Web Development and Database Administration Level-IV

Based on March 2022, Curriculum Version II

Module Title: - Database System Testing

Module code: EIS WDDBA4 06 1123

Nominal duration: 24 Hour

Prepared by: Ministry of Labor and Skill

September, 2023

Addis Ababa, Ethiopia

Table of Content

TABLE OF CONTENTS

ACKNOWLEDGMENT

Ministry of Labor and Skills wish to extend thanks and appreciation to the many representatives of TVET instructors and respective industry experts who donated their time and expertise to the development of this Teaching, Training and Learning Materials (TTLM).

ACRONYM

DBMS ---------- Database Management System |

SQL ---------- Structured Query

SDLC ---------- Software Development Life Cycle

INTRODUCTION TO THE MODULE

This module provides a comprehensive exploration of the principles and practices involved in assessing the quality, functionality, and performance of database systems. Participants will delve into various aspects of the testing process, including the recognition of potential quality problems, identification of risks, assessment of critical control points, and the detection of quality variations.

This module covers the units:

- Test Preparation and Planning
- Conducting test
- Reporting Quality-Affecting Issues

Learning Objective of the Module

At the end of the module the trainee will be able to:

- set up and configure a test environment
- Align the determination of the software life cycle with foundational work principles
- Understand system architecture for effective modularization
- Understand and apply quality benchmarks
- observe, identify, and articulate potential or existing quality problems
- Apply organization and industry standards effectively in testing processes
- meticulously review and examine expected results and requirements

Module Instruction

For effective use these modules trainees are expected to follow the following module instruction:

1. Read the information written in each unit
2. Accomplish the Self-checks at the end of each unit
3. Perform Operation Sheets which were provided at the end of units
4. Do the “LAP test” giver at the end of each unit and

UNIT ONE: TEST PREPARATION AND PLANNING

This unit is developed to provide you the necessary information regarding the following content coverage and topics:

- Test Environment Preparation
- Software Life Cycle Determination
- Test Plan and Tool Selection
- System Modularization for Live Scenario Mirroring
- Gathering and Preparing Logs, Result Sheets
- Announcements for Scheduled Tests
- Preparation of Test Scripts
- Review of Expected Results and Requirements

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- set up and configure a test environment
- align the determination of the software life cycle with foundational work principles
- define a comprehensive test plan
- Understand the system architecture for effective modularization
- identify and collect relevant logs
- create and maintain a comprehensive log inventory
- design comprehensive and effective test cases within the scripts
- meticulously examine expected results and requirements
- ensure that expected results adhere to established standards and guidelines

1.1. TEST ENVIRONMENT PREPARATION

A testing environment is a setup of software and hardware on which the testing team is going to perform the testing of the newly built software/hardware product.

This setup consists of the physical setup which includes hardware, and logical setup that includes Server Operating system, client operating system, database server, front end running environment (interface) or any other software components required to run the new product.

Test environment preparation is a crucial phase in the database system testing process. It involves setting up an environment that mimics the production environment as closely as possible, ensuring comprehensive and accurate testing. This phase ensures that the database system can perform optimally and reliably under different scenarios.

1.1.1 Key Steps in Test Environment Preparation

- **Environment Analysis**

Before setting up the test environment, it's essential to analyze the production environment to identify key components, configurations, and dependencies. This analysis helps in replicating the production environment accurately.

- Identify Hardware and Software Components
  - ✓ List all hardware components such as servers, storage, and networking equipment.
  - ✓ Document software components, including the operating system, database management system (DBMS), and any middleware.
- Capture Configurations
  - ✓ Document configuration settings for the database server, application servers, and other related components.
  - ✓ Pay special attention to parameters affecting performance, security, and scalability.

1.1.2. Database testing

A database has two main parts - the data structures (the schema) that store the data AND the data itself.

Database testing involves finding out the answers to the following questions:

Is the data organized well logically?

Does the database perform well?

Do the database objects like views, triggers, stored procedures, functions and jobs work correctly?

Does the database implement constraints to allow only correct data to be stored in it?

Is the data secured from unauthorized access?

- **Types of testing and processes**
  - Black box testing

    It involves testing interfaces and the integration of the database. Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

    Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users “don’t care” how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

  - White Box testing

    White box testing mainly deals with the internal structure of the database. The specification details are hidden from the user.

    - ✓ It involves the testing of database triggers and logical views.
    - ✓ It performs module testing of database functions, triggers, views, SQL queries etc.
    - ✓ It validates database tables, data models, database schema etc.

- ✓ It checks rules of Referential integrity.
- ✓ It selects default table values to check on database consistency.

The main advantage of white box testing in database testing is that coding error are detected, so internal bugs in the database can be eliminated.

Fig: 1.1. Database testing

1.2. SOFTWARE LIFE CYCLE

1.2.1. Software Life Cycle Determination

The software life cycle is a general model of the software development process, including all the activities and work process required to develop a software system.
Software life cycle describe phases of the software cycle and the order in which those phases are executed.

The software life cycle is the process of planning, creating, testing, deploying, and maintaining an information system. Determining the appropriate software life cycle model is

crucial for effective software development and testing. Different projects may require different life cycle models based on their specific needs and constraints.

