Victoria University
of Bangladesh

Final Assessment

Md Bakhtiar Chowdhury

ID: 2121210061

Department: CSE

Semester: Summer 2023

Batch: 21th

Course Title: Software Engineering

Course Code: CS1 321

Submitted To:

Umme Khadiza Tithi

Lecturer, Department of Computer Science & Engineering

Victoria University of Bangladesh

Submission Date: 10 October, 2023

Answer to the question no 1(a)

 # Define Software Engineering?

Answer: Software Engineering is a systematic and disciplined approach to designing, developing, testing, deploying, and maintaining software applications, systems, and solutions. It encompasses a set of principles, practices, methodologies, and tools that aim to ensure the creation of high-quality software that meets user needs, is reliable, scalable, and maintainable, and is completed within specified timeframes and budgets.

Key aspects of software engineering include:

- Requirements Engineering
- Design
- Implementation
- Testing
- Deployment
- Maintenance
- Documentation
- Project Management
- Quality Assurance
- Configuration Management
- Software Development Life Cycle (SDLC)

Answer to the question no 1(b)

 # Describe Waterfall model and Validation process ?

Answer:

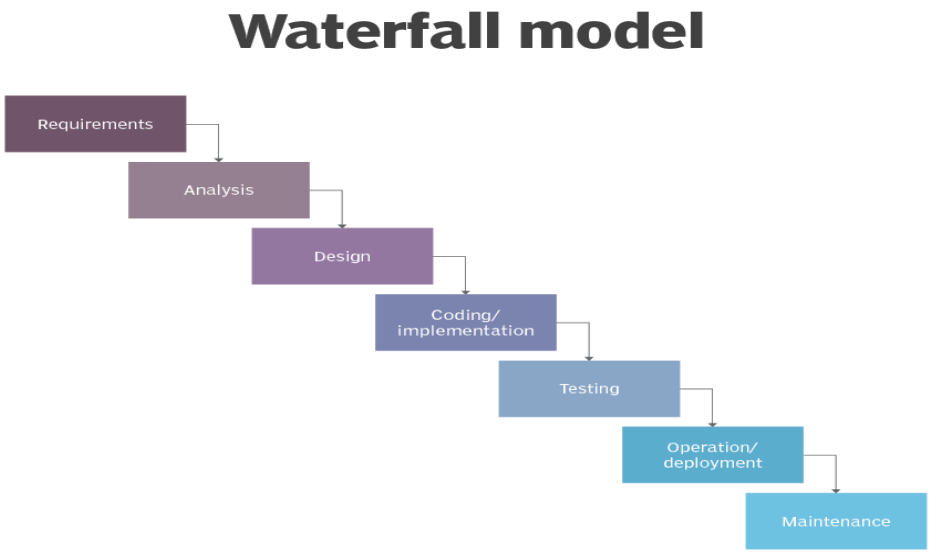
Waterfall Model:

The Waterfall model is a traditional and linear approach to software development. It consists of a sequential series of phases, where each phase must be completed before the next one begins. It is often used when the requirements of a project are well-understood and unlikely to change significantly throughout the development process. Here are the main phases of the Waterfall model:

1. **Requirements Gathering:** In this initial phase, the project team works closely with stakeholders to gather and document all project requirements. This phase aims to establish a clear understanding of what the software should do and how it should behave.
2. **System Design:** Once the requirements are finalized, the system architecture and design are planned. This phase involves creating detailed specifications for the software's structure, components, and interfaces.
3. **Implementation (Coding):** After the design phase, developers start coding based on the design specifications. This phase involves writing the actual code for the software system.
4. **Testing:** In the testing phase, the developed software is thoroughly tested to identify and rectify any defects or issues. Testing includes unit testing, integration testing, system testing, and user acceptance testing.
5. **Deployment (Installation):** Once the software is deemed ready and meets all requirements, it is deployed to the production environment. This phase involves installing the software and making it available to users.

6. **Maintenance:** After deployment, the software enters the maintenance phase, where it is continuously monitored, and any necessary updates, patches, or improvements are made.

The Waterfall model is known for its rigidity and lack of flexibility when changes are required after the project has started. It's most suitable for projects with well-defined and stable requirements.



