RQF LEVEL 5

GENQA501

SOFTWARE DEVELOPMENT

Quality Assurance Application

TRAINEE'S MANUAL

October, 2024

QUALITY ASSURANCE APPLICATION

KOICA Korea International Cooperation Agency

TQUM TVET Quality Management Project

2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

The competent development body of this manual is Rwanda TVET Board ©, reproduce with permission

All rights reserved

- This work has been produced initially with the Rwanda TVET Board with the support from KOICA through TQUM Project
- This work has copyright, but permission is given to all the Administrative and Academic Staff of the RTB and TVET Schools to make copies by photocopying or other duplicating processes for use at their own workplaces.
- This permission does not extend to making of copies for use outside the immediate environment for which they are made, nor making copies for hire or resale to third parties.
- The views expressed in this version of the work do not necessarily represent the views of RTB. The competent body does not give warranty nor accept any liability
- RTB owns the copyright to the trainee and trainer's manuals. Training providers may reproduce these training manuals in part or in full for training purposes only. Acknowledgment of RTB copyright must be included on any reproductions. Any other use of the manuals must be referred to the RTB.

© Rwanda TVET Board

Copies available from:

- *HQs: Rwanda TVET Board-RTB*
- *Web: www.rtb.gov.rw*
- **KIGALI-RWANDA**

Original published version: October 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate (RQF Level V) in software development, specifically for the module " GENQA501: QUALITY ASSURANCE APPLICATION ".

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher

Production Team

Authoring and Review

MUKAMARIDI Marie Rose

MUKASHYAKA Christine

Validation

MUKARIBENZI Leoncie

MUKAMUHOZA Liberee

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

MANIRAKORA Alexis

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

ACRONYMS

API: Application Programmable Interface

CI/CD: Continuous Integration /Continuous Deployment

CLI: Command Line Interface

CRM: Customer Relationship Management

GUI: Graphical User Interface

ISO: International Standards Organisation

KPIs: Key Performance Indicator

PIN: Project Initiation Note

QA: Quality Assurance

QC: Quality Control

REST: Representation State Transfer

RTB: Rwanda TVET Board

RTM: Requirement Traceability Matrix

SOAP: Simple Object Access Protocol

TDP: Test Delivery Plan

To R: Terms of Reference

TQUM Project: TVET Quality Management Project

UAT: User Acceptance testing

INTRODUCTION

This trainer's manual includes all the methodologies required to effectively deliver the module titled **"Quality Assurance Application".** Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies.

The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainer's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainer, you will begin by asking questions related to the activities to encourage critical thinking and guide trainees toward real-world applications in the labor market. The manual also outlines essential information such as learning hours, didactic materials, and suggested methodologies.

This manual outlines the procedures and methodologies for guiding trainees through various activities as detailed in their respective trainee manuals. The activities included in this training manual are designed to offer students opportunities for both individual and group work. Upon completing all activities, you will assist trainees in conducting a formative assessment known as the end learning outcome assessment. Ensure that students review the key reading and the points to remember section.

MODULE CODE AND TITLE: GENQA501 AND QUALITY ASSURANCE APPLICATION

Learning Outcome 1: Perform Requirements Analysis
Learning Outcome 2: Test the System
Learning Outcome 3: Generate Test Documentation

Indicative contents

1.1 Introduction to Quality Assurance

1.2 Analysing Terms of Reference (ToR)

1.3 Examining requirement specification

1.4 Analyzing inception report

Key Competencies for Learning Outcome 1: Perform Requirements Analysis

Knowledge	Skills	Attitudes
• Description of Quality Assurance(QA) • Identify tools and techniques • Description of the terms of Reference • Description of inception report	• Documenting a ToR analysis report • Examining requirement specification • Analysing inception report	• Having Curiosity in Requirements Analysis • Having Open-mindedness in Requirements Analysis • Having Attention in Requirements Analysis • Having Collaboration in Requirements Analysis • Having Analytical Thinking in Requirements Analysis • Having Adaptability in Requirements Analysis • Having Critical Thinking in Requirements Analysis • Having Commitment in Requirements Analysis

Duration: 10 hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly the Quality Assurance according to industry quality assurance standards.
2. Analyse properly Terms of Reference (ToR) according to industry quality assurance standards.
3. Examine properly requirement specification used in Quality Assurance based on industry standards, project objectives, and stakeholder needs.
4. Analyse properly inception report used in Quality Assurance

Resources

Equipment	Tools	Materials
• Computer	• Checklist	• Electricity • Pen • Paper • Internet

Indicative content 1.1: Introduction to the Quality Assurance

Duration:2

Theoretical Activity 1.1.1: Description of the Quality Assurance

Tasks:

1. you are requested to answer the following questions related to the Quality Assurance:

i. What do you understand about Quality Assurance?
ii. What are the stages/Processes of Quality Assurance?
iii. What are the main types of Quality Assurance?
iv. Mention the methods used in Quality assurance.
v. List down the standards used in quality assurance.
vi. What are the benefits of Quality Assurance?
vii. Identify Quality Assurance classification.
viii. Differentiate Quality assurance from Quality control.
ix. What are tools used in Quality Assurance?
x. Describe techniques of Quality Assurance?

2. Provide the answers on papers or flipcharts.

3.Present the findings/answers to the whole class.

4.Ask questions for more clarifications if any.

5.Read the key reading 1.1.1 from your manuals for more clarification.

Key readings 1.1.1: Introduction of Quality Assurance

1.Definition of Quality assurance (QA)

Quality assurance (QA) is a crucial (systematic) process for ensuring that a product or service meets the desired level of quality.

Quality assurance **(QA)** is a process whereby you verify that the system architecture is correct and meets all the requirements

It involves implementing processes, procedures, and standards to ensure that a product or service is consistent, reliable, and meets customers' expectations.

2.Main Stages in the QA Process

The main stages in the quality assurance process typically include the following:

i. Analyse the Requirements

The QAs must understand and analyze the project requirements, including functional and non-functional aspects. Fixing a bug at the early stages of development will cost less when compared to fixing a bug at the testing/production stage.

That is why QAs must be involved in the initial stages of requirement analysis so they understand the requirements clearly and design the tests accordingly.

ii. Plan the Tests

Now, the information gathered in the requirements analysis phase will be used for test planning. The test plan includes the testing strategy, scope, project budget, and deadlines. Also, it should include information about the required types and levels of testing and the testing tools. And when the test plan is ready, the test manager will allocate responsibilities to individual testers.

iii. Design the Tests

Now, the QA teams create test cases and checklists based on the software requirements. Each test case includes conditions, data, and steps to validate functionality. Testers compare actual results with expected results to ensure accuracy.

If automation is part of the test scope, this is the stage where automation testing scenarios are created.

Also, the testing team should prepare the staging environment for test execution in this stage. The staging environment should closely resemble the production environment, including hardware, software, network configurations, databases, and system settings.

iv. Execute Tests and Report Defects

Tests begin with developers conducting unit tests at the individual code component level. Following that, the QA testing team carries out tests at the API and UI levels. Manual tests are conducted based on pre-designed test cases. All identified bugs will be recorded in a defect-tracking system for efficient management. Moreover, test automation engineers can utilize automated test tools like Testsigma to execute test scripts and generate detailed test reports.

v. Run Re-Tests and Regression Tests

After bugs have been identified, reported, and resolved, QA will again re-test the functions to ensure thorough validation and ensure they didn't miss any user scenario. Additionally, they perform regression tests to confirm that the fixes haven't caused any negative impact on the existing functionalities.

vi. Run Release Tests

After developers notify about a release, the QA team identifies the affected functionalities. They create new test suites to cover the changes. The team also conducts smoke tests to check stability. If the tests pass, they run the modified test suites and generate a report.

3.The main Types of QA

i. Functional QA:

- Focuses on testing specific functionalities of the computer sytem against defined requirements.
- Includes unit testing, integration testing, system testing, and acceptance testing.

ii. Non-Functional QA:

- Assesses non-functional aspects such as performance, usability, security, and scalability.
- Types include performance testing, load testing, stress testing, and security testing.

iii. Automated QA:

- Involves using automation tools and scripts to conduct tests.
- Enhances efficiency and accuracy, particularly for regression and repetitive tests.

iv. Manual QA:

- Conducted by human testers who execute test cases without automation tools.
- Valuable for exploratory testing, usability testing, and scenarios requiring human judgment.

v. Regression QA:

- Ensures that recent changes or fixes do not negatively impact existing functionalities.
- Involves re-running previously executed tests to verify stability.

vi. User Acceptance Testing (UAT):

- Involves end-users testing the software to confirm it meets their needs and requirements.
- Focuses on real-world use cases and user experience.

vii. **Compliance Testing**:

- Ensures that the software meets industry standards, regulations, and compliance requirements.
- Important in sectors like finance, healthcare, and data security.

4.Standards of Quality Assurance (QA) in Software Development

Here are some of the key standards and frameworks commonly used in computer system QA:

International Standards Organization (ISO) Standards in software development

i. **ISO 9001**:

- A general quality management standard that applies to all industries, including software manufacturing.
- Focuses on meeting customer requirements and continuous improvement of processes.

ii. **ISO/IEC 17025**:

- Specifies requirements for testing and calibration laboratories.
- Ensures that laboratories produce valid and reliable results, critical for software of quality assurance.

iii. **MIL-STD-810**:

- A military standard that outlines environmental testing for software.
- Specifies test methods to ensure software can withstand environmental stresses.

iv. **ISO 26262**:

- A standard for functional safety in automotive systems.
- Addresses the safety lifecycle of software components and systems used in vehicles.

v. **Six Sigma**: A methodology that focuses on reducing defects and improving process quality.

vi. **IEEE Standards:** Guidelines for software engineering and quality assurance in software development.

5. Methods of Quality Assurance:

The list of methods used for Software Quality Assurance are:

i. AUDIT:

Involves the inspection of the progress and the product to determine whether the set of actions written in the original document that should be followed are being followed or not!

ii. REVIEW:

- Process in which both developers and customers gather for a meeting to:
- examine the progress and product
- clients give their approval
- comments
- suggests the implementation he wants to his product

iii. BUG DETECTION:

- Professional and trained group of people go through a static set of tests to find the bugs and defects in the project based on the rules and criteria of the project.
- These inspections are kept confidential from the clients.

iv. DESIGN INSPECTION:

Developers produce a list according to which the software is designed that examines the following areas of software design:

- General requirements and design
- Functional and Interface specifications
- Conventions
- Requirement traceability
- Structures and interfaces
- Logic
- Performance
- Error handling and recovery
- Testability, extensibility
- Coupling and cohesion

6.Benefits of Quality Assurance

Quality Assurance provides numerous benefits, including:

i. **Prevention of Defects:** QA focuses on preventing defects rather than fixing them after they occur.

ii. **Customer Satisfaction:** Ensuring high-quality products that meet or exceed customer expectations leads to higher satisfaction.
iii. **Efficiency:** Streamlined processes reduce errors and rework, saving time and resources.
iv. **Compliance:** QA ensures that products or services meet regulatory and industry standards.
v. **Continuous Improvement:** The QA process encourages ongoing evaluation and improvement, leading to better products over time.

7.Classification of quality assurance

7.1. Industry-Specific QA

Different industries have specialized QA practices tailored to their specific needs:

- **Software QA:** Focuses on testing software for bugs, functionality, performance, and security.
- **Manufacturing QA:** Ensures that the production process meets specified quality standards and regulations.
- **Healthcare QA:** Ensures medical devices, pharmaceuticals, and healthcare services comply with regulatory requirements (e.g., FDA standards).
- **Construction QA:** Focuses on materials, structural integrity, and safety regulations during the building process.

7.2. Process and Product QA

- **Process QA:** Ensures that the processes followed during production are efficient and consistent with quality standards. It focuses on preventing defects in the design, development, and manufacturing stages.
- **Product QA:** Involves verifying and validating that the final product meets customer requirements and specifications. It focuses on defect detection after the product has been produced.

7.3. Internal vs. External QA

- **Internal QA:** Conducted by the organization's own quality team to ensure compliance with internal standards and processes. This includes self-inspections, internal audits, and regular reviews.
- **External QA:** Conducted by third-party organizations or auditors to ensure that the company meets industry standards and regulations. External QA provides an unbiased evaluation of quality, often required for certifications.

7.4. Manual vs. Automated QA

- **Manual QA:** Quality testing is conducted by human testers who manually check for defects and issues. This includes exploratory testing, visual inspections, and manual execution of test cases.
- **Automated QA:** Uses automated tools to perform testing. Automated testing is efficient for repetitive tasks, regression testing, and performance testing. It ensures faster and more consistent results compared to manual QA.
- **Quality Assurance Vs Quality Control**

8.Difference between Quality Assurance and Quality Control

System quality assurance is (also known as QA) a sequence of tasks to prevent

defects and ensure that the techniques, methods, approaches, and processes are

designed for a specific application must be implemented correctly. This is an ongoing

process within the development of a software system.

The development of units of an application is checked under the quality assurance

specifications in the sequence of their development.

Quality control is the process of evaluating units to determine if they satisfy the specifications for

the final product. QC is important because it identifies and corrects problems and defects as they occur to make sure that the final product is the highest quality possible.

Points	Quality Assurance	Quality Control
Definition	QA is a group of activities which ensures that the quality of processes which is used during the development of the software always be maintained.	QC is a group of activities to detect the defects in the developed software.

Focus	The focus of QA is to **prevent defects** in the developing software by paying attention to processes.	The focus of QC is to **identify defects** in the developed software by paying attention to testing processes.
How	Establishment of the high-quality management system and periodic audits for conformance of the operations of the developing software.	Detecting and eliminating the quality problem elements by using testing techniques and tools in the developed software.
What	QA ensures prevention of quality problem elements by using systematic activities including documentation.	QC ensures identification and elimination of defects by using processes and techniques to achieve and maintain high quality of the software.
Orientation	QA is process oriented**.**	QC is product oriented**.**
Type of process	QA is a proactive process. It concerns to improve development so; defects do not arise in the testing period.	QC is a reactive process because it concerns to identify defects after the development of product and before its release.
Responsibility	Each and every member of the development team is responsible for QA	Only the specific testing team is responsible for QC

Example	Verification is the example of QA	Validation is the example of QC

9.Identification of tools used in software development

i. **Test Management Tools**:

- **JIRA**: Used for issue tracking and project management; can integrate with test management plugins.
- **TestRail**: A dedicated test management tool that allows for test case creation, execution, and reporting.

ii. **Automated Testing Tools**:

- **Selenium**: An open-source tool for automating web applications for testing purposes.
- **JUnit/TestNG**: Frameworks for unit testing in Java applications.
- **Cypress**: An end-to-end testing framework for web applications.

iii. **Performance Testing Tools**:

- **JMeter**: An open-source tool for performance testing, useful for load testing web applications.
- **LoadRunner**: A commercial tool for performance testing across various protocols.

iv.**Static Analysis Tools**:

- **SonarQube**: An open-source platform for continuous inspection of code quality, detecting bugs and vulnerabilities.
- **ESLint**: A static code analysis tool for identifying problematic patterns in JavaScript code.

v. **Continuous Integration/Continuous Deployment (CI/CD) Tools**:

- **Jenkins**: An open-source automation server that supports building, deploying, and automating projects.
- **CircleCI**: A CI/CD platform that automates the testing and deployment process.

vi.**Version Control Systems**:

- **Git**: A widely used version control system for tracking changes in source code.

- **GitHub/GitLab**: Platforms that provide Git repository hosting with additional features for collaboration and CI/CD.

vii.**Bug Tracking Tools**:

- **Bugzilla**: An open-source bug tracking system.
- **Redmine**: A flexible project management web application that includes issue tracking.

viii.**Security Testing Tools**:

- **OWASP ZAP**: An open-source tool for finding security vulnerabilities in web applications.
- **Burp Suite**: A comprehensive platform for web application security testing.

10.Identification of techniques of quality assurance in software development

i. Static Testing:

- **Code Reviews**: Manual inspection of code by peers to identify issues early.
- **Static Analysis**: Automated analysis of code to detect potential errors, vulnerabilities, and adherence to coding standards without executing the program.

ii. Dynamic Testing:

- **Unit Testing**: Tests individual components or functions of the software to verify they work as intended.
- **Integration Testing**: Focuses on verifying the interactions between integrated components or systems to identify interface defects.
- **System Testing**: Validates the complete and integrated software system against specified requirements.

iii. User Acceptance Testing (UAT):

- Conducted by end-users to ensure the software meets their requirements and is ready for deployment. It often involves real-world scenarios.

iv. Exploratory Testing:

- A hands-on approach where testers explore the application to identify defects, leveraging their experience and intuition rather than predefined test cases.

v. Regression Testing:

- Re-running previously executed tests to ensure that new code changes haven't adversely affected existing functionality.

vi. Performance Testing:

- Evaluates the application's responsiveness, stability, and scalability under varying loads. Types include:
    - **Load Testing**: Assesses how the system performs under expected user loads.
    - **Stress Testing**: Tests the system's limits by pushing beyond normal operational capacity.
    - **Endurance Testing**: Checks for performance degradation over extended periods.

vii. Security Testing:

- Involves testing the application for vulnerabilities and security weaknesses. Techniques include penetration testing, vulnerability scanning, and threat modeling.

viii. Compatibility Testing**:**

- Ensures that the software functions correctly across different environments, including various operating systems, browsers, and devices.

ix. Smoke Testing:

- A preliminary test to check the basic functionality of the application before further testing is conducted. Often referred to as a "sanity check."

x. Automation Testing:

Involves using **scripts** and tools to automate the execution of tests, improving efficiency and consistency, particularly for repetitive tasks and regression tests.

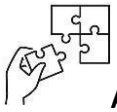
xi. Boundary Value Analysis:

- A testing technique that focuses on the values at the boundaries of input ranges, as errors often occur at these limits.

xii. Equivalence Partitioning:

Points to Remember

- Quality assurance (QA) is a crucial (systematic) process for ensuring that a product or service meets the desired level of quality.
- There are differences between quality control and quality assurance where Quality control is the focus of QC is to identify defects in the developed software system by paying attention to testing processes and Quality assurance is The focus of QA is to prevent defects in the developing software by paying attention to processes.
- In quality assurance we have different types which include**.** Functional QA**,** Non-Functional QA**,** Automated QA**,** Manual QA**,** Regression QA**,** User Acceptance Testing (UAT)**,** Compliance Testing**.**



Application of learning 1.1

ABA company is a company that develop an application to their clients. They want someone to explain them:

a) Quality assurance,
b) Types of Quality Assurance,
c) Difference between the Quality control vs quality assurance.

