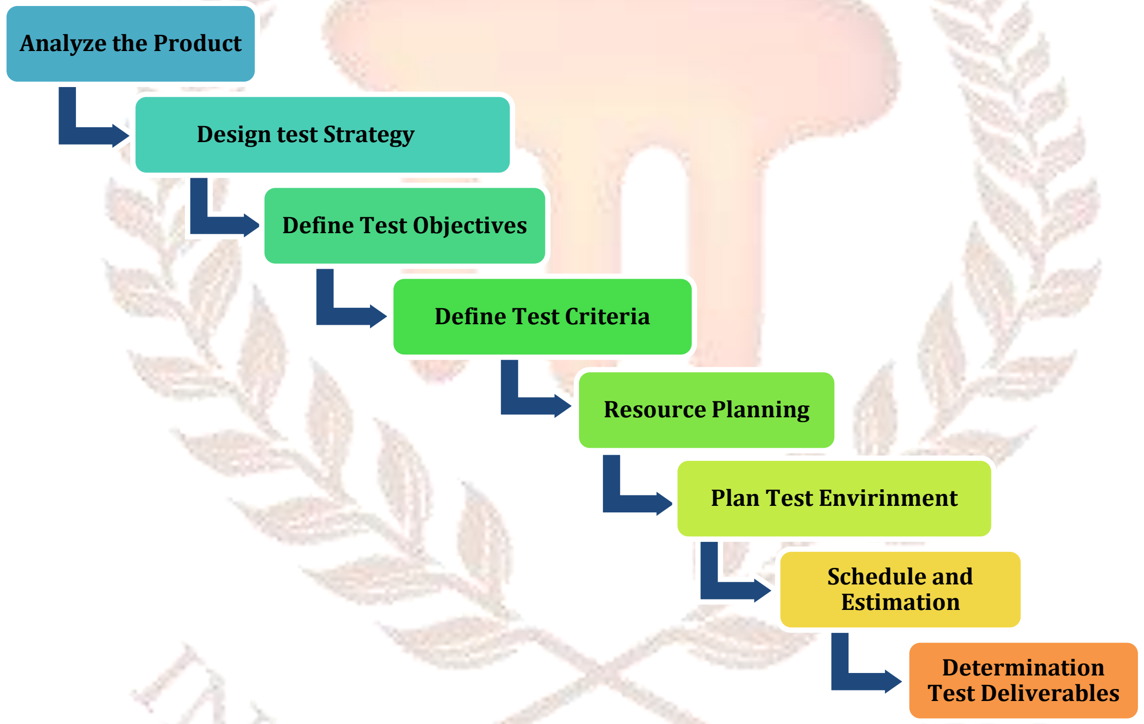

Figure 4: Steps to be followed to write a test plan.

i. ***Analyse the product:*** Understand the product's requirements, architecture, and functionality. Identify key features, risks, and critical areas that need to be tested.

ii. ***Design the Test Strategy:*** Define the overall approach and scope of testing. Determine the test levels, techniques, and methodologies to be used. Consider factors like test prioritization, regression testing, and defect management.

iii. ***Define the Test Objectives:*** Clearly state the goals and objectives of the testing effort. Specify what you aim to achieve through testing, such as identifying defects, validating functionality, or ensuring performance and reliability.

iv. ***Define Test Criteria:*** Establish the criteria that will be used to determine when testing is complete. This includes specifying the exit criteria for each test level, such as achieving a certain level of code coverage or meeting predefined quality metrics.

v. ***Resource Planning:*** Determine the resources required for testing, including the number of testers, their skills, and the availability of testing tools and equipment. Consider factors like time, budget, and expertise needed for effective testing.

vi. ***Plan Test Environment:*** Define the test environment, including hardware, software, network configurations, and any special testing tools or simulators required. Ensure that the test environment accurately reflects the production environment to simulate real-world conditions.

vii. ***Schedule & Estimation:*** Develop a timeline for testing activities, including milestones, deadlines, and dependencies. Estimate the effort required for each testing phase and allocate resources accordingly. Consider factors like testing dependencies, risks, and other project constraints.

viii. ***Determine Test Deliverables:*** Specify the deliverables that will be produced during testing, such as test plans, test cases, test scripts, defect reports, and test summary reports. Define the format and content of each deliverable to ensure consistency and completeness.

5.3 Test Strategies

Test strategies outline how testing will be conducted, what techniques and methodologies will be employed, and how the testing effort will be organized and managed.