- **Key Considerations**
  - Project Characteristics
    - ✓ Size and Complexity
    - ✓ Larger and more complex projects may benefit from a more structured and phased approach, such as the Waterfall model.
    - ✓ Smaller projects or those with evolving requirements may be better suited for agile methodologies.
    - ✓ Criticality
    - ✓ Projects with high criticality and strict regulatory requirements may favor a more rigorous and documentation-centric life cycle, like V-Model.
  - Client Collaboration and Feedback
    - ✓ Client Involvement
    - ✓ Determine the level of client involvement and feedback required throughout the development process.
  - Flexibility of Requirements
    - ✓ Changing Requirements
    - ✓ Agile models accommodate changing requirements more readily than traditional models like Waterfall.
  - Risk Tolerance
    - ✓ Risk Management
    - ✓ Evaluate the project's tolerance for risks and uncertainties.
    - ✓ Iterative models like Spiral address risk management throughout the life cycle.

There are six phases in every Software development life cycle model:

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

1. **Requirement gathering and analysis:** Business requirements are gathered in this phase. The general questions that need answer during a requirements gathering phase include:
   - Who is going to use the system?
   - How will they use the system?
   - What data should be input into the system?
   - What data should be output by the system?

After requirements are gathered and analyzed for their validity, requirements Specification document is created which serves the purpose of guideline for the next phase of the model.

2. **Design:** In this phase, the system and software design is prepared from the requirement specifications documents which were studied in the first phase. System Design helps in specifying hardware and overall system architecture.
3. **Implementation/Coding:** On receiving system design documents, the work is divided in modules/units and actual coding is started. This is the longest phase of the software development life cycle.
4. **Testing:** After the code is developed, it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.

During this phase unit testing, integration testing, system testing, acceptance testing are done.

5. **Deployment:** After successful testing, the product is delivered/deployed to the customer for their use.

6. **Maintenance:** Once when the customers starts using the developed system, then the actual problems comes up and needs to be solved from time to time.

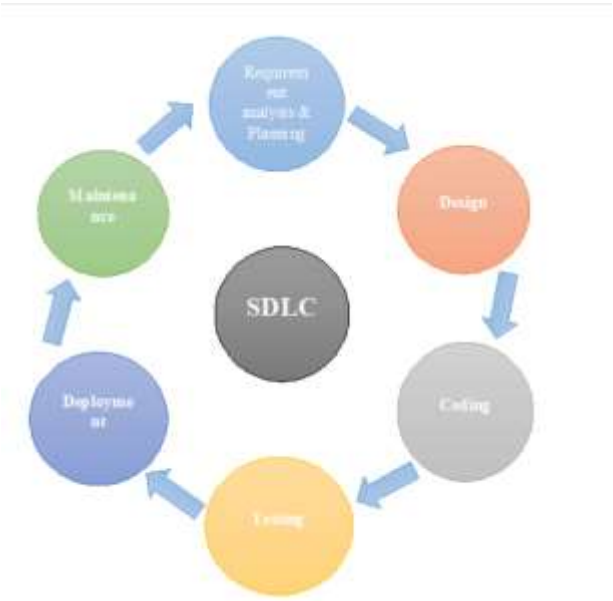

Fig: 1.2 Software Development Life Cycle

1.3. GATHERING AND PREPARING LOGS, RESULT SHEETS

A test plan is a document detailing a systematic approach to testing a system such as a machine or software. test plan can be defined as a document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

Database testing means test engineer should test the data integrity, data accessing, query retrieving, modifications, updating and deletion etc.

Database testing basically include the following.

- Data validity testing - you should be good in SQL queries.
- Data Integrity testing - should know about referential integrity and different constraint.
- Performance related to database - you should have idea about the table structure and design.

- Testing of Procedure, triggers and functions.
- Checking the integrity of UI data with Database Data
- Checking execution of stored procedures with the input values taken from the database tables
- Data accessing, query retrieving, modifications, updating and deletion etc.

Database testing usually consists of a layered process, including the user interface (UI) layer, the business layer, the data access layer and the database itself.

1.4. SYSTEM MODULARIZATION FOR LIVE SCENARIO MIRRORING

This may involve breaking down the database system into modular components to facilitate more focused and realistic testing scenarios. Mirroring live scenarios helps in assessing how the system performs under conditions similar to actual usage.

- **Modularization Overview:**

System modularization is the process of dividing a complex system into smaller, manageable modules or components. Each module represents a distinct functionality or feature of the system.

- **Identification of Modules:** The first step is to identify the key modules within the database system. These modules are typically based on functional areas, such as user authentication, data retrieval, data processing, and system security.
- **Isolation of Modules:** Once identified, each module is isolated to ensure that it can be tested independently. This isolation helps in focusing on specific functionalities without interference from other parts of the system.
- **Testing Scenarios:** The modularization process is guided by the desire to mirror live usage scenarios. Testing scenarios are created to simulate how users interact with different modules of the system in real-world situations.
- **Integration Points:** While testing individual modules is essential, attention is also given to the integration points between modules. Testing these integration points ensures that

the modules work seamlessly together and do not introduce unexpected issues when combined.

