Lecture 2

Lecture Overview

- Key principles QA in software development
- Key practices of QA in software development
- Software Quality Models and Standards

Key principles of QA in software development

Principles of QA in Software Development:

The key principles of Quality Assurance (QA) in software development focus on ensuring that software meets quality standards and customer requirements efficiently and effectively. Here are the primary principles:

1. Customer Focus

- **Principle**: Quality is defined by the customer's needs and expectations, meaning QA processes aim to ensure the software fulfils these requirements.
- **Application**: QA teams work closely with stakeholders to understand customer requirements, translate them into quality objectives, and verify that the final product aligns with these expectations.

2. Prevention over Detection:

- **Principle:** Emphasize preventing defects rather than identifying and fixing them later in the development process.
- **Application:** Implementing quality controls in each phase of the Software Development Life Cycle (SDLC), like code reviews, static analysis, and adherence to coding standards, helps to prevent defects early on.

3. Team Collaboration and Accountability

- **Principle**: Quality is the responsibility of the entire team, not just the QA team.
- **Application**: In Agile and DevOps, QA is integrated into development teams, encouraging cross-functional collaboration. Each team member, from developers to testers, shares accountability for quality, promoting a culture of quality ownership.

Key principles of QA in software development

4. Continuous Improvement:

- **Principle:** QA is an ongoing process that requires regular refinement and enhancement.
- **Application**: By using metrics, feedback, and meetings, teams can identify areas for improvement and adjust QA processes. Continuous improvement aligns with standards like ISO 9001 and practices like Agile.

5. Process-Oriented Approach

- **Principle**: A strong process foundation enables better consistency and predictability in software quality.
- **Application**: QA involves establishing and following standardized processes, such as testing protocols, development methodologies, and quality audits, to ensure each project follows a repeatable approach.

6. Factual Decision Making

- **Principle**: Decisions related to quality should be based on data and analysis rather than assumptions.
- **Application**: Using metrics (e.g., defect density, test coverage, and reliability scores) and data from quality audits allows for informed decision-making and accurate assessment of quality status.

7. Standardization and Compliance

- **Principle**: Adhering to established standards (like IEEE, ISO, or CMMI) provides a structured framework that supports quality and consistency.
- **Application**: Following standards ensures that quality practices are consistent and that software meets industry and regulatory requirements.

Key principles of QA in software development

8. Factual Decision Making

- **Principle**: QA decisions should be data-driven, based on analysis rather than assumptions.
- **Application**: Use quality metrics (e.g., defect density, code coverage, test pass rates) and insights from testing to make informed decisions about release readiness and quality status.

9. Risk Management

- **Principle**: QA should identify and address potential risks to ensure that they do not impact quality.
- **Application**: Risk-based testing helps prioritize testing efforts based on the potential impact and likelihood of risks, focusing QA resources on the most critical areas.

10. Transparency and Traceability

- **Principle**: Quality processes and results should be visible and traceable to maintain clarity and accountability.
- **Application**: Use traceability matrices, requirement mapping, and defect tracking to ensure transparency, making it easier to audit, review, and monitor quality.

11. Early Testing

- **Principle**: Testing should begin as early as possible in the SDLC to identify and address defects sooner.
- **Application**: Practices like unit testing during coding, integration testing after each feature, and continuous integration help detect and resolve defects early, saving time and reducing cost.

Key practices of QA in software development

1. Requirements Analysis and Validation

- **Practice**: Ensure that requirements are clear, complete, and aligned with customer needs before development begins.
- **Application**: Conduct requirements workshops, validation meetings, and reviews to clarify requirements, reduce ambiguity, and establish testable criteria.

2. Test Planning and Strategy

- **Practice**: Develop a comprehensive test plan and strategy that aligns with project goals, risks, and constraints.
- **Application**: Define test objectives, scope, resources, timelines, and responsibilities in a test plan. The strategy should cover types of testing (functional, performance, security) and align with risk prioritization.

3. Code Reviews and Static Analysis

- **Practice**: Review code early to catch issues related to functionality, security, and maintainability.
- **Application**: Use code reviews (peer reviews or pull requests) and static analysis tools to identify bugs, coding standard violations, and security vulnerabilities before testing.

4. Continuous Integration and Continuous Testing

- **Practice**: Integrate code frequently and run automated tests on each integration to catch defects early.
- **Application**: Set up CI/CD pipelines to automate builds and run unit, integration, and regression tests with every code change. This practice ensures that new code does not break existing functionality.

Key practices of QA in software development

5. Automated Testing

- **Practice**: Automate repetitive, high-coverage tests to improve efficiency and accuracy.
- **Application**: Use automated tests for unit, functional, and regression testing. Focus on high-priority and frequently used features to save time and resources in testing cycles.