Some common test strategies used in software engineering are:

i. ***Black-box Testing:*** This strategy focuses on testing the software without knowing its internal structure or implementation details. Testers evaluate the software based on its inputs and outputs, comparing the expected results with the actual results. It ensures that the software functions correctly from a user's perspective.

ii. ***White-box Testing:*** This strategy involves testing the internal structure, code, and logic of the software. Testers know the software's internal workings and design test cases to exercise specific code paths, branches, and conditions. It helps uncover any defects or vulnerabilities within the software's implementation.

iii. ***Gray-box Testing:*** Gray-box testing combines elements of both black-box and white-box testing. Testers have limited knowledge of the software's internal structure and use this information to design test cases. It allows for a more targeted and effective approach to testing, focusing on critical areas of the software.

iv. ***Regression Testing:*** Regression testing involves retesting the software after changes or modifications have been made to ensure that existing functionalities have not been adversely affected. The goal is to identify any regression defects introduced during the development process.

v. ***User Acceptance Testing (UAT):*** UAT is performed by end-users or stakeholders to determine whether the software meets their requirements and expectations. This strategy validates the software from a user's perspective and ensures its usability, functionality, and overall user experience.

vi. ***Exploratory Testing:*** Exploratory testing is an ad-hoc and simultaneous testing approach where testers actively explore the software's functionalities and features. Testers learn about the system under test as they design and execute tests, providing quick feedback and discovering defects in real time.

vii. ***Performance Testing:*** Performance testing focuses on evaluating the software's performance characteristics, such as responsiveness, scalability, and resource usage, under various workload conditions. It helps ensure that the software meets performance requirements and can handle anticipated user loads.

viii. ***Security Testing:*** Security testing aims to identify vulnerabilities and weaknesses in the software's security mechanisms. It includes techniques such as penetration testing, vulnerability scanning, and security code reviews to assess the software's resistance to potential attacks and unauthorized access.

ix. ***Risk-Based Testing:*** This strategy involves prioritizing testing efforts based on the identification and assessment of risks associated with the software. Testers focus on areas that pose the highest risks to system functionality, user experience, security, or critical business processes.

x. ***Model-Based Testing:*** Model-based testing involves creating models or representations of the software's behaviour and using these models to generate test cases. It helps ensure test coverage, reduces manual effort, and improves the efficiency of the testing process.

6. TESTING METHODS AND TOOLS

In software testing assurance, various testing methods and tools are employed to assess the quality, functionality, performance, and reliability of software applications. Here are some commonly used testing methods and tools:

Testing Methods:

i. ***Unit Testing:*** This method involves testing individual components or modules of the software to ensure they function correctly in isolation. It typically focuses on verifying the smallest units of code, such as functions or methods, through automated or manual testing.

ii. ***Integration Testing:*** Integration testing aims to test the interaction and compatibility between different components or modules of the software. It ensures that the integrated system functions as expected, detecting any interface or communication issues.

iii. ***System Testing:*** System testing involves testing the entire software system. It assesses the software's compliance with specified requirements, functional correctness, and overall system behavior.

iv. ***Acceptance Testing:*** Acceptance testing is performed to validate that the software meets the requirements and expectations of end-users or stakeholders. It typically involves user acceptance testing (UAT) where users execute test cases or scenarios to ensure the software satisfies their needs.

Testing Tools:

i. ***Test Management Tools:*** Test management tools help in organizing and managing testing activities. They assist in creating and maintaining test plans, test cases, test scripts, and test execution reports. These tools often include features for test case management, test coverage analysis, and defect tracking.
Examples include *TestRail, Zephyr, and QTest.*

ii. ***Test Automation Tools:*** Test automation tools enable the creation and execution of automated test cases, reducing manual effort and accelerating testing cycles. These tools provide scripting environments, record and playback capabilities, and integration with test frameworks.

Popular test automation tools include *Selenium, Appium, and TestComplete*.

iii. ***Performance Testing Tools:*** Performance testing tools simulate high user loads and measure the performance characteristics of software applications. They generate virtual users, monitor system behaviour, and collect performance metrics such as response time and throughput.
Commonly used performance testing tools include *Apache JMeter, LoadRunner, and Gatling.*

iv. ***Security Testing Tools:*** Security testing tools assist in identifying vulnerabilities and weaknesses in software applications. They perform scans, tests, and analyses to uncover potential security risks and issues. These tools often include features for penetration testing, vulnerability assessment, and security code review.
Examples include *OWASP ZAP, Burp Suite, and Nessus.*

v. ***Code Analysis Tools:*** Code analysis tools analyze the source code to identify potential defects, coding issues, and adherence to coding standards. They help improve code quality, identify vulnerabilities, and enforce best practices.
Popular code analysis tools include *SonarQube, PMD, and FindBugs.*

vi. ***Test Data Management Tools:*** Test data management tools help in generating and managing test data for testing activities. They provide functionalities for creating, organizing, and manipulating test data, ensuring test coverage and repeatability. Examples include GenRocket, Tricentis Tosca, and Informatica Test Data Management.

