Object Oriented Software Engineering

MCA Semester III

Department of Computer Science

Gujarat University

Software Quality and Metrics

- **Definition::**
- Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software. It performs all functions as laid out in the SRS document.
- Example: A mobile banking app that is secure, user-friendly, and performs transactions accurately.

Software Quality Attributes

Software Quality Attributes

- **Functionality:** The ability of the software to perform its intended functions.
- Example: An e-commerce website that allows users to browse products, add them to a cart, and complete purchases.
- **Reliability:** The consistency of the software's performance over time.
- Example: A weather forecasting app that provides accurate weather updates without crashing.
- **Usability:** The ease with which users can learn and use the software.
- Example: A photo editing app with an intuitive interface that allows users to edit photos easily.

Software Quality Attributes

- Efficiency: The software's ability to perform its functions with optimal use of resources.
- Example: A video streaming service that loads videos quickly without buffering.
- Maintainability: The ease with which the software can be modified to fix defects or add new features.
- Example: A content management system that allows administrators to update website content without needing extensive technical knowledge.

Software Quality Attributes

- Portability: The ability of the software to run on different hardware or operating systems.
- Example: A game that can be played on both Android and iOS devices.

Software Quality Models

- Software quality models are frameworks used to evaluate and ensure the quality of software products.
- McCall's Quality Model: Focuses on factors like correctness, reliability, and efficiency.
- Example: Evaluating a payroll system based on its accuracy in calculating salaries and its uptime.
- Boehm's Quality Model: Emphasizes characteristics such as maintainability, usability, and portability.
- Example: Assessing a project management tool for its ease of use and ability to run on various platforms.

Software Quality Models

- ISO/IEC 9126: A standard for software quality that includes six quality characteristics.
- Example: Using ISO/IEC 9126 to evaluate a healthcare management system for functionality, reliability, and usability.

McCall's Quality Model

McCall's Quality Model, introduced by Jim McCall in 1977, is a framework for evaluating software quality. It categorizes software quality into three main areas, each with specific factors:

- **Product Operation Factors:**
- Correctness: How well the software meets its specifications and user needs.
- Reliability: The software's ability to perform its required functions under stated conditions for a specified period.

McCall's Quality Model

- Efficiency: The software's ability to use system resources effectively.
- Integrity: The protection of the software from unauthorized access and data breaches.
- Usability: The ease with which users can learn and use the software.

McCall's Quality Model

- **Product Revision Factors:**
- Maintainability: The ease with which the software can be modified to correct faults, improve performance, or adapt to a changed environment.
- Flexibility: The effort required to modify the software to meet new requirements.
- Testability: The ease with which the software can be tested to ensure it performs correctly.

McCall's Quality Model

- **Product Transition Factors:**
- Portability: The ease with which the software can be transferred from one environment to another.
- Reusability: The extent to which parts of the software can be used in other applications.
- Interoperability: The ability of the software to interact with other systems

Boehm’s Quality Model

- In 1978, B.W. Boehm introduced his software quality model. The model represents a hierarchical quality model similar to the McCall Quality Model to define software quality using a predefined set of attributes and metrics, each of which contributes to the overall quality of software.

Boehm's Quality Model

- **Primary Uses of Boehm's Model:**
- As is the utility: The extent to which, we can use software as-is.
- Maintainability: Effort required to detect and fix an error during maintenance.
- Portability: Effort required to change the software to fit in a new environment.

Boehm's Quality Model

- **Quality Factors Associated with Boehm's Model**

1) Portability: Effort required to change the software to fit in a new environment.
2) Reliability: The extent to which software performs according to requirements.
3) Efficiency: Amount of hardware resources and code required to execute a function.

ISO/IEC 9126

- ISO 9126 is an international standard for the evaluation of software. The standard is divided into four parts which addresses, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics.

ISO/IEC 9126

- The ISO 9126–1 software quality model identifies 6 main quality characteristics, namely:
- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

ISO/IEC 9126

- Functionality
- Functionality is the essential purpose of any product or service.
- e.g. an ATM machine, then the more complicated it becomes to define it's functionality. For software a list of functions can be specified, i.e. a sales order processing systems should be able to record customer information so that it can be used to reference a sales order. A sales order system should also provide the following functions:
- Record sales order product, price and quantity.
- Calculate total price.
- Calculate appropriate sales tax.
- Calculate date available to ship, based on inventory.

