Course Material No. 9

SOFTWARE ENGINEERING 2

FE LARWA-HABLANIDA

Course Instructor

SOFTWARE QUALITY ASSURANCE 1

9

LEARNING OUTCOMES

At the end of the lesson, the learner will be able to:

- Explain the concept of software quality and its importance in software development.
- Identify and apply SQA activities, standards, and best practices.
- Utilize tools and techniques for quality assurance and testing.

RESOURCES NEEDED

For this lesson, you would need the following resources:

- PPT/Module
- Pencil and Paper

DISCUSSION:

INTRODUCTION TO SOFTWARE QUALITY ASSURANCE

Software Quality

Software Quality is the degree to which a software product meets specified requirements and satisfies user needs.

- Software Quality is the degree to which a software product meets:
  a. **Specified requirements** – it performs the functions described in the requirements document.
  b. **User expectations/needs** – it satisfies customers by being reliable, efficient, and easy to use.

➢ **Two Perspectives**

a. **Product Quality**
- The characteristics of the delivered software itself.
- Attributes like reliability, usability, performance, and security.

b. **Process Quality**
- The effectiveness of the processes used to develop the software.

Example:
- *Proper documentation, testing coverage, and adherence to coding standards*
- *How the software is built and maintained.*

➢ **Why Software Quality Matters**

a. **Customer satisfaction** – Meets expectations and builds trust.
b. **Cost savings** – Fewer defects mean lower maintenance costs.
c. **Business reputation** – Quality products attract and retain users.
d. **Compliance & safety** – Meets industry standards and avoids risks.
e. **Reduced failures** – Prevents disasters in safety-critical software (e.g., healthcare, banking).

➢ **Common Causes of Poor Software Quality**

a. Incomplete or ambiguous requirements
b. Lack of testing or rushed releases
c. Poor communication among team members
d. Ignoring coding and documentation standards

e. Insufficient reviews, audits, or SQA activities

⮱ Software Quality Assurance

Software Quality Assurance (SQA) is a planned and systematic set of activities carried out to ensure that software processes and the resulting products meet defined quality standards and requirements.

- SQA makes sure the software is built correctly and that the right software is built.
- It is proactive — it focuses on preventing defects instead of just finding them later.

➢ Activities of SQA

SQA activities span the entire SDLC

a. **Quality Planning**
   - Define quality goals and metrics.
   - Develop the **Software Quality Assurance Plan (SQAP).**

b. **Standards and Procedures**
   - Ensure adherence to coding standards, documentation standards, e.g., IEEE, ISO).

c. **Reviews and Audits**
   - Requirement reviews
   - Design and code inspections
   - Process and product audits

d. **Testing and Verification**
   - Support systematic test planning and execution.

e. **Defect Management**
   - Track, analyze, and resolve defects (e.g., using Jira, Bugzilla).

f. **Metrics and Reporting**
   - Collect and analyze data to measure product and process quality.

g. **Continuous Process Improvement**
   - Identify weaknesses in processes and recommend improvements.

⮱ Benefits of SQA

a. **High-Quality Software:** Fewer defects and failures

b. **Cost-Effectiveness:** Fixing problems early reduces costs

c. **Customer Satisfaction:** Meets or exceeds user expectations

d. **Improved Team Productivity:** Standardized processes reduce rework

e. **Compliance and Safety:** Meets legal, industry, and safety requirements

Example:

A banking application project applies SQA by:

- *Performing requirement reviews to prevent misunderstandings.*
- *Enforcing coding standards for consistency.*
- *Using automated testing to check performance.*
- *Tracking all defects in a defect tracking tool.*
- *Conducting audits before deployment.*

Result: *Fewer production issues, happier customers, and lower maintenance costs.*

SOFTWARE QUALITY MODELS AND STANDARDS

In Software Quality Assurance, quality models define what "quality" means in measurable terms, while standards describe how to achieve and assess that quality.

- **Quality Models** → Define the attributes of a quality product.
- **Standards** → Provide guidelines and best practices to ensure those attributes are met.

➯ Software Quality Models

Quality Models help organizations evaluate and improve the quality of software by focusing on specific quality attributes.

1. **McCall's Quality Model (1977)**

   Software quality is a multidimensional concept — it's not just about "fewer defects," but also about how well the software meets user needs and performs under different conditions.