vii. ***Continuous Integration/Continuous Delivery (CI/CD) Tools:*** CI/CD tools automate the build, integration, and deployment of software applications. They enable continuous testing by integrating testing activities into the software development lifecycle. These tools often include features for version control, build automation and deployment pipelines.
Popular CI/CD tools include *Jenkins, GitLab CI/CD, and CircleCI.*

viii. ***Test Reporting and Analytics Tools:*** Test reporting and analytics tools provide insights and metrics related to testing activities. They generate reports, dashboards, and visualizations to track testing progress, test coverage, and defect trends.
Examples include *TestFLO, qTest Insights, and Xray.*

7. SUMMARY

- Quality audits involve systematic examinations of software objects and processes to ensure compliance with quality standards and best practices. Configuration audits assess the consistency and integrity of software configuration items. Both audits contribute to ensuring the quality and reliability of software products.
- Performance monitoring involves measuring and analyzing performance metrics to evaluate the responsiveness, throughput, resource utilization, and other performance aspects of software applications. It helps identify bottlenecks, optimize system resources, and ensure that performance requirements are met.
- Verification refers to tasks that ensure the software is built correctly and implements specific functions. Validation ensures that the software being built aligns with customer requirements. V&V encompasses activities that ensure software quality and traceability to customer needs.
- A Test Plan is a document that outlines the approach, objectives, scope, and resources for testing activities. It guides the testing process and ensures consistency, adherence to standards, and proper sequencing of testing phases.
- Test strategies define the overall approach and guidelines for testing. They specify the test levels, techniques, and methodologies to be used, helping ensure effective testing and alignment with project goals.
- Testing methods include techniques used to verify software functionality, performance, security, and usability. Examples include unit testing, integration testing, acceptance testing, and performance testing. Testing methods are selected based on project requirements and objectives.
- Testing tools assist in planning, executing, and managing testing activities. They automate tasks, provide frameworks, and offer features that enhance testing efficiency and effectiveness. Examples include test management tools, test automation tools, performance testing tools, and security testing tools.

8. SELF–ASSESSMENT QUESTIONS

1. Quality audits are conducted to:

   a) Ensure compliance with coding standards

   b) Verify the accuracy of test cases

   c) Assess software performance metrics

   d) Examine software artefacts and processes to ensure compliance with quality standards

2. Configuration audits aim to:

   a) Assess software performance under varying load conditions

   b) Evaluate the consistency and integrity of software configuration items

   c) Identify security vulnerabilities in the software

   d) Determine the effectiveness of the software testing process

3. Performance monitoring is used to:

   a) Analyze the efficiency of code implementation

   b) Identify software defects during the development phase

   c) Evaluate the responsiveness and resource utilization of software

   d) Ensure compliance with coding standards

4. Verification and validation (V&V) in software engineering refer to:

   a) Ensuring that the software is built correctly and implements specific functions

   b) Assessing the responsiveness and scalability of the software

   c) Evaluating the software's compliance with industry regulations

   d) Testing the software for usability and user experience

5. A test plan is a document that:

   a) Identifies and assesses risks associated with the testing process

   b) Guides the testing process and outlines the approach, objectives, and scope of testing activities

   c) Measures the extent to which the source code is tested during test execution

   d) Evaluates the performance characteristics of the software under varying load conditions

6. Test strategies in software engineering include:

   a) Test case prioritization and risk-based testing

b) Assessing the consistency and integrity of software configuration items

c) Identifying security vulnerabilities through penetration testing

d) Measuring and analyzing performance metrics of the software

7. Test automation tools are used to:

a) Evaluate the performance characteristics of the software

b) Create and execute automated test cases, reducing manual effort

c) Assess the compliance of software with coding standards

d) Monitor the software's responsiveness and resource utilization

8. Which testing method focuses on testing the software's internal structure and code?

a) Black-box testing

b) Unit testing

c) System testing

d) Regression testing

9. SELF-ASSESSMENT ANSWERS

1. d) Examine software artefacts and processes to ensure compliance with quality standards
2. b) Evaluate the consistency and integrity of software configuration items
3. c) Evaluate the responsiveness and resource utilization of software
4. a) Ensuring that the software is built correctly and implements specific functions
5. b) Guides the testing process and outlines the approach, objectives, and scope of testing activities
6. a) Test case prioritization and risk-based testing
7. b) Create and execute automated test cases, reducing manual effort
8. b) Unit testing

10. TERMINAL QUESTIONS

1. Explain the key aspects of a quality audit and Configuration audit.
2. Briefly explain the important performance monitoring elements in the Assurance of Software testing

3. Define Verification and Validation. Explicate SQA activities.
4. Elucidate the importance of the test plan.
5. With clear steps explain how to write a test plan.
6. Explicate the test strategies that are used in software engineering.
7. Explain different testing methods and testing tools with suitable examples.

11. TERMINAL ANSWERS

1. Refer to section 2.
2. Refer to section 3.
3. Refer to section 4.
4. Refer to section 5.1
5. Refer to section 5.2
6. Refer to section 5.3
7. Refer to section 6.