- **Data Flow and Interactions:** Understanding the flow of data between modules is crucial. Testing should cover scenarios where data moves between modules, ensuring that data integrity is maintained throughout the system.
- **Scalability Considerations:** The modularization process may also consider scalability. For example, testing may involve scenarios where the system experiences increased load or usage to assess how well it scales with growing demands.
- **User Behavior Emulation:** The testing scenarios aim to emulate user behavior in live scenarios. This includes not only typical user interactions but also edge cases, error conditions, and stress conditions to ensure the robustness of the system.
- **Real-Time Conditions:** The goal is to create conditions that closely resemble the live environment. This might involve using real data, incorporating actual usage patterns, and simulating concurrent user interactions to mirror the complexities of real-world scenarios.
- **Documentation of Modular Tests:** Throughout this process, documentation is critical. Each modular test scenario, its expected outcomes, and the results of the testing process are documented. This documentation aids in analysis, reporting, and future testing efforts.

1.5. GATHERING AND PREPARING LOGS, RESULT SHEETS

The collection and preparation of logs and result sheets involve establishing mechanisms to capture relevant data during testing. This data, including logs and results, is crucial for analysis, debugging, and reporting.

- **Log Collection:** Logs are records of events, actions, or messages generated by the system during testing. These may include logs related to database queries, system events, error messages, and other relevant information. Gathering logs is crucial for understanding the system's behavior during testing.

- **Result Sheet Preparation:** A result sheet is a document that captures the outcomes of individual test cases or scenarios. It includes information such as the test case ID, description, steps taken during testing, expected results, actual results, and the pass/fail status. Result sheets serve as a comprehensive record of testing activities.
- **Automation Tool Output:** In cases where automated testing tools are used, logs and results are often generated automatically by the testing tools. These outputs need to be collected, organized, and prepared for analysis.
- **Consistent Naming and Labeling:** To facilitate easy reference and analysis, logs and result sheets should follow a consistent naming and labeling convention. This ensures that each log or result sheet can be easily associated with the corresponding test case or testing activity.
- **Capture Relevant Information:** Logs and result sheets should capture relevant information for each test case, including any deviations from expected behavior, error messages, and performance metrics. This information is valuable for troubleshooting and debugging.
- **Organize by Test Categories:** If testing is conducted in different categories (e.g., functional, performance, security), logs and result sheets should be organized accordingly. This organization helps in identifying specific areas of the system that may require attention.
- **Timestamping:** Logs and result sheets should include timestamps to indicate when each test was executed. Timestamps aid in understanding the sequence of events and can be useful for identifying patterns or correlations between different testing activities.
- **Version Control:** If multiple versions of the software are being tested, logs and result sheets should be labeled with the corresponding software version. This ensures that results are tied to the specific version under test.

- **Collaboration and Communication:** Logs and result sheets serve as a basis for collaboration between testing and development teams. Clear communication about the availability and location of logs and results ensures that teams can efficiently review and address identified issues.

1.6. ANNOUNCEMENTS FOR SCHEDULED TESTS

Scheduling is the process of deciding how to commit resources between varieties of possible tasks.

Schedule Test means arrange or plan (an event) to take place at a particular time.

Scheduling through Task Scheduler allows you to automatically perform routine tasks on a chosen schedule. The Task Scheduler does this by monitoring whatever criteria you choose to initiate the tasks (daily, weekly etc and time also) and then execute the task when the criteria is met. With Scheduled Tasks, you can schedule any script, program, or document to run at a time that you specify when creating the task

This component likely involves planning and communicating the schedule for various testing activities. Scheduled announcements help ensure that all stakeholders are aware of when testing will occur and can plan accordingly.

- **Test Schedule Planning:** Before making announcements, the testing team establishes a detailed schedule outlining when different testing activities will take place. This includes start and end dates for testing phases, specific testing sessions, and any planned interruptions or breaks.
- **Communication Channels:** The team determines the channels through which test announcements will be communicated. This could include project management tools, collaboration platforms, emails, or team meetings. The goal is to choose channels that reach all relevant stakeholders effectively.
- **Stakeholder Identification:** Identifying the key stakeholders is crucial. This includes members of the testing team, development team, project managers, and any other individuals or groups with a vested interest in the testing process and its outcomes.

- **Announcement Content:** Announcements should include details such as the purpose of the upcoming testing phase, the scope of testing, specific areas or functionalities being tested, and any critical information stakeholders need to be aware of (e.g., testing constraints or special considerations).
- **Advance Notice:** Providing advance notice is important to allow stakeholders sufficient time to prepare. This may include preparing test environments, reviewing relevant documentation, and aligning their schedules with the testing activities.
- **Frequency of Announcements:** The frequency of announcements depends on the testing timeline and the nature of the project. For longer testing phases, regular updates may be necessary, while for shorter, more intensive phases, concise and timely announcements are crucial.
- **Feedback Mechanisms:** The announcement process should include mechanisms for stakeholders to provide feedback or seek clarification. This promotes open communication and allows stakeholders to raise any concerns or questions related to the testing schedule.
- **Emergency Notifications:** In cases where unexpected changes to the schedule occur, such as the need for urgent testing due to critical issues, a mechanism for emergency notifications should be in place. This ensures that stakeholders are informed promptly.

- **Integration with Project Calendar:** The testing schedule should be integrated into the overall project calendar. This allows stakeholders to see how testing activities align with other project milestones and activities, fostering a holistic view of project progress.
- **Consistency:** Consistency in the format and style of announcements contributes to clarity. Stakeholders become accustomed to a certain communication pattern, making it easier for them to digest and act upon the information provided.