As a one of software system team, you are tasked to perform that activity.

Indicative content 1.2: Analysing Terms of Reference (ToR)

Duration:3

Theoretical Activity 1.2.1: Description of Terms of Reference

Tasks:

1. you are requested to answer the following questions related to the Quality Assurance:

   i. What do you understand by “terms of reference”?
   ii. Give the reason why the Terms of reference is necessary in Quality Assurance.
   iii. Outline the elements of Terms of reference.
   iv. What is the importance of Terms of reference?
   v. What is test case in terms of reference?
   vi. Describe document analysis report in terms of reference.

2. Provide the answers on papers or flipcharts.

3.Present the findings/answers to the whole class.

4.Ask questions for more clarifications if any.

5.Read the key reading 1.2.1 from your manuals for more clarification.

Key readings 1.2.1 Description of Terms of Reference (ToR)

1.Definition

Terms of Reference (ToR) is a formal document that outlines the objectives, scope, deliverables, roles, and responsibilities for a specific project, task, or engagement.

ToR sets clear guidelines for all parties involved, helping ensure alignment on goals and expectations before work begins

2.Purpose

- Clarifies Scope: Defines the boundaries of the work to be undertaken.
- Establishes Objectives: Outlines the specific goals and outcomes to be achieved.

- Provides Guidelines: Offers instructions and parameters for completing the task.
- Allocates Responsibilities: Assigns roles and duties to individuals or teams.
- Sets Deadlines: Establishes timelines for completion of different phases.
- Defines Deliverables: Specifies the expected outputs or results.
- Facilitates Communication: Ensures everyone is on the same page regarding expectations.

3.Elements of ToR

i. Background: Provides context and rationale for the project.
ii. Objectives: States the specific goals to be accomplished.
iii. Scope: Defines the boundaries of the project, including what is and is not included.
iv. Methodology: Describes the approach or techniques to be used.
v. Timeline: Outlines the project schedule with key milestones and deadlines.
vi. Deliverables: Specifies the expected outputs or results.
vii. Resources: Lists the necessary human, financial, and material resources.
viii. Roles and Responsibilities: Assigns tasks and duties to individuals or teams.
ix. Reporting Requirements: Defines the frequency and format of progress reports.
x. Evaluation Criteria: Specifies the metrics or standards for measuring success.

4.Importance of ToR

- Reduces Misunderstandings: Ensures everyone is aligned on expectations.
- Increases Efficiency: Provides a clear roadmap for the project.
- Enhances Accountability: Defines responsibilities and expectations.
- Facilitates Decision-Making: Offers a framework for evaluating progress and making adjustments.

Improves Project Management**:** Helps in planning, organizing, and controlling the project.

5.Test Case in Terms of Reference (ToR)

A test case is a collection of criteria or variables that will be used to determine whether an

application, software system, or one of its features is working as intended. For example, a test case can be written to ensure that the login functionality of a website works properly.

A test case generally has three components:

1. The name of the test case which describes its purpose.

2. The input data required to execute the test case.

3. The expected result of the test case.

6. Document analysis report in terms of reference

i. Title Page

- Title**:** Clearly state "Document Analysis Report."
- Subtitle (if any): Indicate the specific focus of the analysis.
- Prepared by**:** Names and titles of the authors.
- Date**:** Completion date of the report.

ii. Table of Contents

- Section Titles and Page Numbers**:** List all major sections and subsections with corresponding page numbers.

iii. Executive Summary

- Purpose: Summarize the objectives of the document analysis.
- Key Findings: Highlight the main conclusions of the analysis.
- Recommendations: Provide a brief overview of suggested actions or improvements.

iv. Introduction

- Context: Explain the background and the reason for the document review.
- Purpose of Analysis: Define the objectives and scope of the document analysis.
- Scope**:** Describe what is included and excluded from the analysis.

v. Methodology

- Approach: Detail the methods used to analyze the document.
- Criteria for Evaluation: Describe the specific criteria or standards used to assess the document (e.g., clarity, completeness, compliance, feasibility).
- Process: Outline the steps taken during the analysis.

vi. Findings

- Clarity and Completeness: Discuss issues related to the document's clarity and completeness.
  Examples: Provide examples to illustrate areas that were well-addressed or lacking.
- Alignment with Stakeholder Expectations: Evaluate how well the document meets stakeholder needs and expectations.

Gaps: Identify any discrepancies between stakeholder requirements and document content.

- Feasibility and Risks:
  - Feasibility: Assess the practicality of implementing the document's proposals or solutions.
  - Risks**:** Identify potential risks and their impact on the document's goals.

vii. Assumptions and Constraints

- Assumptions: Document any assumptions made during the analysis.
- Constraints: Identify any limitations or constraints that affect the document's implementation or analysis.

viii. Recommendations

- Actions: Provide specific recommendations to address issues identified during the analysis.
- Prioritization: Suggest the priority for addressing each issue or recommendation.

ix. Conclusion

- Summary of Findings: Recap the main findings and their implications.
- Overall Assessment: Provide an overall assessment of the document based on the analysis.

x. Appendices

- Supporting Information: Include additional documents, data, or information that supports the analysis.
- Glossary: Define any specialized terms used in the report.

xi. References

- Sources Cited: List all sources, documents, or references used in preparing the analysis report.

Practical Activity 1.2.2: Analyse Terms of Reference (TOR) document

Task:

Perform the following activity:

1. Read the key reading 1.2.2.
2. Referring to the theoretical activity 1.2.1, you are requested to do the task Below:
   i. As an software system developer, you are asked to analyse the given ToR document according to the requirement specification.
3. Referring to the task on point(i), make the ToR analysis report
4. Present to the trainer your ToR analysis report
5. Ask clarification where necessary if any
6. Perform application of learning 1.2.

Key readings 1.2.2: Analyse Terms of Reference (TOR) document

Steps to analyze the ToR in software system development

In software system development, **Terms of Reference (ToR)** is a formal document that outlines the purpose, structure, and scope of a system project. It provides a clear framework for stakeholders, project managers, and developers to understand the objectives, deliverables, and timelines of the project. Analyzing a TOR is an important step before the actual software system development begins. Here are the steps to analyze the ToR:

1. Understand the Scope and Objectives

- **Review the Project Background:** Analyze the context or reason why the project is being initiated, the business need, or the problem that the application is intended to solve.
- **Identify the Objectives:** Ensure that the key objectives of the software project are clearly stated. The objectives should be measurable and aligned with the overall goals of the company or business.
- **Clarify the Scope:** The ToR should clearly define what is included in the project and what is outside its boundaries (out of scope). Understand the functionalities,

systems, and modules that are part of the development.

2. Analyze Stakeholders' Roles and Responsibilities

- **List of Stakeholders:** Identify the key stakeholders involved, including clients, users, project sponsors, and team members.
- **Roles and Responsibilities:** Understand the specific responsibilities of each stakeholder, such as decision-making authority, approval processes, and communication points.

3. Examine Deliverables and Milestones

- **Deliverables:** Analyze the list of deliverables expected at each stage of the project. These can include design documents, source code, testing plans, and final software products.
- **Milestones and Deadlines:** Ensure the ToR outlines key project milestones and their corresponding deadlines. This helps in tracking the project's progress and ensures timely delivery.

4. Assess the Project Timeline and Phases

- **Timeline:** Examine the overall project timeline. Make sure it is realistic, and check whether the ToR specifies enough time for each phase of development (e.g., planning, coding, testing, deployment).
- **Phases:** Break down the software development into phases such as requirement gathering, design, development, testing, deployment, and post-launch support.

5. Review the Technical Specifications

- **Technical Requirements:** Check the technical requirements such as platforms, languages, databases, frameworks, and APIs. These specifications should align with the project's objectives and the company's technology stack.
- **System Integration:** If the software integrates with other systems or databases, make sure the ToR specifies those details and any technical dependencies.

6. Check Risk Management and Contingency Plans

- **Risk Identification:** Look for any risks identified in the ToR, such as technical, operational, or scheduling risks.
- **Mitigation Plans:** Ensure there are contingency plans or risk mitigation strategies in place to handle potential issues during the software development lifecycle.

7. Understand the Quality Assurance (QA) Strategy

- **Testing Requirements:** Review the testing plans to ensure there are appropriate testing methods outlined (unit testing, integration testing, user acceptance

testing).

- **Quality Standards:** Analyze the quality standards and performance metrics that the final software must meet.

8. Evaluate the Budget and Resources

- **Budget:** Check if the ToR defines the budget for the project, including costs related to development, testing, deployment, and any additional resources.
- **Resource Allocation:** Review how resources (both human and technical) will be allocated to meet the project's goals. Ensure that there are enough resources to complete the project within the defined timeframe.

9. Analyze the Communication Plan

- **Communication Channels:** Review how communication will flow between the team and stakeholders (meetings, reports, updates).
- **Reporting Schedule:** Ensure that the ToR includes a reporting schedule, detailing when and how progress updates should be shared with stakeholders.

10. Legal and Compliance Requirements

- **Licensing and Compliance:** Check for any legal and compliance requirements mentioned in the ToR, such as data privacy laws , software licenses, or intellectual property considerations.
- **Contracts and Agreements:** Make sure the ToR includes details of any contracts, third-party agreements, or service level agreements involved in the project.

11. Analyze Success Criteria

- **Success Metrics:** Ensure that there are clear criteria that define the success of the software project. This could include performance benchmarks, user satisfaction, or financial returns.
- **Final Approval Process:** Understand how the final approval or sign-off process will occur once the project is complete

Points to Remember

- Terms of Reference is a formal document that outlines the objectives, scope, deliverables, roles, and responsibilities for a specific project, task, or engagement.

- In Terms of Reference (ToR), there are Elements which are: Background, Objectives, Scope, Methodology, Timeline, Deliverables, Resources, Roles and Responsibilities, Reporting Requirements, Evaluation Criteria

- We use the Terms of Reference (ToR) for the following Purposes:
    - Clarifies Scope, Establishes Objectives, Provides Guidelines, Allocates Responsibilities, Sets Deadlines, Defines Deliverables, Facilitates Communication

- **Steps to analyse the ToR in software system development:**

1. Understand the Scope and Objectives
2. Analyse Stakeholders' Roles and Responsibilities
3. Examine Deliverables and Milestones
4. Assess the Project Timeline and Phases
5. Review the Technical Specifications
6. Check Risk Management and Contingency Plans
7. Understand the Quality Assurance (QA) Strategy
8. Evaluate the Budget and Resources
9. Analyse the Communication Plan
10. Legal and Compliance Requirements
11. Analyse Success Criteria

Application of learning 1.2.

ABA company develop an software system to their clients. They created a new software for a local banks in Rwanda but they have a problem of ensuring that the project aligns with the client's expectations and requirements. They want someone to analyse a Terms of Reference (ToR) document for that created application. As the one of the software development team, you are hired to perform this task.

Indicative content 1.3: Examining Requirement Specification

Duration:

Theoretical Activity 1.3.1: Description of requirement specification

Tasks:

1. you are requested to answer the following questions related to the Quality Assurance:
    i. What are the components of requirement specification document?
    ii. Describe functional and non-functional requirements for stakeholder.
    iii. Differentiate consistency and completeness requirement specification.
    iv. Identify dependencies and interactions in requirement specification.

2. Provide the answers on papers or flipcharts.

3.Present the findings/answers to the whole class.

4.Ask questions for more clarifications if any.

5.Read the key reading 1.3.1 from your manuals for more clarification.

Key readings 1.3.1: Description of requirement specification

1. Review the entire document

Means thoroughly reading and analysing the whole requirement specification document to ensure that it is comprehensive, well-organized, and fully addresses the project's needs.

This review aims to ensure that the document is organized, coherent, and comprehensive before diving into detailed analysis or evaluation.

Key components of requirement specification document include:

I. Title Page
II. Table of Contents
III. Executive Summary or Abstract
IV. Introduction
V. Quality Assurance Objectives
VI. Scope of Work
VII. Roles and Responsibilities

VIII. QA Methodology and Approach
IX. Test Plan
X. Test Cases and Procedures
XI. Test Data
XII. Risk Management
XIII. Quality Metrics and Reporting
XIV. Compliance and Standards
XV. Change Management
XVI. Documentation and Record-Keeping
XVII. Approval and Sign-Off
XVIII. Glossary
XIX. Appendices

2. Description of stakeholder requirements

1.Functional: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

2.Non-Functional: These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example 1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** 1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

3. Verify consistency and completeness

Consistency and **completeness** are two critical aspects of a well-written requirement specification. They ensure that the document is clear, accurate, and provides all the necessary information for successful project execution.

checking that the document is well-structured, logically coherent, and covers all necessary requirements without leaving out any important details. Here's a breakdown:

1. Verify Consistency

- Ensure that the document is consistent in language, structure, and information throughout.

- **What to Look for:**
    - **Terminology:** Terms, phrases, and acronyms should be used consistently across the document. For instance, if the term “user authentication” is used in one part, it should be consistently used in the same way throughout the document (and not called "login" in some sections).
    - **Requirement Formatting:** Functional and non-functional requirements should follow the same structure and format, making them easy to understand.
    - **No Contradictions:** Ensure that no requirements contradict one another. For example, the system shouldn't be required to allow both unlimited and limited access in different sections.
    - **Stakeholder Alignment:** The needs and priorities of different stakeholders should align consistently throughout the document.

2. Verify Completeness

- Ensure that the document fully covers all aspects of the system without missing any key information.
- **What to Look for:**
    - **Functional Requirements:** All expected system features, behaviors, and functions should be described. Every task the system must perform should be covered.
    - **Non-Functional Requirements:** Performance, scalability, security, and other qualities of the system should be clearly defined.
    - **Business and Stakeholder Needs:** Ensure that every stakeholder’s requirements are captured and no essential need is overlooked.
    - **Edge Cases and Exceptions:** Check that the document accounts for exceptional or unusual scenarios, including how the system will handle errors or unexpected conditions.
    - **Dependencies and Assumptions:** Ensure the document covers any dependencies on other systems or assumptions that need to be considered.

Why It Matters:

- **Consistency** ensures that everyone interprets the requirements the same way, reducing confusion or errors during implementation.
- **Completeness** ensures that no important requirement is left out, preventing the risk of incomplete system development or missed stakeholder expectations.

4. Identify dependencies and interactions

Means analyzing how different requirements, components, or systems rely on each other and how they interact. This is critical for understanding the

relationships between various parts of the system and ensuring that the requirements can be implemented without conflict or unexpected issues.

Identifying Dependencies

- **Direct Dependencies:**
  - Sequential Dependencies: Requirements that must be completed in a specific order.
  - Parallel Dependencies: Requirements that can be worked on simultaneously.
  - Conditional Dependencies: Requirements that depend on the outcome of other requirements.
- **Indirect Dependencies:**
  - Shared Resources: Requirements that rely on shared resources, such as hardware, software, or personnel.
  - Shared Constraints: Requirements that are subject to common constraints, such as budget, time, or performance.

Identifying Interactions

- Positive Interactions: When the fulfillment of one requirement enhances the value or benefit of another.
- Negative Interactions: When the fulfillment of one requirement conflicts with or hinders the fulfillment of another.
- Neutral Interactions: When there is no significant impact on the fulfillment of other requirements.

Here are some techniques to identify dependencies and interactions:

- **Dependency Mapping:** Create a visual representation of the relationships between requirements, using techniques like dependency diagrams or dependency matrices.
- **Impact Analysis:** Assess the potential impact of changes to one requirement on other requirements.
- **Risk Assessment:** Identify potential risks associated with dependencies and interactions.
- **Traceability Matrix:** Create a matrix to track the relationships between requirements and other project artifacts, such as design documents or test cases.

Practical Activity 1.3.2: Generation of findings document

Task:

Perform the following activity:

1. Read the key reading 1.3.2.

2. Referring to the theoretical activity 1.3.1, you are requested to do the task below:

   i) As a software developer you are asked to generate findings document the specification requirement

3. Referring to the task on point(i), Generate findings document

4.Present to the trainer your findings document

5. Ask clarification where necessary if any

6. Perform application of learning 1.3.

Key readings 1.3.2: Generation of findings document

When reviewing a requirement specification for a software project, generating a findings document involves the identification of inconsistencies, missing information, and overall completeness.

Steps of Generating a Findings Document

1. Review the Entire Document:

- **Read thoroughly:** Carefully read through the entire requirement specification, paying attention to each section and its content.
- **Take notes:** Make notes on key points, areas of concern, and potential issues.

2. Analyze Stakeholder Requirements:

- **Evaluate clarity:** Ensure that the requirements are clearly stated and understandable to all stakeholders.
- **Assess completeness:** Verify that all necessary requirements are included and that there are no gaps or omissions.
- **Check alignment:** Ensure that the requirements align with the project's goals and objectives.

3. Verify Consistency and Completeness:

- **Internal consistency:** Check for contradictions or inconsistencies within the document.
- **External consistency:** Verify that the requirements align with other project documents, such as the project charter or feasibility study.

- **Completeness:** Ensure that all necessary details are provided for each requirement.

4. Identify Dependencies and Interactions:

- **Dependency mapping:** Create a visual representation of the relationships between requirements.
- **Impact analysis:** Assess the potential impact of changes to one requirement on other requirements.
- **Risk assessment:** Identify potential risks associated with dependencies and interactions.

5. Conduct a gap Analysis:

- **Compare requirements:** Compare the requirements to the project's goals, objectives, and constraints.
- **Identify gaps:** Identify any gaps or inconsistencies between the requirements and the project scope.

6. Assess Feasibility:

- **Evaluate feasibility:** Determine if the requirements are achievable within the project's budget, timeline, and resources.
- **Consider constraints:** Identify any constraints that may impact the feasibility of the requirements.

7. Identify Potential Risks:

- **Risk assessment:** Identify potential risks associated with the requirements, such as technical challenges or dependencies.
- **Risk mitigation:** Develop strategies to mitigate identified risks.

8. Gather Feedback:

- **Stakeholder input:** Collect feedback from stakeholders to ensure that their requirements are accurately captured.
- **Address concerns:** Address any concerns or questions raised by stakeholders.