   - One of the earliest models that linked software quality factors to user needs.
   - **Strength:** Easy to understand; user-centric.
   - **Limitation:** Outdated for modern software (e.g., lacks security focus).

   ➢ **Categories of Quality Factors**

   a. **Product Operation** – How well the software operates:
      - *Correctness, Reliability, Efficiency, Integrity, Usability*

   b. **Product Revision** – How easy it is to modify:
      - *Maintainability, Flexibility, Testability*

   c. **Product Transition** – How easily it adapts to new environments:
      - *Portability, Reusability, Interoperability*

2. **Boehm's Quality Model (1978)**

   Software quality involves more than just having a program that runs without errors — it must also be maintainable, efficient, user-friendly, and adaptable to future changes.

   - It is one of the earliest hierarchical quality models, providing a structured way to understand and evaluate software quality attributes.
   - Focused on maintainability and ease of change.
   - Broke down quality into high-level characteristics (e.g., As-is utility, Maintainability, Portability).
   - Useful for understanding long-term software evolution.

3. **ISO/IEC 9126 (1991)**

   The model defines quality characteristics and sub-characteristics to evaluate both internal (code-level), external (run-time), and quality in use (user experience) attributes of software.

   - An international standard that defines six main quality characteristics:
     a. Functionality
     b. Reliability
     c. Usability
     d. Efficiency
     e. Maintainability
     f. Portability

4. **ISO/IEC 25010 (2011) – Modern Standard**

   The **ISO/IEC 25010** standard is part of the SQuaRE (Software Product Quality Requirements and Evaluation) series, which replaces ISO/IEC 9126.

   - The current global standard for software quality, improving upon ISO 9126.
   - Most widely used model for modern software products.
   - It defines **8 Product Quality Characteristics**:
     a. **Functional Suitability** – Meets stated requirements.
     b. **Reliability** – Performs consistently over time.
     c. **Usability** – Easy to learn and use.
     d. **Performance Efficiency** – Optimal use of resources.
     e. **Security** – Protects data and resists unauthorized access.

f. **Compatibility** – Works well with other systems.
g. **Maintainability** – Easy to modify and update.
h. **Portability** – Can be transferred to other platforms.

⇘ Software Quality Standards

Standards provide **frameworks, guidelines, and best practices** for ensuring and assessing quality in software development and maintenance.

1. **ISO Standards**
   a. **ISO/IEC 25010** – Defines product quality attributes.
   b. **ISO 9001** – Focuses on **quality management systems** at the organizational level.
   c. **ISO/IEC 12207** – Provides **software lifecycle process standards**.
   d. **ISO/IEC 27001** – For **information security management** (important for secure software).
2. **IEEE Standards**
   a. **IEEE 730** – Standard for **Software Quality Assurance Plans (SQAP)**.
   b. **IEEE 829 / 29119** – Standards for **software testing documentation**.
   c. **IEEE 1012** – Standard for **Software Verification and Validation (V&V)**.
3. **Capability Maturity Model Integration (CMMI)**
   - Developed by Carnegie Mellon SEI.
   - Focuses on process maturity to improve software development practices.

   ➢ **Levels of Maturity**
      a. **Initial** – Ad hoc processes
      b. **Managed** – Basic project management
      c. **Defined** – Standardized processes
      d. **Quantitatively Managed** – Measured and controlled
      e. **Optimizing** – Continuous process improvement
4. **Other Frameworks**
   a. **Six Sigma** – Reduces variation and defects.
   b. **TQM (Total Quality Management)** – Organization-wide quality approach.
   c. **ITIL** – Best practices for IT service management.

SQA ACTIVITIES ACROSS THE SOFTWARE LIFECYCLE

Software Quality Assurance (SQA) ensures that quality is built into the product at every stage of the Software Development Life Cycle (SDLC) — not just tested at the end.

- SQA is proactive, focusing on defect prevention rather than just defect detection.

⇘ SQA Activities in SLDC Phase

1. **Requirements Phase**
   - Ensure that the requirements are complete, clear, testable, and feasible.

   ➢ **SQA Activities**

   a. Requirements reviews and inspections
   b. Traceability matrices (linking requirements to design and test cases)
   c. Identification of ambiguous or conflicting requirements
   d. Verification that requirements meet standards (IEEE 830, ISO/IEC 29148)

   ➢ **Common Errors Prevented**
   - Incomplete, inconsistent, or unclear requirements that often lead to rework.