ISO/IEC 9126

- Reliability
- Once a software system is functioning, as specified, and delivered the reliability characteristic defines the capability of the system to maintain its service provision under defined conditions for defined periods of time.
- For example if the network goes down for 20 seconds then comes back the system should be able to recover and continue functioning.

ISO/IEC 9126

- Usability
- Usability only exists with regard to functionality and refers to the ease of use for a given function.
- For example a function of an ATM machine is to dispense cash as requested. Placing common amounts on the screen for selection, i.e. $20.00, $40.00, $100.00 etc

ISO/IEC 9126

- Maintainability
- The ability to identify and fix a fault within a software component is what the maintainability characteristic addresses.
- Portability
- This characteristic refers to how well the software can adopt to changes in its environment or with its requirements.

Measurement Basics

- In Software Engineering, Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.
- Code Quality
- Reliability
- Performance
- Usability
- Correctness

Measurement Basics

- Maintainability
- Integrity
- Security

Measurement Basics

- Code Quality – Code quality metrics measure the quality of code used for software project development.
- Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development.
- In code quality, both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, and maintainability, etc are measured.

Measurement Basics

- Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).
- Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

Measurement Basics

- Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.
- Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.

Measurement Basics

- Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include the time required to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.
- Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyberattacks.

Measurement Basics

- Security – Security metrics measure how secure the software is. In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

- Example: Tracking the number of bugs found in a software release.

Analysis of Metric Data

- Analyzing metric data in the context of software engineering can help improve code quality, team productivity, and overall project success. Here's a tailored approach:
- Define Clear Objectives: Identify what you want to achieve, such as reducing bug rates, improving code quality, or enhancing team efficiency.

Analysis of Metric Data

- Select Relevant Metrics: Choose metrics that align with your objectives. Common software engineering metrics include:
- Code Quality Metrics: Cyclomatic complexity, code coverage, and technical debt.
- Productivity Metrics: Velocity, sprint burndown, and lead time.
- Performance Metrics: Response time, throughput, and error rates.
- Customer Satisfaction Metrics: User feedback, Net Promoter Score (NPS), and feature adoption rates.

Analysis of Metric Data

- **Gather Accurate Data:** Use tools like version control systems (e.g., Git), continuous integration/continuous deployment (CI/CD) pipelines, and project management tools (e.g., Jira) to collect data.
- **Analyze the Data:** Apply statistical methods and data visualization techniques to understand trends and patterns. For example:
- **Trend Analysis:** Track metrics over time to identify improvements or regressions.
- **Correlation Analysis:** Determine relationships between different metrics, such as the impact of code complexity on bug rates.

Analysis of Metric Data

- Draw Insights and Take Action: Interpret the results to make informed decisions. For instance, if high complexity correlates with increased bug rates, you might prioritize refactoring efforts.
- Monitor and Refine: Continuously monitor the metrics to assess the impact of your actions. Use feedback loops to refine your approach and ensure continuous improvement.

Metrics for Measuring Size and Structure

- A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a characteristic of a software product or the software process.
- Software Measurement Principles
- The software measurement process can be characterized by five activities-
- Formulation: The derivation of software measures and metrics appropriate for the representation of the software that is being considered.

Metrics for Measuring Size and Structure

- Collection: The mechanism used to accumulate data required to derive the formulated metrics.
- Analysis: The computation of metrics and the application of mathematical tools.
- Interpretation: The evaluation of metrics results in insight into the quality of the representation.
- Feedback: Recommendation derived from the interpretation of product metrics transmitted to the software team.

Metrics for Measuring Size and Structure

Need for Software Measurement

- Create the quality of the current product or process.
- Anticipate future qualities of the product or process.
- Enhance the quality of a product or process.
- Regulate the state of the project concerning budget and schedule.
- Enable data-driven decision-making in project planning and control.
- Identify bottlenecks and areas for improvement to drive process improvement activities.

Metrics for Measuring Size and Structure

Classification of Software Measurement

- There are 2 types of software measurement:
- Direct Measurement: In direct measurement, the product, process, or thing is measured directly using a standard scale.
- Indirect Measurement: In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

Measuring Software Quality

Measuring software quality involves using various metrics to assess different aspects of the software.

- Code Quality Metrics include measures like code coverage, which indicates the percentage of code executed during testing, and code churn, which tracks the amount of code changed over time.
- Reliability Metrics measure the software's ability to perform without failure, such as mean time between failures (MTBF) and mean time to repair (MTTR).
- For example, a high MTBF indicates that the software is reliable and experiences fewer failures.