1.7. PREPARATION OF TEST SCRIPTS

This involves creating detailed test scripts that outline the steps to be taken, the data to be input, and the expected results for each test case. Test scripts serve as a guide for executing tests systematically.

1.7.1. online test

- **Online testing**, also known as real-time testing or interactive testing, involves testing a system or application while it is actively running and interacting with users or other systems.
  - Characteristics
    - ✓ Dynamic Interaction: Online testing occurs in real-time, and the system is actively processing user inputs or requests.
    - ✓ User Interaction: Users may be actively interacting with the system during the testing process, and the testing team may simulate user actions.
    - ✓ Continuous Testing: Testing is ongoing as the system is operational, allowing for immediate feedback on changes or updates.
    - ✓ Immediate Detection of Issues: Issues or defects are identified and addressed quickly as they arise during real-time interactions.

  Examples

  Testing an e-commerce website while users are actively browsing, selecting products, and making purchases.

  Testing an online banking system where customers are conducting transactions in real-time.

1.7.2. batch test

- Batch testing involves testing a set of data or transactions in a group or batch. The testing occurs without real-time interaction, and the focus is on processing a predefined set of input data.

- Characteristics
  - ✓ Offline Processing: Testing is done without the direct involvement of users or real-time interactions with the system.
  - ✓ Large Sets of Data: Batch testing often involves processing a large volume of data or transactions in a systematic and automated manner.
  - ✓ Scheduled Execution: Tests are typically scheduled to run at specific times, often during non-peak hours.
  - ✓ Data Integrity and Processing: Emphasis is on ensuring the correctness and integrity of data processing.

Examples:

Performing data validation checks on a nightly batch of records in a database.

Testing the processing of a bulk data import feature that runs at scheduled intervals.

- Considerations
  - Usage Scenario
  - Online testing is suitable for scenarios where real-time user interactions are critical, while batch testing is appropriate for large-scale data processing or background tasks.
- **Feedback Time**: Online testing provides immediate feedback, allowing for quick detection and resolution of issues. Batch testing may have longer feedback cycles.
- **Resource Utilization:** Online testing may require more resources as it involves continuous interaction with the live system. Batch testing, being scheduled, can be optimized for resource usage.
- **Complexity**: Batch testing may be more suitable for complex, time-consuming operations, while online testing is focused on immediate user interactions.

- **Testing Tools:** The choice of testing tools may vary based on whether the testing approach is online or batch. Online testing may use tools for user interface testing, while batch testing tools may focus on automated processing and validation.

Both online and batch testing play crucial roles in the overall testing strategy, and the choice between them depends on the nature of the system, the testing objectives, and the specific requirements of the project. Often, a combination of both approaches is used to comprehensively validate the functionality and performance of a system.

1.8. REVIEW OF EXPECTED RESULTS AND REQUIREMENTS

1.8.1. Review expected results against acceptance criteria and system requirements Documentation

- This step includes a thorough review of expected results against the specified requirements. It ensures that the testing team has a clear understanding of what constitutes correct behavior in the database system.

These components collectively contribute to the systematic planning and preparation required before the actual execution of tests in a database system testing environment.

Acceptance Criteria Evaluation

The expected results are scrutinized to ensure they align with the acceptance criteria defined for the database system. Acceptance criteria outline the conditions that must be met for the system to be considered acceptable or ready for deployment.

- **Consistency with System Requirements:** The review assesses whether the expected results are consistent with the overall system requirements. System requirements provide a comprehensive view of the functionalities and features expected from the database system.
- **Validation of Compliance:** Each expected result is validated to ensure compliance with both acceptance criteria and system requirements. This involves checking that the testing outcomes meet the specific conditions and functionalities specified in these documents.

- **Traceability Verification:** Traceability between expected results, acceptance criteria, and system requirements is verified. This ensures that each expected result can be traced back to the corresponding acceptance criteria and, ultimately, to the system requirements.
- **Gap Analysis:** The review may include a gap analysis to identify any disparities between the expected results and the acceptance criteria or system requirements. This helps in uncovering areas where the testing process may need adjustment or where additional testing might be required.
- **Risk Assessment against Criteria:** The review assesses whether the expected results adequately address high-priority or high-risk areas identified in the acceptance criteria and system requirements. It ensures that critical functionalities are thoroughly tested.
- **Documentation Clarity:** The clarity and comprehensibility of the documentation, including expected results, acceptance criteria, and system requirements, are assessed. Clear documentation is essential for effective communication and understanding among stakeholders.
- **Communication with Stakeholders:** Stakeholders involved in defining acceptance criteria and system requirements may be consulted during the review. This collaborative approach ensures that the testing efforts align with the expectations and goals set by stakeholders.

