Introduction to Software Testing

Understanding Fundamentals of Software Testing and Industry Standards

Eng. Hussein Ali
DTA Board | Unit Manager | Training and Consultation
husseinalicoach@gmail.com
+201004942297

Learning Objectives

- Understand basic concepts of software testing.
- Explore the importance of ISTQB certification.
- Differentiate Types of Testing.

What is Software Testing?

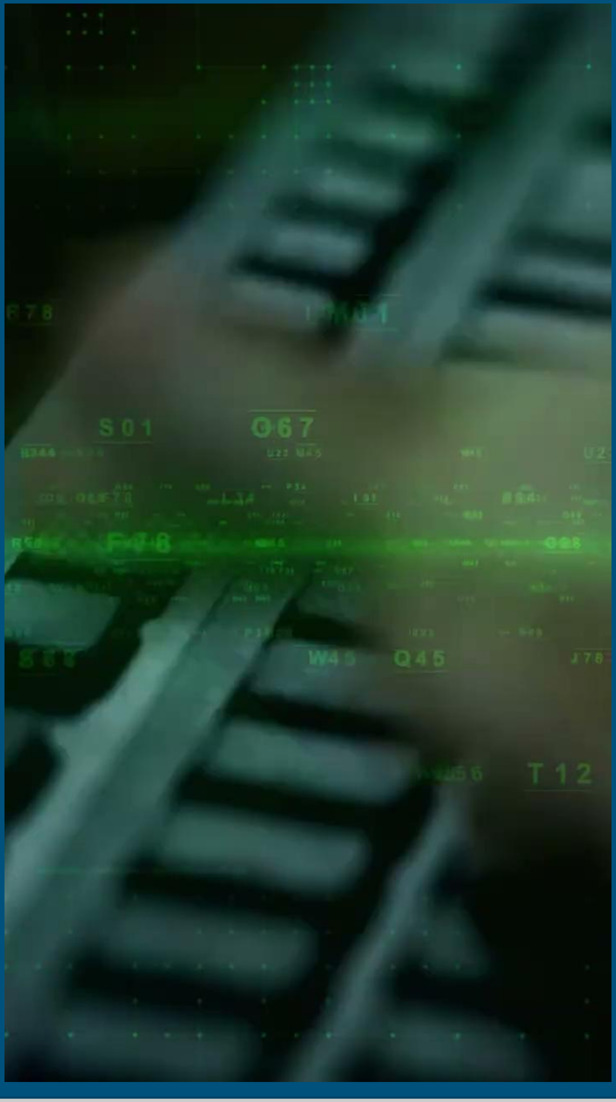
Definition:

Verifies and **Validates** software to ensure it meets requirements.

Goals:

- Identify defects.
- Validate user needs.
- Deliver high-quality products.

