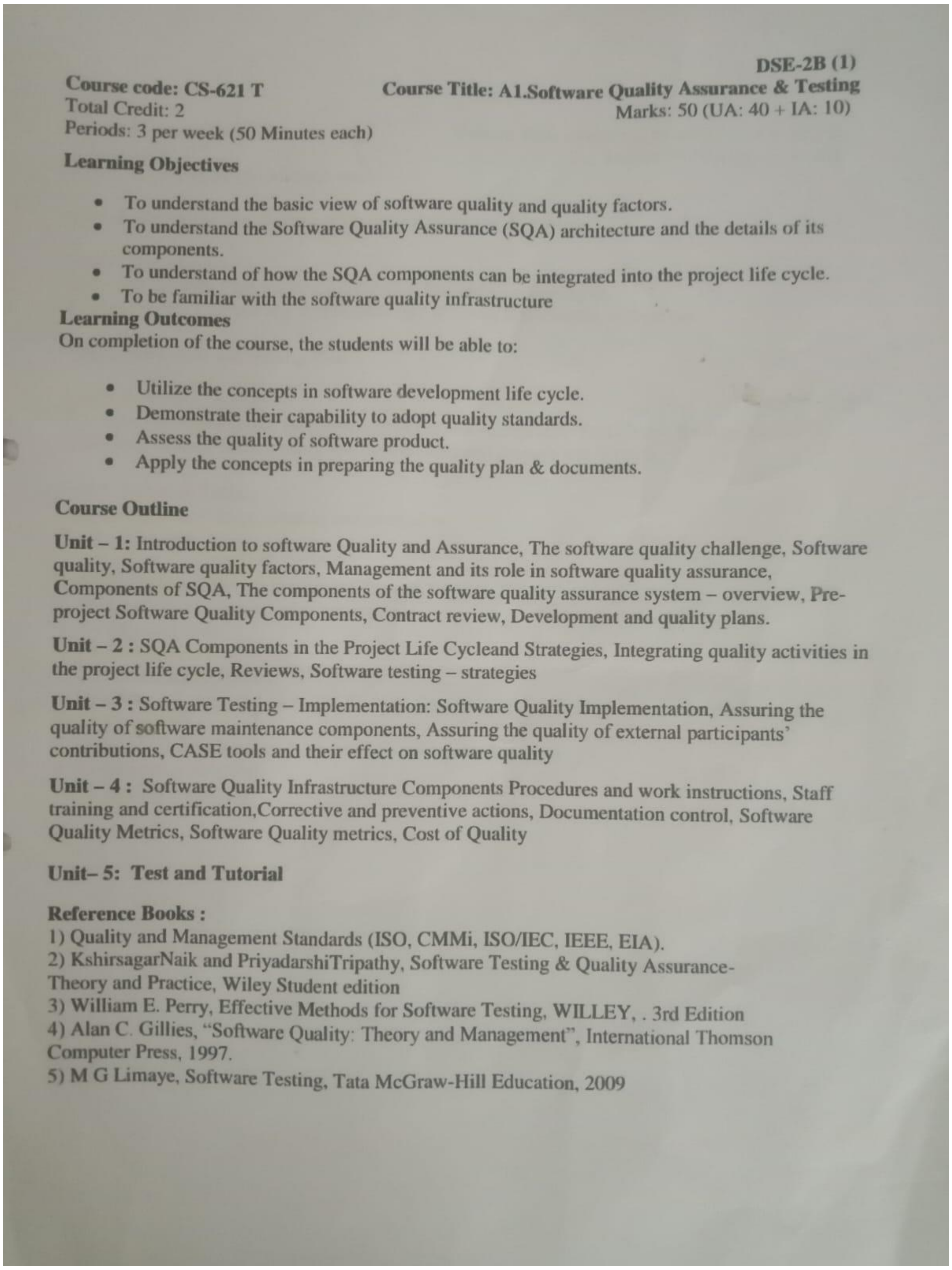


DSE-2B (1)

Course code: CS-621 T **Course Title: A1.Software Quality Assurance & Testing**
Total Credit: 2 Marks: 50 (UA: 40 + IA: 10)
Periods: 3 per week (50 Minutes each)

Learning Objectives

- To understand the basic view of software quality and quality factors.
- To understand the Software Quality Assurance (SQA) architecture and the details of its components.
- To understand of how the SQA components can be integrated into the project life cycle.
- To be familiar with the software quality infrastructure

Learning Outcomes

On completion of the course, the students will be able to:

- Utilize the concepts in software development life cycle.
- Demonstrate their capability to adopt quality standards.
- Assess the quality of software product.
- Apply the concepts in preparing the quality plan & documents.

Course Outline

Unit – 1: Introduction to software Quality and Assurance, The software quality challenge, Software quality, Software quality factors, Management and its role in software quality assurance, Components of SQA, The components of the software quality assurance system – overview, Pre-project Software Quality Components, Contract review, Development and quality plans.

Unit – 2 : SQA Components in the Project Life Cycleand Strategies, Integrating quality activities in the project life cycle, Reviews, Software testing – strategies

Unit – 3 : Software Testing – Implementation: Software Quality Implementation, Assuring the quality of software maintenance components, Assuring the quality of external participants' contributions, CASE tools and their effect on software quality

Unit – 4 : Software Quality Infrastructure Components Procedures and work instructions, Staff training and certification,Corrective and preventive actions, Documentation control, Software Quality Metrics, Software Quality metrics, Cost of Quality

Unit– 5: Test and Tutorial

Reference Books :

1) Quality and Management Standards (ISO, CMMi, ISO/IEC, IEEE, EIA).
2) KshirsagarNaik and PriyadarshiTripathy, Software Testing & Quality Assurance-Theory and Practice, Wiley Student edition
3) William E. Perry, Effective Methods for Software Testing, WILLEY, . 3rd Edition
4) Alan C. Gillies, "Software Quality: Theory and Management", International Thomson Computer Press, 1997.
5) M G Limaye, Software Testing, Tata McGraw-Hill Education, 2009