SELF-CHECK 1

Part-I Multiple choice

1. What is the purpose of a result sheet in testing? a. To set up a test environment
   b. To document the expected results and requirements
   c. To capture outcomes of individual test cases
   d. To announce scheduled tests

2. Which testing approach involves testing a system while it is actively running and interacting with users or other systems?

   a. Batch testing
   b. Online testing
   c. White box testing
   d. Black box testing

3. What is the main advantage of white box testing in database testing? a. It involves testing interfaces and integration
   b. It detects coding errors and eliminates internal bugs
   c. It tests a system with no prior knowledge of its internal workings
   d. It simulates user activity and observes system responses

4. What does the modularization process involve in the context of system testing?

a. Breaking down the system into smaller, manageable modules

b. Creating test scripts for individual modules

c. Performing batch testing on isolated modules

d. Documenting test results for each module

5. Why is providing advance notice for scheduled tests important? a. To ensure stakeholders are aware of when testing will occur

   b. To delay testing activities

   c. To skip testing phases

   d. To keep the testing schedule confidential

Part-II Give short Answer

1. Explain the significance of modularization in the context of system testing.
2. Describe the key considerations in determining the appropriate Software Development Life Cycle (SDLC) model for a project
3. Explain the difference between online testing and batch testing.

UNIT TWO: CONDUCTING TEST

This unit is developed to provide you the necessary information regarding the following content coverage and topics:

- Execution and Documentation of Test Scripts
- Quality Benchmarks and Comparisons
- Organization/Industry Standards Adoption
- Comparison of Actual and Expected Results

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- document the test execution process comprehensively
- Understand quality benchmarks
- test outcomes against established quality benchmarks
- identify standards effectively in the testing processes
- Identify and document discrepancies between actual and expected results.

2.1. EXECUTION AND DOCUMENTATION OF TEST SCRIPTS

2.1.1. Test Script Execution

Test script development involves the same processes and techniques used when constructing software programs, any experience.

A test script is the executable form of a test. It defines the set of actions to carry out in order to conduct a test and it defines the expected outcomes and results that are used to identify any deviance in the actual behavior of the program from the logical behavior in the script.

Test Scripts are step-by-step instructions on how to test a test case. They are detailed and contain individual steps that test for each and every functionality. Test scripts are programs that execute tests on the software product/application.

Test case: a logical description of a test. It details the purpose of the test and the derivation audit trail.

Test Script: the physical, executable, description of the test case.

Automated test script: a program that implements a test.

- **Preparation:** Before executing test scripts, ensure that the test environment is set up according to the specifications outlined in the test plan. This includes configuring databases, servers, and any other necessary components.
- **Test Script Review:** Review each test script to understand the steps involved, input data required, and expected outcomes. Make sure that the test scripts cover various scenarios, including normal operation and potential error conditions.

- **Execution:** Execute the test scripts systematically. Follow the defined steps and provide input data as required. Monitor the system's responses and record any deviations from expected behavior.

- **Logging:** Log relevant information during test script execution. This includes timestamps, inputs provided, actual outputs, and any error messages or unexpected behaviors encountered.
- **Capturing Screenshots/Logs:** Capture screenshots or log files during critical steps or in the case of errors. This documentation aids in the analysis and understanding of the test execution process.

2.1.2. Documentation

Documentation consists of documents which provide proof or evidence of something, or are a record of something

- **Results Recording**: Record the results of each test script. Clearly document whether the test passed, failed, or encountered issues. Include details on any deviations from expected outcomes.

- **Defect Reporting**: If any defects or issues are identified during test script execution, document them in detail. Include information such as steps to reproduce, expected versus actual results, and any relevant system information.

- **Traceability**: Maintain traceability between the executed test scripts and the corresponding requirements. This ensures that all aspects of the system's functionality are being tested.

- **Metrics and Statistics**: Gather metrics and statistics related to test script execution. This could include the time taken for each script, success rates, and any performance metrics that are relevant to the testing goals.

- **Test Execution Summary:** Summarize the overall test script execution, highlighting key findings, successes, and challenges. This summary can be useful for stakeholders who want a high-level overview of the testing progress.
- **Communication:** Communicate the results to relevant stakeholders, including developers and project managers. Clear and timely communication is crucial for addressing issues and ensuring that the necessary actions are taken.

2.2. QUALITY BENCHMARKS AND COMPARISONS

2.2.1. Benchmark Definition

Clearly define quality benchmarks specific to the database system under test. These benchmarks may include criteria related to performance, functionality, security, and other relevant aspects.

- **Performance Metrics:** Identify specific performance metrics, such as response time, throughput, and resource utilization that will be used as benchmarks for evaluating the database system's performance.
- **Functional Criteria:** Define functional criteria that the database system must meet. This could involve ensuring that specific operations (e.g., data retrieval, updates, deletions) are performed accurately and efficiently.
- **Security Standards:** Incorporate security standards into the benchmarks. Evaluate the database system against established security measures, including access controls, encryption, and data integrity.

2.2.2. Comparisons

- **Actual Performance Measurement:** Execute the database system tests and measure its actual performance against the predefined benchmarks. Capture relevant data such as response times and system resource usage.
- **Benchmark Evaluation:** Evaluate the database system's performance and functionality against the established benchmarks. Determine whether the system meets, exceeds, or falls short of the defined criteria.