Why Software Testing is Critical



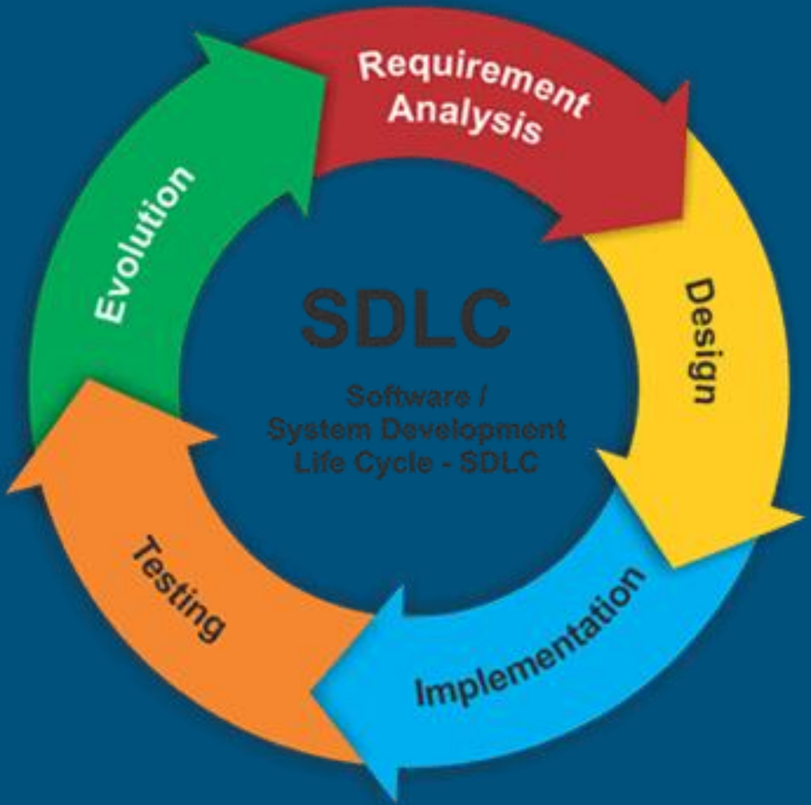
Why Software Testing is Critical

- On the 4th of June 1996 the European Space Agency’s (ESA), Ariane 5 rocket exploded on her maiden voyage at **3700 meters, 37 seconds after launch**. It was one of the most expensive software bugs in history, costing over **US$370 million**.
- Testing **saves time** and **money** by avoiding costly errors.

Software Lifecycle (SDLC)

1. Requirement Analysis.
2. Design.
3. Development.
4. Testing.
5. Deployment

    and Maintenance .



Software Testing Lifecycle (STLC)

1. Test Analysis.
2. Test Planning.
3. Test Case Design.
4. Test Execution.
5. Test Reporting.
6. Test Evaluation/Closure.