Validation Process:

In software development, the validation process is a critical phase that ensures the software meets the specified requirements and functions correctly. Here are the key steps involved in the validation process:

1. **Requirement Validation**: Before development begins, the project team reviews and validates the gathered requirements to ensure they are clear, complete, and feasible. Any ambiguities or inconsistencies are addressed.
2. **Design Validation:** During the design phase, the design specifications are reviewed to verify that they align with the requirements. This validation ensures that the proposed system architecture and design will meet the intended goals.

3. **Code Validation**: In the coding phase, the code undergoes validation through various testing methods, including unit testing, integration testing, and system testing. This step identifies and rectifies any coding errors, bugs, or discrepancies.

4. **User Acceptance Testing (UAT):** This phase involves end-users or stakeholders testing the software in a real-world environment to validate that it meets their needs and expectations. Any issues or discrepancies are documented and addressed.

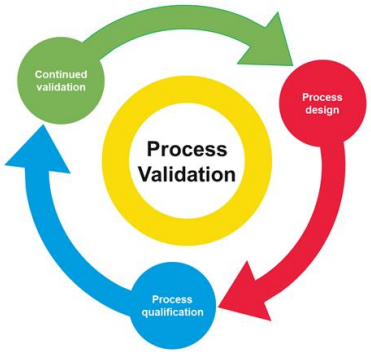
5. **Regression Testing:** After making changes or fixes to the software, regression testing is performed to ensure that the modifications do not introduce new defects or affect existing functionalities.

6. **Documentation Validation:** Validation also extends to the documentation of the software, including user manuals, technical guides, and system documentation. These documents should accurately reflect the software's behavior and usage.

7. **Compliance and Standards Validation**: Depending on the industry or domain, software may need to adhere to specific standards or regulations. Validation ensures that the software complies with these requirements.

8. **Performance and Scalability Testing**: For software systems that require high performance and scalability, validation includes testing for performance bottlenecks and ensuring the software can handle expected loads.

Validation is an ongoing process that continues throughout the software development life cycle. It is essential for delivering a reliable and high-quality software product that meets the needs of its users and stakeholders.



Answer to the question no 1(c)

 # Describe characteristics of SRS?

A Software Requirements Specification (SRS) is a critical document in software engineering that outlines the detailed description of what a software system should accomplish. An SRS serves as a contract between the stakeholders (clients, users, and developers) and provides a clear understanding of the software's functionalities and constraints. The characteristics of a well-written SRS include:

1. Clarity: The SRS should be clear and concise, using unambiguous language and terminology. It should avoid jargon or technical terms that might not be understood by all stakeholders.

2. Completeness: The SRS should be comprehensive and cover all aspects of the software's requirements, including functional and non-functional requirements, constraints, and assumptions. It should leave no room for ambiguity or missing information.

3. Correctness: All information in the SRS should be accurate and free from errors. Requirements should be verifiable, and there should be no contradictions or conflicts within the document.

4. Consistency: The SRS should maintain consistency in its terminology, definitions, and descriptions throughout the document. Inconsistencies can lead to misunderstandings and confusion.

5. Traceability: Each requirement in the SRS should be uniquely identified and traced back to its source, typically through a requirement ID or number. This traceability helps in managing changes and ensuring that all requirements are addressed.

6. Modifiability: The SRS should be designed to accommodate changes and updates to requirements as they evolve over time. It should describe the process for handling changes and their impact on the project.

7. Testability: Requirements in the SRS should be structured in a way that allows for easy testing and validation. This includes specifying acceptance criteria and conditions for each requirement.

8. Feasibility: The SRS should address the feasibility of implementing the specified requirements within the project's constraints, such as time, budget, and available technology.

9. Priority: It is essential to assign priorities to requirements, indicating which ones are critical for the software's core functionality and which ones are optional or can be deferred to future releases.

10. Accessibility: The SRS should be readily accessible to all relevant stakeholders, including developers, testers, project managers, and clients. It should be maintained in a format that is easily distributable and updatable.

11. Version Control: The SRS should be version-controlled to keep track of changes and revisions. This ensures that all stakeholders are working with the most up-to-date document.

12. Legal and Regulatory Compliance: If applicable, the SRS should address legal and regulatory requirements that the software must adhere to, ensuring that the software complies with industry standards and laws.

13. Conciseness: While being comprehensive, the SRS should avoid unnecessary verbosity and should focus on conveying essential information effectively.