- **Scalability Assessment:** Assess the scalability of the database system by comparing its performance under varying workloads. This helps determine how well the system can handle increased data volume or user concurrency.
- **Comparative Analysis:** Perform a comparative analysis between different test scenarios or configurations. For example, compare the system's performance under different database configurations or network conditions.
- **Threshold Identification:** Identify performance thresholds beyond which the system's performance is considered unacceptable. This helps in setting clear boundaries for acceptable performance levels.
- **Documentation of Comparisons:** Document the results of the comparisons, highlighting specific metrics and outcomes. Include both quantitative data (e.g., response times) and qualitative observations (e.g., user experience).
- **Iterative Improvement:** If discrepancies are identified, use the benchmarking data to inform iterative improvements. This may involve adjusting configurations, optimizing queries, or making other enhancements to meet or exceed benchmarks.
- **Reporting to Stakeholders:** Communicate the benchmarking results to relevant stakeholders, providing a clear understanding of how well the database system aligns with quality benchmarks. This information is crucial for decision-making and further development efforts.

By incorporating clear benchmarks and systematically comparing the actual performance against these benchmarks, you ensure a comprehensive evaluation of the database system's quality. This process aids in identifying areas for improvement and making informed decisions about the system's readiness for deployment or further optimization.

2.3. ORGANIZATION/INDUSTRY STANDARDS ADOPTION

- **Organization Standards**

  Many organizations have their own set of standards and guidelines for software development and testing. These standards could cover various aspects, such as coding

conventions, documentation requirements, and testing methodologies. Adhering to these standards ensures consistency and quality across projects within the organization.

- **Industry Standards**

  In addition to internal organization standards, there are often industry-wide standards that define best practices for software development and testing. These standards may be set by recognized bodies or associations in the field. Adhering to industry standards helps ensure that the software or system meets broader expectations for quality, security, and performance.

- **Benefits of Standard Adherence**

  Adhering to standards provides several benefits, including improved interoperability, better maintainability, and increased reliability. It also facilitates communication and collaboration within the development and testing teams and across different projects within the organization.

- **Documentation and Compliance**

  The subunit may involve documentation of how the testing process aligns with specific standards. This documentation serves as evidence of compliance and can be valuable for audits, reviews, or future reference. It may include details on how each testing activity meets the prescribed standards.

- **Continuous Improvement**

  The adoption of standards is often coupled with a commitment to continuous improvement. Teams may regularly review and update their processes to align with evolving standards and incorporate lessons learned from previous projects.

  In summary, this subunit likely emphasizes the importance of following established standards, whether internal to the organization or external industry-wide standards, to ensure that the database system testing process is robust, consistent, and aligned with best practices.

2.4. COMPARISON OF ACTUAL AND EXPECTED RESULTS

Involves the critical process of verifying that the behavior of the database system aligns with the anticipated outcomes. Here's a breakdown of what this subunit might encompass:

- **Test Case Execution**: During the execution of test cases, various actions are performed on the database system, including inputting data, executing queries, and interacting with the system in predefined ways. These actions are based on test scripts created during the test planning phase.
- **Expected Results**: Each test case comes with predefined expected results. These are the outcomes or responses that the testing team anticipates when the test case is executed successfully. Expected results are determined during the test planning phase and serve as a benchmark for evaluating the actual outcomes.
- **Actual Results**: The actual results are the outcomes observed when the test case is executed. Testers carefully compare these actual outcomes with the expected results to identify any discrepancies, errors, or unexpected behaviors. This step is crucial for uncovering defects or issues in the database system.
- **Defect Identification**: Any differences between the actual and expected results are treated as potential defects. Testers document these disparities in detail, including the steps to reproduce the issue and any relevant system conditions. This documentation is then used to communicate the identified defects to the development team for resolution.
- **Regression Testing**: If defects are found and subsequently fixed by the development team, regression testing may be performed. This involves re-executing relevant test cases to ensure that the corrections did not introduce new issues and that the overall system functionality remains intact.
- **Verification and Validation:** The comparison of actual and expected results is part of the broader verification and validation process. Verification confirms that the system meets specified requirements, while validation ensures that it satisfies the needs of the end-users.

- **Iterative Process**: This comparison is often iterative, with test cases being executed multiple times, especially when changes are made to the system. It helps ensure the ongoing reliability and correctness of the database system.

In summary Comparison of Actual and Expected Results" is a crucial step in the testing process, aiming to identify and address any discrepancies between the expected and actual behavior of the database system.

SELF-CHECK 2

Part-I Multiple choice

1. What is the purpose of defining quality benchmarks for a database system?

A. To make the testing process more complicated

B. To set clear boundaries for acceptable performance levels

C. To eliminate the need for actual testing

D. To increase the size of the test dataset

2. What is a test script in the context of software testing?

A. A physical, executable description of a test case

B. A detailed record of test results

C. A document outlining project requirements

D. A program that implements a defect

3. Why is it essential to review test scripts before execution?

A. To increase the complexity of the testing process

B. To ensure all team members understand the test environment

C. To identify deviations from expected behavior

D. To delay the execution phase

Part-II Give Short Answer

1. Define organization/Industry standard

2. What benefits does adherence to organization standards bring to the testing process?
3. Explain the significance of incorporating security standards into the benchmarks for evaluating a database system.