Software Quality Assurance & Testing

Software Quality Assurance - Software Engineering

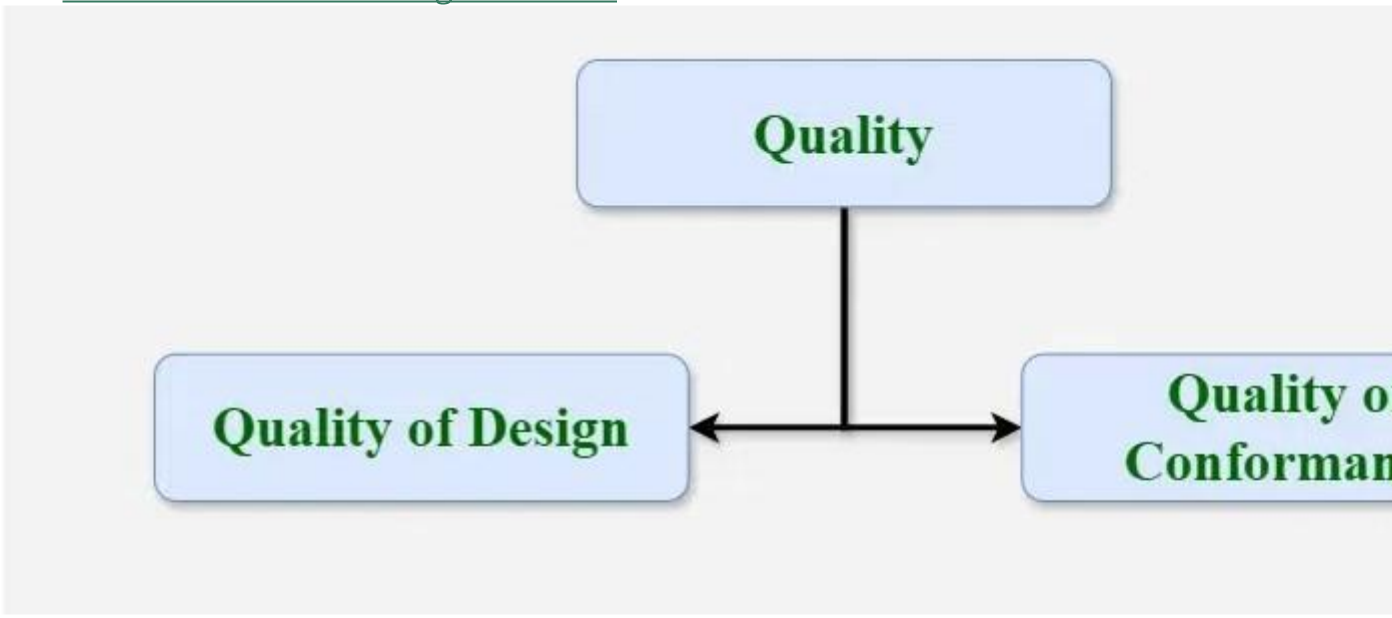
Unit 1

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities that ensure processes, procedures as well as standards are suitable for the project and implemented correctly. It is a process that works parallel to Software Development. It focuses on improving the process of development of software so that problems can be prevented before they become major issues. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

For those looking to deepen their expertise in SQA and elevate their professional skills, consider exploring a specialized training program **Manual to Automation Testing: A QA Engineer's Guide** . This program offers practical, hands-on experience and advanced knowledge that complements the concepts covered in this guide.

Quality
Quality in a product or service can be defined by several measurable characteristics. Each of these characteristics plays a crucial role in determining the overall quality.

Generally, the quality of the software is verified by third-party organizations like international standard organizations .



Quality in Software Engineering

Elements of Software Quality Assurance (SQA)

- **Standards:** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
- **Reviews and audits:** Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel

(people employed in an organization) with the intent of ensuring that quality guidelines are being followed for software engineering work.

- **Testing:** Software testing is a quality control function that has one primary goal to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted for primary goal of software.
- **Error/defect collection and analysis** : SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
- **Change management:** SQA ensures that adequate change management practices have been instituted.
- **Education:** Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement which is key proponent and sponsor of educational programs.
- **Security management:** SQA ensures that appropriate process and technology are used to achieve software security.
- **Safety:** SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
- **Risk management** : The SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

Focus of Software Quality Assurance (SQA)

The Software Quality Assurance (SQA) focuses on the following :

Software Quality Assurance (SQA)

- **Software's Portability:** Software's portability refers to its ability to be easily transferred or adapted to different environments or platforms without needing significant modifications. This ensures that the software can run efficiently across various systems, enhancing its accessibility and flexibility.
- **Software's Usability:** Usability of software refers to how easy and intuitive it is for users to interact with and navigate through the application.

A high level of usability ensures that users can effectively accomplish their tasks with minimal confusion or frustration, leading to a positive user experience.

- **Software's Reusability:** Reusability in software development involves designing components or modules that can be reused in multiple parts of the software or in different projects. This promotes efficiency and reduces development time by eliminating the need to reinvent the wheel for similar functionalities, enhancing productivity and maintainability.
- **Software's Correctness:** Correctness of software refers to its ability to produce the desired results under specific conditions or inputs. Correct software behaves as expected without errors or unexpected behaviors, meeting the requirements and specifications defined for its functionality.
- **Software's Maintainability:** Maintainability of software refers to how easily it can be modified, updated, or extended over time. Well maintained software is structured and documented in a way that allows developers to make changes efficiently without introducing errors or compromising its stability.
- **Software's Error Control:** Error control in software involves implementing mechanisms to detect, handle, and recover from errors or unexpected situations gracefully. Effective error control ensures that the software remains robust and reliable, minimizing disruptions to users and providing a smoother experience overall.

Major Activities in Software Quality Assurance (SQA)

- **SQA Management Plan:** Make a plan for how you will carry out the SQA throughout the project. Think about which set of software engineering activities are the best for project. check level of SQA team skills.
- **Set The Check Points:** SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.
- **Measure Change Impact:** The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to check the compatibility of this fix with whole project.
- **Multi testing Strategy:** Do not depend on a single testing approach. When you have a lot of testing approaches available use them.
- **Manage Good Relations:** In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of SQA team with programmers team will impact directly and badly on project.
- **Maintaining records and reports:** Comprehensively document and share all QA records, including test cases, defects, changes, and cycles, for stakeholder awareness and future reference.

- **Reviews software engineering activities:** The SQA group identifies and documents the processes. The group also verifies the correctness of software product.
- **Formalize deviation handling:** Track and document software deviations meticulously. Follows established procedures for handling variances.

Benefits of Software Quality Assurance (SQA)

- SQA produces high quality software.
- High quality application saves time and cost.
- SQA is beneficial for better reliability.
- SQA is beneficial in the condition of no maintenance for a long time.
- High quality commercial software increase market share of company.
- Improving the process of creating software.
- Improves the quality of the software.
- It cuts maintenance costs. Get the release right the first time, and your company can forget about it and move on to the next big thing. Release a product with chronic issues, and your business bogs down in a costly, time consuming, never-ending cycle of repairs.

Disadvantage of Software Quality Assurance (SQA)

There are a number of disadvantages of quality assurance.

- **Cost:** Some of them include adding more resources, which cause the more budget its not, addition of more resources for betterment of the product.
- **Time Consuming:** Testing and Deployment of the project taking more time which cause delay in the project.
- **Overhead** : SQA processes can introduce administrative overhead, requiring documentation, reporting, and tracking of quality metrics. This additional administrative burden can sometimes outweigh the benefits, especially for smaller projects.
- **Resource Intensive** : SQA requires skilled personnel with expertise in testing methodologies, tools, and quality assurance practices. Acquiring and retaining such talent can be challenging and expensive.
- **Resistance to Change** : Some team members may resist the implementation of SQA processes, viewing them as bureaucratic or unnecessary. This resistance can hinder the adoption and effectiveness of quality assurance practices within an organization.
- **Not Foolproof** : Despite thorough testing and quality assurance efforts, software can still contain defects or vulnerabilities. SQA cannot guarantee the elimination of all bugs or issues in software products.

The software quality challenges

The software quality challenge is the complex, ongoing effort to ensure software meets user needs and standards, balancing features, performance, security, and budget, complicated by high complexity, changing requirements, tight deadlines, and the invisible nature of code, requiring strong collaboration, rigorous testing, and adaptable strategies like Agile and MVP to find the "good enough" middle ground between releasing too late and releasing buggy software.