6. Risk-Based Testing

- **Practice**: Prioritize testing efforts based on the risk and criticality of features.
- **Application**: Assess features for their business importance and potential impact of failure. Allocate more testing resources to high-risk areas, such as security, data handling, or mission-critical functions.

7. Functional Testing

- **Practice**: Verify that the software performs as expected by validating individual functions and their interactions.
- **Application**: Execute test cases covering different aspects of functionality, including input validation, business rules, and edge cases, to ensure software meets requirements.

8. Regression Testing

- **Practice**: Regularly test previously developed functionality to confirm that changes have not introduced new defects.
- **Application**: Maintain a suite of regression tests to verify that updates or bug fixes do not impact existing features, ensuring product stability over time.

Key practices of QA in software development

9. Performance and Load Testing

- **Practice**: Assess software performance under varying loads to ensure it meets speed, scalability, and stability requirements.
- **Application**: Conduct load, stress, and endurance testing to measure response times, system stability, and resource utilization under high user loads and for extended periods.

10. Security Testing

- **Practice**: Identify and mitigate potential security vulnerabilities within the software.
- **Application**: Perform security assessments, penetration tests, and vulnerability scans to detect and fix security flaws, ensuring compliance with security standards and protecting data integrity.

11. Defect Tracking and Management

- **Practice**: Track and manage defects from detection through resolution, maintaining visibility over the quality process.
- **Application**: Use a defect-tracking tool to document, prioritize, assign, and monitor defects. Implement root-cause analysis for recurring issues and work closely with development teams for fixes.

12. User Acceptance Testing (UAT)

- **Practice**: Involve end-users or clients in testing to validate that the software meets real-world needs and is ready for deployment.
- **Application**: Provide users with test cases and scenarios to verify functionality and usability. Use feedback to make final adjustments before release.

Key practices of QA in software development

13. Quality Metrics and Reporting

- **Practice**: Measure and track key quality metrics to evaluate project health and identify improvement areas.
- **Application**: Collect metrics like defect density, test coverage, and pass/fail rates. Generate regular reports to inform stakeholders and guide decisions on quality improvements.

14. Continuous Improvement and Retrospectives

- **Practice**: Regularly review QA processes to identify areas for enhancement.
- **Application**: Conduct retrospectives at the end of each sprint or release cycle to reflect on what worked well and areas that need improvement. Update processes based on feedback to improve future projects.

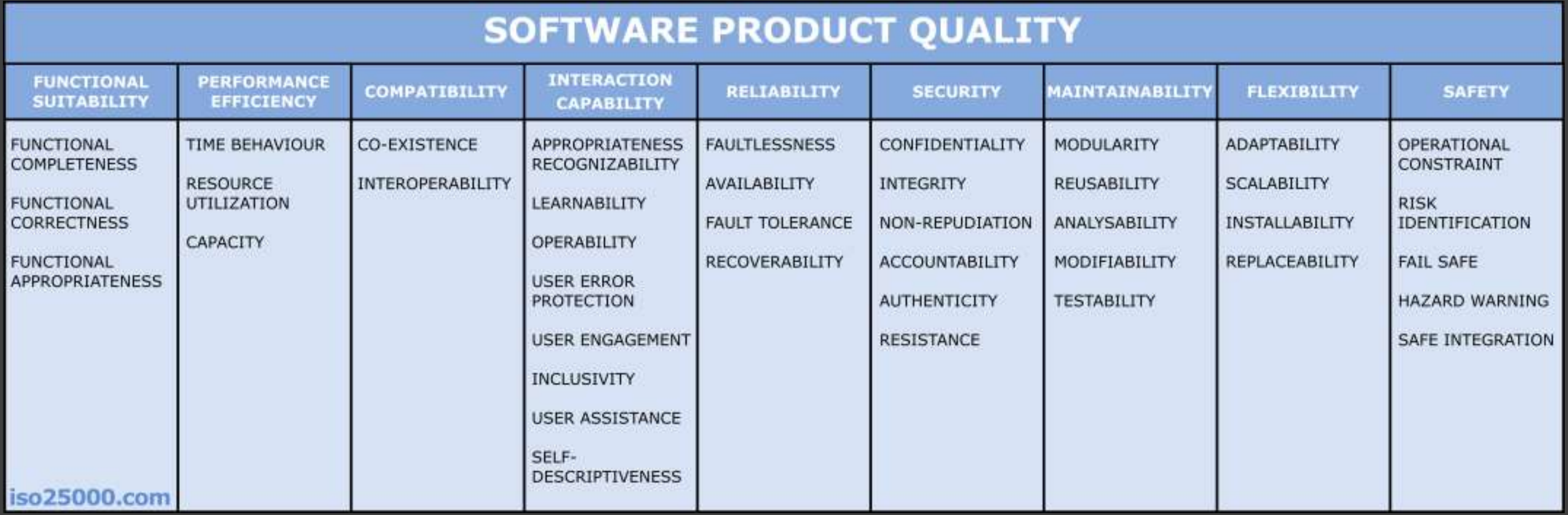
Software Quality Models and Standards

What is a Quality Model?