14. Use Cases and Scenarios: In addition to requirements, the SRS may include use cases, scenarios, and user stories to illustrate how the software will be used and how different functionalities will interact.

15. Sign-off: The SRS should undergo formal sign-off and approval by all relevant stakeholders, indicating their agreement with the documented requirements. Sign-off signifies commitment to the specified scope of the project.

Creating a well-structured and well-documented SRS is crucial for the success of a software project. It serves as a foundation for development, testing, project management, and client communication, ensuring that all parties have a common understanding of what the software should achieve.

Answer to the question no 2(a)

 # Define Categories of Software Maintenance ?

Answer:

Software maintenance is a crucial phase in the software development life cycle that involves making changes and updates to a software system after it has been deployed. Maintenance activities are essential to ensure that the software remains functional, reliable, and up-to-date. There are several categories of software maintenance, each serving a specific purpose:

1. Corrective Maintenance:

- **Purpose:** Corrective maintenance involves fixing defects, errors, and issues identified in the software. These issues can be functional, performance-related, or security vulnerabilities.

- **Activities:** Developers diagnose and troubleshoot problems, identify the root causes of issues, and implement necessary fixes to restore the software's intended functionality.

2. Adaptive Maintenance:

- Purpose: Adaptive maintenance is focused on adapting the software to changes in the environment, such as changes in hardware, operating systems, or external interfaces. It ensures the software remains compatible and operational in evolving conditions.

- **Activities:** Developers modify the software to accommodate new hardware or software platforms, update libraries, and make necessary adjustments to maintain compatibility.

3. Perfective Maintenance:

- **Purpose:** Perfective maintenance aims to enhance the software's functionality, performance, and usability. It involves making improvements and optimizations to meet changing user needs or expectations.

- **Activities:** Developers add new features, optimize code for better performance, improve user interfaces, and enhance existing functionalities to make the software more efficient and user-friendly.

4. Preventive Maintenance:

- **Purpose:** Preventive maintenance is proactively performed to prevent potential issues and mitigate risks. It aims to identify and address vulnerabilities and weaknesses in the software before they lead to problems.

- **Activities:** Activities include code reviews, security audits, performance analysis, and proactive updates to prevent future issues or security breaches.

5. Emergency Maintenance:

- **Purpose:** Emergency maintenance is unplanned and occurs in response to critical issues that require immediate attention. It is often triggered by unexpected system failures or security breaches.

- **Activities:** Developers respond urgently to address critical failures, resolve security breaches, or mitigate risks to minimize downtime and data loss.

6. Routine Maintenance:

- Purpose: Routine maintenance involves regular, scheduled activities to ensure the software's health and reliability. It includes tasks like data backups, database optimization, and system monitoring.

- Activities: Developers and administrators perform routine tasks to keep the software running smoothly and to prevent potential issues from arising.

7. Obsolete Maintenance:

- Purpose: Obsolete maintenance deals with the retirement or decommissioning of software that is no longer in use or supported. It involves archiving data and ensuring a graceful exit for the software.

- Activities: Developers and administrators may assist in data migration, archival, or transitioning users to newer systems when older software becomes obsolete.

Understanding these categories of software maintenance helps organizations effectively manage their software assets, keep systems up-to-date, and ensure that software remains reliable and aligned with changing business needs and technology environments.

Answer to the question no 2(b)

 # What is the difference between Black box testing and white box testing?

Answer:

1. Black Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. Only the external design and structure are tested.
2. White Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested.

Differences between Black Box Testing vs White Box Testing:

Black Box Testing	**White Box Testing**
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
It is mostly done by software testers.	It is mostly done by software developers.

Black Box Testing	**White Box Testing**
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
It is a functional test of the software.	It is a structural test of the software.
This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.

Black Box Testing	**White Box Testing**
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: Search something on google by using keywords	Example: By input to check and verify loops
Black-box test design techniques- • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	White-box test design techniques- • Control flow testing • Data flow testing • Branch testing
Types of Black Box Testing: • Functional Testing • Non-functional testing • Regression Testing	Types of White Box Testing: • Path Testing • Loop Testing • Condition testing
It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

Answer to the question no 2(c)

 # Mention what are the types of documents in SQA ?

Software Quality Assurance (SQA) involves a set of processes, activities, and documents aimed at ensuring that software development and testing meet defined quality standards and objectives. Various types of documents play essential roles in SQA efforts. Here are some of the key types of documents used in SQA:

1. Software Quality Management Plan (SQMP): The SQMP outlines the overall approach and strategy for managing quality throughout the software development life cycle. It defines roles, responsibilities, and processes related to quality assurance.