Key Challenges

- **Complexity & Invisibility:**

  Software is complex, interdependent, and hard to inspect, making defects difficult to find until late in the cycle.

- **Defining "Quality":**

  Quality is subjective (functionality, performance, security, usability) and varies by context, making standards hard to pin down.

- **Requirements Volatility:**

  Requirements often change or are unclear, confusing testers and leading to rework.

- **Time & Resource Constraints:**

  Pressure to release quickly often clashes with the need for thorough testing, creating bottlenecks.

- **Agile & Evolving Processes:**

  Rapid development cycles (Agile, DevOps) require QA to adapt quickly to new methodologies and tools.

- **Data & Environment Diversity:**

  Testing across numerous devices, platforms, and data sets is increasingly difficult.

- **Organizational Culture:**
  Lack of leadership commitment or communication between dev/QA teams hinders quality efforts.
  This video discusses the dilemma of balancing speed and quality in software development:

Common Approaches & Solutions

- **Shift-Left Testing:** Integrating testing earlier in the development lifecycle.
- **Clear Communication:** Fostering collaboration between developers, testers, and stakeholders.
- **Minimum Viable Product (MVP):** Releasing core features first to gather user feedback.
- **Automation:** Using automated testing to handle complexity and speed up regression.
- **Traceability:** Using matrices to link requirements to tests to ensure coverage.
- **Continuous Improvement:** Adopting agile QA practices and staying updated on new tools.

Software quality

Traditionally, a high-quality product is outlined in terms of its fitness of purpose. That is, a high-quality product will specifically be what the users need to try. For code products, the fitness of purpose is typically taken in terms of satisfaction of the wants arranged down within the SRS document.

Though "fitness of purpose" could be a satisfactory definition of quality for some products like an automobile, a table fan, a grinding machine, etc. – for code products, "fitness of purpose" isn't a completely satisfactory definition of quality.

What is Software Quality?

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. But, it has an associate degree virtually unusable program. even though it should be functionally correct, we tend not to think about it to be a high-quality product.

Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as "fitness of purpose" for code products isn't satisfactory.

Factors of Software Quality

The modern read of high-quality associates with software many quality factors like the following:

1. **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code products, etc.
2. **Usability:** A software has smart usability if completely different classes of users (i.e. knowledgeable and novice users) will simply invoke the functions of the products.
3. **Reusability:** A software has smart reusability if completely different modules of the products will simply be reused to develop new products.
4. **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
5. **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the products, and therefore the functionalities of the products may be simply changed, etc
6. **Reliability:** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
7. **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

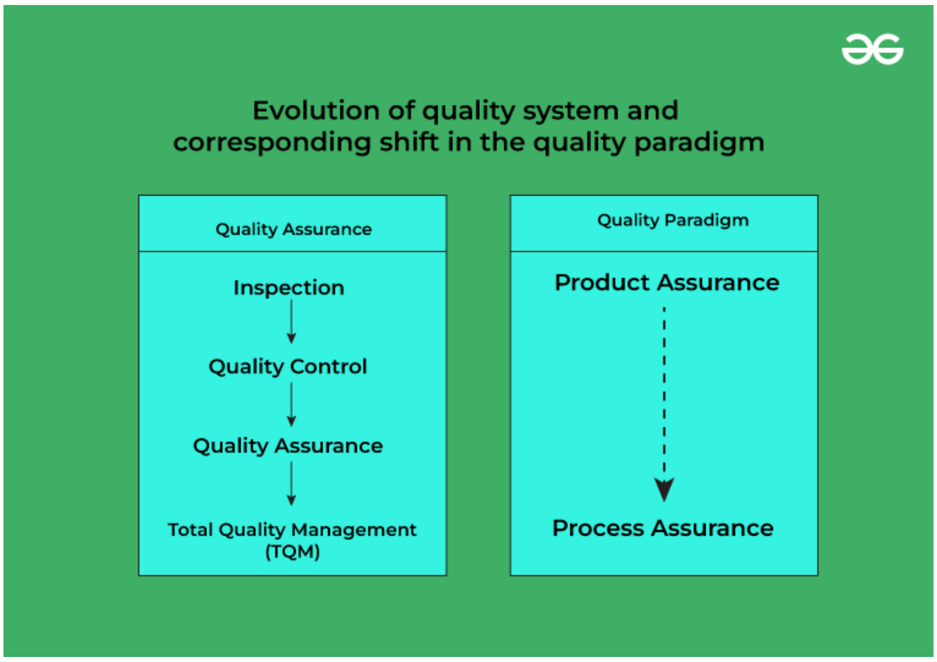
Software Quality Management System
Software Quality Management System contains the methods that are used by the authorities to develop products having the desired quality.
Some of the methods are:

- **Managerial Structure:** Quality System is responsible for managing the structure as a whole. Every Organization has a managerial structure.
- **Individual Responsibilities:** Each individual present in the organization must have some responsibilities that should be reviewed by the top management and each individual present in the system must take this seriously.
- **Quality System Activities:** The activities which each quality system must have been
    - Project Auditing.
    - Review of the quality system.
    - It helps in the development of methods and guidelines.

Evolution of Quality Management System

Quality Systems are basically evolved over the past some years. The evolution of a Quality Management System is a four-step process.

1. **Inspection:** Product inspection task provided an instrument for quality control (QC).
2. **Quality Control:** The main task of quality control is to detect defective devices, and it also helps in finding the cause that leads to the defect. It also helps in the correction of bugs.
3. **Quality Assurance:** Quality Assurance helps an organization in making good quality products. It also helps in improving the quality of the product by passing the products through security checks.
4. **Total Quality Management (TQM):** Total Quality Management(TQM) checks and assures that all the procedures must be continuously improved regularly through process measurements.
5.



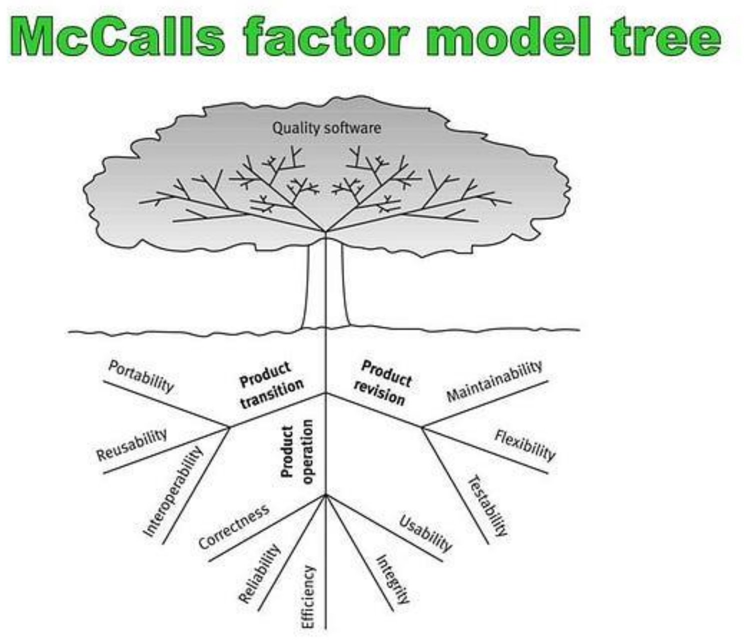
Evolution of Quality Management System

n software development quality assurance is very important thing. Quality of software directly affect for software Engineers, Developers and Users. Because of that software quality factors are very essential for a develop quality software. These factors are important measure of the quality of software and being a good indicator of the need for software development and maintenance. There are soo many factors that affect the quality of the software In this article I’m gong to discuss about these software quality factors.