9. Organize Findings:

- **Structure the document:** Organize your findings into a clear and concise document.
- **Use headings and subheadings:** Use headings and subheadings to organize the

information.

Points to Remember

- Description of functional and Non-functional requirements in requirement specification.
- Verification of consistency and completeness in requirement specifications.
- Identification of dependencies and interactions in requirement specifications.
- While Generating a Findings Document pass through the following steps:

1. Review the Entire Document
2. Analyze Stakeholder Requirements
3. Verify Consistency and Completeness
4. Identify Dependencies and Interactions
5. Conduct a gap Analysis
6. Assess Feasibility
7. Identify Potential Risks
8. Gather Feedback
9. Organize Findings

Application of learning 1.3.

XY team has completed a major phase of their project, which involves building a cloud-based inventory management software for a retail company. After conducting reviews on the system's functionality and performance. As the one of the team, you are tasked for generating a findings document.

Indicative content 1.4: Analysing Inception Report

Duration:

Theoretical Activity 1.4.1: Description of the inception report

Tasks:

1. you are requested to answer the following questions related to the Quality Assurance:

i. What is inception report?
ii. What is document review in inception report?
iii. What do you understand by ”stakeholder expectations”?
iv. What is the clarity and completeness in inception report?
v. What is the feasibility and risks?
vi. Describe document assumptions and constraints
vii. Describe Document analysis findings

2. Provide the answers on papers or flipcharts.

3.Present the findings/answers to the whole class.

4.Ask questions for more clarifications if any.

5.Read the key reading 1.4.1 from your manuals for more clarification.

Key readings 1.4.1: Description of the inception report

1.Inception Report

An inception report is an initial document prepared at the start of a project to outline the approach, scope, and planning details. It provides clarity on the project’s objectives and sets expectations for all stakeholders.

2. Document Review:

The document review section examines all relevant documents, reports, policies, and data available at the start of the project. This review aims to ensure alignment with the project's goals, uncover any pre-existing research or baseline data, and identify gaps or areas that require further exploration.

3. Stakeholder Expectations:

stakeholder expectations is critical to the success of the project. This section details the perspectives, needs, and concerns of all stakeholders involved, including clients, sponsors, beneficiaries, and partners.

4. Clarity and Completeness:

Clarity and completeness section assesses how well the project objectives, deliverables, and milestones have been defined. This part of the inception report ensures that all parties have a common understanding of the project's scope and expected outcomes.

5. Feasibility and Risks

Feasibility and risks refer to evaluating whether the project can realistically be completed within the specified time, budget, and resources. This section also identifies potential risks that could impact the project and suggests mitigation strategies.

6. Assumptions and Constraints:

The assumptions and constraints section outlines key assumptions the project team is making, and any limitations that may affect project delivery.

7. Analysis Findings:

This section presents the analysis findings derived from the initial investigation, document review, stakeholder consultations, and feasibility study. The analysis outlines the key insights gained so far, highlighting critical areas for further focus during the project implementation phase.

Practical Activity 1.4.2: Analysing inception report

Task:

Perform the following activity:

1: Read the key reading 1.4.2.

2: Referring to the theoretical activity 1.4.1, you are requested to do the task below:

i) As a software developer you are asked to analyse the inception report

3: Referring to the task on point(i), analyze inception report

4. present to the trainer your inception report

5: Ask clarification where necessary if any

6: Perform application of learning 1.4.

Key readings 1.4.2: Analysing an Inception Report

Analysing an Inception Report

Analyzing an inception report is a critical step to ensure the project is on the right path from the outset. It involves reviewing the foundational documents, understanding stakeholder expectations, and assessing the project's clarity, feasibility, and risks.

Below is a structured guide to analysing an inception report:

1. Perform Document Review

The document review is the foundation of the inception report analysis, ensuring all relevant materials have been examined for consistency and alignment with the project's objectives.

-Steps:

- Collect all relevant documents, including baseline studies, previous project reports, policies, and frameworks.

- Review how these documents align with the current project objectives.

- Identify gaps, outdated information, or areas that require further exploration.

2. Stakeholder Expectations:

Understanding stakeholder expectations is essential for aligning project goals with the interests and needs of those involved or impacted by the project.

Steps:

1. Engage stakeholders through interviews, surveys, or consultations.
2. Identify the key concerns, needs, and priorities of each stakeholder group (e.g., clients, beneficiaries, partners).
3. Assess any conflicting interests or expectations among stakeholders.

3. Assess Clarity and Completeness:

Assessing clarity and completeness helps ensure that the project's objectives, deliverables, timelines, and roles are well-defined and understood by all participants.

Steps:

1. Review the project scope, goals, and deliverables outlined in the inception report.

2.Check for ambiguities or missing information regarding roles, responsibilities, and timelines.

3.Validate the completeness of milestones and whether all key aspects of the project are accounted for.

4. Evaluate Feasibility and Risks:

Evaluating feasibility and risks is crucial to ensure that the project can be realistically executed within the allocated resources, time, and scope. Identifying risks early helps in developing mitigation strategies.

Steps:

1. Assess the project's timeline, budget, and resource allocation to determine if they are sufficient.
2. Identify potential risks (technical, financial, legal, logistical) that could impact project delivery.
3. Develop a risk management plan, outlining how identified risks will be mitigated.

5. Document Assumptions and Constraints:

Clearly documenting assumptions and constraints ensures that all team members and stakeholders understand the boundaries within which the project will operate, as well as any assumptions that may impact the project's success.

Steps:

1. Identify key assumptions made in the project plan (e.g., availability of resources, stakeholder cooperation, external conditions).

2.Document constraints such as legal, financial, or technological limitations that may affect the project's execution.

6. Document Analysis Findings:

The final step is to synthesize the findings from the document review, stakeholder consultations, risk analysis, and other assessments. This forms the basis for finalizing the project strategy.

Steps:

1.Summarize the key insights and findings from each section of the analysis.

2.Identify opportunities for improvement, areas of concern, and critical success factors.

3.Provide recommendations for refining the project plan based on the analysis findings.

Points to Remember

- An inception report is an initial document prepared at the start of a project to outline the approach, scope, and planning details.
- In Inception Report, there are Key Components which are: Document Review, Stakeholder Expectations, Clarity and Completeness, Feasibility and Risks, Assumptions and Constraints, Analysis Findings
- **Steps of analysing inception report are:**

1.Understand the Project Context and Objectives

2. Examine the Scope of the Project

3. Analyze the Methodology

4. Assess the Work Plan and Timeline

5. Evaluate Stakeholder Engagement and Roles

6. Check Resource Allocation

7. Risk Assessment and Mitigation Strategies

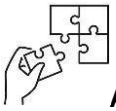
8. Analyze the Monitoring and Evaluation Plan

9. Review the Legal and Compliance Requirements

10. Examine Assumptions and Dependencies

11. Ensure Alignment with Project Goals and Vision

12. Feedback and Validation



Application of learning 1.4.

XZ Company is planning to launch a new educational program focused on learning software system for working professionals. The program aims to offer flexible, high-quality courses in various fields. XZ company's administration has a prepared inception report. As a trainee , you are tasked to analyse this inception report to ensure that the program is viable, aligned with stakeholder expectations, and free of significant risks before proceeding.

Learning outcome 1 end assessment

Written assessment

Q1. **Read the Following statement and answer by true if correct or false otherwise**

1. QA is solely responsible for finding defects in software.
2. QA should be performed only after development is complete.
3. A high defect rate indicates poor software quality.
4. QA teams are responsible for writing requirements and specifications.
5. QA teams should always strive for zero defects.
6.QA is only concerned with functional correctness.
7. Automated testing tools can completely replace manual testing.
8.QA teams should be isolated from development teams to avoid conflicts of interest.
9.Quality Assurance (QA) is primarily focused on preventing defects in a product or service.
10.The main stages of the Quality Assurance process include planning, assurance, and control.

11. Quality Control (QC) is a proactive process, while Quality Assurance (QA) is reactive.

12.ISO 9001 is a widely recognized standard for Quality Management Systems.
13.One of the main benefits of Quality Assurance is improved customer satisfaction.
14.Statistical process control (SPC) is a tool used in Quality Control, not in Quality Assurance.
15.Quality Assurance involves the use of methods like audits and process evaluations.
16.Lean methodology is an example of a Quality Assurance standard.

Q2. Circle the letter corresponding with the correct answer

i.Which of the following is NOT an example of industry-specific QA?

a. Software QA
b. Healthcare QA
c. Financial QA
d. Retail QA

ii. Which of the following is a benefit of manual testing?

a. Faster test execution
b. Higher test coverage
c. Ability to detect subtle defects
d. Lower cost

iii. The main objective of quality assurance is:

a. Proof of fitness product
b. Inspection of quality of product
c. Quality conormance
d. Customer satisfaction

iv. The primary goal of analyzing an inception report is to:

a. Assess the project's feasibility and risks
b. Review the project's documentation
c. Identify stakeholders
d. All of the above

v. What is the purpose of documenting assumptions and constraints in an inception report analysis?

a. To identify potential risks
b. To clarify project scope
c. To ensure project feasibility
d. All of the above

vi. The findings from an inception report analysis are used to:

a. Develop a project plan
b. Identify potential risks
c. Communicate project goals to stakeholders
d. All of the above

Q3**.** What is Quality Assurance (QA) in software development?

Q4. What is the main difference between Quality Assurance (QA) and Quality Control (QC)?

Q5. Q5. Below there are incomplete sentences about quality assurance and quality control.

Choose and fill in the blank space with the appropriate words from the provided ones in the following given box.

1……..is part of quality management focussed on providing confidence that quality requirements will be fulfilled.

a. Quality management
b. Quality
c. Quality assurance
d. Updating

2. Fil The first step in analyzing an inception report is to perform a **……..**

a. document review
b. stakeholder expectations
c. document analysis findings

Q6. What is the difference between functional and non-functional requirements?

Practical assessment

XYZ company, is embarking on a new software development project to launch an advanced customer relationship management (CRM) system. The company wants to ensure the quality of the software. The Quality Assurance (QA) team is responsible for ensuring the product meets both functional and non-functional requirements while adhering to industry standards.

The QA team is tasked with

1. Implementing quality assurance processes,
2. Analysing the Terms of Reference (ToR),
3. Reviewing the requirement specifications,
4. Assessing the project's inception report.

END

Reference

ABBAS, T. (2023). *How to write terms of refernce for project.* TAHIR ABBAS.

Dayton, D. (2020, January 15). *how to write an inception report in requirement specification*. Retrieved from bizfluent: https://bizfluent.com

Enov8. (2024, September 16). Retrieved from www.enov8.com: https://www.enov8.com

GeeksforGeeks. (2024, october 11). *Functional vs No functional requirements*. Retrieved from www.geeksforgeeks.org.

Krawczyk, B. (2022). *Quality Assurance in computer system.* Bart Krawczyk.

Lambda. (2024). *COMPUTER SYSTEM QUALITY ASSURANCE.* Lambda TEST.

Lambdatest. (2024, June 13). *What Is Test Plan? Its Components, Types, and Example*. Retrieved from www.lambadatest.com: https://www.lambdatest.com/learning-hub/test-plan

MORKOVICH, E. (2023). *TERMS OF REFERENCE IN SOFTWARE.* ERIC MORKOVICH.

Singh, R. (2024, July 31). *top software testing tools 2024*. Retrieved from www.headspin.io/: https://www.headspin.io/blog/top-software-testing-tools

Testing, G. A. (2024, March n.d). *www.globalapptesting.com/*. Retrieved from globalapptesting: https://www.globalapptesting.com/


SOFTWARE TESTING
Performance Testing
Security
SOFTWARE TESTING

Indicative contents

2.1 Preparation of Test Plan
2.2 Preparation of Testing Environment
2.3 Perform Testing

Key Competencies for Learning Outcome 2: Test the System

Knowledge	Skills	Attitudes
• Description of Test plan. • Identification of tools. • Identification of techniques • Identification of test cases. • Identification of test criteria. • Preparation of Test Execution Scripts • Identification of key components of a Testing Environment • Description of performing test • Identification of testing tools • Description of test delivery plan	• Prepare Test Data • Scheduling test delivery plan • Selecting test techniques • Selecting software testing tools • Setting up and Configuration testing environment • Perform Testing	• Having Curiosity in System testing • Having Open-mindedness in System testing • Having Attention in System testing • Having Collaboration in System testing • Having Analytical Thinking in System testing • Having Critical Thinking in System testing • Having Commitment in System testing

Duration: 10 hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Test clearly Plan used in software system Testing.
2. Test properly Environment used in software system Testing
3. perform correctly Testing in software system Testing

Resources

Equipment	Tools	Materials
• Computer • Smartphone • Tablet	• Microsoft Visio • LucidChart • DrawIO	• Pen • Notebook

Indicative content 2.1: Preparation of Test Plan

Duration:

Theoretical Activity 2.1.1: Description of test plan

Tasks:

1.Answer the following question:

i. What do you understand by " test plan" ?
ii. What are types of test plan?
iii. What are the components of test plan?
iv. Mention tools used for testing the system?
v. Identify test criteria to consider during testing system.
vi. Identify test techniques used for testing the system?
vii. Describe Test cases in testing system.
viii. What is test data in system testing.
ix. Describe types of test data used in testing system.
x. Give the importance of test data.
xi. Describe test execution scripts.
xii. What are components of test delivery plan?

2.Participate in group formulation

3.Present the findings to your class or colleagues

4.Ask the clarification if any

5.Read the trainee manual key reading on 2.1.1

Key readings 2.1.1: Description of test plan

1.Definition of test plan

Is a formal document that outlines the overall strategy, objectives, resources, and scope of testing activities for a computer system project. It serves as a roadmap for the testing process and defines how testing will be carried out to ensure that the software meets its requirements and functions correctly. These documents help guide teams through the software testing life cycle, ensuring they have the time and resources required for solid testing.

2. Type of test plan

i. Master Test Plan

In this type of test plan, includes multiple test strategies and has multiple levels of testing. It goes into great depth on the planning and management of testing at the various test levels and thus provides a bird's eye view of the important decisions made, tactics used, etc. It includes a list of tests that must be executed, test coverage, the connection between various test levels, etc.

ii. Test Phase Plan

In this type of test plan, emphasis is on any one phase of testing. It includes further information on the levels listed in the master testing plan. Information like testing schedules, benchmarks, activities, templates, and other information that is not included in the master test plan is included in the phase test plan.

iii. Specific Test Plans

Plans for conducting particular testing, such as performance and security tests. For instance, performance testing is software testing that aims to ascertain how a system responds and performs under a specific load. Security testing is software testing that aims to ascertain the system's vulnerabilities and whether its data and resources are safe from potential intruders.

3.Components of test plan

A test plan usually comprises various elements that offer in-depth details regarding the testing procedures for a particular software application .

The essential components of a test plan include:

- **Scope:** Outlines the goals of the specific project and provides information about user scenarios intended for testing purposes. The scope may also specify scenarios or issues that will not be addressed by the project if required.
- **Schedule:** Specifies the commencement dates and deadlines for testers to provide outcomes.
- **Resource Allocation:** Specifies the allocation of specific tests to individual testers.
- **Environment:** Describes the characteristics, setup, and accessibility of the test environment.
- **Tools:** Specifies the tools that will be employed for testing, reporting bugs, and other pertinent activities.
- **Defect Management:** Outlines the procedure for reporting bugs, including the designated recipients and the required accompanying elements for each bug report. This may include specifications such as whether bugs should be reported with screenshots, textual logs, or videos demonstrating their occurrence in the code.
- **Risk Management:** Specifies potential risks that could arise during the testing of the software and the risks the software might face if released without adequate testing.
- **Exit Parameters:** Specifies the cessation point for testing activities. This section delineates anticipated outcomes from QA operations, providing testers with a standard against which they can measure actual results.

4. Identify software testing tools

i. **Unit Testing Frameworks**:

- Tools for testing individual components of code. Examples include:
    - **Unity**: A lightweight framework for C.
    - **CppUTest**: A testing framework for C++.

ii. **Integration Testing Tools**:

- These tools test interactions between integrated components. Common options include:
    - **Catch2**: A modern C++ testing framework.
    - **Google Test**: A widely used C++ testing framework.

iii. **Simulation Tools**:

- Tools that simulate the behavior of software systems without physical hardware. Examples include:
    - **MATLAB/Simulink**: Used for modeling, simulation, and analysis.
    - **QEMU**: An open-source emulator that simulates various platforms.

iv. **Software-in-the-Loop (HIL) Testing Tools**:

- These tools use simulation to test software in real-time. Examples include:
    - **dSPACE**: Provides HIL testing solutions.
    - **NI VeriStand**: A platform for developing HIL simulations.

v. **Debugging Tools**:

- Essential for identifying and fixing issues in software. Examples include:
    - **GDB (GNU Debugger)**: A versatile debugger for various programming languages.

vi.**Static Analysis Tools**:

- Tools that analyze code without executing it to find potential errors. Examples include:
    - **Coverity**: A static analysis tool for C/C++.
    - **PC-lint**: A static code analysis tool for C/C++.

vii. **Dynamic Analysis Tools**:

- These tools analyze the program during execution to monitor performance and behavior. Examples include:
    - **Valgrind**: A tool for memory debugging and profiling.
    - **Gprof**: A performance analysis tool for C/C++ programs.

viii. Performance Testing Tools:

- Tools that assess the performance characteristics of software. Examples include:
    - **Benchmarking tools**: Such as lmbench and sysbench.

ix. **Test Automation Tools**:

- Tools that automate the testing process to improve efficiency. Examples include:
    - **Robot Framework**: A generic test automation framework.
    - **Jenkins**: For continuous integration and testing.

x. **Code Coverage Tools**:

- Tools that measure how much of the code is executed during tests, helping identify untested paths. Examples include:
    - **gcov**: A code coverage analysis tool for C/C++.
    - **lcov**: A graphical front-end for gcov.

5. Identify software test techniques

Testing techniques are essential methods used to design test cases and validate a system's functionality, performance, and security. Selecting appropriate testing techniques ensures comprehensive coverage, identifies defects, and improves the overall quality of the system

Software testing techniques are broadly classified into two main categories based on the running of the codebase. Those are as follows:

- Static testing
- Dynamic testing

1.Static Testing

Static testing is mainly used to find the error or flaw in the software application without actually running the code. In other words, static testing is a software testing approach that involves reviewing and analyzing software documentation, design, or code without executing the software. It's a method of checking for errors and defects in the software by inspecting it in a static state rather than during active operation.