UNIT THREE: REPORTING QUALITY-AFFECTING ISSUES

This unit is developed to provide you the necessary information regarding the following content coverage and topics:

- Recognition of Potential or Existing Quality Problems
- identification of potential risks and critical control points
- Identification of Quality Variations
- reporting quality variations and potential problems

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Observe and recognize potential or existing quality problems during the testing process
- Identify and document variations that may impact the quality of the system
- Articulate and express identified variations and potential problems clearly and comprehensively.
- Recognize potential or existing quality problems

3.1. RECOGNITION OF POTENTIAL OR EXISTING QUALITY PROBLEMS

It involves the awareness and acknowledgment of issues that might impact the quality of the database system. Testers and quality assurance professionals need to be vigilant in recognizing potential problems, whether they are derived from requirements, design, or other aspects of the system. Include:

- **Requirement Analysis**

  Testers and quality assurance professionals carefully analyze the requirements for the database system. This includes understanding the functional and non-functional requirements to ensure that they are clear, complete, and feasible.

- **Early Testing Stages**

  Quality problems can be recognized early in the testing process. This involves conducting reviews and inspections of project documentation, such as requirements and design specifications, to identify potential issues before the actual testing of the system begins.

- **Use of Testing Techniques**

  Testing techniques, such as static testing methods (reviews, inspections) and dynamic testing methods (test case execution), are employed to identify potential issues. Dynamic testing helps simulate real-world usage scenarios and exposes issues related to functionality, performance, and other quality attributes.

- **Traceability**

  Establishing traceability between requirements and test cases helps ensure that all aspects of the requirements are covered during testing. This also allows for the early detection of any discrepancies or gaps in the requirements.

- **Collaboration with Stakeholders**

Effective communication and collaboration with stakeholders, including developers, product owners, and end-users, are essential for recognizing potential quality problems. Input from different perspectives can uncover issues that may not be apparent from a testing standpoint alone.

- **Monitoring System Metrics**

  Continuous monitoring of system metrics, such as performance and resource usage, can help identify potential quality problems. Deviations from expected metrics may indicate areas that require further investigation.
- **Feedback Mechanisms:**

  Establishing feedback mechanisms within the testing team and between the testing and development teams facilitates the early recognition of potential issues. Regular team discussions and status updates contribute to a proactive approach in addressing quality problems.
- **Documentation of Observations**

  Testers document their observations and concerns regarding potential quality problems. This documentation serves as a basis for reporting and provides a record of the thought process behind recognizing these issues.

3.2. IDENTIFICATION OF POTENTIAL RISKS AND CRITICAL CONTROL POINTS

The identification of potential risks and critical control points involves assessing areas of the database system that may be susceptible to issues or failures. This proactive approach allows for the implementation of measures to mitigate risks and establish control points to monitor and manage potential problems.

- **Risk Analysis:** Testers and quality assurance professionals conduct a thorough analysis to identify potential risks associated with the database system. Risks may include factors such as technical complexities, uncertainties in requirements, external dependencies, and resource constraints.

- **Risk Identification Techniques:** Various techniques, such as brainstorming sessions, interviews, and the use of historical data, may be employed to identify potential risks. By involving key stakeholders, the team can gather diverse perspectives and insights to identify risks that might not be immediately apparent.

- **Documentation of Identified Risks:** Each identified risk is documented along with relevant details, such as the nature of the risk, potential impact on the project, and the likelihood of occurrence. This documentation forms the basis for developing risk mitigation and management strategies.
- **Critical Control Points:** Critical control points are specific stages or aspects of the database system where control measures can be applied to mitigate or monitor identified risks. These control points act as checkpoints during the development and testing process, ensuring that potential risks are actively managed.
- **Mitigation Strategies:** Once risks are identified, the team collaboratively develops mitigation strategies. These strategies outline the actions to be taken to reduce the likelihood or impact of the identified risks. Mitigation plans are proactive measures to handle potential issues before they escalate.
- **Continuous Monitoring:** The identification of critical control points involves setting up mechanisms for continuous monitoring of the project. This allows the team to stay vigilant for any changes in risk factors and to adjust mitigation strategies accordingly.
- **Communication with Stakeholders:** Clear communication with stakeholders, including project managers, developers, and business analysts, is essential for understanding and addressing potential risks. Regular updates on risk identification and mitigation efforts foster transparency and collaboration.
- **Integration with Test Planning:** The identified risks and critical control points should be integrated into the overall test planning process. Test cases and scenarios can be designed to specifically address high-priority risks, ensuring thorough testing in areas where the potential impact is significant.

3.3. IDENTIFICATION OF QUALITY VARIATIONS

Quality variations refer to differences or deviations from expected levels of quality. This could involve variances in performance, functionality, security, or other relevant quality attributes. Testers aim to identify and document these variations during the testing process.

- **Quality Attributes:** Testers identify and assess various quality attributes of the database system. These attributes may include but are not limited to functionality, performance, reliability, security, usability, and maintainability.
- **Testing Scenarios:** Specific testing scenarios and test cases are designed to evaluate the different quality attributes of the system. These scenarios are tailored to simulate real-world usage conditions and interactions to uncover potential variations in quality.
- **Performance Metrics:** Quality variations may be identified through the measurement of performance metrics. This involves monitoring factors such as response times, throughput, and resource utilization to ensure that the system meets performance expectations.
- **Usability Testing:** For user-centric quality variations, usability testing may be conducted. This involves evaluating how easily users can interact with the system, including aspects such as user interface design, navigation, and overall user experience.
- **Security Testing:** Security-related quality variations are identified through security testing. This involves assessing the system for vulnerabilities, potential breaches, and compliance with security standards and best practices.
- **Functional Testing:** Functional testing is performed to ensure that the database system meets its specified functional requirements. Any deviations from expected functionality are documented as quality variations.
- **Regression Testing:** As changes are made to the system, regression testing is conducted to ensure that new modifications do not introduce variations in the existing functionality. This helps maintain the overall quality and integrity of the system.