When we discuss about software quality factors can’t miss-out software quality factor models. There are many software quality factor models. The classic model of software quality factors, introduced by McCall in 1977. Later Similarly, models to similar McCall model were suggested by Deutsch and Willis and by Evans and Marciniak. But these models not much different from McCall’s model. In software requirement The McCall factor model provides a practical, up-to-date method for classifying software requirement.

McCall’s Factor Model

In this model we can Classify software requirements into software quality factors. There are 11 requirements divided into software quality factors. The 11 factors are grouped into three main categories there are product operation, product revision, and product transition factors.
Press enter or click to view image in full size



Product operation factors:

Correctness, Reliability, Efficiency, Integrity, Usability.

Product revision factors:

Maintainability, Flexibility, Testability.

Product transition factors:

Portability, Reusability, Interoperability.

Product operation factors

Product operation factors includes five software requirements. These requirements directly affect to the software operations such as operational performance, ease of usage and etc. These 5 factors help to provide a better user experience to users.

Correctness

Correctness means a program satisfy all its specifications. Correctness is the capability of software to perform the operations that were designed to perform.

In the correctness consider what are the things need to output, Accuracy of output ,Completeness of output information , Up-to-date of the information and Availability of the information

Reliability

Reliability means a program capable to be maintained so that it can perform its exact function. Reliability requirements deal with software product failure.
Quality assurance (QA) and management roles in software engineering focus on ensuring software products meet quality standards, are functional, and reliable through both proactive process improvements and reactive testing. Key roles include QA Engineers, Analysts, and Managers who design test plans, execute manual/automated tests, report defects, and collaborate with developers to ensure high-quality software delivery.

Key Quality Assurance and Management Roles:

- **QA Engineer/Software Tester:** Responsible for creating test plans, designing test cases, and performing manual or automated testing to identify bugs before release.
- **Automation Engineer:** Focuses on writing scripts using tools like Selenium, Cypress, or Playwright to automate repetitive testing processes, improving efficiency.
- **QA Analyst:** Analyzes software requirements and business logic to ensure that testing covers all user requirements and functionality.

- **QA Lead/Manager**: Oversees the entire testing process, manages QA team activities, ensures adherence to standards, and communicates with project managers regarding risks and product quality.
- **Software Quality Assurance (SQA) Engineer**: Focuses on the entire software development lifecycle (SDLC), implementing processes and standards to *prevent* defects rather than just finding them.

Core Responsibilities:

- **Defect Tracking:** Identifying, documenting, and verifying fixes for bugs.
- **Test Execution:** Performing various types of tests, including regression, functional, usability, and security testing.
- **Process Improvement:** Auditing development processes to ensure best practices are followed.
- **Collaboration:** Working closely with developers and product managers to ensure requirements are met.

Key Skills:

- **Technical Knowledge:** Proficiency in automated testing tools, bug tracking systems (e.g., Jira), and programming languages (e.g., Python, Java).
- **Analytical Skills:** Ability to analyze software requirements and identify potential risks.
- **Communication:** Strong skills for reporting bugs and explaining technical issues to non-technical stakeholders.

These roles ensure that the software is reliable, secure, and user-friendly, ultimately protecting the organization's reputation.

Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes. It ensures that developed software meets and complies with the defined or standardized quality specifications. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets the desired quality measures.

SQA practices are implemented in most types of software development, regardless of the underlying software development model being used. SQA incorporates and implements software testing methodologies to test the software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development, until the software is complete. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards. SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies.

It includes the following activities −

- Process definition and implementation
- Auditing
- Training

Processes could be −

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing, etc.

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities −

- Identify the weaknesses in the processes
- Correct those weaknesses to continually improve the process

Components of SQA System

An SQA system always combines a wide range of SQA components. These components can be classified into the following six classes −

Pre-project components

This assures that the project commitments have been clearly defined considering the resources required, the schedule and budget; and the development and quality plans have been correctly determined.

Components of project life cycle activities assessment

The project life cycle is composed of two stages: the development life cycle stage and the operationmaintenance stage.

The development life cycle stage components detect design and programming errors. Its components are divided into the following sub-classes: Reviews, Expert opinions, and Software testing.

The SQA components used during the operationmaintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality to improve the maintenance tasks.

Components of infrastructure error prevention and improvement

The main objective of these components, which is applied throughout the entire organization, is to eliminate or at least reduce the rate of errors, based on the organizations accumulated SQA experience.

Components of software quality management

This class of components deal with several goals, such as the control of development and maintenance activities, and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.

Components of standardization, certification, and SQA system assessment

These components implement international professional and managerial standards within the organization. The main objectives of this class are utilization of international professional knowledge, improvement of coordination of the organizational quality systems with other organizations, and assessment of the achievements of quality systems according to a common scale. The various standards may be classified into two main groups: quality management standards and project process standards.

Organizing for SQA the human components

The SQA organizational base includes managers, testing personnel, the SQA unit and the persons interested in software quality such as SQA trustees, SQA committee members, and SQA forum members. Their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements

Software Quality Assurance (SQA) system in software engineering comprises six main component classes designed to ensure product quality throughout the

development lifecycle. These include pre-project planning, lifecycle-focused reviews, error-preventing infrastructure, management controls, standards certification, and human organizational components.
Key components of an SQA system include:

- **Pre-project Quality Components**: Define contract requirements, project development plans, and quality plans to ensure feasibility.
- **Project Life Cycle Quality Components**: Include formal design reviews, code inspections, and testing (unit, integration, system) to detect errors early.
- **Infrastructure Error Prevention and Improvement**: Involves procedures, training, and tools that prevent errors, such as documentation management and configuration management.
- **Software Quality Management**: Controls project progress, manages risks, and uses metrics to assess quality levels.
- **Standardization and Certification**: Ensures compliance with international standards (e.g., ISO) and internal quality guidelines.
- **Organizational Components**: Involves the management team, SQA staff, and practitioners responsible for implementing, monitoring, and improving the quality system.

A project is a temporary endeavor undertaken to create a unique product, service or result. Temporary means, every project has a definite beginning and a definite end. The end is reached when the project's objectives have been achieved or when it becomes clear that the project objectives will not or cannot be met and the project is terminated. A Project creates unique deliverables, which are products, services or results. The presence of repetitive elements does not change the fundamental uniqueness of the project work. A product or artifact that is produced, is quantifiable, and can be either an end item in itself or a component item.