2. Quality Policy and Objectives: These documents establish the organization's quality policy and objectives, emphasizing the commitment to delivering high-quality software.

3. Quality Standards and Guidelines: Quality standards and guidelines specify the criteria and expectations for software quality. They can include industry-specific standards (e.g., ISO 9001), coding standards, and best practices.

4. Process Documentation: Process documents describe the software development and testing processes, including workflow diagrams, procedures, and work instructions. Examples include software development life cycle (SDLC) documentation and test process documentation.

5. Quality Assurance Plans (QAP): QA plans outline the specific quality assurance activities, tasks, and schedules for a particular project. They may include details on testing strategies, test environments, and resource allocation.

6. Test Plans: Test plans detail the approach, scope, objectives, and resources for testing a specific software project. They also outline the test strategy, test cases, and schedules for testing activities.

7. Test Cases and Test Scripts: These documents specify the detailed steps, input data, expected results, and conditions for executing test cases and scripts during software testing.

8. Test Data: Test data documents include sets of data and scenarios used for testing purposes, including valid and invalid inputs, boundary conditions, and edge cases.

9. Defect Reports: When defects or issues are discovered during testing or quality assurance activities, defect reports document the details, severity, and status of each defect. These reports help prioritize and track defect resolution.

10. Traceability Matrix: A traceability matrix links requirements to test cases, ensuring that each requirement is tested and that all test cases have corresponding requirements. It helps maintain coverage and ensures that no requirements are missed.

11. Metrics and Measurement Plans: These documents define the key performance indicators (KPIs) and metrics used to assess software quality. Metrics can include defect density, code coverage, and test pass rates.

12. Audit Reports: If internal or external audits are conducted to evaluate the quality processes and practices, audit reports summarize the findings, recommendations, and actions taken.

13. Configuration Management Plan: This document outlines how software configurations are managed, including version control, release management, and change management procedures.

14. Review and Inspection Reports: Documents related to peer reviews and inspections, including meeting minutes, checklists, and summaries of findings.

15. Training Materials: Materials for training personnel in quality assurance processes and practices, including training manuals, presentations, and training plans.

16. SQA Records and Logs: Records and logs document activities related to quality assurance, such as test execution logs, defect tracking records, and change request records.

17. Release Documentation: Documentation for software releases, including release notes, installation guides, and user manuals.

18. SQA Metrics and Reports: Reports summarizing quality metrics, trends, and the overall status of software quality assurance activities.

These documents collectively help ensure that software development and testing processes are well-defined, well-executed, and aligned with quality goals, standards, and best practices. Effective SQA documentation is crucial for maintaining transparency, traceability, and accountability throughout the software development life cycle.

Answer to the question no 3(a)

 # What is Software project management?

Software project management is the discipline of planning, organizing, and overseeing the development, testing, and delivery of software products or systems. It encompasses a set of activities, processes, and techniques aimed at successfully completing a software project within defined scope, budget, and time constraints while meeting quality and performance objectives. In essence, it is the management of all activities related to developing software from inception to delivery, ensuring that the project is completed efficiently and effectively.

Answer to the question no 3(b)

 # Define project manager role and responsibilities?

Answer:

The Role of a Project Manager involves overseeing the entire project lifecycle, from initiation to closure. Project managers are responsible for ensuring that projects are completed successfully, meeting objectives, staying within scope, on time, and within budget. They play a crucial role in coordinating resources, managing teams, and communicating effectively with stakeholders. Here's an overview of the role and associated responsibilities:

Role of a Project Manager:

- Leadership: Providing guidance and direction to the project team, motivating them to achieve project goals.

- Coordination: Ensuring efficient allocation of resources, both human and material, to accomplish project tasks.

- Communication: Facilitating clear and effective communication among team members, stakeholders, and clients.

- Risk Management: Identifying potential risks and developing strategies to mitigate or manage them.

- Problem Solving: Addressing challenges and issues that arise during the project, making decisions to keep the project on track.

- Decision Making: Making informed decisions related to project scope, changes, and priorities.

- Stakeholder Management: Building and maintaining positive relationships with project stakeholders and managing their expectations.