Static testing includes two techniques:

- Reviews
- Static Analysis

Reviews

Reviews are a crucial aspect of static testing, ranging from informal peer reviews between developers/testers on artefacts (code/test cases/test data) to formal inspections. They allow testers to identify defects and issues in documentation, such as requirements and design, at an early stage. Reviews are executed manually by various project members, including architects, designers, managers, moderators, and reviewers.

Static Analysis

Static analysis, a software testing technique, involves reviewing code without executing it. Essentially, it's a form of code analysis performed to understand the code structure and adhere to coding standards. Mainly, it is regarded as the debugging process that aims to test the source code to find any structural error in the developer's code:

- Unused variables
- Syntax violations
- Non-compliance with coding standards
- Dead code

Dynamic Testing

Dynamic testing involves analyzing the dynamic behavior of code within a software application. During this process, it's crucial to provide input and output values as expected when executing a test case. This can be done manually or through automation. Additionally, compiling and executing the software code is necessary.

The main objective of dynamic testing is to validate the software, ensuring it functions correctly without any defects post-installation. Essentially, dynamic testing verifies the overall functionality and performance of the software application, aiming for stability and consistency.

Dynamic testing includes three techniques:

- White box testing
- Black box testing
- Grey box testing

White Box Testing

White box testing involves verifying the internal structure and code of the software. This necessitates coding expertise as it includes evaluating the internal code implementation and algorithms of software applications.

The following are the different software testing techniques under white box testing:

- **Statement coverage:** A primary objective of white box testing is to have a maximum portion of the source code. Code coverage indicates the extent to which an application's code is subjected to unit tests that verify its functionality.

  Statement coverage=(Total number of statements/Number of executed statements)×100

- **Branch coverage:** In programming, a branch refers to a point in the code where the program's execution flow can diverge and follow different paths based on certain conditions. Consequently, in branch coverage, we verify whether each branch is executed at least once.

  Branch coverage=(Total number of branches/Number of executed branches)×100

- **Path coverage:** In this software testing technique, all the paths within the software application are explicitly tested. This technique ensures that each potential application path is traversed at least once, offering more effectiveness compared to branch coverage.
- **Decision coverage:** It is a technique that verifies the true or false outcomes of every boolean expression in the source code. Decision coverage aim is to cover and validate all available source codes by ensuring that each branch of every potential decision point is visited at least once.
- **Condition coverage:** This software testing technique is also known as expression coverage. It evaluates the variables or sub-expressions in a conditional statement. Its goal is to evaluate the outcome of each logical condition.
- **Multiple condition coverage:** This testing method evaluates all possible combinations of conditions for each decision.
- **Finite state machine coverage:** It is challenging to achieve a finite state machine due to its focus on the design's functionality. This approach involves tallying the frequency of state visits or transitions and determining the number of sequences within a finite state system.
- **Control flow testing:** This testing technique aims to ascertain the program's execution sequence using its control structure. Test cases are formulated based on the program's control structure, and a specific section of the program is selected to construct the testing path.

Black Box Testing

Black box testing involves validating the software application without access to its internal structure, code, or design. Thus, programming expertise is not required for this testing method. Its goal is to confirm the functionality of software applications under test and execute the entire test suite without programming knowledge, adhering to requirements and specifications.

Following are the different software testing techniques under black box testing:

- **Boundary value analysis:** This testing technique focuses on values at the boundaries of input ranges to verify if a system handles extreme values within acceptable ranges correctly. By testing values like the minimum, maximum, and just beyond these boundaries, testers can uncover potential issues related to boundary conditions.

  There are two types of boundary testing:

  - **Inner boundary testing**: It focuses on input values just inside the input domain, including the minimum and maximum allowed values.
  - **Outer boundary testing**: It focuses on input values slightly outside the input domain, such as values slightly above or below the minimum and maximum allowed values.

- **Equivalence class partitioning:** This technique functions as a method in software testing for dividing potential inputs into sets of equivalence classes, aiming to identify and test a representative selection of inputs from each class.

- **Decision table-based testing:** This technique is used to visually illustrate various combinations of inputs and outputs, focusing more on business rules. It's employed in scenarios where diverse combinations of test input conditions yield different outputs.

- **State transition testing:** This technique ensures software application behaves as expected during transitions between different system states. Initially, it identifies a finite set of states the software application can occupy and then tests how it transitions between them based on varying input conditions.

Grey Box Testing

Grey box testing involves integrating aspects of both white box and black box testing. Here, testers engage in both types of testing procedures, hence the term

"grey box." In this approach, testers have partial visibility into the internal coding, resembling a semi-transparent grey box.

The following are the different software testing techniques under grey box testing:

- **Matrix testing:** It involves evaluating business and technical risks developers identify in software applications. Developers specify all program variables, each carrying inherent technical and business risks and potentially utilized with varying frequencies throughout its lifecycle.
- **Pattern testing:** It involves analysis to determine the root causes within the code. The analysis template includes defect reasons, facilitating proactive identification of potential failures before production.
- **Orthogonal array testing:** It involves numerous permutations and combinations of test data. It is used when maximal coverage is necessary with few test cases and extensive test data, proving valuable in evaluating complex software applications.
- **Regression testing:** It involves evaluating software applications after each change to ensure that modifications or new functionalities do not adversely impact existing software application workings. It also confirms that rectifying defects does not disrupt other software functionalities.
- **API testing:** It involves testing the software applications' exposed interfaces. The primary aim is to verify that the API accepts various input formats and functions as intended.

6. Identification of test cases

Test cases are designed to verify that your application is operating as expected. Test case writers design test cases so testers can determine whether an app or software system's feature is working correctly.

1.functional testings Test the system's overall functionality to ensure it meets requirements.

- **Valid Input:** Test with valid input data to ensure the system processes it correctly.
    - Example: Test login with valid username and password.
- **Invalid Input:** Test with invalid input data to verify error handling and data validation.
    - Example: Test login with invalid username or password.
- **Boundary Values:** Test at the boundaries of input data ranges to identify potential issues.

Example: Test password length limits.

- **Equivalence Classes:** Test representative values from different equivalent classes of input data.

  Example: Test login with different types of usernames (e.g., email, username).

- **Decision Table Testing:** Test different combinations of input conditions using decision tables.

Example: Test different combinations of user roles and permissions.

2.Non-Functional Testing: Test performance, security, usability, and other non-functional aspects.

- **Performance Testing:**

  - Load testing: Test the system's performance under expected workloads.
  - Example: Test the system's response time under peak load.
  - Stress testing: Test the system's behavior under extreme conditions.
  - Example: Test the system's performance with excessive data.
  - Endurance testing: Test the system's performance over a long period.
  - Example: Test the system's reliability after running for 24 hours.

- **Security Testing:**

  - Vulnerability scanning: Identify potential security vulnerabilities.

Example: Scan the system for SQL injection vulnerabilities.

  - Penetration testing: Simulate attacks to assess the system's security defenses.

Example: Attempt to gain unauthorized access to the system.

- **Usability Testing:** Test how easily users can complete tasks.
  - Example: Test how long it takes users to complete a registration process.
  - Error messages: Evaluate the clarity and helpfulness of error messages.

    Example: Test if error messages are clear and provide guidance.

  - User satisfaction: Gather feedback on the system's overall usability.

Example: Conduct user surveys to assess satisfaction.

3.Integration Testing

- **Interface Testing:** Test the interactions between different components of the system.
  - Example: Test the communication between the frontend and backend.
- **Data Flow Testing:** Test the flow of data through the system.
  - Example: Test the data flow from input to output.
- **Dependency Testing:** Test the system's behavior when dependent components are unavailable or malfunctioning.

Example: Test the system's behavior when the database is unavailable.

4.Acceptance Testing

- **Alpha Testing:** Test the system by internal users within the organization.
  - Example: Have developers and testers use the system to identify issues.
- **Beta Testing:** Test the system by external users who represent the target audience.
  - Example: Distribute the system to a group of potential users for feedback.

7. Identification of test criteria

Test criteria in a test plan define the conditions that must be met during the testing process to ensure the software's quality and readiness for release.

These criteria are divided into main categories:

1. **Entry Criteria**

Entry criteria refer to the conditions that must be fulfilled before testing can begin. This ensures that the testing environment is stable, the test artifacts are ready, and all necessary prerequisites have been met. Common entry criteria include:

- **Test Environment Setup**: The test environment is correctly configured and all necessary software, hardware, and network configurations are in place.
- **Test Data Availability**: All required test data is prepared and validated.
- **Test Plan Approval**: The test plan, test strategy, and test cases are reviewed and approved by relevant stakeholders.
- **Code Deployment**: The build to be tested is successfully deployed in the test environment.

- **Dependencies Resolved**: All external dependencies, such as third-party services or APIs, are available and stable.
- **Tools and Resources Availability**: All testing tools, scripts, and resources are accessible and functional.
- **Bug Tracking System Setup**: The bug tracking system is ready, and testers can log and track defects.

2. **Exit Criteria**

Exit criteria specify the conditions that must be satisfied before testing can be concluded and the product is considered ready for release. It ensures the completeness and quality of the testing process. Common exit criteria include:

- **Test Case Execution**: All planned test cases are executed (or a significant percentage, like 95-100%, depending on risk and coverage).
- **Pass/Fail Rate**: A specific percentage of test cases have passed (e.g., 95% of test cases must pass before the product is released).
- **Critical Bugs Closed**: All critical and high-severity defects must be resolved and verified. Medium or low-priority bugs may be accepted depending on the business impact.
- **Test Coverage**: The required level of test coverage is achieved (e.g., code coverage, functional coverage).
- **Performance Testing**: Performance and load tests meet the defined benchmarks for response time, scalability, and resource utilization.
- **Regression Testing**: No new critical bugs are found in regression tests.
- **User Acceptance Testing (UAT)**: The product has been approved by users or stakeholders after UAT.
- **Compliance with Standards**: The product meets any regulatory, legal, or company-defined standards.
- **Sign-off**: Testing has been signed off by the test manager or key stakeholders, indicating readiness for release.

3. **Suspension and Resumption Criteria**

These criteria determine when testing activities should be paused or resumed.

- **Suspension Criteria**: Conditions under which testing is temporarily stopped, such as:
    - Build instability or test environment failures.
    - Unavailability of key resources (e.g., testers or test environments).
- **Resumption Criteria**: Conditions under which testing can resume, such as:

- The critical issues that caused the suspension are resolved.
- Environment or build stability is restored.

8. Allocate resources

Prioritization of testing activities is essential to effectively allocate resources. It is crucial to identify critical functionalities and high-risk areas of the software to allocate resources accordingly. By identifying potential risks and impact, you can allocate resources to where they are most needed.

How to allocate resources on a project?

The following five steps are important when allocating available resources as part of project management:

1. **Plan.** Project managers should first map out the project. They must divide the project into separate tasks and identify what skills are needed. They also must examine any constraints, such as the deadlines and budget. Project managers should also identify potential team members based on their skills and availability.
2. **Gauge availability.** Sick time, vacation time, holidays and other projects all impact a team's availability. During this stage, managers must establish lines of communication with team members so resource allocation and shifts and changes in the project or its schedule can be communicated.
3. **Schedule.** Managers assign tasks and develop project timelines. They use resource management tools to automate and streamline this process and improve workload management.
4. **Track.** Once the project begins, it's important to track the performance of team members and monitor how effectively they complete tasks. Resource allocations should be adjusted to maximize efficiency and take advantage of new opportunities that arise.
5. **Evaluate.** The success of the project is evaluated based on metrics that show how well it met expectations. Data from these findings can be used to refine resource allocation strategies in new projects.

9.Types of Test Data

In software testing, Test data is a set of data used to validate the correctness, completeness, and quality of a software program or system.

There are different types of test data which are :

1. Valid Test Data: Inputs that meet the requirements and are expected to yield successful outcomes (e.g., correct usernames and passwords).

2. Invalid Test Data: Inputs that do not meet requirements and should trigger error messages or failures (e.g., incorrect passwords or invalid email formats).
3. Boundary Test Data: Data that tests the limits of input fields (e.g., the maximum and minimum values).
4. Null or Empty Test Data: Inputs that test how the application handles null or empty fields.
5. Performance Test Data: Large datasets used to assess the application's performance under load (e.g., simulating thousands of users).
6. Security Test Data: Data designed to test security measures, including malicious inputs (e.g., SQL injection strings).

10. Importance of test data

Here are a few important benefits of test data:

- **Offers the ability to identify coding errors:** Test data can help researchers identify coding errors quickly before the release of a program. It can also help improve the security of programs.
- **Provides a foundation for additional testing:** Test data provides a foundation to develop further data tests. It first tests the most basic inputs before moving on to the program's purpose.
- **Identifies redundancy or unnecessary duplication:** Test data can help designers find redundancies or unnecessary duplications of code. This can help lighten code and create a more efficient site.
- **Provides flexibility in managing applications:** Collecting test data can give designers flexibility when managing many applications, especially on several platforms.

11. Description of Test Execution Scripts

A test script should include all the steps to test a specific functionality or action in the required order of testing. In a test script, each test step should include necessary entries, like test data, and it should also specify the expected results.

Key Components of a Test Execution Script

1. Test Case ID: A unique identifier for the test case.
2. Test Case Description: A brief explanation of the test case's purpose.
3. Test Steps: A sequential list of actions to be performed.
4. Expected Results: The anticipated outcomes of each test step.
5. Actual Results: The observed results after executing the test step.
6. Status: The status of the test case (e.g., Passed, Failed, Blocked).

7. Comments: Additional notes or observations.

Example of Test Execution Script

Test Case ID	Test Case Description	Test Steps	Expected Results	Actual Results	Status	Comments
TC001	Login Functionality	1. Enter valid username	Login successful	Login successful	Passed	N/A
		2. Enter valid password		Login successful	Passed	N/A
		3. Click Login button		User is redirected to the dashboard	Passed	N/A
TC002	Invalid Login	1. Enter invalid username	Login Failed	Login failed	Passed	N/A
		2. Enter valid password		Login failed	Passed	N/A
		3. Click Login button		Error message displayed	Passed	N/A

12. Components of Test Delivery Plan

Key components of a test delivery plan include:

1. **Project Overview**: A brief description of the project and its objectives.
2. **Scope of Testing**: Specifies what features, functions, or areas will be tested.
3. **Testing Strategy**: An overview of the testing methods and techniques to be employed (e.g., manual vs. automated testing).
4. **Test Schedule**:
    - **Milestones**: Key dates for deliverables and activities (e.g., completion of test planning, execution start, UAT).
    - **Testing Phases**: Breakdown of testing into phases (e.g., unit testing, integration testing).
5. **Resource Allocation**: Identification of team members involved in the testing process, along with their roles and responsibilities.
6. **Risk Management**: Potential risks associated with the testing process and strategies to mitigate them.
7. **Communication Plan**: Guidelines for how updates and reports will be communicated to stakeholders.
8. **Deliverables**: Expected outputs from the testing process (e.g., test reports, defect logs, final summary).

Practical Activity 2.1.2.: Schedule test delivery plan

Task:

Perform the following activity:

1. Read the key reading 2.1.2

2. Referring to the theoretical activity 2.1.2, you are requested to do the task below:

    i) As a software developer you are asked to Schedule test delivery plan

3. Referring to the task on point(i), Schedule test delivery plan

4:Present to trainer your Schedule test delivery plan

5. Ask clarification where necessary if any

6. Perform application of learning 2.1.

Key readings 2.1.2.: Schedule test delivery plan

A Test Delivery Plan (TDP) is a comprehensive document outlining the strategies, resources, and timelines for executing software testing activities within a project. It serves as a roadmap for the testing team, ensuring that testing efforts are well-planned, coordinated, and aligned with the overall project goals.

Steps to Create a Test Delivery Plan

1. **Define Testing Objectives:** Clearly articulate the goals of testing, such as identifying defects, assessing quality, and ensuring compliance with requirements.
2. **Identify Testing Scope:** Determine the specific components, features, and functionalities to be tested.
    - Consider the testing levels (unit, integration, system, acceptance) that will be performed.
3. **Develop Test Strategy:** Outline the overall approach to testing, including the testing methodologies, techniques, and tools to be used.
    - Define the roles and responsibilities of the testing team members.
4. **Create Test Plan:** Develop a detailed plan that outlines the test cases, test data, test environment, and test schedule.
    - Consider the dependencies on other project activities, such as development and deployment.
5. **Allocate Resources:** Identify the necessary resources, including personnel, hardware, software, and infrastructure.
    - Assign tasks and responsibilities to team members.
6. **Define Test Environment:** Specify the hardware, software, and network configurations required for testing.
    - Ensure that the test environment is representative of the production environment.
7. **Develop Test Data:** Create or acquire appropriate test data to cover various scenarios and test cases.
    - Consider data privacy and security requirements.
8. **Establish Test Metrics:** Define the key performance indicators (KPIs) to measure the effectiveness of testing.
    - Examples include defect density, test coverage, and test execution time.

9. **Schedule Test Activities:** Create a timeline for test execution, including test case design, test data preparation, test environment setup, test execution, and defect resolution.
    - Consider dependencies on other project activities and potential risks.
10. **Risk Management:** Identify potential risks that could impact the testing process.

- Develop mitigation strategies to address these risks.

11. **Communication Plan:** Define the communication channels and frequency for sharing information with stakeholders.

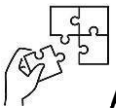
- Ensure that all team members are kept informed of the testing progress and any issues that arise.

Points to Remember

- Test plan: Is a formal document that outlines the overall strategy, objectives, resources, and scope of testing activities for a computer system project.
- There is Types of test plan which are**:** Master Test Plan , Test Phase Plan, Specific Test Plans
- There are the Components of test plan which are Scope, Schedule, Resource Allocation, Environment, Tools, Defect Management, Risk Management
- There are software testing tools used in software development which are: Test Management tools, Automated Testing tools, Performance Testing tools, Cross-browser Testing tools, Integration Testing tools ,Unit Testing tools ,Mobile Testing tools ,GUI Testing tools ,Bug Tracking tools ,Security Testing tools and hardware testing tools.
- In Test plan there are software testing tools used in software development which are**:** Unit Testing, Integration Testing, Functional Testing, System Testing, Acceptance Testing, Performance Testing, Regression Testing, Security Testing, Usability Testing, Smoke Testing, Exploratory Testing, Static Testing, Dynamic Testing
- In test plan there are the Benefits of test data which are: Offers the ability to identify coding errors, Provide a foundation for additional testing, identifies redundancy or unnecessary duplication, Provides flexibility in managing applications.
- In description of Test Execution Scripts, There are Key Components which are :Test Case ID, Test Case Description, Test Steps, Expected Results, Actual Results, Status and Comments.