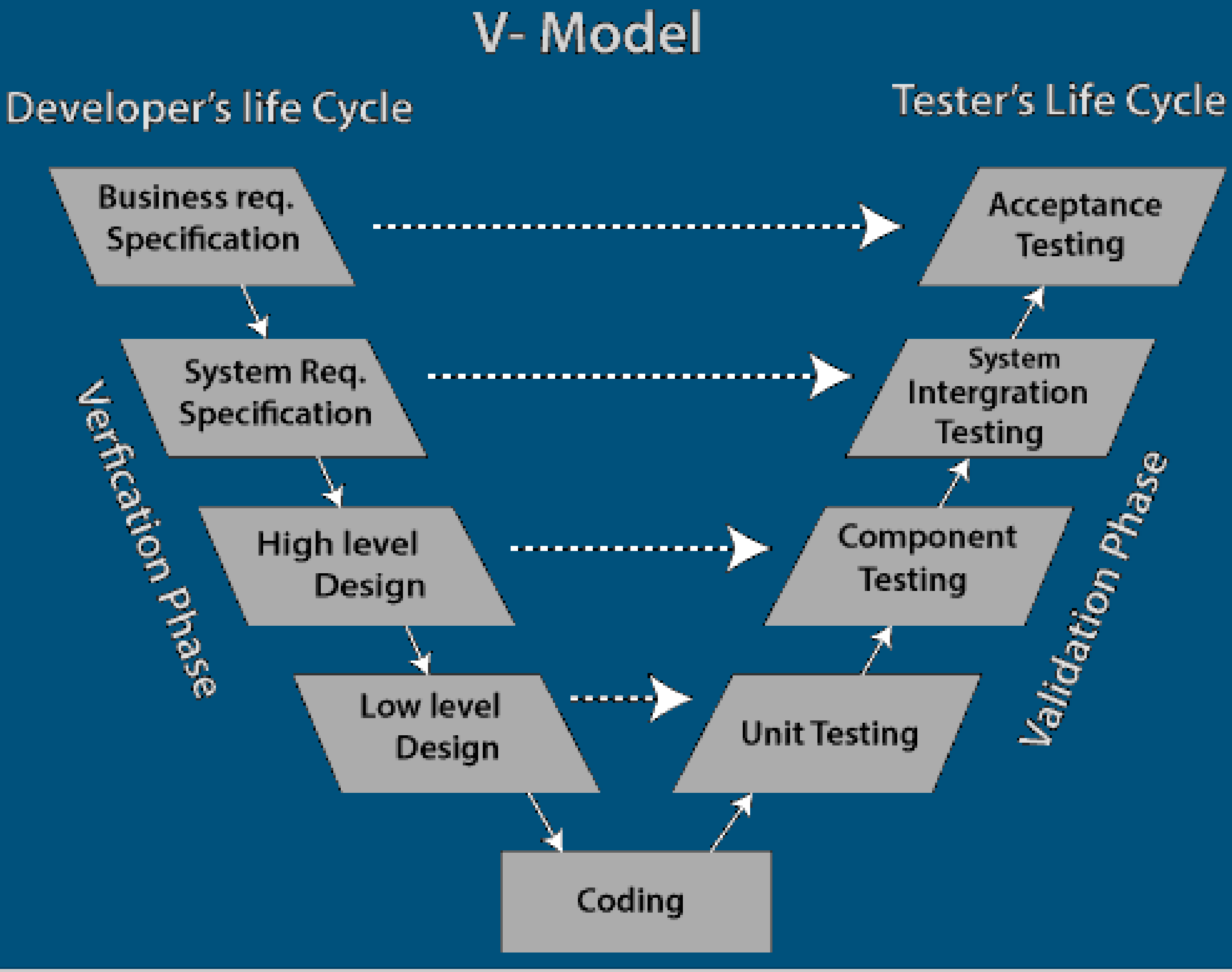
   Entry/Exit Criteria

   Define start and stop points

Software Testing Lifecycle

- Phases:
  - Test Planning: Define the overall testing strategy, including test objectives, scope, resources, and schedule.
  - Test Analysis: Understand the software requirements to identify testable features.
  - Test Case Design: Create test cases based on requirements and design specifications.
  - Test Execution: Execute test cases and record results.
  - Test Reporting: Document test results, defects, and progress reports.
  - Test Evaluation (Entry/Exit Criteria): Analyze test results and determine if the software meets quality standards.

Software Lifecycle (STLC)



ISTQB Overview

What is ISTQB?

- International Software Testing Qualifications Board
- Sets global standards for software testing certifications

Why ISTQB Certification?

- Industry-recognized certification
- Demonstrates knowledge and skills
- Improves career prospects

ISTQB Overview

ISTQB
Certified
Tester

- Foundation-Level Syllabus covers:
  - **Testing Principles:** General principles that apply to all testing activities, such as the principle of early testing and the impossibility of exhaustive testing.
  - **Testing Process:** A structured approach to planning, designing, executing, and evaluating testing activities.
  - **Test Levels:** Different levels of testing, including unit, integration, system, and acceptance testing.
  - **Test Techniques:** Specific methods used to design and execute tests, such as equivalence partitioning, boundary value analysis, and state transition testing.

ISTQB- Testing Principles

1. **Testing shows the presence of defects, not their absence**

- defects are present, but cannot prove that there are no defects.
- Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, testing is not a proof of correctness.

ISTQB- Testing Principles

2. Exhaustive testing is impossible

- Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Rather than attempting to test exhaustively, risk analysis, test techniques, and priorities should be used to **focus test efforts**.

ISTQB- Testing Principles

3. Early testing saves time and money

- To find defects early, both static and dynamic test activities should be started as early as possible in the software development lifecycle.
- Early testing is sometimes referred to as shift left.
- Testing early in the software development lifecycle helps reduce or eliminate costly changes.

ISTQB- Testing Principles

4. Defects cluster together

- A small number of modules usually contains most of the defects discovered during pre-release testing or is responsible for most of the operational failures.
- Predicted defect clusters, and the actual observed defect clusters in test or operation, are important input into a risk analysis used to focus the test effort.

ISTQB- Testing Principles

5. Beware of the pesticide paradox

- If the same tests are repeated over and over again, eventually these tests no longer find any new defects.
- To detect new defects, existing tests and test data may need changing, and new tests may need to be written. (Tests are no longer effective at finding defects, just as pesticides are no longer effective at killing insects after a while.)
- In some cases, such as automated regression testing, the pesticide paradox has a beneficial outcome, which is the relatively low number of regression defects.

ISTQB- Testing Principles

6. Testing is context-dependent

- Testing is done differently in different contexts. For example, safety-critical industrial control software is tested differently from an e-commerce mobile app.
- As another example, testing in an Agile project is done differently than testing in a sequential software development lifecycle project.

ISTQB- Testing Principles

7. Absence-of-errors is a fallacy

- Some organizations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible.
- Further, it is a fallacy (i.e., a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system.
- For example, thoroughly testing all specified requirements and fixing all defects found could still produce a system that is difficult to use, that does not fulfill the users’ needs and expectations, or that is inferior compared to other competing systems.