To have a successful software project, the project objectives should be clearly defined as Specific, that is, concrete and well-defined, in terms of Measurable, satisfaction of the objective can be objectively judged ; Achievable, it is within the power of the individual or group concerned to meet the target; Relevant, the objective must relevant to the true purpose of the project;

Time constrained, there is defined point in time by which the objective should be achieved. The Management involves the following activities

- Planning – deciding what is to be done Organizing – making arrangements
- Staffing – selecting the right people for the job Directing – giving instructions
- Monitoring – checking on progress
- Controlling – taking action to remedy hold-ups
- Innovating – coming up with solutions when problems emerge
- Representing – liaising with clients, users, developers and other stakeholders

Project management is the application of knowledge, skills, tools and techniques to project activities to meet project requirements. Project management is accomplished through the application and integration of the project management processes of initiating, planning, executing, monitoring and controlling, and closing.Managing a Project Includes:

- Identifying requirements
- Establishing clear and achievable objectives
- Balancing the competing demands for quality, scope, time and cost
- Adapting the specifications, plans, and approach to the different concerns and Expectations of the various stakeholders.

<u>Contract and Contract Review</u>

Contract is the basis for considerable litigation when delivered software does not perform as expected. The purpose is to arrive at a realistic budget and timetable and discovering pitfalls early and then, to arrive at final contract draft with these important parameters revealed.

Several situations can lead a software company to sign a contract with a customer. The most common are:

- Participation in a tender.
- Submission of a proposal according to the customer's RFP.(Request For Proposal)
- Receipt of an order from a company's customer.
- Receipt of an internal request or order from another department in the organization.

Request for Proposal is a solicitation made, often through a bidding process, by an agency or company interested in procurement of a commodity, service or valuable asset, to potential suppliers to submit business proposals. A good RFP allows contractors or a project team to understand what the customer expects so that they can prepare a thorough proposal that will satisfy the customer's requirement at a realistic price. An RFP must provide a statement of work (SOW) A SOW deals with the scope of the project, outlining the tasks or work elements the customer wants the contractor or project team to perform. The RFP must include the customer requirements which define specifications and attributes. The RFP should state what deliverable the customer expects the contractor or project team to provide. The RFP should list any customer-supplied items. The RFP might state the approvals required by the customer. Some RFPs mention the type of contract the customer intends to use. An RFP might state the payment terms the custome intends to use. The RFP should state the required schedule for completion of the project. The RFP should provide instructions for the format and content of the contractor proposals. The RFP should indicate the due date by which the customer expects potential contractors to submit proposals.

A bad contract is characterized by loosely defined requirements, and unrealistic budgets and schedules and is expected to yield low-quality software. SQA program performs its preventive quality assurance efforts with a review of the Proposal draft, Contract draft. The two reviews are aimed at improving the budget and timetable.

The objective of the proposal draft review is to make sure that the following activities have been satisfactorily carried out. Customer requirements have been clarified and documented. Alternative approaches for carrying out the project have been examined and formal aspects of the relationship between the customer and the software firm have been specified.

The three contract draft review objectives that make sure the following activities have been satisfactorily carried out:

1. No unclarified issues remain in the contract draft.
2. All understandings reached subsequent to the proposal are correctly documented.
3. No “new” changes, additions, or omissions have entered the contract draft.

Contract reviews vary in their magnitude, depending on the characteristics of the proposed project. This complexity may be either technical or organizational. Accordingly, different levels of professional effort are justified for the various contract reviews. The task of contract review can be completed by various individuals, listed here in ascending order, according to the complexity of the project:

- The leader or another member of the proposal team.

- The members of the proposal team.
- An outside professional or a company staff member.
- A team of outside experts.

Implementation of a contract review process for a major project usually involves substantial organizational difficulties. The major difficulties in carrying out the contract reviews for major proposals are:

- Time Pressure.
- Proper contract review requires substantial professional work.
- The potential contract review team members are very busy.

The careful planning of contract reviews is required for their successful completion, they are:

- The contract review should be scheduled.
- A team should carry out the contract review.
- A contract review team leader should be appointed.

The contract review should be scheduled. Contract review activities should be included in the proposal preparation schedule, leaving sufficient time for the review and the ensuing corrections to be made. Teamwork makes it possible to distribute the workload among the team members so that each member of the contract review team can find sufficient time to do his or her share.

A substantial number, if not the majority, of software projects are internal projects — "in-house" projects – carried out by one unit of an organization for another unit of the same organization. In such cases, the software development unit is the supplier, while the other unit can be considered the customer.

The loose relationships maintained between the internal customer and the internal developer increase the probability of project failure. This trend can be reduced by adequate procedures that will define the preparation and by applying the same guidelines used for external project contract review. Loose Relationships are usually characterized by:

- Inadequate definition of project requirements.
- Poor estimates of required resources.
- Poor timetable/scheduling.
- Inadequate awareness of development risks.

A substantial number, if not the majority, of software projects are internal projects — "in-house" projects – carried out by one unit of an organization for another unit of the same organization. In such cases, the software development unit is the supplier, while the other unit can be considered the customer. Internal software development projects are not based on what would be considered a complete customer–supplier relationship. In many cases, these projects are based on general agreements, with goodwill playing an important role in the relationships between the two units.

Contract reviews examine many subjects, based on the contract review objectives. Checklists are useful devices for helping review teams to organize their work and achieve high coverage of the relevant subjects. It is clear that many of the subjects on these lists are irrelevant for any specific project. At the same time, even a comprehensive checklist may exclude some important subjects relevant to a given project proposal. It is the task of the contract review team, but especially of its leader, to determine the list of subjects pertinent for the specific project proposal

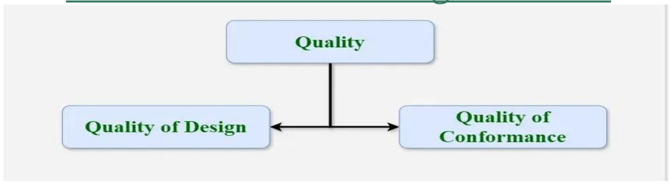
Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities that ensure processes, procedures as well as standards are suitable for the project and implemented correctly. It is a process that works parallel to Software Development. It focuses on improving the process of development of software so that problems can be prevented before they become major issues. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

For those looking to deepen their expertise in SQA and elevate their professional skills, consider exploring a specialized training program **Manual to Automation Testing: A QA Engineer's Guide** . This program offers practical, hands-on experience and advanced knowledge that complements the concepts covered in this guide.

Quality

Quality in a product or service can be defined by several measurable characteristics. Each of these characteristics plays a crucial role in determining the overall quality.

Generally, the quality of the software is verified by third-party organizations like international standard organizations .



Quality in Software Engineering

Elements of Software Quality Assurance (SQA)

- **Standards:** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
- **Reviews and audits:** Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel (people employed in an organization) with the intent of ensuring that quality guidelines are being followed for software engineering work.
- **Testing:** Software testing is a quality control function that has one primary goal to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted for primary goal of software.
- **Error/defect collection and analysis** : SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
- **Change management:** SQA ensures that adequate change management practices have been instituted.
- **Education:** Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement which is key proponent and sponsor of educational programs.
- **Security management:** SQA ensures that appropriate process and technology are used to achieve software security.
- **Safety:** SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
- **Risk management** : The SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

Focus of Software Quality Assurance (SQA)
The Software Quality Assurance (SQA) focuses on the following :
Software Quality Assurance (SQA)

- **Software's Portability:** Software's portability refers to its ability to be easily transferred or adapted to different environments or platforms without needing significant modifications. This ensures that the software can run efficiently across various systems, enhancing its accessibility and flexibility.
- **Software's Usability:** Usability of software refers to how easy and intuitive it is for users to interact with and navigate through the application. A high level of usability ensures that users can effectively accomplish their tasks with minimal confusion or frustration, leading to a positive user experience.
- **Software's Reusability:** Reusability in software development involves designing components or modules that can be reused in multiple parts of the software or in different projects. This promotes efficiency and reduces

development time by eliminating the need to reinvent the wheel for similar functionalities, enhancing productivity and maintainability.