- **While scheduling test plan, follow these Steps to Create a Test Delivery Plan which are:**

    1. Define Testing Objectives
    2. Identify Testing Scope
    3. Develop Test Strategy
    4. Create Test Plan
    5. Allocate Resources
    6. Define Test Environment
    7. Develop Test Data
    8. Establish Test Metrics
    9. Schedule Test Activities
    10. Risk Management
    11. Communication Plan



Application of learning 2.1.

XYZ Company is a software development company that specializes in electronics and tech gadgets. With a rapidly growing user base and an expanding product catalog, it decided to develop a new feature for its application by prepare the test plan to ensure that the feature meets business requirements and user expectations while maintaining the overall integrity of the website. As software developer, help them to Prepare the test plan.

Indicative content 2.2: Preparation of Testing Environment

Duration: 4 hrs

Theoretical Activity 2.2.1: Description of testing environment

Tasks:

1.you are requested to answer the following questions related to the Description of Testing Environment:

i. What are Key Components of a Testing Environment?
ii. Explain the hardware infrastructure used in testing environment?
iii. Explain are software infrastructure used in testing environment?
iv. What are the dependencies need for testing?
v. Explain computer system testing techniques used in testing environment.
vi. What are computer system testing tools used in testing environment?
vii. What is computer system functionalities?
viii. Match the following computer system Test Techniques with their corresponding system functionalities.

Functionality Type	**Recommended Test Technique**
User Interface	Automated Testing (Regression)
Core Business Logic	Manual Testing / Exploratory Testing
New or Unstable Features	Automated Testing (Performance Testing)
Performance Critical Paths	Exploratory Testing
Cross-Platform Compatibility	Manual testing
Ad-hoc Changes	Parallel testing

2. Provide the answers on papers/flipcharts.

3. Present your findings to your classmates and trainer

4. For more clarification, read the key readings 2.2.1. In addition, ask questions where necessary.

Key readings 2.2.1. Description of Testing Environment

Identification of Key Components of a Testing Environment

Creating a robust testing environment involves several key components which are:

Hardware:

Physical servers or virtual machines that replicate the production environment.

Network infrastructure to simulate real-world usage.

Software:

Operating systems that match production configurations.

Necessary application software, middleware, and databases.

Testing tools and frameworks (e.g., Selenium, JUnit).

Configuration Management:

Version control systems to manage code changes.

Configuration files that specify environment settings.

Test Data:

Realistic datasets that mimic production data while ensuring privacy and security.

Data generation tools or scripts to create test data.

Testing Tools:

Automated testing tools for functional, performance, and security testing.

Defect tracking and management tools (e.g., JIRA, Bugzilla).

Network Setup:

Configuration of firewalls, load balancers, and proxies.

Simulated network conditions (e.g., latency, bandwidth constraints).

Environment Isolation:

Sandboxed environments to prevent interference with production systems.

Clear separation of different testing stages (e.g., unit, integration, system).

Monitoring and Logging:

Tools for monitoring performance and resource usage during tests.

Logging mechanisms to capture application behavior and errors.

Documentation:

Clear documentation of the testing environment setup.

Test plans, cases, and scripts for reference.

Access Control:

User permissions and roles to ensure only authorized personnel can access the environment.

Backup and Recovery:

Systems in place for backing up the environment and restoring it in case of failures.

2. Hardware Infrastructure used in testing environment

Servers:

Application Server: To host the application under test.

Database Server: For managing test databases.

Load Balancers: To distribute testing load evenly across servers.

Workstations:

Testers' Machines: Individual computers for testers to execute tests.

Devices for Mobile Testing: Smartphones and tablets with different OS versions.

Networking Equipment:

Routers and Switches: For creating a reliable network to mimic production environments.

Firewalls: To ensure security and control access.

Backup Solutions:

Backup Servers: For data recovery in case of failure.

Cloud Storage: Optional for redundancy and accessibility.

3. Software Infrastructure used in testing environment

Operating Systems:

Variety of OS: Install multiple versions (e.g., Windows, Linux, macOS) to ensure compatibility.

Testing Tools:

Test Management Tools: JIRA, TestRail for managing test cases and defects.

Automation Frameworks: Selenium, Cypress for automated testing.

Performance Testing Tools: LoadRunner, JMeter for performance assessments.

Databases:

Test Database: Set up an isolated environment with sample data relevant to the application.

Backup Tools: Software to manage database snapshots and recovery.

Application Software:

Version Control: Ensure the correct version of the application is deployed for testing.

APIs: Access to necessary APIs for integration testing.

4. Dependencies Needed for Testing environment

Third-Party Libraries:

Ensure all required libraries and frameworks are installed and compatible with the application.

Configuration Files:

Environment-specific configuration settings that define how the application should run (e.g., database connection strings, API keys).

Test Data:

Anonymized Data: Use data that mimics production but does not expose sensitive information.

Data Migration Tools: Software to populate the test database with necessary data sets.

Access to External Services:

Ensure availability of any external APIs, services, or systems that the application interacts with during testing.

5. Test Techniques used in testing environment

When selecting testing techniques, it's crucial to consider the specific needs of your project and the functionalities of the system. Here's a breakdown of various techniques and how they can be effectively utilized:

a. Manual Testing

Description: Involves human testers executing test cases without automation tools.

Use Cases:

Usability Testing: Assessing user experience and interface design.

Ad-hoc Testing: Informal testing without a formal test plan.

New Feature Testing: When exploring new features that are still evolving.

Advantages:

Flexibility in test execution.

Human insight can catch issues automated tests might miss.

b. Automated Testing

Description: Uses automation tools to execute predefined test cases.

Use Cases:

Regression Testing: Ensuring existing features work after changes.

Performance Testing: Running tests to evaluate how the system performs under load.

Continuous Integration/Continuous Deployment (CI/CD): Automated tests can be integrated into the build process.

Advantages:

Speed and efficiency for repetitive tasks.

Consistency and repeatability in testing.

c. Exploratory Testing

Description: Simultaneously learning about the application while testing it.

Use Cases:

New Applications: When testing an application for the first time.

Dynamic Features: Features that evolve and change frequently.

Risky Areas: High-risk components that may not have comprehensive test cases.

Advantages:

Encourages creativity and intuition.

Can uncover hidden issues not captured by scripted tests.

d. Parallel Testing

Description: Running multiple test scenarios simultaneously across different environments.

Use Cases:

Cross-Platform Testing: Ensuring the application works on various devices and browsers.

Performance Load Testing: Simulating multiple users interacting with the system.

Advantages:

Reduces overall testing time.

Provides faster feedback on system performance.

6. software functionalities

System functionalities in software refer to the specific capabilities and behaviors that the system is designed to perform.

These functionalities define what the system can do and how it interacts with users and other systems. Here are some key aspects of system functionalities:

Input Processing: Accepting and processing user inputs, data from sensors, or other systems.

Data Storage: Storing, retrieving, and managing data in databases or file systems.

Data Manipulation: Performing operations on data, such as calculations, transformations, and filtering.

User Interface: Providing a graphical or textual interface for users to interact with the system (e.g., GUI, CLI).

Communication: Enabling communication between users and the system, or between different systems (e.g., APIs, messaging protocols).

Security: Implementing authentication, authorization, and data encryption to protect sensitive information.

Error Handling: Detecting and responding to errors or exceptions, providing user feedback, and maintaining system stability.

Reporting and Analytics: Generating reports and analytics based on processed data for decision-making.

Integration: Connecting with other systems, software, or services to enhance functionality (e.g., third-party APIs).

Performance Optimization: Ensuring efficient resource usage and fast response times, including load balancing and caching strategies.

Maintenance and Updates: Supporting system updates, patches, and maintenance without significant downtime.

Scalability: Allowing the system to handle increased load or users without performance degradation.

7. Matching Test Techniques with System Functionalities

Hardware system functionalities refer to the specific tasks and capabilities that hardware components provide within a computing environment. These functionalities enable software applications to operate effectively.

Identification: Assess the key functionalities of the system and categorize them based on complexity, risk, and usage frequency.

Functionality Type	**Recommended Test Technique**
Core Business Logic	Automated Testing (Regression)
User Interface	Manual Testing / Exploratory Testing
Performance Critical Paths	Automated Testing (Performance Testing)
New or Unstable Features	Exploratory Testing
Cross-Platform Compatibility	Parallel testing
Ad-hoc Changes	Manual testing

8. Select Testing Tools

Choosing the right testing tools is critical for ensuring effective and efficient testing processes. Here's a breakdown of recommended tools for both hardware and software aspects of testing.

a. Hardware Testing Tools

i. Load Generators

Apache JMeter: An open-source tool for performance testing that can simulate multiple users on various services and applications.

LoadRunner: A comprehensive performance testing tool that allows simulating thousands of users.

ii. Monitoring Tools

Nagios: A powerful monitoring system that checks the health of hardware components and network services.

Zabbix: Monitors various network services and hardware performance metrics in real time.

iii. Benchmarking Tools

PassMark PerformanceTest: Tests and compares the performance of computer hardware.

SiSoftware Sandra: Offers benchmarking and diagnostic capabilities for hardware components.

iv. Network Testing Tools

Wireshark: A network protocol analyser that helps troubleshoot network issues and analyse traffic.

iperf: A tool for measuring bandwidth and performance of network connections.

v. Compatibility Testing Tools

Browser Stack: Provides a cloud-based platform to test hardware configurations and compatibility across different browsers and devices.

b. Software Testing Tools

i. Test Management Tools

JIRA: Primarily for issue tracking but can also be used for managing test cases and tracking bugs.

TestRail: A dedicated test management tool for organizing test cases, plans, and results.

ii. Automated Testing Tools

Selenium: An open-source framework for automating web applications for testing purposes.

Cypress: A modern testing framework for front-end applications, particularly suitable for end-to-end testing.

iii. Performance Testing Tools

Gatling: An open-source load testing tool designed for ease of use and performance metrics.

LoadNinja: A cloud-based load testing tool that allows for real browser load testing.

iv. Security Testing Tools

OWASP ZAP: An open-source security scanner for web applications.

Burp Suite: A comprehensive suite for web application security testing.

v. API Testing Tools

Postman: A popular tool for testing APIs and creating automated test scripts.

SoapUI: A tool for testing SOAP and REST web services.

vi. Continuous Integration/Continuous Deployment (CI/CD) Tools

Jenkins: An open-source automation server for building and deploying software.

GitLab CI: Integrated CI/CD tool within GitLab for automated testing and deployment.

Practical Activity 2.2.2: Selecting software testing tools

Task:

Perform the following activity:

1. Read the key reading 2.2.2.:

2. Referring to the theoretical activity 2.2.2., you are requested to do the task below.

i) As software developer you asked to select software testing tools

3. Referring to the task on point(i), select software testing tools

4.Present to the trainer your software testing tools which are selected.

5. Ask clarification where necessary if any

6. Perform application of learning 2.2

Key readings 2.2.2: Selecting software testing tools

Steps of Selecting Hardware Testing Tools:

Step1: Identify Hardware Components: Determine the specific hardware components that need to be tested.
Step2: Define Testing Objectives: Clearly outline the goals of the hardware testing.
Step3: Research and Evaluate Tools**:** Identify potential tools, compare features, and consider tool integration.
Step4: Conduct Proof of Concept: Test tool functionality, assess accuracy, and evaluate team expertise.
Step5: Make a Final Selection: Consider all factors and choose the most suitable tool.

Steps of Selecting Software Testing Tools:
Step1: Define Testing Requirements: Identify the specific testing needs (e.g., functional, performance, security).
Step2: Identify Tool Categories Needed: Determine which types of tools are required (e.g.,
test management, automation, performance, security).
Step3: Evaluate Tool Features and Capabilities: Assess the tool's core features: ease of use, automation support, reporting, and analytics.
Step4: Check Compatibility: Verify the tool's compatibility with different platforms, environments, and devices.
Step5: Assess Learning Curve and Training Needs: Evaluate how easy it is for your team to learn and adopt the tool.
Step6: Review Vendor and Community Support: Assess the quality of vendor support, including response time and support channels.
Step7: Conduct a Cost-Benefit Analysis: Analyze the total cost of ownership, including licensing, maintenance, and training costs.
Step8: Perform Proof of Concept (PoC): Test the tool in a small-scale project or

environment to validate its performance.

Step9: Gather Feedback from Stakeholders: Collect input from testers, developers, and QA managers to understand usability and effectiveness.

Practical Activity 2.2.3. Set up and configure testing environment

Task:

Perform the following activity:

1. Read the key reading 2.2.3.:
2. Referring to the theoretical activity 2.2.3, you are requested to do the task below:

   i)As a full stack developer you asked to Set up and configure testing environment
3. Referring to the task on point(i), Set up and configure testing environment.
4. Present to trainer your testing environment which are configured.
5. Ask clarification where necessary if any
6. Perform application of learning 2.2.

Key readings 2.2.3: Set Up and Configure a Testing Environment

A well-configured testing environment is crucial for effective system testing. It should replicate the production environment as closely as possible to ensure accurate results.

Steps to Set Up and Configure a Testing Environment

1.Identify the requirements

The first step is to identify the requirements for the testing environment, such as the hardware, software, network, data, and security specifications. You should also consider the type and scope of testing you will perform, such as functional, performance, integration, or regression testing.

2.Create the test plan

The second step is to create a test plan that defines the objectives, scope, schedule, resources, and procedures for testing. The test plan should also include the criteria for success, the expected results, the risks, and the contingency plans. The test plan

should be aligned with the requirements and the project goals, and should be reviewed and approved by the stakeholders.

3.Configure the test environment

The third step is to configure the test environment according to the requirements and the test plan. This involves installing and setting up the hardware, software, network, data, and security components, as well as creating and managing the user accounts, roles, and permissions.

4.Validate the test environment

The fourth step is to validate the test environment by performing some basic tests to ensure that it is functioning properly and meets the expectations. You should check that the test environment is compatible with the production environment, that it has the correct data and configuration, and that it can support the test scenarios and cases.

5.Execute the test cases

The fifth step is to execute the test cases according to the test plan and the test schedule. You should follow the test procedures and protocols, and record the test results and outcomes. You should also monitor the test environment and report any errors, bugs, or defects that you encounter.

6.Maintain the test environment

The sixth and final step is to maintain the test environment by keeping it updated, stable, and secure. You should perform regular backups, restores, and audits of the test environment, and apply any patches, fixes, or enhancements that are necessary. You should also review and evaluate the test environment periodically, and make any changes or improvements that are required.

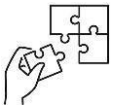
Points to Remember

Description of testing environment

- In description of testing environment, we can identify key components used in testing software environment which are: Hardware Infrastructure, Software Infrastructure
- In software development, there is Dependencies which are Libraries, APIs, Configuration Files
- When you Select Test Techniques it will depends on the nature of the system, project requirements, and available resources such as: Manual Testing, Automated Testing, Exploratory Testing, Parallel Testing

- **Steps of Selecting Hardware Testing Tools are:**
    1. Identify Hardware Components
    2. Define Testing Objectives
    3. Research and Evaluate tools
    4. Conduct Proof of Concept
    5. Make a Final Selection
- **Steps of Selecting Software Testing Tools which are:**
    1. Define Testing Requirements
    2. Identify Tool Categories Needed
    3. Evaluate Tool Features and Capabilities
    4. Check Compatibility
    5. Assess Learning Curve and Training Needs
    6. Review Vendor and Community Support
    7. Conduct a Cost-Benefit Perform Proof of Concept (PoC)
    8. Gather Feedback from Stakeholders

- **The Steps to Set Up and Configure a Testing Environment which are:**
    1. Identify the requirements
    2. Create the test plan
    3. Configure the test environment
    4. Validate the test environment
    5. Execute the test cases
    6. maintain the test environment.



Application of learning 2.2.

XYZ Company is software company that specializes in electronics and tech gadgets. With a rapidly growing user base and an expanding product catalog, it decided to develop a new feature for its website by Preparing the Testing Environment. As software developer, help them to Prepare Testing Environment.

Indicative content 2.3: Perform Testing

Duration: 4 hrs

Theoretical Activity 2.3.1: Description of testing

Tasks:

1. As Trainee you are requested to answer the following questions related to the Description of Testing:

i. Describe the review test plan.
ii. What are key aspects to select test cases?
iii. Describe test execution.
iv. Explain Record defects
v. What is the Purpose of Recording Defects?
vi. Outline Key Steps to Record Defects.
vii. What are Best Practices for Recording Defects?
viii. What is Monitoring test progress?
ix. What is the Purpose of Monitoring test progress?
x. Mention Key components of Monitoring test progress.
xi. What are Best Practices for Recording Defects?
xii. Explain document test result?

2. Provide the answers on papers/flipcharts.

3. Present your findings to your classmates and trainer

4. For more clarification, read the key readings 2.3.1. In addition, ask questions where necessary.

Key readings 2.3.1. Description of Testing

ɾiew test plan

The review of the test plan involves a thorough examination of the document that outlines the strategy, scope, resources, and schedule for testing a system application. Key aspects include:

1. Objectives: Ensure the testing goals are clear and aligned with project objectives.

2. Scope: Confirm what is included in testing and what is excluded to avoid misunderstandings.
3. Test Environment: Verify that the environment accurately reflects production settings.
4. Resources: Check that the necessary tools, personnel, and technologies are available.
5. Schedule: Review timelines for realism and clarity, including milestones.
6. Risk Assessment: Examine identified risks and their mitigation strategies.
7. Test Design Techniques: Ensure appropriate methodologies are documented for creating test cases.
8. Exit Criteria**:** Confirm that criteria for concluding testing are clearly defined.