2. **Design Phase**
   - Assure the software architecture and design will meet the specified requirements.

   ➢ **SQA Activities**

   a. Design reviews (high-level and detailed design)
   b. Checking for modularity, scalability, security, and maintainability
   c. Conformance to design standards and best practices
   d. Risk analysis (e.g., performance bottlenecks, security gaps)

   ➢ **Common Errors Prevented**
   - Poor architecture, weak interfaces, and performance issues.

3. **Implementation / Coding Phase**
   - Ensure the code is correct, maintainable, and adheres to coding standards.

   ➢ **SQA Activities**

   a. Code reviews/peer reviews
   b. Static code analysis (checking for syntax, security, and style issues)
   c. Unit testing to verify individual components
   d. Enforcing version control and change management

e. Ensuring adherence to coding standards (e.g., naming conventions, security guidelines)

- **Common Errors Prevented**
  - Logic errors, security vulnerabilities, and non-standard code.

4. **Testing Phase**
   - Ensure the integrated software works as intended and meets user expectations.

- **SQA Activities**
  a. Development of a Test Plan (IEEE 829 standard)
  b. Integration testing, System testing, and Acceptance testing
  c. Test case reviews and traceability to requirements
  d. Defect reporting and tracking (Defect Life Cycle)
  e. Ensuring use of test automation tools where appropriate

- **Common Errors Prevented**
  - Missed defects, incomplete coverage, performance, and usability issues.

5. **Deployment / Delivery Phase**
   - Ensure the software is correctly released to the production environment.

- **SQA Activities:**
  a. Release readiness reviews (checking documentation, training materials, licenses)
  b. Validation of installation procedures
  c. Configuration management to ensure the correct versions are released
  d. Post-deployment verification (sanity testing, smoke testing)

- **Common Errors Prevented**
  - Deployment failures, mismatched versions, incomplete release documentation.

6. **Maintenance Phase**
   - Maintain quality during updates, patches, and enhancements after release.

- **SQA Activities**
  a. Regression testing after changes
  b. Impact analysis for bug fixes and updates
  c. Monitoring defect reports and customer feedback
  d. Ensuring updated documentation remains accurate

e. Continuous process improvement based on lessons learned

➢ **Common Errors Prevented:**

- Breaking existing features, introducing new defects.

⇘ Cross-Cutting SQA Activities

Some SQA practices occur throughout the entire SDLC:

a. Process audits and compliance checks (e.g., ISO 9001, CMMI)
b. Metrics collection and analysis (e.g., defect density, test coverage)
c. Risk management and mitigation
d. Training and awareness programs for development teams
e. Configuration management and version control
f. Continuous improvement initiatives

⇘ Benefits of SQA Across the Life Cycle

a. Early defect detection → reduces rework costs
b. Improves customer satisfaction with higher-quality products
c. Ensures compliance with standards and regulations
d. Enhances maintainability and reliability of software
e. Builds trust in the development process

SQA TOOLS AND TECHNIQUES

Software Quality Assurance (SQA) tools and techniques are used to plan, control, measure, and improve software quality throughout the Software Development Life Cycle (SDLC).

⇘ Categories of SQA Tools

SQA tools can be grouped according to their purpose during the SDLC:

1. **Project Management and Planning Tools**
   - Used to plan and monitor SQA activities, schedules, and resources.
   - **Helps With:** Project scheduling, milestone tracking, risk management.

   Examples:
   - **Jira, Trello, Asana** – *for project tracking and task assignments*
   - **Microsoft Project** – *for scheduling and resource allocation*
2. **Requirement Management Tools**
   - Ensure that requirements are well-documented, traceable, and testable.

- **Helps With:** Requirement traceability matrices, change tracking.

Examples:
  - *IBM DOORS, Jama Software, Helix RM*
  - *Jira (with plugins), ReqView*

3. **Design and Modeling Tools**
   - Support software architecture, modeling, and validation of design.
   - **Helps with:** Validating design consistency, scalability, and security.

   Examples:
     - *Enterprise Architect, Lucid chart, Visual Paradigm (UML diagrams)*
     - *MATLAB/Simulink (for simulations)*

4. **Static Analysis and Code Quality Tools**
   - Analyze code without executing it to find errors, vulnerabilities, and enforce coding standards.
   - **Helps With:** Early detection of bugs, security gaps, and maintainability issues.