ISTQB- Testing Process

Test Planning and Control

- **Define the testing objectives:** Clearly outline the goals of the testing effort, such as finding defects, verifying requirements, or assessing performance.
- **Develop a test strategy:** Create a high-level plan that outlines the overall approach to testing, including the testing scope, resources, schedule, and risks.
- **Create a test plan:** Develop a detailed plan that specifies the test cases, test procedures, test environments, and test data.
- **Establish exit criteria:** Define the criteria that must be met to conclude the testing phase, such as the number of defects found or the percentage of test cases passed.
- **Allocate resources:** Assign resources (e.g., testers, tools, hardware) to the testing activities.

ISTQB- Testing Process

Test Analysis and Design

- **Review test basis:** Analyze the requirements, design specifications, and other relevant documents to identify testable features and functions.
- **Design test cases:** Create detailed test cases that specify the test inputs, expected outputs, and test procedures.
- **Prioritize test cases:** Determine the order in which test cases should be executed based on risk and importance.
- **Review test cases:** Conduct peer reviews to ensure the quality and completeness of the test cases.

ISTQB- Testing Process

Test Implementation and Execution

- **Develop test procedures:** Create step-by-step instructions for executing test cases.
- **Set up test environment:** Configure the hardware and software required for testing.
- **Execute test cases:** Run test cases according to the test procedures and record the results.
- **Log defects:** Report any defects found during testing, including detailed information about the defect, its severity, and its impact.
- **Retest defects:** Retest fixed defects to ensure they have been resolved correctly.

ISTQB- Testing Process

Evaluating Exit Criteria and Reporting

- **Evaluate test results:** Analyze the test results to determine if the testing objectives have been met.
- **Assess test coverage:** Determine the extent to which the software has been tested.
- **Prepare test reports:** Document the test results, defects, and overall test progress.
- **Make a decision to release or not to release:** Based on the test results and exit criteria, decide whether the software is ready for release.