- **Software's Correctness:** Correctness of software refers to its ability to produce the desired results under specific conditions or inputs. Correct software behaves as expected without errors or unexpected behaviors, meeting the requirements and specifications defined for its functionality.
- **Software's Maintainability:** Maintainability of software refers to how easily it can be modified, updated, or extended over time. Well maintained software is structured and documented in a way that allows developers to make changes efficiently without introducing errors or compromising its stability.

UNIT 2

SQA components in the project life cycle are structured activities embedded throughout development and maintenance to ensure high software quality. Key components include formal design reviews, peer reviews (inspections/walkthroughs), expert opinions, comprehensive software testing (functional/system), and specialized maintenance activities to detect and prevent errors.

These components span the two primary stages of the project life cycle:

- **Development Life Cycle Stage:** Focuses on detecting design and programming errors early through:
  - **Formal Design Reviews:** Structured evaluations to approve project phases.
  - **Peer Reviews (Inspections/Walkthroughs):** Colleague-based reviews of code and documentation.
  - **Expert Opinions:** Insights from specialists to resolve complex issues.
  - **Software Testing:** Manual and automated testing (unit, integration, system) to validate functionality and performance.
- **Operation and Maintenance Stage:** Ensures long-term quality via:
  - **Corrective Maintenance:** Fixing faults found after release.
  - **Adaptive Maintenance:** Updating software for new environments.
  - **Functionality Improvement:** Enhancing performance and features.

Key SQA Sub-components & Activities:

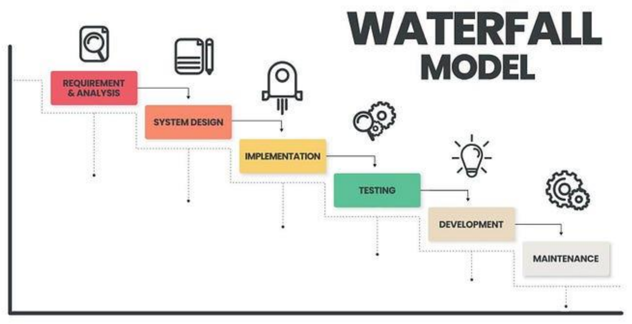
- **Verification and Validation:** Ensuring the product meets specifications and user needs.
- **Configuration Management:** Controlling changes to software items.
- **Documentation Control:** Maintaining proper documentation standards.
- **Quality Metrics & Reporting:** Evaluating exit criteria and tracking quality trends

Integrating quality activities in project life cycle of each

outputs are reviewed and evaluated by the developer as well as, in many cases, by the customer. The outcomes range from approval of the phase results and continuation to the next phase, to demands to correct, redo or alter parts of the respective phase.

1. The software development life cycle (SDLC) model

The Software Development Life Cycle model (the SDLC model) is the classic model (still applicable today); it provides the most comprehensive description of the process available. The model displays the major building blocks for the entire development process, described as a linear sequence that begins with requirements definition and ends with regular system operation and maintenance.



- **System Design**. This stage involves the detailed definition of the outputs, inputs and processing procedures, including data structures and databases, software structure, etc.

- **Implementation (Coding)**. In this phase, the design is translated into a code. Coding involves quality assurance activities such as inspection, unit tests and integration tests.

- **Testing (System tests)**. System tests are performed once the coding phase is completed. The main goal of testing is to uncover as many software errors as possible so as to achieve an acceptable level of software quality once corrections have been completed. System tests are carried out by the software developer before the software is supplied to the customer.

- **Development (Installation and conversion)**. After the software system is approved, the system is installed to serve as firmware, that is, as part of the information system that represents a major component of the expanded system.

- **Maintenance (Regular operation)**. Regular software operation begins once installation and conversion have been completed. Throughout the regular operation period, which usually lasts for several years or until a new software generation appears on the scene, maintenance is needed.

The classic waterfall model was suggested by Royce (1970) and later presented in its commonly known form by Boehm (1981). It provides the foundations for the majority of the major software quality assurance standards employed, such as IEEE Std 1012 (IEEE, 1998) and IEEE Std 12207 (IEEE, 1996, 1997a, 1997b), to mention just two

2. The prototyping model

The prototyping model is based on replacement of one or more SDLC model phases by an evolutionary process, where software prototypes are used for communication between the developer and the users and customers. Prototypes are submitted to user representatives for evaluation.

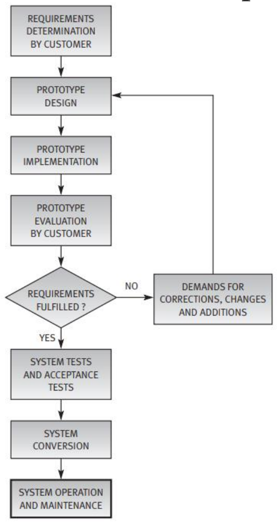

REQUIREMENTS DETERMINATION BY CUSTOMER
PROTOTYPE DESIGN
PROTOTYPE IMPLEMENTATION
PROTOTYPE EVALUATION BY CUSTOMER
REQUIREMENTS FULFILLED ?
NO
DEMANDS FOR CORRECTIONS, CHANGES AND ADDITIONS
YES
SYSTEM TESTS AND ACCEPTANCE TESTS
SYSTEM CONVERSION
SYSTEM OPERATION AND MAINTENANCE

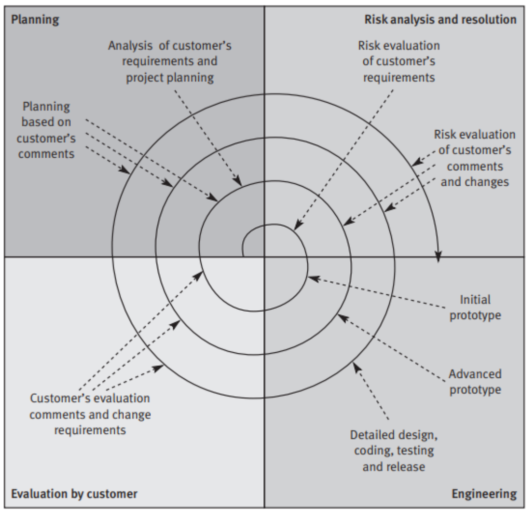
Figure 3: Prototyping model

Prototyping as a software development methodology has been found to be efficient and effective mainly for small- to medium-sized software development projects.