   Examples:
     - *SonarQube, Check style, PMD, ESLint, Pylint*
     - *Fortify SCA (security analysis)*

5. **Testing Tools**
   - Automate or assist test planning, execution, and reporting.
   - **Helps With:** Faster and consistent test execution, better coverage, and early defect detection.

   a. **Test Management Tools**
   - Jira Xray, TestRail, Zephyr

   b. **Automated Functional Testing Tools**
   - Selenium, Cypress, Playwright (web apps)
   - Appium (mobile apps), JUnit, NUnit, PyTest

   c. **Performance & Load Testing Tools**
   - JMeter, LoadRunner, Gatling

   d. **Security Testing Tools**
   - OWASP ZAP, Burp Suite

6. **Defect Tracking and Configuration Management Tools**
   - Track and manage defects, version control, and changes.
   - **Helps With:** Organizing defect reports, maintaining software integrity.

   Examples:
     - *Bugzilla, MantisBT, Redmine – for defect tracking*

- *Git, GitHub, GitLab, Bitbucket – for version control*
- *Jenkins – for build automation*

7. **Continuous Integration / Continuous Delivery (CI/CD) Tools**
   - Automate builds, testing, and deployment, reducing human error.
   - **Helps With**: Ensuring consistent releases and rapid feedback on changes.

   Examples:
   - *Jenkins, Travis CI, CircleCI, GitHub Actions*

8. **Measurement, Metrics, and Reporting Tools**
   - Provide quantitative insights into product and process quality.
   - **Helps With:** Tracking defect density, test coverage, code complexity, productivity.

   Examples:
   - *SonarQube dashboards (code metrics)*
   - *Grafana, Kibana (visualization)*
   - *Power BI, Tableau (reporting)*

➯ COMMON SQA TECHNIQUES

SQA techniques are methodologies and practices used alongside tools to ensure quality.

1. **Reviews and Inspections**
   - **Requirements Reviews** – ensure clarity and completeness
   - **Design Reviews** – ensure architecture correctness
   - **Code Reviews / Peer Reviews** – catch logic errors early
   - **Formal Inspections** – structured meetings to detect defects
   - **Benefit:** Early defect detection before coding or testing.

2. **Audits**
   - Conducted to check **compliance** with standards, policies, and processes.

   ➢ **Types**

   a. **Process Audit**
      - Checks adherence to SDLC procedures
      - Focus on the development and maintenance processes used to create the software.
      - Ensure that teams follow documented procedures, standards, and best practices.

      Example:

- *Verifying that a project team is following the Agile Scrum process as defined in the company's development policy.*

b. **Product Audit**

- Ensures product meets specifications
- Focus on the **software product itself** (code, documentation, test results).
- Ensures that the **final deliverable meets specifications and requirements.**

Example:

- *Reviewing a mobile app's UI, functionality, and test results to ensure it meets the client's requirements before release.*

3. **Testing Techniques**

a. **Black-box Testing:** Focuses on inputs/outputs (functional testing)

b. **White-box Testing:** Focuses on internal code logic

c. **Grey-box Testing:** Combines both approaches

d. **Regression Testing:** Ensures new changes don't break existing features

4. **Metrics and Measurement**
   - Collect and analyze quantitative data to assess quality and process efficiency:
     - Defect density
     - Mean time to failure (MTTF)
     - Test coverage percentage
     - Cyclomatic complexity (code complexity)
5. **Risk Management Techniques**
   - Risk identification and analysis (e.g., Failure Mode and Effect Analysis – FMEA)
   - Prioritize risks based on severity and probability
   - Plan mitigation strategies
6. **Configuration Management**
   - Maintains integrity and traceability of software versions, builds, and documents.
   - Often implemented using Git, Subversion, Mercurial.

REFERENCES

- Pressman, Roger S, Software Engineering: A Practitioner's Approach, 9th Edition, Published by Mc Graw-Hill (2019)
- Sommerville, Ian, Software Engineering 10th Edition, Published by Pearson Education, Inc (2016)
- Bass, Len, Clements, Paul, & Kazman, Rick., Software Architecture in Practice (3rd Edition). Addison-Wesley, 2012.