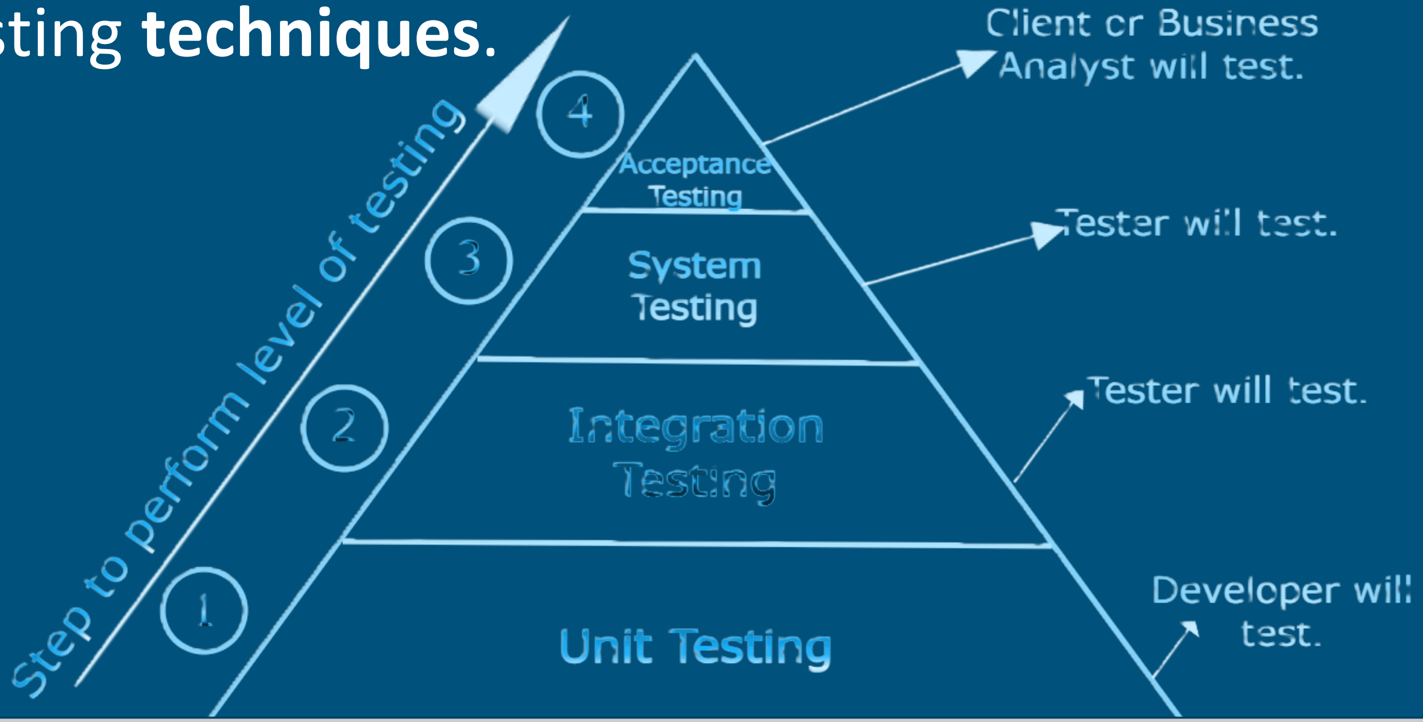
ISTQB- Testing Process

Test Closure Activities

- **Evaluate the testing process:** Analyze the effectiveness of the testing process and identify areas for improvement.
- **Archive test documentation:** Store test artifacts for future reference.
- **Release test environment:** Release the test environment for other purposes.

ISTQB- Testing Levels and Techniques

Testing levels are different phases of testing that are performed during the software development lifecycle. Each **level** focuses on specific **objectives** and uses different testing **techniques**.



ISTQB- Testing Levels and Techniques

Component Testing (Unit Testing):

Objective: To test individual software components in isolation.

Scope: Focuses on the smallest testable units of code, such as functions, procedures, or classes.

Techniques: White-box testing techniques like statement coverage, branch coverage, and path coverage are commonly used.

Responsibility: Typically performed by developers.

ISTQB- Testing Levels and Techniques

Integration Testing:

Objective: To test the interaction between integrated components.

Scope: Focuses on the interfaces between components and how they work together.

Techniques: Both black-box and white-box techniques can be used.

Responsibility: Can be performed by developers or testers.

ISTQB- Testing Levels and Techniques

System Testing:

Objective: To test the entire system as a whole.

Scope: Focuses on the system's behavior as a complete product.

Techniques: Black-box techniques like functional testing, non-functional testing (performance, security, usability, etc.), and system integration testing are commonly used.

Responsibility: Typically performed by testers.

ISTQB- Testing Levels and Techniques

Acceptance Testing:

Objective: To verify that system meets user requirements.

Scope: Focuses on user's perspective and how system meets their needs.

Techniques: Black-box techniques like functional testing and usability testing are commonly used.

Responsibility: Can be performed by users, customers, or a dedicated acceptance testing team.

ISTQB- Testing Levels and Techniques

Additional Testing Levels

Alpha Testing: Early user testing conducted by a limited number of users within the organization.

Beta Testing:

Late-stage user testing conducted by a larger group of users outside the organization.

Common Testing Terminology

- Bug vs. Defect vs. Failure.
- Test Case and Test Suite.
- Types:
    - Regression
    - Functional
    - Non-functional testing...

Bug vs. Defect vs. Failure

Defect:

- A flaw in a component or system that can cause the component or system to fail to perform its required function.
- It's a deviation from the expected behavior of the software.
- **Example:** A typo in a code line that prevents a specific function from executing correctly.

Bug vs. Defect vs. Failure

Bug:

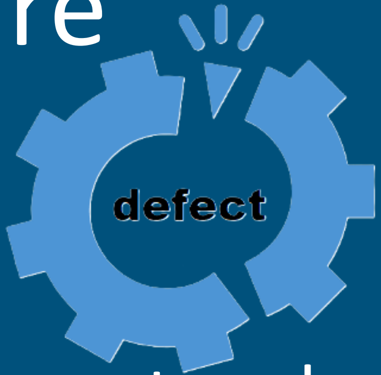
- A specific instance of a defect that has been identified and reported.
- It's a manifestation of a defect that causes the software to behave incorrectly.
- **Example:** A user reports that a particular button on a website is not working as expected. This is a bug caused by the underlying defect in the code.

Bug vs. Defect vs. Failure

Failure:

- An event in which a system or system component does not perform a required function within specified limits.
- Observable consequence of a defect or bug.
- **Example:** A website crashes due to a memory leak, preventing users from accessing it. This is a failure caused by a defect in the memory management code.

Bug vs. Defect vs. Failure



Relationship between the Three:

- A **defect** can lead to a **bug** when it is encountered during testing or usage.

- A **bug** can cause a **failure** if it impacts the software's functionality and prevents it from performing its intended task.