- Planning: Creating a detailed project plan, including tasks, timelines, milestones, and resource requirements.

- Quality Assurance: Ensuring that project deliverables meet established quality standards**.**

- Budget Control: Monitoring project expenses and ensuring that the project stays within the allocated budget.

- Scope Management: Preventing scope creep and managing changes to project scope.

- Time Management: Monitoring project progress against the schedule and making adjustments as needed.

- Documentation: Keeping accurate records of project plans, decisions, and communications.

- Closure and Evaluation: Concluding the project, conducting evaluations, and documenting lessons learned for future projects.

Responsibilities of a Project Manager:

- Define project goals, objectives, and scope.

- Develop a comprehensive project plan.

- Identify and allocate necessary resources.

- Assemble and lead project teams.

- Monitor and manage project progress.

- Communicate project status to stakeholders.

- Mitigate risks and resolve issues.

- Make timely decisions to keep the project on track.

- Ensure quality and adherence to requirements.

- Manage project budgets and expenses.

- Handle changes and scope adjustments.

- Maintain stakeholder relationships.

- Ensure effective communication within the team.

- Prepare for project closure and evaluation.

- Document project processes and outcomes.

In summary, a project manager is responsible for overseeing all aspects of a project, from planning and execution to closure and evaluation. They ensure that projects are completed successfully while managing resources, risks, scope, and communication. Effective leadership, communication, problem-solving, and organizational skills are essential for a project manager to excel in their role.

Answer to the question no 3(c)

 # Define Iterative Model advantage and disadvantage?

Answer:

The Iterative Model is a software development methodology in which a project is divided into smaller, manageable cycles or iterations. Each iteration involves a complete software development cycle, including planning, requirements, design, coding, testing, and deployment. The primary advantage of the Iterative Model is that it allows for incremental and flexible development, but it also comes with some disadvantages. Here are the key advantages and disadvantages of the Iterative Model:

Advantages of the Iterative Model	**Disadvantages of the Iterative Model**
1. **Flexibility**: Allows adaptation to changing requirements and insights.	1. **Complexity**: Adds complexity to project management.
2. **Early Deliveries:** Provides working software in each iteration.	2. **Higher Initial Cost:** May result in higher initial development costs.
3. **Improved Risk Management:** Addresses risks iteratively.	3. **Potential Scope Creep:** Flexibility may lead to scope expansion.
4. **Enhanced Collaboration:** Encourages collaboration with stakeholders.	4. **Uncertainty in Duration:** Challenging to estimate project duration.
5. **Continuous Improvement:** Lessons learned lead to a refined product.	5. **Resource Intensive:** May require additional resources.
6. **Partial Functionality:** Provides partial functionality in early stages.	6. **Client Involvement:** Success depends on consistent client involvement.

In summary, the Iterative Model offers benefits such as flexibility, early deliveries, improved risk management, and enhanced collaboration. However, it also comes with challenges related to complexity, cost, scope management, and resource requirements.

The suitability of the Iterative Model depends on the project's specific requirements, the degree of uncertainty, and the willingness of stakeholders to engage in an iterative development process.

Answer to the question no 4(a)

 # What is Quality ? Define Importance of Quality?

Quality refers to the degree of excellence or superiority of a product, service, or process. It involves meeting or exceeding customer expectations and adhering to established standards and specifications. Quality can be assessed through various attributes such as reliability, performance, durability, consistency, and user satisfaction.

Importance of Quality:

Quality is of paramount importance in various aspects of business, manufacturing, and service industries for several reasons:

1. Customer Satisfaction: High-quality products and services lead to increased customer satisfaction. Satisfied customers are more likely to become repeat customers and brand advocates, which can boost sales and reputation.

2. Competitive Advantage: Maintaining high quality sets a business apart from competitors. It can lead to a competitive advantage by attracting customers who are willing to pay a premium for quality.

3. Reduced Costs: Quality control helps identify and rectify defects early in the production process, reducing the costs associated with rework, repairs, and warranty claims.

4. Enhanced Reputation: Consistently delivering quality products or services enhances a company's reputation and builds trust among customers, partners, and stakeholders.

5. Compliance and Standards: Quality often aligns with industry standards and regulatory requirements. Adhering to quality standards ensures compliance with legal and safety regulations.