Figure 4: Prototyping versus SDLC methodology — advantages and disadvantages (mainly for small to medium-sized projects)

3. The spiral model

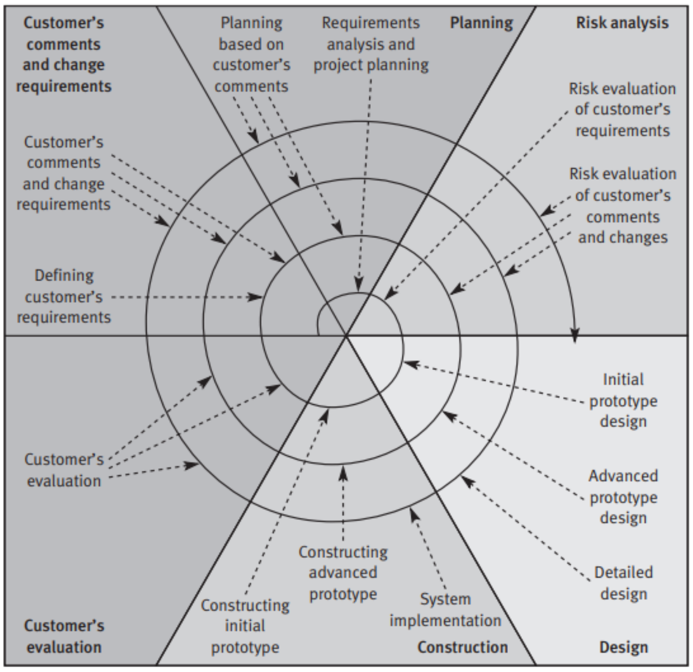
The spiral model, as revised by Boehm (1988, 1998), offers an improved methodology for overseeing large and more complex development projects displaying higher prospects for failure, it combines an iterative model that introduces and emphasizes risk analysis and customer participation into the major elements of SDLC and prototyping methodologies.



Planning
Risk analysis and resolution
Analysis of customer's requirements and project planning
Risk evaluation of customer's requirements
Planning based on customer's comments
Risk evaluation of customer's comments and changes
Initial prototype
Advanced prototype
Customer's evaluation comments and change requirements
Detailed design, coding, testing and release
Evaluation by customer
Engineering

Figure 5: The spiral model (Boehm, 1988)

An advanced spiral model, the Win–Win Spiral model (Boehm, 1998), enhances the Spiral model (Boehm, 1988) still further. The advanced spiral model (the Win–Win model) places extra emphasis on communication and negotiation between customer and developer. The customer wins by improving chances to receive a system that satisfies most of his needs while the developer wins by improving chances of completing the project within budgetary and timetable constraints.



Customer's comments and change requirements
Planning based on customer's comments
Requirements analysis and project planning
Planning
Risk analysis
Risk evaluation of customer's requirements
Customer's comments and change requirements
Risk evaluation of customer's comments and changes
Defining customer's requirements
Initial prototype design
Customer's evaluation
Advanced prototype design
Detailed design
Constructing advanced prototype
Constructing initial prototype
System implementation
Customer's evaluation
Construction
Design

Figure 6: The advanced spiral model (Boehm, 1998)

4. The object-oriented mode

According to this model, the development process begins with a sequence of object-oriented analysis and design activities. The design phase is followed by acquisition of a reusable software library together with “regular” development of the unavailable software components.

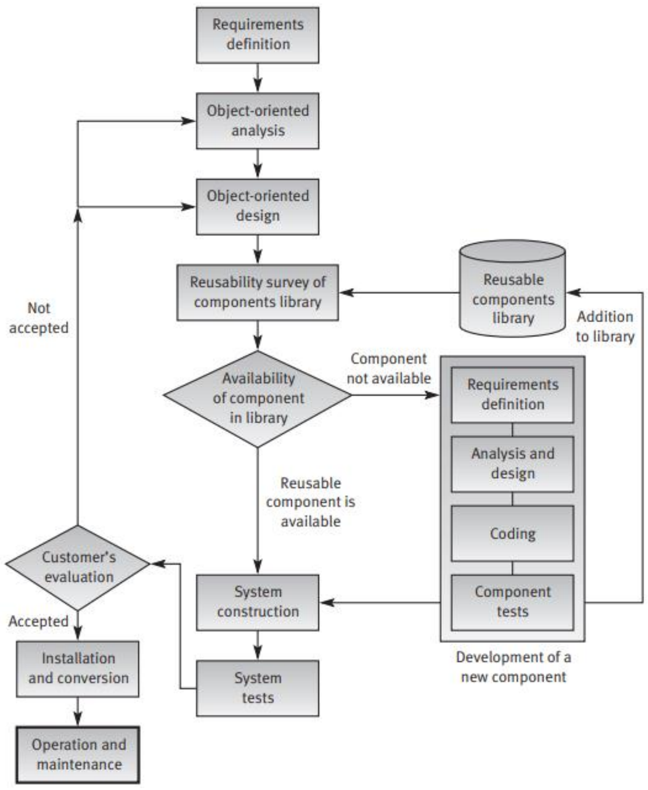


Figure 7: The object-oriented model

Factors affecting intensity of quality assurance activities in the development process

The decision about the number of quality assurance activities to be applied is affected by project and team factors. Project factors include project magnitude, its complexity and difficulty, extent of reusable software components, and the severity of the outcomes if the project fails. Team factors include its professional qualifications as well as acquaintance with the project and related experience, availability of professional support, and staff familiarity with team members.

Verification, validation and qualification

Quality assurance activities examine three different aspects of quality by means of software product verification, validation and qualification.

- Verification examines the consistency of current development activities with the products from previous phases. Doing so enables the examiner to confirm whether the developer has fulfilled his requirements while disregarding deviations from the original requirements that may have arisen during development.

- Validation represents the customer's interests by examining the extent to which the customer's original requirements have been fulfilled.
- Qualification focuses on operational aspects, where maintenance is the main issue. Qualification reviews project application of professional standards and coding procedures, based on the assumption that applying these standards facilitates maintenance. Quality assurance activity planners are required to determine which of these aspects should be examined in each of the planned quality assurance activities

A model for SQA defect removal effectiveness and cost

The model deals with two quantitative aspects of an SQA plan designed for a specific project: (1) Total effectiveness of defect removal. (2) Total cost of defect removal.

The model is based on the following assumptions:

- The development process is linear and sequential (the waterfall model).
- A number of “new” defects are introduced in each development phase.
- Various review and test software assurance activities serve as filters, removing a percentage of the entering defects while allowing the rest to pass to the next software assurance activity.
- Incoming defects are the sum of defects passed from the former quality assurance activity together with “new” defects created in the current development phase.
- The cost of defect removal is calculated by multiplying the number of defects removed by the relative cost of removing a defect.
- Defects passed to the customer will be detected by him or her; their full removal at this phase will incur heavy cost

Software Testing Strategies

- **Software testing** is the process of evaluating a software application to identify if it meets specified requirements and to identify any defects. **The following are common testing strategies:**