Test Case vs. Test Suite

Test Case

A single unit of testing that verifies a specific functionality.

- **Components**:
  - **Test ID**: A unique identifier for the test case.
  - **Test Case Description**: A brief description of the test case's purpose.
  - **Test Steps:** Detailed sequential steps to be followed to execute test.
  - **Test Data:** Input data required for the test.
  - **Expected Results:** Expected outcomes of executing the test steps.
  - **Actual Results:** Actual outcomes recorded after executing the test.
  - **Pass/Fail:** A status indicating whether the test passed or failed.

- **Purpose:** To validate a specific feature or requirement of the software.

Test Case vs. Test Suite

Test Suite

A collection of related test cases organized into a logical group.

- **Components:**
  - A set of related test cases.
  - A description of the test suite's purpose.
  - Information about the test environment and configuration.
  - Execution order of test cases.
  - Test data requirements.
- **Purpose:**
  - To improve test case management and organization.
  - To streamline test execution and reporting.
  - To group test cases based on specific functionalities, modules, or user scenarios.

Types of Testing

- Functional Testing:
- Focus: Validating requirements.

  - Examples: Unit, integration testing.

- Non-Functional Testing:
- Focus: Performance, usability, security.

- Manual vs. Automated Testing:

When to use each.

Types of Testing

1. **Functional Testing:**

Testing the software's functionality to ensure it performs as expected.

- **Unit Testing:** Testing individual software components in isolation.
- **Integration Testing:** Testing the interaction between integrated components.
- **System Testing:** Testing the entire system as a whole.
- **Acceptance Testing:** Testing the system to ensure it meets user requirements.

Types of Testing

2. Non-Functional Testing:

Testing non-functional attributes of the software.

- **Performance Testing:** Testing the system's response time, throughput, and resource utilization.
- **Security Testing:** Testing the system's vulnerability to security threats.
- **Usability Testing:** Testing the system's ease of use and user experience.
- **Compatibility Testing:** Testing the system's compatibility with different hardware, software, and environments.

Types of Testing

Manual Testing vs. Automated Testing

3. **Manual Testing:** Testing performed by human testers.

4. **Automated Testing:** Testing performed by automated tools.

When to Automate:

- Repetitive tests
- Time-consuming tests
- Tests that require high precision
- Tests that are difficult to perform manually

Case Study on Testing Approaches

- Scenario: E-commerce website with checkout delays and unresponsive buttons.
- Task:
  - Identify the issues.
  - Recommend testing types (functional, performance).
  - Discuss manual vs. automated approaches.

Wrap-Up

- Testing ensures quality and reduces risks.
- ISTQB is the gold standard for testing certification.
- Mastering lifecycle and terminology is essential.

Day-2

1. **Static Testing vs. Dynamic Testing**
2. **Agile Methodologies Testing**
3. **Risk Management**
4. **Testing Roles**

Static Testing vs. Dynamic Testing

- **Static Testing:**
  - Performed **without executing** the software.
  - Involves manual or automated examination of the software's source code, design documents, and other static artifacts.
  - **Examples:**
    - **Code reviews:** Manual inspection of the code by peers or experts.
    - **Static analysis:** Automated techniques like code analysis tools to detect potential issues such as syntax errors, security vulnerabilities, and code style violations.
    - **Design reviews:** Examining the software design documents for completeness, consistency, and correctness.
    - **Data flow analysis:** Tracing the flow of data through the software to identify potential errors.

Static Testing

- **Dynamic Testing:**

- Involves **executing** the software.
- Observing the actual behavior of the software under test.

- **Examples:**

- **Unit testing:** Testing individual components or modules of the software.
  Integration testing: Testing the interaction between different components or modules.
- **System testing:** Testing the entire system as a whole.
- **Acceptance testing:**
- Testing the software against[1] the user's requirements to ensure it meets their needs.
- **Performance testing:** Evaluating the software's performance under various workloads.
- **Security testing:** Assessing the software's vulnerability to security threats.

Severity vs. Priority

- **Severity:**
- Describes the **technical impact** of the defect on the software.
- Focuses on the **seriousness of the issue** itself.
- Examples:
    - **Critical:** Application crash, data loss, security breach.
    - **Major:** Significant functionality loss, major usability issues.
    - **Minor:** Cosmetic issues, minor usability problems.
    - **Trivial:** Spelling errors, minor cosmetic issues.
- **Priority:**
- Describes the **business impact** and **urgency** of fixing the defect.
- Focuses on how quickly the issue **needs to be resolved**.
- Examples:
    - **High:** Urgent fix required to prevent major business disruption.
    - **Medium:** Should be fixed in the next release.
    - **Low:** Can be fixed in a future release.