2. Selection of test cases

cases involves identifying the most relevant scenarios to validate the system. The process includes:

1. Understand Requirements: Review functional and non-functional requirements to ensure comprehensive coverage.
2. Prioritize Based on Risk: Focus on high-risk areas that could significantly impact users or business operations.
3. Use Test Design Techniques: Apply methods like equivalence partitioning, boundary value analysis, and decision tables to identify effective test cases.
4. Consider Compliance Needs: Include test cases that address any legal or regulatory requirements.
5. Incorporate Negative Test Cases: Ensure robustness by testing how the application handles invalid inputs or unexpected scenarios.
6. Review Existing Test Cases: Adapt and reuse test cases from previous projects or releases.
7. Engage Stakeholders: Collaborate with team members to gather insights on critical functionalities.
8. Limit Scope: Select a manageable number of test cases that maximize coverage without overwhelming the testing process.

3.Test Execution

Test execution is the phase where selected test cases are run to validate the software system.

Here's a detailed breakdown of the process of making test execution:

1. Prepare the Test Environment: Ensure that the testing environment is correctly configured to mimic production conditions.
2. Review Test Cases: Familiarize yourself with the selected test cases, expected results, and prerequisites.
3. Set Up Test Data: Prepare the necessary data for various testing scenarios, including edge cases.
4. Execute Test Cases: Run the tests according to the defined procedures, documenting the outcomes.
5. Record Results: Document whether each test case passed or failed, capturing discrepancies for failed tests.
6. Log Defects: For any failures, log defects in a tracking system, providing detailed information for resolution.
7. Monitor Progress: Track the execution status and defect resolution to keep stakeholders informed.
8. Retesting and Regression Testing: Once defects are resolved, retest failed cases and conduct regression tests to ensure overall system stability.
9. Document and Review Results: Compile a summary of execution results and discuss findings with the team to improve future testing efforts.

4.Record Defects

Recording defects is a critical part of the software testing process. It involves documenting any issues or discrepancies found during testing to facilitate their resolution. Here's a detailed explanation of the steps and considerations involved in this process:

5.Purpose of Recording Defects

- Communication**:** To clearly inform the development team about issues that need to be addressed.
- Traceability**:** To maintain a history of defects, which helps in tracking their resolution and impact on the project.
- Quality Improvement**:** To analyze defect trends and root causes for future improvement in software quality.

6.Key Steps to Record Defects

1. Identify the Defect
   - After executing a test case, if the actual result does not match the expected result, identify the issue as a defect.

2. Use a Defect Tracking System

- Utilize a defect management tool (e.g., JIRA, Bugzilla, or Trello) to log the defect. These tools facilitate organization and tracking.

3. Document Essential Information

   For each defect, include the following details:

   - **Title/Summary**: A brief, descriptive title that summarizes the defect.
   - **Description**: Detailed information about the defect, including:
     - Steps to reproduce the issue (specific actions taken that led to the defect).
     - Expected result (what should have happened).
     - Actual result (what actually happened).
   - **Severity and Priority**:
     - **Severity**: How severe the defect is (e.g., critical, major, minor).
     - **Priority**: How quickly the defect needs to be fixed (e.g., high, medium, low).
   - **Environment Details**: Specify the environment where the defect was found (e.g., OS, browser version, device type).
   - **Attachments**: Include screenshots, logs, or any other relevant files that help in understanding the defect.

4. **Assign the Defect**
   - Assign the defect to the appropriate developer or team responsible for fixing it. This may also include tagging it with relevant labels (e.g., module, component).
5. **Status Tracking**
   - Track the status of the defect as it progresses through the resolution workflow. Common statuses include:
     - New
     - Assigned
     - In Progress
     - Resolved
     - Reopened
     - Closed
6. **Follow-Up**
   - Regularly follow up on the status of reported defects, especially high-severity issues, to ensure timely resolution.
7. **Retesting**
   - Once a defect is marked as resolved, retest the affected functionality to confirm that the fix is effective and that no new issues have been introduced.

7.Best Practices for Recording Defects

- Clarity**:** Be clear and concise in your documentation to avoid misunderstandings.
- Consistency**:** Follow a consistent format and procedure for logging defects.
- Collaboration**:** Work closely with developers and other stakeholders to ensure defects are understood and prioritized appropriately.
- **Analysis**: Regularly analyse defect data to identify patterns, which can inform future development and testing practices.

8. Monitor Test Progress

Monitoring test progress is a crucial aspect of the testing process that helps ensure that testing activities are on track, issues are identified early, and stakeholders are kept informed. Here's a detailed overview of what it involves, its purpose, and key practices:

Purpose of Monitoring Test Progress

1. **Track Execution**: To assess the status of test case execution and ensure that planned activities are being carried out as scheduled.
2. **Identify Issues Early**: To detect any blockers, delays, or unexpected challenges in the testing process.
3. **Measure Quality**: To gather metrics on defect discovery, test coverage, and overall software quality.
4. **Facilitate Communication**: To provide timely updates to stakeholders, ensuring transparency and enabling informed decision-making.

Key Components of Monitoring Test Progress

1. **Establish Metrics**
    - Define key performance indicators (KPIs) to measure testing progress, such as:
        - Test case execution rates (e.g., how many test cases have passed, failed, or are yet to be executed).
        - Defect counts (e.g., total defects found, defects resolved, and defects reopened).
        - Test coverage (e.g., percentage of requirements tested).
        - Test cycle time (e.g., time taken to complete a test phase).
2. **Use Dashboards and Reports**
    - Utilize dashboards in test management tools (e.g., JIRA, TestRail) to visualize real-time progress.

- Generate regular reports summarizing testing activities, highlighting:
  - Execution status (e.g., completed, in progress).
  - Defect status (e.g., severity levels, resolution progress).
  - Overall testing trends (e.g., comparison to previous cycles).

3. **Regular Stand-up Meetings**
   - Conduct daily or weekly stand-up meetings to discuss:
     - Current progress and any blockers.
     - Next steps and priorities.
     - Team collaboration and support needed.
   - Encourage open communication to quickly address issues.
4. **Review Test Execution**
   - Continuously monitor the execution of test cases:
     - Check for any deviations from the test plan (e.g., delays, unplanned changes).
     - Identify any test cases that consistently fail, indicating potential areas of concern in the application.
5. **Adjust Testing Strategies**
   - Based on monitoring data, be prepared to adjust testing strategies if necessary:
     - Reallocate resources to critical areas or increase focus on high-risk features.
     - Prioritize retesting of critical defects that could impact project timelines.
6. **Communicate with Stakeholders**
   - Provide regular updates to stakeholders, including:
     - Progress reports on testing activities and timelines.
     - Key findings, including critical defects or risks.
     - Recommendations for decision-making (e.g., whether to proceed to production).

7. **Maintain Documentation**
   - Keep thorough records of all monitoring activities, including test execution logs, defect reports, and meeting notes.
   - Document any changes made to the test plan or strategy based on monitoring insights.

Best Practices for Monitoring Test Progress

- **Be Proactive**: Actively seek out potential issues rather than waiting for them to escalate.

- **Engage the Team**: Encourage all team members to contribute to monitoring efforts and share insights.
- **Focus on Quality**: Look beyond just metrics; ensure that the quality of testing is maintained and that the software meets user needs.
- **Stay Agile**: Be flexible and ready to adapt your monitoring strategies as the project evolves.

6. Document Test Results

- **Objective**: Maintain clear and comprehensive documentation of test outcomes.
- **Key Actions**:
    - Summarize the results of test execution, including pass/fail rates and defect counts.
    - Compile a report detailing findings, including critical issues and recommendations.
    - Ensure documentation is accessible for future reference and audits.
- **Outcome**: A thorough record of testing activities that informs stakeholders and supports continuous improvement.

Practical Activity 2.3.2.: Selection test cases and test execution

Task:

Perform the following activity:

1. Read the key reading 2.3.2.

2. Referring to the theoretical activity 2.3.2, you are requested to do the task below:

    i)As software developer you are asked to Select test cases and test execution.

3. Referring to the task on point(i), Select test cases and test execution.

4.Present to the trainer your selected test cases and test execution

5. Ask clarification where necessary if any

6. Perform application of learning 2.3

Key readings 2.3.2: Selection test cases and test execution

Select test cases

1.Selecting test cases involves identifying specific scenarios that will be executed during testing to validate that the system meets its requirements. Choose the specific test cases that will be executed based on the test plan. Focus on critical functionalities, edge cases, and high-risk areas of the system.

The steps for selecting test cases in system testing are as follows:

1. Define the test objectives: Identify the goals of the system testing and determine what aspects of the system need to be tested.
2. Identify test scenarios: Determine the different scenarios that need to be tested, such as different user inputs, system responses, and edge cases.
3. Create test cases: Develop detailed test cases for each scenario, including the steps to be taken, the expected results, and any prerequisites or assumptions.
4. Prioritize test cases: Determine the order in which the test cases should be executed, based on their importance and the system requirements.
5. Execute test cases: Run the test cases and record the results, noting any defects or issues that arise.
6. Analyze results: Review the results of the test cases and determine if the system meets the requirements and functions as expected.
7. Report defects: Document any defects or issues found during testing and work with the development team to address them.
8. Repeat testing: If necessary, repeat the testing process to ensure that the defects have been addressed and that the system is fully functional.

2.Test execution

Test execution is the phase where selected test cases are run to validate the software system. Key steps include:

1. Prepare the Test Environment: Ensure that the testing environment is correctly configured to mimic production conditions.
2. Review Test Cases: Familiarize yourself with the selected test cases, expected results, and prerequisites.

3. Set Up Test Data: Prepare the necessary data for various testing scenarios, including edge cases.
4. Execute Test Cases: Run the tests according to the defined procedures, documenting the outcomes.
5. Record Results: Document whether each test case passed or failed, capturing discrepancies for failed tests.
6. Log Defects: For any failures, log defects in a tracking system, providing detailed information for resolution.
7. Monitor Progress: Track the execution status and defect resolution to keep stakeholders informed.
8. Retesting and Regression Testing: Once defects are resolved, retest failed cases and conduct regression tests to ensure overall system stability.
9. Document and Review Results: Compile a summary of execution results and discuss findings with the team to improve future testing efforts.

Points to Remember

In software testing there is:

Review Test Plan is a process where a document outlining the testing strategy, objectives, scope, and procedures for a software project is critically examined.

- Process test refers to executing the selected test cases in accordance with the defined test plan.
- Recording defects is an essential part of system testing where any issues or bugs identified during test execution are documented systematically.
- Monitoring test progress involves tracking the execution of tests against the planned schedule and assessing overall quality metrics throughout the testing phase.
- Documenting test results in system testing is a critical process that ensures the quality, reliability, and usability of software products.
- There are Key Components of Test Result Documentation that are: Test Case Identification, Test Execution Summary, Detailed Test Results, Defect Summary, Test Coverage Information, Conclusion and Recommendations.

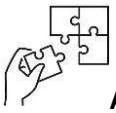
- Selecting test cases identifies specific scenarios that will be executed during testing to validate that the system meets its requirements.
- **There are steps for selecting test cases in software testing which are following:**

1. Define the test objectives
2. Identify test scenarios
3. Create test cases
4. Prioritize test cases
5. Execute test cases
6. Analyze results
7. Report defects
8. Repeat testing

- Test execution is the phase where selected test cases are run to validate the software system.

The steps for Test execution in system software testing are as follows:

1. Prepare the Test Environment
2. Review Test Cases
3. Set Up Test Data
4. Execute Test Cases
5. Record Results
6. Log Defects
7. Monitor Progress
8. Retesting and Regression Testing
9. Document and Review Results



Application of learning 2.3.

XYZ Company is a software company specializing in electronics and tech gadgets. The company is rolling out a new search feature aimed at improving user experience by providing more accurate, faster search results but they have a problem of ensuring if the feature meets the required quality standards. They want someone to perform testing by testing process involves:

1. Reviewing the test plan,
2. Selecting appropriate test cases,
3. Processing the tests,
4. Recording defects,
5. Monitoring progress,
6. Documenting the results.

As software system developer, you are hired to perform this task.

Learning outcome 2 end assessment

Written assessment

Q1. What is software testing?

Q2. Match the following terms in column A with their corresponding descriptions column B. Write the letter of the correct answer in the provided blank space in Column C.

Column A	Column B	Column C
1. Description of test plan	A. The process of identifying and selecting appropriate test tools	……
2. Identify test tools	B. A document outlining the scope, objectives, and approach for testing a software product	……
3. Identify test techniques	C. The process of creating specific test cases to execute	……
4. Identify test cases	D. The process of defining the criteria that will determine if a test is successful or failed	……
5. Identify test criteria	E. The process of defining the specific methods and procedures that will be used to conduct testing	……
6. Allocate resources	F. The process of assigning personnel, equipment, and other resources needed for testing	……
7. Prepare Test Data	G. The process of creating a timeline for completing the testing process	……
8. Prepare Test Execution Scripts	H. The process of creating detailed instructions for executing test cases	……
9. Schedule test delivery plan	I. The process of creating or obtaining data that will be used during testing	……

Q3. Circle the letter corresponding with the correct answer

1. Which of the following test techniques is most suitable for regression testing in a large-scale software system?

a) Manual Testing

b) Automated Testing
c) Exploratory Testing
d) Parallel Testing

2. What is the main advantage of using automated testing over manual testing?

a) Automated tests can explore new functionalities not previously planned.
b) Automated tests are cheaper to write than manual tests.
c) Automated tests can be run repeatedly with less effort and in less time.
d) Automated tests require no maintenance after creation.

3. In which scenario would you typically choose parallel testing?

a) To verify that new software versions perform consistently with the old version.
b) To test a system under various conditions using random inputs.
c) To perform security and load testing for critical applications.
d) To verify that the system responds quickly under heavy loads.

Q4. Read the Following statement and answer by true if correct or false otherwise

a) Preparation of a testing environment involves setting up both hardware and software infrastructure.

b)Hardware infrastructure is not essential for testing as most testing is done on virtual environments.

c)Proper preparation of a testing environment helps prevent configuration-related issues during testing.

d) Hardware and software requirements in a testing environment should always match those in the production environment.

e) Testing environments should be isolated from development environments to avoid interference.

Q5. By using a table, differentiate Software testing tools and Hardware testing tools in software development testing.

Practical assessment

A healthcare technology company is developing a hybrid system architecture for a patient management application that integrates both hardware (medical devices) and software (application interface). To ensure this software system meets the required quality standards,

a comprehensive test plan is prepared to outline the testing strategies, tools, techniques, and schedules. As a software developer, you are tasked to :

1. Prepare the test plan,
2. Setup of the testing environment,
3. Execute testing.

END

Reference

Enov8. (2024, September 16). Retrieved from www.enov8.com: https://www.enov8.com

Lambda. (2024). *COMPUTER SYSTEM QUALITY ASSURANCE.* Lambda TEST.

Lambdatest. (2024, June 13). *What Is Test Plan? Its Components, Types, and Example*. Retrieved from www.lambadatest.com: https://www.lambdatest.com/learning-hub/test-plan

Singh, R. (2024, July 31). *top software testing tools 2024*. Retrieved from www.headspin.io/: https://www.headspin.io/blog/top-software-testing-tools

Testing, G. A. (2024, March n.d). *www.globalapptesting.com/*. Retrieved from globalapptesting: https://www.globalapptesting.com/


TEST TEST DOCUMENTATION
TEST PLANS
TEST REPORTS

Indicative contents

3.1 Consolidation of test results
3.2 Providing User Acceptance Testing Report
3.3 Generating recommendation Report

Key Competencies for Learning Outcome 3: Generate Test Documentation

Knowledge	Skills	Attitudes
• Description of the Test results • Description of User Acceptance Testing Report • Identification of usability testing outcomes • Definition of Security vulnerability findings • Analysing overall test results • Definition of recommendation report • Discussion of Benefits and Impact of recommendation report	• Analysing and evaluating test results • Creating test summary report • Measuring performance testing (speed, stability) • Generating UAT report • Generating recommendation Report • Documenting and archiving test artifacts	• Having Self Confidence in test Documentation • Being Good observer in test documentation • Being Creative • Having Critical thinking in test documentation • Being Problem solver in test documentation • Being Continuous learner in test documentation • Being Innovative in test documentation • Being Teamwork in test documentation • Speak with Relevance in test documentation

Duration:10 hrs

Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Consolidate properly test results as used in quality assurance

2. Provide clearly the User Acceptance Testing Report as used in quality assurance

3.Generate properly the recommendation Report as used in quality assurance

Resources

Equipment	Tools	Materials
• Computer	• Version control systems: Git, Subversion • Screenshot/recording tools: Snagit, Screencastify	• None

Indicative content 3.1: Consolidation of Test Results

Duration: 3 hrs

Theoretical Activity 3.1.1: Description of the test results

Tasks:

1: Answer the following question:

i. Describe test data collection in software development?
ii. Talk about Test Artifacts in software development
iii. Describe test execution data
iv. Explain Test Logs in test results
v. Explain Test Findings in test results
vi. What is Defects in test results?
vii. Define Test Metrics in test results

2: Participate in group formulation

3: Present the findings to your class or colleagues

4: Ask the clarification if any

5: Read the trainee manual key reading on 3.1.1

Key readings 3.1.1: Description of the test results

1.**Test data collection**

i. Collecting Test Data

Collecting test data involves gathering the inputs, conditions, and environmental factors that will be used during testing. This data can include:

- **Input Parameters**: Values fed into the system to simulate various conditions.
- **System State Information**: Current configurations, settings, and statuses of the software system being tested.
- **External Conditions**: Factors such as temperature, voltage, or signal integrity that may affect system performance.

Effective data collection is crucial for creating realistic test scenarios that can uncover potential issues in the system.

ii. Collecting Test Artifacts

Test artifacts are the documents and materials produced during the testing process, which may include:

- **Test Plans**: Documents outlining the scope, approach, resources, and schedule of intended test activities.
- **Test Cases**: Detailed descriptions of specific tests to be conducted, including input data, execution steps, and expected outcomes.
- **Test Scripts**: Automated scripts created for executing tests in a consistent and repeatable manner.
- **Test Logs**: Records of test execution, including pass/fail results, error messages, and execution timestamps.

Collecting these artifacts helps ensure traceability, supports future testing efforts, and provides documentation for compliance and quality assurance purposes.