6. Efficiency and Productivity: Quality processes streamline operations, reduce errors, and increase productivity. Employees can work more efficiently when they have confidence in the quality of their work.

7. Innovation and Continuous Improvement: A focus on quality encourages innovation and drives continuous improvement efforts. It promotes a culture of learning and adaptation to changing customer needs.

8. Risk Mitigation: High quality reduces the risk of product recalls, legal disputes, and customer dissatisfaction, which can have severe financial and reputational consequences.

9. Customer Loyalty: Quality products and services cultivate customer loyalty, reducing the need for aggressive marketing and customer retention efforts.

10. Market Expansion: A reputation for quality can open doors to new markets and partnerships, as businesses with strong quality management systems are seen as reliable and trustworthy.

11. Long-Term Viability: Quality-focused organizations tend to have long-term viability and sustainability, as they are better equipped to withstand market fluctuations and economic challenges.

12. Employee Morale: A commitment to quality can boost employee morale and job satisfaction. Employees are more engaged when they see their efforts contribute to a successful and respected organization.

In summary, quality is not just a desirable attribute but a fundamental driver of success in business and various industries. It contributes to customer satisfaction, cost reduction, competitiveness, reputation, and overall organizational effectiveness. As a result, organizations that prioritize and maintain high levels of quality are more likely to thrive and prosper in the long run.

Answer to the question no 4(b)

 # Describe SQA Activities?

Software Quality Assurance (SQA) activities are a set of systematic and planned actions performed throughout the software development life cycle to ensure that a software product meets specified quality standards and requirements. These activities are essential for building reliable, high-quality software that satisfies customer needs and minimizes defects. Here are the key SQA activities:

1. Quality Planning:

- Define the quality objectives and quality standards for the project.
- Develop a Quality Management Plan (QMP) that outlines how quality will be ensured throughout the project.

2. Process Definition and Compliance:

- Define and document software development processes, including coding standards, testing procedures, and configuration management practices.
- Ensure that development teams adhere to established processes and standards.

3. Requirements Analysis and Validation:

- Review and validate software requirements to ensure they are clear, complete, and testable.
- Verify that the software design and architecture align with the specified requirements.

4. Design and Code Reviews:

- Conduct design and code reviews to identify and address issues, defects, and non-compliance with coding standards.
- Ensure that design and code reviews are performed by peers or experts.

5. Testing and Test Planning:

- Develop test plans, test cases, and test scripts based on requirements.
- Execute various testing types, including unit testing, integration testing, system testing, and user acceptance testing.
- Monitor and report on testing progress and results.

6. Defect Tracking and Management:

- Identify and document defects and issues discovered during testing and reviews.
- Track the status of defects and manage them through resolution and validation.

7. Configuration Management:

- Establish and maintain version control for software artifacts.
- Manage changes, releases, and baselines to ensure consistency and traceability.

8. Quality Metrics and Reporting:

- Define and collect software quality metrics, such as defect density, code coverage, and test pass rates.
- Generate reports and dashboards to communicate quality status to stakeholders.

9. Audits and Assessments:

- Conduct internal and external audits and assessments to evaluate compliance with quality standards and processes.
- Identify areas for improvement and corrective actions.

10. Documentation and Documentation Reviews:

- Maintain comprehensive project documentation, including requirements, design documents, and user manuals.
- Review and validate documentation for accuracy and completeness.

11. Process Improvement:

- Continuously assess and improve software development processes and practices.
- Implement process improvements based on lessons learned and best practices.

12. Training and Competence Development:

- Provide training and skill development programs to ensure that team members are competent in quality-related activities and processes.

13. Risk Management:

- Identify, assess, and mitigate risks related to software quality and project success.
- Develop risk management plans and strategies.

14. Change Management:

- Manage changes to requirements, design, and code through a formal change control process.
- Ensure that changes are evaluated, approved, and properly communicated.

15. Customer Feedback and Satisfaction Surveys:

- Gather feedback from customers and end-users to assess satisfaction and identify areas for improvement.
- Incorporate customer feedback into the development process.

16. Post-Release Monitoring:

- Monitor the performance and reliability of the software in production environments.
- Address any issues or defects that arise post-release.

These SQA activities are essential for ensuring that software projects are well-managed, well-documented, and produce high-quality software that meets or exceeds customer expectations. They help minimize defects, enhance reliability, and contribute to overall project success.

>>>>>>END<<<<<<