Severity vs. Priority

- **Key Differences:**
- **Severity:** Technical impact on the software.
- **Priority:** Business impact and urgency of the fix.

- **Example:**
- **Scenario:** A spelling error in a minor, infrequently used help section.
  - **Severity:**
  - **Priority:**
- **Scenario:** A critical bug that crashes the application and causes data loss.
  - **Severity:**
  - **Priority:**

Severity vs. Priority

- **Key Differences:**
- **Severity:** Technical impact on the software.
- **Priority:** Business impact and urgency of the fix.

- **Example:**
- **Scenario:** A spelling error in a minor, infrequently used help section.
  - **Severity:** Trivial
  - **Priority:** Low
- **Scenario:** A critical bug that crashes the application and causes data loss.
  - **Severity:** Critical
  - **Priority:** High

Agile Development Methodologies

- **Test-Driven Development (TDD):**
- A development approach where tests are written **before** the actual code.
- The cycle involves writing a test case that fails, then writing the minimum amount of code to pass the test, and finally refactoring the code to improve its design and maintainability.
- **Focus:** Primarily on unit testing.

Agile Development Methodologies

- **Acceptance Test-Driven Development (ATDD):**
- Involves collaboration between business stakeholders, developers, and testers to define acceptance criteria for user stories or features.
- Acceptance tests are written based on these criteria **before** development begins.
- **Focus:** On high-level requirements and user acceptance.

Agile Development Methodologies

- **Behavior-Driven Development (BDD):**
- An extension of TDD that emphasizes the behavior of the software from the perspective of the user or customer.
- Uses a common language (like Gherkin) to describe scenarios and expected outcomes in a human-readable format (e.g., "Given...When...Then").
- **Focus:** On collaboration and communication within the development team and with stakeholders.

Agile Development Methodologies

- **Similarities between TDD, ATDD, and BDD:**
- All three emphasize **testing early and often**.
- They promote **continuous feedback** and **iterative development**.
- They foster **collaboration** among stakeholders.
- They contribute to **higher quality software** by identifying and addressing issues early in the development cycle.

Agile Development Methodologies

- **Extreme Programming (XP)**
- A lightweight, agile development methodology that emphasizes **customer satisfaction** and **rapid feedback**.
- **Key practices:**
  - **Pair programming:** Two developers work together at one workstation.
  - **Test-driven development:** Tests are written before any code is written.
  - **Continuous integration:** Frequent integration of code changes into a shared repository.
  - **Collective ownership:** Any developer can improve any part of the code.
  - **Simple design:** Focus on the current requirements and avoid over-engineering.
  - **Refactoring:** Continuously improving the code design without changing its functionality.

Agile Development Methodologies

- **Feature-Driven Development (FDD)**
- A model-driven development process that focuses on the rapid delivery of small, customer-valued features.
- **Key activities:**
  - **Develop an overall model:** Create a high-level model of the system.
  - **Build a list of features:** Identify and prioritize features based on customer value.
  - **Plan by feature:** Plan the development of each feature in detail.
  - **Design by feature:** Design the components and classes required for each feature.
  - **Build by feature:** Develop and test each feature independently.

Risk Management

- **Scenario:**
- A company is developing a new e-commerce website for online clothing sales. The website will include features such as product browsing, user registration, shopping cart, payment gateway integration, and order tracking.

Risk Management

- **Risk Identification:**

1. **Technical Risks:**
    - **Performance Issues:** The website may experience slow loading times or crashes under heavy traffic during peak shopping seasons (e.g., Black Friday, Cyber Monday).
    - **Security Vulnerabilities:** The website may be susceptible to hacking attacks (e.g., SQL injection, cross-site scripting) or data breaches, compromising customer information.
    - **Compatibility Issues:** The website may not be compatible with all major browsers and devices (desktops, laptops, tablets, smartphones).
    - **Integration Issues:** There may be problems integrating with the payment gateway, leading to failed transactions.
2. **Project Risks:**
    - **Schedule Delays:** Unexpected delays in development, testing, or deployment could lead to missed deadlines and impact the launch date.
    - **Budget Constraints:** Budget overruns could force compromises on testing activities or the use of less-qualified personnel.
    - **Communication Issues:** Poor communication between developers, testers, and stakeholders could lead to misunderstandings and rework.
    - **Staffing Issues:** Lack of skilled testers or key personnel leaving the project could impact testing progress and quality.