2. Compiling Test Execution Data

Compiling test execution data involves gathering and organizing the results of test executions. This can include:

- **Execution Results**: Outcomes of test cases, indicating whether they passed or failed.
- **Performance Metrics**: Data such as execution time, resource usage, and response times.
- **Error Logs**: Detailed information about any failures or issues encountered during testing, including stack traces or fault conditions.
- **Comparative Analysis**: Summarizing results against expected outcomes, historical data, or benchmarks.

3.Test execution data

- Test execution data refers to the information generated and collected during the execution of test cases in the software testing process.

4. Test logs are detailed records generated during the execution of test cases in the software testing process. They document the step-by-step actions taken, the results of those actions, and any issues or anomalies encountered during testing. Test logs provide a comprehensive trace of what was tested, how it was tested, and the outcomes of those tests.

5. Test findings refer to the results and observations derived from executing test cases during the software testing process.

6. A **defect** in software development refers to any flaw, error, or inconsistency in a software product that causes it to behave unexpectedly or fail to meet the specified requirements.

7. Test metrics are quantitative measures used to assess the effectiveness, efficiency, and overall quality of the software testing process.

- **Practical Activity 3.1.2: Consolidating test results**

Task:

Perform the following activity:

1.Read the key reading 3.1.2.

2.Referring to the theoretical activity 3.1.1, you are requested to do the task below:

i) As a software developer, you asked to consolidate the test result as used in software development.

3.Referring to the task on point(i), make the test result

4. Present to the trainer your test results

5.Ask clarification where necessary if any

6.Perform application of learning 3.1.

Key readings 3.1.2: Consolidating test results

- The steps to consolidate test results are:

1. Collect All Test Results

✓ **Retrieve Test Execution Logs**: Gather the results from all test executions (manual and automated). This includes the status of test cases (pass/fail), error logs, screenshots, performance metrics, and any other test artifacts.

✓ **Include Different Types of Tests**: Consolidate results from unit tests, integration tests, system tests, user acceptance tests (UAT), performance tests, etc.

2. Organize Results by Test Suites

- ✓ **Group by Test Suite/Module**: Organize the results according to the test suites or modules they belong to. This helps in mapping the results to specific features or areas of the system.
- ✓ **Categorize by Test Types**: Within each test suite, categorize results by test type (e.g., functional, non-functional, regression).

3. Summarize the Test Case Status

- ✓ **Classify Test Outcomes**: For each test case, classify it into one of the following categories:
  - **Passed**: Test case executed successfully, and the result matches the expected outcome.
  - **Failed**: Test case failed because the actual result did not meet the expected outcome.
  - **Blocked**: Test case could not be executed due to environmental or dependency issues.
  - **Deferred**: Test case execution was postponed due to changes in scope or priorities.
- ✓ **Use a Summary Table**: Create a table summarizing the total number of test cases and their status for easy understanding:

Test Suite	Total	Passed	Failed	Blocked	Deferred |
Login Function	50	45	2	1	2
Payment System	70	65	3	0	2

4. Analyze Defects

- ✓ **Link Defects to Test Cases**: For failed test cases, document the defects raised. Include a reference to the defect ID from your defect tracking tool (e.g., JIRA, Bugzilla).
- ✓ **Severity and Priority of Defects**: Categorize defects by severity (critical, major, minor) and priority. This helps in determining the most critical areas of failure.
- ✓ **Defect Summary Table**: Provide a summary of the defects found during testing:

Defect ID	Test Case ID	Severity	Status	Description
101	TC-Login-05	Critical	Open	Login fails on retry

5. Include Test Metrics

- ✓ **Test Case Execution Rate**: Calculate the rate of test execution (e.g., total test cases executed vs. total test cases planned).
- ✓ **Pass/Fail Percentage**: Include pass/fail percentages to give an overall snapshot of the test success rate.
    - **Pass Rate (%) = (Number of Passed Test Cases / Total Executed Test Cases) x 100**
    - **Fail Rate (%) = (Number of Failed Test Cases / Total Executed Test Cases) x 100**
- ✓ **Defect Density**: Include defect metrics, such as the number of defects found per test case or per requirement.
    - **Defect Density = (Number of Defects / Number of Test Cases Executed)**

6. Evaluate Test Coverage

- ✓ **Map Tests to Requirements**: Ensure test results are mapped to the system's requirements or user stories to check for test coverage.
- ✓ **Identify Coverage Gaps**: Highlight areas of the application that were not tested or had incomplete testing.
- ✓ **Test Coverage Table**: Summarize the test coverage:

Mathematica

Requirement	**Total Test Cases**	**Passed**	**Failed**	**Untested**
Login	20	18	2	0

7. Summarize Key Findings

- ✓ **Highlight Critical Issues**: Provide a summary of any critical issues that may impact the release or require immediate attention.

- ✓ **Discuss Testing Constraints**: Mention any blockers or environmental issues that impacted the testing process (e.g., lack of test data, unavailable systems).
- ✓ **Risk Assessment**: Discuss the risks associated with unresolved defects or untested areas, and their potential impact on the product.

8. Generate Test Summary Report

- ✓ **Executive Summary**: Provide a high-level overview of the testing effort, key results, and overall product quality.
- ✓ **Test Metrics Summary**: Summarize test execution statistics, defect trends, and other key metrics.
- ✓ **Recommendations**: Offer actionable recommendations based on the test results. These may include further testing, bug fixes, or a decision on whether to release the product.

9. Attach Supporting Artifacts

- ✓ **Attach Logs and Screenshots**: If relevant, include links or attachments of detailed logs, screenshots of failures, performance results, and any other supporting artifacts that provide additional context.
- ✓ **Reference Documentation**: Link to the test plan, test cases, and defect tracking documents for cross-referencing.

10. Review and Distribute

- ✓ **Internal Review**: Before distributing the final test result documentation, review it internally with the QA and development teams to ensure accuracy.
- ✓ **Distribute to Stakeholders**: Once finalized, distribute the consolidated test result documentation to key stakeholders, including project managers, developers, and business owners.

- **Create Test Summary Report:** Compile all findings, metrics, and summaries into a formal Test Summary Report. This report should include:

  - Introduction: Overview of the testing objectives and approach.
  - Test Execution Summary: Results of test execution, including passed, failed, and blocked test cases.
  - Defect Summary: Detailed information about defects identified, their severity, and status.
  - Key Findings: Highlight any critical issues or major successes from testing.
  - Recommendations: Suggested actions for resolving critical defects or improving system performance.

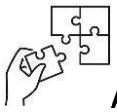
- Conclusion: Final assessment of the system's readiness for release or further testing.

Points to Remember

- Test artifacts refer to the documents, reports, tools, and other deliverables created and maintained during the software testing process.
- Test execution data refers to the information generated and collected during the execution of test cases in the software testing process.
- Test logs are detailed records generated during the execution of test cases in the software testing process.
- Test findings refer to the results and observations derived from executing test cases during the software testing process.
- A defect in software development refers to any flaw, error, or inconsistency in a software product that causes it to behave unexpectedly or fail to meet the specified requirements.
- Test metrics are quantitative measures used to assess the effectiveness, efficiency, and overall quality of the software testing process.
- The steps to consolidate test results are:

    1. Collect All Test Results
    2. Organize Results by Test Suites
    3. Summarize the Test Case Status
    4. Analyze Defects
    5. Include Test Metrics
    6. Evaluate Test Coverage
    7. Summarize Key Findings
    8. Generate Test Summary Report
    9. Attach Supporting Artifacts
    10. Review and Distribute

- **Create Test Summary Report:** Compile all findings, metrics, and summaries into a formal Test Summary Report.



Application of learning 3.1.

XYZ- solutions ltd is a software development company that specialises in building different softwares. However, the developer is facing with challenges of improving the system by applying the quality assurance. As a student of level 5 in software development, you are tasked to consolidate the test result.

Indicative content 3.2: Prepare User Acceptance Testing Report

Duration:

Theoretical Activity 3.2.1: Description of User Acceptance

Tasks:

1.Answer the following question:

i. What is the User Acceptance Testing (UAT)
ii. What are the Characteristics of UAT ?
iii. What are the Objectives of UAT ?
iv. Discuss on the UAT Process
v. What are the Benefits of UAT?
vi. What are the Types of UAT?
vii. What is the Usability testing?
viii. What is the A security vulnerability?

2.Participate in group formulation

3.Present the findings to your class or colleagues

4.Ask the clarification if any

5.Read the trainee manual key reading on 3.2.1

Key readings 3.2.1: Description of User Acceptance Testing (UAT) Report

1.User Acceptance Testing (UAT) is the final phase of the software testing process, where the end-users or business stakeholders validate whether the software meets their requirements and functions as expected in real-world scenarios. UAT is designed to ensure that the software satisfies business needs and is ready for deployment.

2.Characteristics of UAT

1. **End-User Involvement:** UAT is carried out by the end users (clients, customers, or business stakeholders) who will use the software in their day-to-day operations.
    - Users execute test cases to verify that the system works as required from a business perspective.
2. **Real-World Scenarios:** Testing is based on real-life scenarios and workflows that the users will encounter during normal operations.

- It ensures the software behaves as expected in a production-like environment.

3. **Validation Against Requirements:** UAT validates the software against the originally defined business requirements and ensures that all functional and non-functional requirements are met.
4. **Final Check Before Release:** UAT is often the final testing phase before the software is approved for release into production.
    - A successful UAT signifies that the product is ready to be deployed and used by the business.

3.Objectives of UAT

a. **Confirm Business Requirements:** Ensure that the software meets all business requirements and functions according to the specifications.
b. **Verify Usability:** Validate that the software is user-friendly and intuitive, allowing users to perform their tasks efficiently.
c. **Identify Any Gaps:** Detect any missing functionality, defects, or issues that may have been overlooked during earlier testing phases.
d. **Ensure Readiness for Production:** Confirm that the software is stable, secure, and ready to be used in the actual business environment.

4.UAT Process

a. **Title Page**:
    - **Project Name**: Include the title of the project.
    - **Report Title**: Clearly state that this is a "User Acceptance Testing Report."
    - **Prepared by**: Name of the person/team who prepared the report.
    - **Date**: Date when the report was completed.

b. **Introduction**:
    - **Purpose**: Describe the objective of UAT and the purpose of the report.
    - **Scope**: Briefly define what was tested, such as specific modules, features, or workflows.
    - **Test Environment**: Specify the environment where UAT was conducted (e.g., production-like, staging).
    - **Participants**: List the stakeholders and end-users involved in the UAT process.

c. **Acceptance Criteria**: Define the **acceptance criteria** against which the software was evaluated. This includes the requirements, key features, and performance metrics that the software must meet.

d. **Test Plan Summary**: Summarize the UAT **test plan**, including the total number of test cases, test scenarios, and the overall testing strategy.

e. **Test Execution Details**:

- **Test Case Summary**: Provide a high-level summary of the test cases, including the number of test cases executed, passed, failed, and blocked.
- **Pass/Fail Criteria**: Explain the criteria used to determine whether a test case was successful (pass) or unsuccessful (fail).

f. Test Results:

- **Overall Test Results**: Provide a detailed summary of the test case execution results, focusing on key findings and results.
- **Defects Found**: List the defects or issues encountered during UAT, along with their priority (high, medium, low) and status (open, closed, in-progress).

g. **User Feedback**: Include any feedback from users regarding usability, performance, or functionality. Highlight positive feedback as well as pain points identified during testing.

h. **Risk Assessment**: Identify potential risks and issues that could arise based on UAT findings. Assess the likelihood of these risks and provide recommendations to mitigate them.

i. **Recommendations**:

- Based on the UAT findings, provide clear recommendations for improvement or action steps to resolve any critical defects or usability issues.
- State whether the system is ready for deployment or if additional work is required.

j. **Sign-Off Section**:

- Include a sign-off section where key stakeholders, business users, and the project manager can approve the results of the UAT.
- This section formally acknowledges that the software meets the acceptance criteria and is ready for production.

5.Benefits of UAT

a. Business Confidence:

- UAT ensures that the software meets business needs, providing confidence to stakeholders that the product will function as required in the real world.

b. Reduction in Post-Release Defects:

- By identifying issues before deployment, UAT reduces the risk of post-release defects and ensures a smoother go-live process.

c. Improved User Satisfaction:

- Since UAT involves real users, their feedback ensures that the software is aligned with their expectations and is user-friendly.

d. Better Risk Management:

- UAT helps uncover any potential risks, such as system flaws or usability challenges, that could affect business operations if left unchecked.

6.Types of UAT

a. Alpha Testing:

- Conducted internally by the development or testing team with the involvement of business stakeholders. It's typically done in a controlled environment before the product is released to real users.

b. Beta Testing:

- Performed by a select group of actual users in a real-world environment. This helps to gather user feedback and identify any last-minute issues before the software's official release.

7.Usability testing

It is a type of software testing that focuses on evaluating how easy and user-friendly a software application is. It assesses how well end-users can interact with the system, navigate its interface, and complete specific tasks.

The goal of usability testing is to ensure that the software provides a smooth, intuitive, and efficient experience for users.

8.**A security vulnerability** is a weakness or flaw in a system, software, or network that can be exploited by attackers to gain unauthorized access, disrupt operations, or compromise the integrity, confidentiality, or availability of data.

Practical Activity 3.2.2: Prepare the User Acceptance Testing Report

Task:

Perform the following activity:

1. Read the key reading 3.2.2.
2. Referring to the theoretical activity 3.2.1, you are requested to do the task below:
   i) As software developer, you asked to prepare User Acceptance Testing Report
3. Referring to the task on point(i), make User Acceptance Testing Report
4. Present to trainer your User Acceptance Testing report.

5. Ask clarification where necessary if any
6. Perform application of learning 3.2.

Key readings 3.2.2: Providing the User Acceptance Testing Report

The steps to generate a comprehensive UAT report:

1. Define the Purpose

- **Purpose of the Report**: Start with a clear statement about the goal of the UAT report. This typically includes documenting UAT results and evaluating whether the system meets business requirements and is ready for deployment.
- **Scope of UAT**: Specify what was tested during UAT, such as specific features, modules, or processes. Clearly outline the boundaries of the testing.

2. Include UAT Test Overview

- **Objectives**: Highlight the main objectives of UAT. These usually include verifying that the system meets business needs, confirming that key workflows function as expected, and validating that the system is ready for release.
- **Key Areas Tested**: List the critical business areas or functionalities that were the focus of UAT.

3. Identify Test Participants

- **List of Participants**: Provide details of UAT testers and their roles (e.g., business users, product owners, subject matter experts).
- **Stakeholders**: Mention key stakeholders who are involved in the UAT process and the decision-making around go/no-go recommendations.

4. Detail the UAT Environment

- **Test Environment Details**: Provide information on the environment where UAT took place, such as hardware configurations, software versions, databases, and network settings.
- **Test Data**: Indicate whether real or simulated data was used for testing and any constraints encountered due to data availability or quality.

5. Summarize Test Scenarios and Cases

- **Test Scenario Summary**: Create a summary table of test cases or scenarios executed during UAT. Include the following:
    - Test Case/Scenario ID

- Scenario description
- Expected result
- Actual result
- Status (Pass/Fail/Blocked)

Example Table:

Test Case ID	Scenario Description	Expected Result	Actual Result	Status
UAT-001	User login via web portal	User successfully logs in	User logged in correctly	Pass
UAT-002	Add item to shopping cart	Item added to cart	Item added to cart	Pass
UAT-003	Apply discount during checkout	Discount applied successfully	Discount not applied	Fail

6. Defect Summary

- **List of Identified Defects**: Document any defects that were identified during UAT. Include details such as:
  - **Defect ID**
  - **Description of the issue**
  - **Severity level** (Critical, Major, Minor)
  - **Status** (Open, Closed, Deferred)
  - **Comments** or resolution details.

Example Defect Table:

Defect ID	Test Case ID	Description	Severity	Status	Comments
DEF-001	UAT-003	Discount not applied at checkout	Major	Open	Needs fix before release

7. Test Metrics

- **Test Execution Summary**: Provide key metrics such as:
    - Total number of test cases
    - Number of test cases passed
    - Number of test cases failed
    - Blocked test cases due to external factors

Example Metrics Table:

Total Test Cases	Passed	Failed	Blocked	Pass Rate (%)	Fail Rate (%)
50	45	3	2	90%	6%

8. Evaluate Risks and Impact

- **Risk Assessment**: Identify any risks associated with unresolved defects or areas that weren't fully tested. Discuss potential impacts on the system if the identified defects are not resolved.
- **Impact of Test Failures**: Highlight the severity of any failed test cases and how they might affect the business or the end-user experience.

9. Recommendations

- **Go/No-Go Decision**: Based on the test results, make a recommendation on whether the system is ready for production. If there are critical blockers, provide reasons for delaying the release.
- **Further Actions**: Suggest additional testing, bug fixes, or changes that need to be completed before a final go-live decision can be made.

10. Provide Conclusion

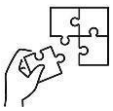
- **Summary of Findings**: Offer a high-level summary of the UAT results, emphasizing the overall quality of the system and any major findings.
- **Final Sign-Off**: Include space for key stakeholders, such as business owners or project managers, to sign off on the UAT report. This indicates agreement that the system meets the necessary requirements or requires further action.

11. Attach Supporting Documentation

- **Supporting Artifacts**: Attach any logs, screenshots, or detailed test case execution results that provide additional context for the report.
- **References**: Include links to the defect tracking system (e.g., JIRA) and any other relevant documentation.

Points to Remember

- In software testing process, User Acceptance Testing (**UAT**) is the final phase where the end-users or business stakeholders validate whether the software meets their requirements and functions as expected in real-world scenarios.
- There are Objectives of UAT which are Confirm Business Requirements, Verify Usability, Identify Any Gaps, Ensure Readiness for Production
- UAT Process are Title Page, Introduction, Acceptance Criteria, Test Plan Summary, Test Execution Details, Test Results, User Feedback, Risk Assessment, Recommendations, Sign-Off Section:
- In UAT there are two types which are : Alpha Testing, Beta Testing
- **The steps to generate a comprehensive UAT report:**
    1. Define the Purpose
    2. Include UAT Test Overview
    3. Identify Test Participants
    4. Detail the UAT Environment
    5. Summarize Test Scenarios and Cases
    6. Defect Summary
    7. Test Metrics
    8. Evaluate Risks and Impact
    9. Recommendations
    10. Provide Conclusion
    11. Attach Supporting Documentation



Application of learning 3.2.