Measuring Software Quality

Performance Metrics assess how well the software performs under different conditions, such as response time, throughput, and resource utilization.

- Usability Metrics evaluate how easy and intuitive the software is for users, including measures like task completion rate and user satisfaction.
- Maintainability Metrics measure the ease with which the software can be modified, such as the number of defects fixed per release or the time taken to implement changes.
- By using these metrics, organizations can gain a comprehensive understanding of their software's quality and identify areas for improvement.

Object-Oriented Metrics

Object-oriented metrics are used to measure the quality and complexity of object-oriented software.

- Coupling measures the degree of interdependence between classes, with lower coupling indicating better modularity and easier maintenance.
- For example, a class that relies heavily on other classes has high coupling, making it harder to change without affecting other parts of the system.
- Cohesion measures how closely related the responsibilities of a single class are, with higher cohesion indicating better design.
- A class with high cohesion performs a single task or a group of related tasks, making it easier to understand and maintain.

Object-Oriented Metrics

Inheritance metrics assess the use of inheritance in the software, such as the depth of inheritance tree (DIT) and the number of children (NOC).

- For instance, a deep inheritance tree can indicate complex relationships that are harder to understand and maintain.
- Polymorphism metrics measure the use of polymorphic methods, which allow objects to be treated as instances of their parent class.
- By using these metrics, developers can assess the quality of their object-oriented design and identify areas for improvement.

Cost Impact of Software Defects

The cost impact of software defects varies depending on the phase in which they are detected and fixed.

- Defects found during the requirements phase are generally cheaper to fix, as they involve changes to documentation rather than code.
- For example, clarifying a requirement early on can prevent costly rework later.
- Defects found during the design phase are more expensive, as they may require changes to the software architecture.
- Defects found during the coding phase are even costlier, as they involve modifying the code and retesting the software.

Cost Impact of Software Defects

Defect amplification refers to the phenomenon where defects introduced in earlier phases become more costly to fix in later phases.

- For instance, a requirement defect that goes undetected until the testing phase can result in significant rework and delays.
- Defect removal involves identifying and fixing defects as early as possible to minimize their impact.
- Techniques like code reviews, static analysis, and automated testing can help detect defects early and reduce their cost impact.

Elements of Software Quality Assurance (SQA)

Software Quality Assurance (SQA) involves a set of activities designed to ensure that software meets specified quality standards.

- SQA tasks include planning, monitoring, and controlling the software development process to ensure quality.
- For example, an SQA team might develop a quality plan that outlines the standards, procedures, and tools to be used.
- SQA goals include preventing defects, ensuring compliance with standards, and improving the software development process.
- Metrics like defect density and customer satisfaction can be used to measure progress toward these goals.

Elements of Software Quality Assurance (SQA)

SQA activities include reviews, inspections, and testing. Reviews involve examining documents and code to identify defects, while inspections are more formal and involve a structured process.

- Testing involves executing the software to identify defects and ensure it meets requirements.
- By performing these tasks and activities, SQA helps ensure that software is of high quality and meets the needs of its users and stakeholders.

Formal Approaches to SQA

Formal approaches to SQA involve using structured methods to ensure software quality. Inspections are a formal review process where a team examines software artifacts to identify defects.

- For example, a code inspection might involve a team of developers reviewing a piece of code line by line to identify issues. Walkthroughs are less formal and involve the author of the software artifact leading a review session to explain the work and gather feedback.
- Formal methods involve using mathematical techniques to specify, develop, and verify software. For instance, formal specification languages can be used to describe software behavior precisely, and formal verification techniques can prove that the software meets its specifications.

Statistical Software Quality Assurance

Statistical Software Quality Assurance (SSQA) involves using statistical methods to monitor and control the quality of software processes and products.

- Statistical Process Control (SPC) is a key technique in SSQA, which uses control charts to track process performance over time.
- For example, a control chart might be used to monitor the number of defects found in each software build.
- If the number of defects exceeds a certain threshold, it indicates that the process is out of control and corrective actions are needed.

Statistical Software Quality Assurance

Control charts help identify trends and variations in the process, allowing teams to take proactive measures to improve quality.