A Quality Model is a set of criteria or standards used to measure software quality.
Think of it as a checklist that helps ensure software meets certain quality goals.

Key Software Quality Models

- **ISO/IEC 25010 Quality Model**



SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS	TIME BEHAVIOUR	CO-EXISTENCE	APPROPRIATENESS RECOGNIZABILITY	FAULTLESSNESS	CONFIDENTIALITY	MODULARITY	ADAPTABILITY	OPERATIONAL CONSTRAINT
FUNCTIONAL CORRECTNESS	RESOURCE UTILIZATION	INTEROPERABILITY	LEARNABILITY	AVAILABILITY	INTEGRITY	REUSABILITY	SCALABILITY	RISK IDENTIFICATION
FUNCTIONAL APPROPRIATENESS	CAPACITY		OPERABILITY	FAULT TOLERANCE	NON-REPUDIATION	ANALYSABILITY	INSTALLABILITY	FAIL SAFE
			USER ERROR PROTECTION	RECOVERABILITY	ACCOUNTABILITY	MODIFIABILITY	REPLACEABILITY	HAZARD WARNING
			USER ENGAGEMENT		AUTHENTICITY	TESTABILITY		SAFE INTEGRATION
			INCLUSIVITY		RESISTANCE			
			USER ASSISTANCE					
			SELF-DESCRIPTIVENESS					