XYZ- solutions ltd is a software development company that specialises in building different softwares. However, the developer is facing with challenges of improving the system by

applying the quality assurance. As a student of level 5 in software development, you are tasked to make User Acceptance Testing Report for that company.

Indicative content 3.3: Generating Recommendation Report

Duration:

Theoretical Activity 3.3.1: Description of recommendation

Tasks:

1: Answer the following question:

- **i.** What is the recommendation report ?
- **ii.** What are the components of recommendation report?
- **iii.** What are the Benefits and impacts of a Recommendation Report?
- **iv.** Describe the archive test artifacts.

2: Participate in group formulation

3: Present the findings to your class or colleagues

4: Ask the clarification if any

5: Read the trainee manual key reading on 3.3.1

Key readings 3.3.1: Recommendation Report

1. recommendation report

is a structured document that presents a problem or issue, evaluates potential solutions, and recommends the best course of action.

2.Components of recommendation report

1. **Introduction**:
   - Brief overview of the issue or decision that needs to be addressed.
   - Purpose of the report.
2. **Background**:
   - Context and details about the issue.
   - Relevant data and information.
3. **Evaluation Criteria**:
   - Criteria used to evaluate potential solutions or options.
4. **Options or Solutions**:
   - Description of each alternative or solution.
   - Analysis of the pros and cons of each option.
5. **Recommendations**:
   - Suggested course of action based on the evaluation.
   - Justification for why this recommendation is preferred.
6. **Implementation Plan** (if applicable):

- Steps and actions needed to implement the recommendation.
- Resources required and timeline.

7. **Conclusion**:
   - Summary of the findings and recommendation.
8. **Appendices** (if needed):
   - Additional data, charts, or documents that support the report.

3.Benefits and Impact of recommendation report

Benefits of a Recommendation Report:

Informed Decision-Making

A well-researched recommendation report provides decision-makers with data-driven insights and analysis, enabling them to make informed choices.

It highlights the best course of action after comparing different alternatives and evaluating their pros and cons.

1.Clarity and Focus

- The report presents information clearly and concisely, helping to focus on the most viable solutions and avoid distractions from less relevant issues.
- It offers structured insights that are easy to understand, making complex decisions more approachable.

2.Objective Analysis

- A recommendation report provides an impartial analysis, as it typically relies on evidence, data, and factual information rather than subjective opinions.
- It evaluates various alternatives objectively, ensuring that personal biases are minimized in decision-making.

3.Cost and Resource Optimization

- By analyzing different options, including the costs, benefits, and risks associated with each, the report helps organizations make cost-effective decisions.
- It supports efficient allocation of resources, reducing waste and improving overall organizational productivity.

4.Risk Mitigation

- A recommendation report often includes risk assessments, helping stakeholders understand potential issues or challenges associated with each option.

- By identifying and proposing mitigation strategies, the report helps prevent costly mistakes.

5.Enhances Communication

- The report serves as a communication tool between stakeholders, teams, and decision-makers, ensuring everyone has access to the same information.
- It can facilitate discussions and consensus by presenting a balanced evaluation of the options.

6. Accountability

- A formal recommendation report documents the reasoning behind a decision. This ensures that decision-makers are accountable for their choices, with a record of the factors that led to a specific decision.
- It also serves as a reference document for future reviews or audits.

7.Time-Saving

- The recommendation report synthesizes large amounts of data, analysis, and options into a single document, saving stakeholders from the need to gather information on their own.
- It enables faster decision-making since the report narrows down the choices and offers a clear path forward.

Impact of a Recommendation Report:

1.Improved Business Performance

- By suggesting the most effective and efficient strategies, recommendation reports can lead to improved operational and financial performance.
- When the recommended solutions are implemented, organizations can achieve their goals faster and with fewer obstacles.

2.Strategic Alignment

- Recommendation reports often align proposed solutions with the overall strategy and objectives of the organization.
- This ensures that the chosen actions contribute to long-term goals and are in harmony with the company's vision and mission.

3.Reduction in Decision Errors

- The structured analysis in a recommendation report helps minimize decision-making errors by evaluating the feasibility, risks, and benefits of each option.

- It lowers the chances of costly mistakes by providing a clear, logical rationale for decisions.

4.Innovation and Growth

- The report often suggests innovative solutions based on market trends, technology advancements, or industry best practices, promoting growth and development.
- It may offer novel approaches to solving problems or expanding into new markets, driving innovation within the organization.

5.Enhanced Stakeholder Confidence

- Stakeholders, including investors, clients, or board members, are more likely to trust and support decisions backed by a thorough recommendation report.
- The transparency of the report fosters confidence in the decision-making process, improving relationships with internal and external parties.

6.Better Risk Management

- By identifying potential risks and proposing mitigation strategies, recommendation reports help organizations avoid or prepare for negative outcomes.
- The proactive nature of these reports means that risk management is embedded into the decision-making process.

7.Guidance for Future Actions

- Recommendation reports provide a roadmap for future actions, guiding the implementation of strategies or solutions.
- They serve as benchmarks for progress and help ensure that actions are aligned with the organization's strategic direction

4. The archiving of test artifacts is a critical step in the overall testing and project lifecycle, ensuring that key testing documents, data, and results are securely stored for future reference, audits, or post-release analysis.

Types of Test Artifacts to Archive

The following key test artifacts are recommended for archiving:

- **Test Plans**: Documents outlining the scope, objectives, strategies, and schedules for UAT.
- **Test Cases**: All test cases executed during UAT, including details of each test case ID, description, expected outcomes, and actual outcomes.

- **Test Results**: The detailed results of each test case execution, categorized into Passed, Failed, or Blocked, along with any supporting evidence (e.g., screenshots, system logs).
- **Defect Logs**: A list of all defects identified during UAT, including descriptions, severity, priority, and resolution status.
- **Change Requests**: Records of any changes or feature requests raised during UAT, including the decisions made on these requests.
- **Test Scripts** (for automated tests): Any test scripts used for automation along with logs of their execution.
- **Communication Records**: Emails, meeting minutes, or any documentation of key discussions and decisions made during the testing phase.

Practical Activity 3.3.2: Generating recommendation Report

Task:

Perform the following activity:

1: Read the key reading 3.3.2.

2: Referring to the theoretical activity 3.3.1, you are requested to do the task below:

i. As software developer you asked to generate recommendation report as used in software development

3: Referring to the task on point(i), make the test result

4:Present to the trainer your test results

5: Ask clarification if any

6: Perform application of learning 3.3.

Key readings 3.3.2: Generating a Recommendation Report

- **Steps to generate a Recommendation Report are**:

1. Provide Recommendations

This section is where you present the core recommendations based on the results

of the testing or evaluation.

a) Proposing Changes

- **Identify Necessary Changes**: Outline specific changes or improvements that are needed based on the testing or findings.
    - Example: "It is recommended that the login functionality be refactored to handle high concurrent user loads to prevent system crashes during peak times."
- **Prioritize Recommendations**: If multiple recommendations are being made, rank them based on priority (e.g., Critical, Major, Minor).
    - Example: "Critical: Fix the checkout process bug, which impacts all users and causes transaction failures."

b) Present Supporting Evidence

- **Justify Each Recommendation**: For each proposed change, provide supporting data, test results, or evidence that explains why this recommendation is necessary.
    - Example: "In the last round of User Acceptance Testing (UAT), 4 out of 5 test cases for the checkout process failed due to a system timeout error, indicating a significant issue with payment gateway integration."
- **Use Data and Metrics**: Include relevant metrics, such as defect counts, pass/fail rates, performance benchmarks, or user feedback, to strengthen your argument.
    - Example: "Our performance testing results show a 50% increase in page load times during high traffic, suggesting the need for server optimization."

c) Outline Action Steps

- **List Specific Steps**: Clearly define the steps needed to implement each recommendation. Make sure the steps are actionable and measurable.
    - Example:
        1. Refactor the login service to optimize database queries.
        2. Conduct a load test with 10,000 concurrent users after the refactor.
        3. Implement a queue system to handle login requests during peak times.

- **Assign Responsibility**: Indicate who will be responsible for each action (e.g., development team, testing team, business analysts).
    - Example: "The development team will address the backend refactoring, while the QA team will conduct retesting once changes are implemented."
- **Define a Timeline**: Provide an estimated timeline or deadlines for each action step.
    - Example: "Complete refactor of the login system within 2 weeks, followed by a performance test in week 3."

2. Conclusion

- Recap the **key recommendations**, emphasizing the importance of implementing the suggested changes to ensure the system's readiness for production or further development.
- Highlight the **benefits of following the recommendations**, such as improved performance, reduced defects, or better user satisfaction.

Example Outline for a Recommendation Section:

1. Recommendation: Improve server response times during peak traffic periods.

- Proposed Change: Refactor the backend code and optimize database queries.

- Supporting Evidence: Performance testing shows that the server response time exceeds 5 seconds when traffic exceeds 5,000 concurrent users.

- Action Steps:

1. Review and optimize slow-running queries in the database.

2. Conduct stress tests after optimization to ensure response times are within acceptable limits.

3. Implement a caching mechanism for frequently accessed data.

- Responsibility: Development team to perform optimizations; testing team to conduct stress tests.

- Timeline: All actions to be completed within 4 weeks.

2. Recommendation: Fix the critical checkout process bug.

- Proposed Change: Investigate and fix the bug causing transaction failures during

the checkout process.

- Supporting Evidence: 4 out of 5 test cases related to the checkout process failed due to a system timeout error.

- Action Steps:

1. Investigate the timeout issue in the payment gateway integration.
2. Deploy a patch to address the bug.
3. Retest the checkout process to ensure the issue is resolved.

- Responsibility: Development team to fix the bug; QA team to retest.

- Timeline: Complete within 1 week.

- **Steps for documenting and archiving test artifacts**

1. Identify Key Test Artifacts for Archiving

In this step, list and describe the test artifacts that need to be archived. These typically include:

- **Test Plan**: Details the objectives, strategy, scope, and schedule for testing.
- **Test Cases**: Specific test scenarios with expected outcomes.
- **Test Results**: Outcome of the test cases (Pass/Fail/Blocked).
- **Defect Logs**: Records of defects encountered during testing, along with severity, status, and resolution.
- **Change Requests**: Any enhancements or adjustments identified during testing.
- **Automated Test Scripts**: Scripts used for automation (if applicable).
- **UAT Report**: Final summary of the User Acceptance Testing outcomes.
- **Communication Records**: Meeting notes, email discussions, and decisions.

2. Define the Documentation Format

Detail how each artifact should be documented for archiving:

- **Test Plans and Reports**: Use consistent document templates with headings for objectives, scope, test environment, and schedule.
- **Test Cases**: Ensure test cases are labeled with unique IDs and descriptions, expected outcomes, and actual outcomes.
- **Test Results**: Clearly show pass/fail status, along with supporting evidence such as screenshots or logs.

- **Defect Logs**: Document defect IDs, descriptions, severity, priority, and resolution status.
- **Automated Test Scripts**: Properly comment and save scripts in a readable and executable format.

3. Establish Organization and Naming Conventions

For ease of access and retrieval, you need to recommend clear **folder structures** and **naming conventions**:

This step ensures the archiving process is organized, and artifacts can be easily retrieved in the future.

4. Select the Storage Medium

Recommend secure and accessible storage options for archiving the test artifacts:

- **Digital Storage**: Cloud solutions (e.g., Google Drive, SharePoint) or dedicated test management tools (e.g., TestRail, Zephyr).
- **Version Control**: For test scripts, use a version control system like Git to track changes.
- **Physical Storage**: If physical documents are involved, provide recommendations for securely storing them.

5 Establish Access Control and Data Security

Recommend security measures to protect sensitive test artifacts:

- **Access Control**: Limit access to archived materials based on roles and permissions, ensuring that only authorized personnel can view or modify documents.
- **Encryption and Backups**: Recommend using encrypted storage for sensitive materials and ensure regular backups are implemented to prevent data loss.

6. Define a Retention Policy

Recommend a clear **retention policy** for archived artifacts:

- **Retention Period**: Define how long each artifact should be stored (e.g., 1–2 years after project completion).
- **Disposal Policy**: Provide guidelines for securely disposing of outdated or irrelevant test artifacts in compliance with organizational and regulatory

requirements.

7. Archive References and Provide Accessibility

Ensure that archived materials are easy to access:

- **Link to Artifacts**: In the UAT report or other final documentation, include references or hyperlinks to the archived test artifacts.
- **Searchability**: Recommend using a searchable repository to allow stakeholders to easily find and retrieve the archived documents.

8. Review and Audit Preparation

Recommend conducting periodic reviews to ensure the completeness and accuracy of archived artifacts:

- **Review Process**: Propose a periodic review schedule to ensure artifacts are archived correctly and are still relevant.
- **Audit Trail**: Suggest implementing audit trails to track who accessed or modified the archived documents.

Points to Remember

- Recommendation report is a structured document that presents a problem or issue, evaluates potential solutions, and recommends the best course of action.
- In Recommendation report there are different components which are : Introduction, Background, Evaluation Criteria, Options or Solutions, Recommendations, Implementation Plan (if applicable),Conclusion, Appendices (if needed).
- In Recommendation report there are Benefits that are: Informed Decision-Making, Clarity and Focus, Objective Analysis, Cost and Resource Optimization, Risk Mitigation, Enhances Communication, Accountability, Time-Saving
- In Recommendation report there are impact that are: Improved Business Performance, Strategic Alignment, Reduction in Decision Errors, Innovation and Growth, Enhanced Stakeholder Confidence, Better Risk Management, Guidance for Future Actions
- In Recommendation report, archiving of test artifacts is a critical step in the overall testing and project lifecycle, ensuring that key testing documents, data, and results are securely stored for future reference, audits, or post-release analysis.

- **Steps to generate a Recommendation Report are**:
  1 . Provide Recommendations
  2. Conclusion
- **Steps for documenting and archiving test artifacts**

  1. Identify Key Test Artifacts for Archiving

  2. Define the Documentation Format

  3. Establish Organization and Naming Conventions

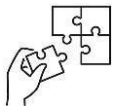
  4. Select the Storage Medium

  5 Establish Access Control and Data Security

  6. Define a Retention Policy

  7. Archive References and Provide Accessibility

  8. Review and Audit Preparation



Application of learning 3.3.

XYZ- solutions ltd is a software development company that specialises in building different applications. However, the programmer is facing challenges of improving the system by applying the quality assurance. As a student of level 5 software development, you are tasked to generate recommendation report for that company.

Learning outcome 3 end assessment

Written assessment

Q1. Circle the letter corresponding with the correct answer

1.Which methodology is used to performed Maintenance testing?

a) Breadth test and depth test
b) Confirmation testing
c) Retesting
d) Sanity testing

2.Which of the following is not part of the Test document?

a) Test Case
b) Requirements Traceability Matrix [RTM]
c) Test strategy
d) Project Initiation Note [PIN]

3.Which term is used to define testing?

a) Evaluating deliverable to find errors
b) Finding broken code
c) A stage of all projects
d) None of the above

4.Which of the following testing is related to the boundary value analysis?

a) White box and black box testing
b) White-box testing
c) Black box testing
d) None of the above

5.What are the different levels of Testing?

a) Integration testing
b) Unit testing
c) System testing
d) All of the above

Q2. Match the Recommendation Elements in column A to Their Corresponding Descriptions in column B.

Answer	Column A (Recommendation Elements)	Column B (Descriptions)
……B……	1.Executive Summary	A. Outlines the steps and specific actions proposed to achieve desired outcomes
……D……	2.Problem Statement	B. Provides a brief overview of the report and the key points of the recommendation.
……E……	3.Proposed Solutions	C. Explains the reasons and supporting arguments for the recommended course of action.
……A……	4. Action Plan	D. Summarizes the issue or challenge that the recommendation addresses.
……F……	5.Supporting Evidence	E. Includes the main suggestions or recommendations made in the report to solve the problem.
……G……	6. Conclusion	F. Offers data, research, or examples that back up the recommendations made in the report.
……C……	7.Recommendations	G. Wraps up the report by reinforcing the key points and the importance of acting on the recommendations.

Q3. What is the primary purpose of a User Acceptance Testing (UAT) report?

Q4. Who are the typical participants involved in UAT, and what roles do they play?

Q5**.** How are test cases designed for UAT, and what factors are considered while creating them?

Q6**.** What is the difference between "Passed" and "Failed" test cases in the UAT report?

Q7**.** Why is it important to include both functional and business requirements in UAT?

Q8. What are some common issues identified during UAT, and how are they typically resolved?

Q9**.** How do delays in UAT affect the overall project timeline and deployment?

Q10. What actions should be taken after UAT if certain test cases fail?

Q11**.** What role does test data play in the success of UAT?

Q12**.** Why is it important to document the actual results alongside the expected results in UAT?

Practical assessment

XYZ software system company, is preparing to release a new product recommendation feature for its software system. Before launch, the QA team is tasked with ensuring that the feature works as expected by :

1. Consolidating test results,
2. Conducting User Acceptance Testing (UAT)
3. Providing a recommendation report to stakeholders.

The goal is to verify system functionality, performance, and usability, ensuring the new feature meets both business and user expectations. As the one of QA team , you are tasked to perform the above task.

END

Reference

Elazar, E. (2018, April 23). *panaya.com/blog/testing/what-is-uat-testing*. Retrieved from www.panaya.com/blog/testing/what-is-uat-testing: https://www.panaya.com/blog/testing/what-is-uat-testing/#:~:text=User%20Acceptance%20Testing%20(UAT)%2C,do%20in%20real%2Dworld%20situations.

Felice, S. (2024, May 21). *browserstack.com/guide/what-is-test-data*. Retrieved from www.browserstack.com: https://www.browserstack.com/guide/what-is-test-data#:~:text=Test%20data%20is%20a%20crucial,exceptions%20and%20error%2Dhandling%20cases.

Team, I. E. (2023, June 29). *indeed.com/career-advice*. Retrieved from www.indeed.com: https://www.indeed.com/career-advice/career-development/how-to-write-recommendation-report

Technologies, Q. (2024, June 06). *q3tech.com/blogs*. Retrieved from www.q3tech.com: https://www.q3tech.com/blogs/what-is-user-acceptance-testing-uat-a-complete-guide/

RTB | RWANDA TVET BOARD

October 2024