- Other statistical methods used in SSQA include hypothesis testing, regression analysis, and design of experiments.
- By applying these techniques, organizations can gain insights into their software processes, identify areas for improvement, and ensure that their software meets quality standards.

Software Reliability

Software reliability refers to the probability that software will function correctly under specified conditions for a given period.

- Reliability metrics include measures like Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR).
- MTBF is the average time between failures, indicating how reliable the software is.
- For example, a high MTBF means that the software experiences fewer failures and is more reliable. MTTR is the average time taken to repair the software after a failure, with a lower MTTR indicating quicker recovery.

Software Reliability

Reliability models like the Weibull model and the exponential distribution model are used to predict software reliability and plan maintenance activities.

- By measuring and analyzing software reliability, organizations can identify potential issues, improve their software's robustness, and ensure that it meets user expectations.

ISO 9000 Quality Standards

The ISO 9000 family of quality management standards provides guidelines for ensuring that organizations meet customer and regulatory requirements.

- ISO 9001 is the most well-known standard in this family, focusing on quality management systems (QMS).
- It outlines requirements for processes, documentation, and continuous improvement.
- For example, an organization might implement ISO 9001 to ensure that its software development processes are well-documented, consistent, and continuously improving.

ISO 9000 Quality Standards

ISO 9000 standards are widely recognized and can help organizations demonstrate their commitment to quality.

- By achieving ISO 9001 certification, organizations can gain a competitive advantage, improve customer satisfaction, and enhance their overall quality management practices.

SQA Plan Topics

An SQA plan outlines the activities, roles, and responsibilities for ensuring software quality.

- Purpose and Scope define the objectives and boundaries of the SQA plan, such as ensuring compliance with quality standards and improving software reliability. SQA Activities include tasks like reviews, inspections, testing, and audits.
- For example, the plan might specify that code reviews will be conducted for all critical modules.

SQA Plan Topics

Roles and Responsibilities outline who is responsible for each SQA activity, such as the SQA team, developers, and project managers.

- Tools and Techniques describe the tools and methods used to support SQA activities, like automated testing tools and static analysis tools.
- By developing a comprehensive SQA plan, organizations can ensure that all aspects of software quality are addressed and that their software meets the required standards.

Assignment -4

- Explain the importance of software quality attributes such as functionality, reliability, usability, efficiency, maintainability, and portability. Provide examples of how each attribute can impact the user experience.
- Compare and contrast McCall's Quality Model, Boehm's Quality Model, and ISO/IEC 9126. Discuss how each model can be applied in a real-world software development project.
- Define software metrics and explain the difference between product, process, and project metrics. Provide examples of each type of metric and discuss their significance in software development.
- Describe the process of collecting and analyzing metric data in software engineering. Discuss the importance of data visualization techniques such as control charts in understanding metric data.
- Explain the concepts of Lines of Code (LOC), Function Points, and Cyclomatic Complexity. Discuss the advantages and limitations of each metric in measuring software size and structure.
- Identify and explain various metrics used to measure software quality, including code quality metrics, reliability metrics, performance metrics, usability metrics, and maintainability metrics. Provide examples of how these metrics can be applied.
- Discuss the importance of object-oriented metrics such as coupling, cohesion, inheritance, and polymorphism. Provide examples of how these metrics can be used to assess the quality of object-oriented software design.
- Analyze the cost impact of software defects at different stages of the software development lifecycle. Explain the concepts of defect amplification and defect removal, and discuss strategies to minimize the cost impact of defects.

Assignment -4

- Describe the key elements of Software Quality Assurance, including SQA tasks, goals, and metrics. Discuss the role of SQA in ensuring software quality and provide examples of SQA activities.
- Explain the formal approaches to Software Quality Assurance, such as inspections, walkthroughs, and formal methods. Discuss the benefits and challenges of using these approaches in software development.
- Describe the principles of Statistical Software Quality Assurance (SSQA) and explain how statistical methods such as Statistical Process Control (SPC) and control charts are used to monitor and improve software quality.
- Define software reliability and explain the importance of reliability metrics such as Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR). Discuss how these metrics can be used to assess and improve software reliability.
- Provide an overview of the ISO 9000 family of quality management standards, with a focus on ISO 9001. Discuss the relevance of these standards to software quality and the benefits of achieving ISO 9001 certification.
- Outline the key topics that should be included in a Software Quality Assurance (SQA) plan. Discuss the purpose and scope of an SQA plan, and provide examples of SQA activities, roles, and responsibilities.