iso25000.com

Software Quality Models and Standards

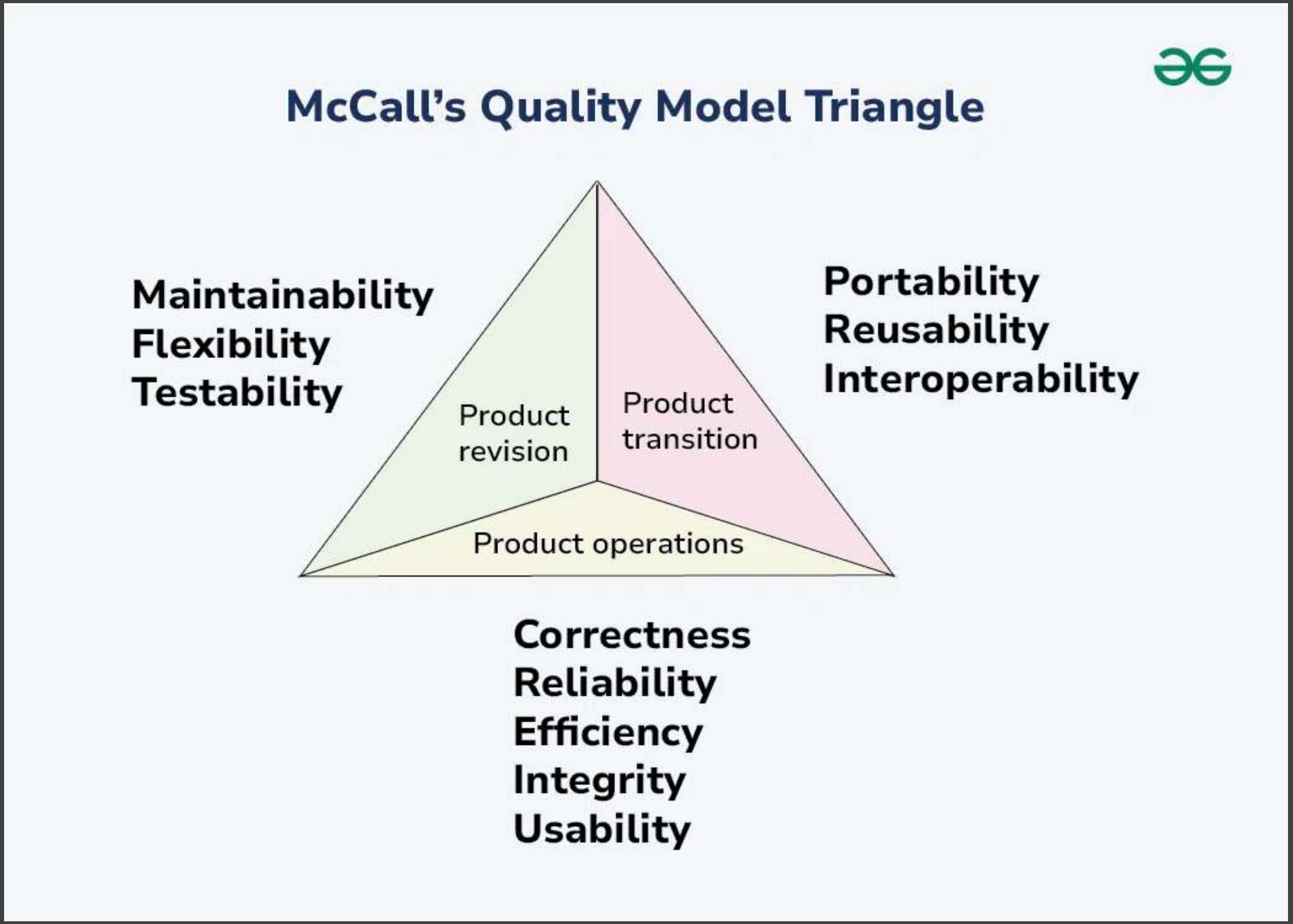
ISO/IEC 25010 Quality Model

This model is used to evaluate software by focusing on several quality areas (like performance, usability, and security).

Example: Imagine a mobile app that is fast, secure, easy to use, and rarely crashes. ISO/IEC 25010 helps ensure the app meets all these criteria.

Software Quality Models and Standards

CMMI (Capability Maturity Model Integration)



Software Quality Models and Standards

CMMI (Capability Maturity Model Integration)

1.Product Operation

- Product Operation includes five software quality factors, which are related to the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage, and correctness. These factors help in providing a better user experience.
- **Correctness:** The extent to which software meets its requirements specification.
- **Efficiency:** The number of hardware resources and code the software, needs to perform a function.
- **Integrity:** The extent to which the software can control an unauthorized person from accessing the data or software.
- **Reliability:** The extent to which software performs its intended functions without failure.
- **Usability:** The extent of effort required to learn, operate, and understand the functions of the software.

Software Quality Models and Standards

CMMI (Capability Maturity Model Integration)

2. Product Revision

- Product Revision includes three software quality factors, which are required for testing and maintenance of the software. They provide ease of maintenance, flexibility, and testing efforts to support the software to be functional according to the needs and requirements of the user in the future.
- **Maintainability:** The effort required to detect and correct an error during maintenance.
- **Flexibility**: The effort needed to improve an operational software program.
- **Testability**: The effort required to verify software to ensure that it meets the specified requirements.

Product Transition

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

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

Software Quality Models and Standards

Important Software Quality Standards

- **ISO 9001: Quality Management Standard**
  - ISO 9001 is a widely-used standard that provides guidelines for setting up a Quality Management System (QMS).
  - **Purpose**: Ensures consistent quality in products and services.
  - **Example**: A software company might use ISO 9001 to ensure they have a repeatable process for developing and testing software.
- **IEEE 730: Standard for Software Quality Assurance Plans**
  - IEEE 730 provides a structure for creating a Software Quality Assurance Plan (SQAP), which defines how quality will be ensured in a project.
  - **Example**: When creating an SQAP, a team outlines roles, responsibilities, and steps for reviewing code, testing, and documentation to maintain high quality.
- **ISO/IEC 12207: Software Lifecycle Processes**
  - ISO/IEC 12207 standardizes the steps in software development, from planning to maintenance.
  - **Purpose**: Defines best practices for each phase of the software lifecycle to ensure quality.
  - **Example**: This standard ensures that all teams follow a similar process, making it easier to manage, track, and improve development.

Software Quality Models and Standards

Why Use These Models and Standards?

- **Higher Quality**: Ensures software meets a high standard.
- **Lower Costs**: Reduces the risk of costly bugs later.
- **Customer Satisfaction**: Creates a better product for users.
- **Consistency**: Standardized processes lead to more predictable results.

Activity: Applying a Quality Model

- **Scenario**: You are building a mobile app for booking train tickets.
- **Objective**: Use the **ISO/IEC 25010** model to evaluate quality.
- **Steps**:
  - **Define Usability**: Ensure users can easily find and book trains.
  - **Reliability**: Minimize crashes, especially during booking.
  - **Security**: Ensure payment information is securely processed.
- **Outcome**: A checklist based on ISO/IEC 25010 characteristics to guide development and testing.

Summary

- Quality Models (like ISO/IEC 25010 and McCall’s) and Standards (like ISO 9001 and IEEE 730) help create high-quality software.
- Models focus on **what** makes software high quality, while standards provide a **how-to guide** for achieving that quality.