- **User Feedback:** User feedback and observations play a crucial role in identifying quality variations. End-users may provide insights into aspects of the system that impact their experience, leading to the recognition of variations in quality.
- **Automated Testing Tools:** Automated testing tools may be employed to systematically execute test cases and identify variations in quality. These tools can efficiently perform repetitive tests and provide quick feedback on the system's behavior.
- **Documentation of Variations:** Testers document each identified quality variation, including details such as the specific condition under which the variation occurred, the observed behavior, and the expected behavior. This documentation is valuable for reporting and addressing the variations.

3.4. REPORTING QUALITY VARIATIONS AND POTENTIAL PROBLEMS

Once potential problems, risks, and quality variations are identified, this subunit likely involves the formal reporting of these issues. Testers generate reports that document the nature of the variations, potential problems, or risks. These reports may include detailed information to aid developers in understanding and addressing the issues.

- **Documentation of Variations**: Testers document the identified quality variations, detailing the specific nature of each variation, the conditions under which it occurred, and any relevant information needed for analysis. This documentation serves as the basis for the subsequent reporting process.

- **Severity and Priority Assessment**: Each documented quality variation may be assessed in terms of severity and priority. Severity indicates the impact of the variation on the system, while priority determines the order in which variations should be addressed. This assessment guides the development team in allocating resources effectively.

- **Formalized Reports**: Testers generate formalized reports that compile information about the documented quality variations and potential problems. These reports may include summary tables, graphs, or charts to provide a visual representation of the state of the system.

- **Communication with Development Team**: Testers collaborate with the development team to communicate the documented quality variations. Clear and effective communication ensures that the development team understands the nature of the variations, facilitating their timely resolution.
- **Root Cause Analysis**: In some cases, the reporting process may involve conducting a root cause analysis to determine the underlying reasons for quality variations. Understanding the root causes helps in implementing effective corrective and preventive measures.
- **Feedback Loop**: Reporting quality variations establishes a feedback loop between the testing and development teams. The development team reviews the reports, addresses the identified issues, and may provide feedback to the testing team for clarification or additional information.
- **Continuous Monitoring**: The reporting process contributes to continuous monitoring of the system's quality. As new variations are identified and reported, the testing and development teams work collaboratively to maintain and improve the overall quality of the database system.
- **Documentation for Auditing and Compliance:** The reports generated during this process may also serve as documentation for auditing purposes and to demonstrate compliance with quality standards and requirements. This documentation can be valuable for regulatory compliance and process improvement initiatives.
- **Integration with Test Planning:** The reported variations may influence the test planning process for subsequent testing

SELF-CHECK 3

Part-I choose the best answer

1. Why is the documentation of identified quality variations important in the testing process?
   A) To increase paperwork.
   B) To fulfill a procedural requirement.
   C) To serve as a basis for reporting and aid in analysis.
   D) To impress stakeholders with documentation skills.

2. What is the purpose of assessing the severity and priority of identified quality variations?

   A) To add complexity to the reporting process.

   B) To allocate resources effectively for issue resolution.

   C) To confuse the development team.

   D) To impress stakeholders with technical details.

3. What is the primary purpose of identifying critical control points during risk analysis in the testing process?

   A) To add complexity to the testing process.

   B) To establish checkpoints for continuous monitoring and risk mitigation.

   C) To create unnecessary documentation.

   D) To impress project managers with technical details.

Part-II Give Short Answer

1. Explain the role of collaboration with stakeholders in the early recognition of potential quality problems during the testing process.

2. Why is continuous monitoring of system metrics important in the recognition of potential quality problems?

REFERENCES

Books

1. Manual Testing Step by Step Tutorial G C Reddy Technologies (www.gcreddy.com)
2. INTRODUCTION TO SOFTWARE TESTING Paul Amman and Jeff Offutt
3. SOFTWARE TESTING Autonomous Institution – UGC, Govt. of India

URL

https://www.imperva.com/lear n/application-security/black-box-testing/

https://www.gcreddy.com/wp-content/uploads/2018/12/Manual-Testing-Step-by-Step-Tutorial.pdf

DEVELOPER'S PROFILE

No	Name	Qualification	Field of Study	Organization/ Institution	Mobile number	E-mail
1	Frew Atkilt	M-Tech	Network & Information Security	Bishoftu Polytechnic College	0911787374	frew.frikii@gmail.com
2	Gari Lencha	MSc	ICT Managment	Gimbi Polytechnic	0917819599	Garilencha12@gmail.com
3	Kalkidan Daniel	BSc	Computer Science	Entoto Polytechnic	0978336988	kalkidaniel08@gmail.com
4	Solomon Melese	M-Tech	Computer Engineering	M/G /M/Polytechnic College	0918578631	solomonmelese6@gmail.com
5	Tewodros Girma	MSc	Information system	Sheno Polytechnic College	0912068479	girmatewodiros @gmail.com
6	Yohannes Gebeyehu	BSc	Computer Science	Entoto Polytechnic College	0923221273	yohannesgebeyehu73@gmail.com