Risk Management

- **Risk Assessment:**
- For each identified risk, assess the following:
- **Likelihood:** How likely is this risk to occur? (e.g., High, Medium, Low)
- **Impact:** What would be the potential consequences of this risk occurring? (e.g., High, Medium, Low)

Risk Management

- **Risk Mitigation:**
- Based on the risk assessment, implement appropriate mitigation strategies:
- **Performance Issues:**
  - **Mitigation:** Conduct performance testing under simulated peak load conditions. Use load testing tools
- **Security Vulnerabilities:**
  - **Mitigation:** Perform security testing, including penetration testing and vulnerability scanning.
- **Compatibility Issues:**
  - **Mitigation:** Test the website on a variety of browsers and devices. Use cross-browser testing tools to ensure compatibility.
- **Integration Issues:**
  - **Mitigation:** Conduct thorough integration testing between the website and the payment gateway.
- **Schedule Delays:**
  - **Mitigation:** Use agile development methodologies to adapt to changes and minimize delays.
- **Budget Constraints:**
  - **Mitigation:** Prioritize testing activities based on risk. Consider using cost-effective testing techniques such as risk-based testing.
- **Communication Issues:**
  - **Mitigation:** Establish clear communication channels and regular meetings between stakeholders.
- **Staffing Issues:**
  - **Mitigation:** Ensure adequate staffing with qualified testers. Provide training and mentoring to junior testers.

Risk Management

- **Risk Monitoring:**
- Continuously monitor the identified risks throughout the project lifecycle.
- Track any changes in the likelihood or impact of risks.
- Adjust mitigation strategies as needed based on the changing project environment.
- Document all risk-related activities and decisions.

Continuous Integration and Continuous Delivery

- **Continuous Integration (CI)**
- **Definition:** A software development practice where developers frequently integrate their code changes into a shared repository (e.g., Git).
- **Key Principles:**
  - **Frequent Integrations:** Developers integrate their code changes multiple times a day.
  - **Automated Builds:** Automated builds and tests are triggered automatically after each code commit.
  - **Rapid Feedback:** Quick feedback is provided to developers on the success or failure of the build and tests.
  - **Early Defect Detection:** Issues are identified and addressed early in the development cycle.

Continuous Integration and Continuous Delivery

- **Continuous Delivery (CD)**
- **Definition:** An extension of CI where all code changes that pass the automated tests are automatically deployed to a staging or production environment.
- **Key Principles:**
  - **Automated Deployments:** Automated scripts are used to deploy code changes to various environments.
  - **Release Readiness:** Software is always in a releasable state.
  - **Reduced Release Cycles:** Shorter release cycles enable faster delivery of value to customers.

Tester Roles

- **Facilitator:**
- **Responsibilities:**
    - Guides the meeting process and ensures it stays on track.
    - Creates a safe and inclusive environment for open discussion.
    - Encourages participation from all members.
    - Manages time effectively and keeps the meeting moving forward.
    - Summarizes key points and decisions.
    - May use various facilitation techniques (e.g., brainstorming, mind mapping, voting).

Tester Roles

- **Author:**
- **Responsibilities:**
  - Responsible for creating and delivering presentations, reports, or other written materials related to the meeting's topic.
  - May conduct research, gather information, and draft the initial content.
  - May be responsible for presenting findings and recommendations to the group.

Tester Roles

- **Scribe:**
- **Responsibilities:**
    - Records the key discussions, decisions, action items, and conclusions of the meeting.
    - May also capture visual information like diagrams or mind maps.
    - Ensures accurate and comprehensive documentation of the meeting proceedings.

Tester Roles

- **Manager:**
- **Responsibilities:**
  - Oversees the overall meeting process and ensures it aligns with project goals.
  - May be responsible for setting the meeting agenda, inviting participants, and allocating resources.
  - Ultimately accountable for the successful outcome of the meeting

Q & A

Thanks

Eng. Hussein Ali
DTA Board | Unit Manager | Training and Consultation
husseinalicoach@gmail.com
+201004942297