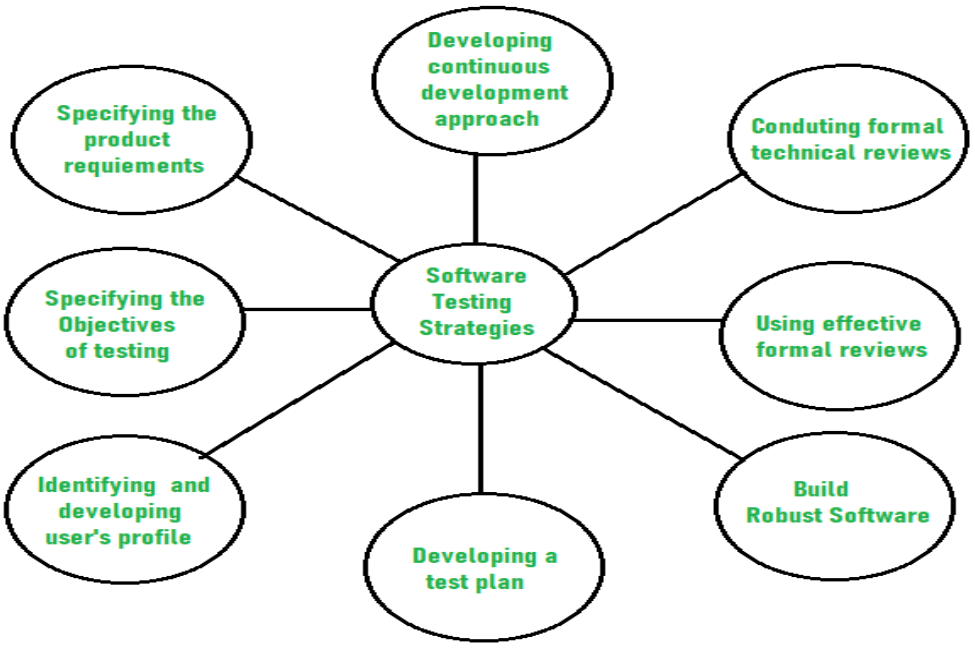
1. **Black box testing**- Tests the functionality of the software without looking at the internal code structure.
2. **White box testing** - Tests the internal code structure and logic of the software.

3. **Unit testing** - Tests individual units or components of the software to ensure they are functioning as intended.
4. **Integration testing**- Tests the integration of different components of the software to ensure they work together as a system.
5. **Functional testing**- Tests the functional requirements of the software to ensure they are met.
6. **System testing**- Tests the complete software system to ensure it meets the specified requirements.
7. **Acceptance testing** - Tests the software to ensure it meets the customer's or end-user's expectations.
8. **Regression testing** - Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
9. **Performance testing** - Tests the software to determine its performance characteristics such as speed, scalability, and stability.
10. **Security testing**- Tests the software to identify vulnerabilities and ensure it meets security requirements.

Software Testing is a type of investigation to find out if there are any defects or errors present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specified requirements or not.
According to Glen Myers, software testing has the following objectives:

- The process of investigating and checking a program to find whether there is an error or not and does it fulfills the requirements or not is called testing.
- When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of a good test case.
- Finding an unknown error that wasn't discovered yet is a sign of a successful and good test case.

Software testing Strategies
The main **objective** of software testing is to **design the tests** in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for the development of the software.
The overall strategy for testing software includes:


Developing continuous development approach
Specifying the product requiements
Conduting formal technical reviews
Software Testing Strategies
Specifying the Objectives of testing
Using effective formal reviews
Identifying and developing user's profile
Build Robust Software
Developing a test plan

Software testing Strategies

Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner. Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.

1. **Specifying the objectives of testing in a clear and detailed manner.** Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.
2. **For the software, identifying the user's category and developing a profile for each user.** Use cases describe the interactions and communication among different classes of users and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.
3. **Developing a test plan to give value and focus on rapid-cycle testing.** Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing.
4. **Robust software is developed that is designed to test itself.** The software should be capable of detecting or identifying different classes of errors. Moreover, software design should allow automated and regression testing

which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.

5. **Before testing, using effective formal reviews as a filter.** Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.
6. **Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.** The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.
7. **For the testing process, developing a approach for the continuous development.** As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

Advantages of Software Testing

- **Improves software quality and reliability** - Testing helps to identify and fix defects early in the development process, reducing the risk of failure or unexpected behavior in the final product.
- **Enhances user experience** - Testing helps to identify usability issues and improve the overall user experience.
- **Increases confidence** - By testing the software, developers and stakeholders can have confidence that the software meets the requirements and works as intended.
- **Facilitates maintenance** - By identifying and fixing defects early, testing makes it easier to maintain and update the software.
- **Reduces costs** - Finding and fixing defects early in the development process is less expensive than fixing them later in the life cycle.

Disadvantages of Software Testing

1. **Time-consuming** - Testing can take a significant amount of time, particularly if thorough testing is performed.
2. **Resource-intensive** - Testing requires specialized skills and resources, which can be expensive.
3. **Limited coverage** - Testing can only reveal defects that are present in the test cases, and it is possible for defects to be missed.
4. **Unpredictable results** - The outcome of testing is not always predictable, and defects can be hard to replicate and fix.
5. **Delays in delivery** - Testing can delay the delivery of the software if testing takes longer tha

Unit 3

Software Testing – Implementation

Testing process includes Planning, design and performance of testing are carried out throughout the software development process. The testing process involves the following steps:

- **Determining the test methodology phase**
- **Planning the tests**
- **Test design**
- **Test implementation**

In Test Methodology Phase, Testing is done throughout the development process. Testing is divided into phases beginning in the design phase and ending at the customer's site. The main issues that testing methodology addresses are the appropriate required software quality standard and the software testing strategy. Different standards are required for different software applications. The expected damage resulting from failed software impacts the standard of software quality.

Example 1:

- A software package for a hospital patient bed monitor requires the highest software quality standard considering the possibly severe consequences of software failure.

Example 2:

- A package developed for handling feedback information for an organization's internal employee training program could make do with a medium-level software quality standard, assuming that the cost of failure is relatively low.

Classification of Software Failure:

(a) Damages to Customers and Users:

- Endangers the safety of human beings
- Affects an essential organizational function with no system replacement capability available
- Affects functioning of firmware, causing malfunction of an entire system
- Affects an essential organizational function but a replacement is available
- Affects proper functioning of software packages for business applications
- Affects proper functioning of software packages for a private customer

(b) Damages to the Software Developer:

- **Financial losses**

-
    - Damages paid for physical injuries
    - Purchase cost reimbursed to customers
    - Damages paid to organizations for malfunctioning of software

- **Non-quantitative damages**
    - Expected to affect future sales
    - Substantially reduced current sales

The issues that have to be decided include: The testing strategy: (Big – bang or bottom-up or top-down?) Parts of the testing plan that should be performed according to the white box testing model. Parts of the testing plan that should be performed according to the automated testing model.

Developers need to consider the following issues before initiating a specific test plan:

- What to test?
- Which sources to use for test cases?
- Who is to perform the tests?
- Where to perform the tests?
- When to terminate the tests?

An approach to perfect testing is to recommend a full and comprehensive software test plan that requires:

- Performing unit tests for all the individual units.
- Integration tests for all the unit integrations.
- A system test to test the software package as a whole.

Factors to be considered for Unit and Integration Tests:
Which modules need to be unit tested?

Which integrations should be tested? Low priority applications tested in unit testing may not be needed or included in the system tests. Lots of planning is needed, as testing is a very expensive undertaking.

The methods for modules, integrations and applications to determine their priority in the testing plan are based on two factors:
Factor A: Damage severity level: The severity of results in case the module or application fails.

Factor B: Software risk level: The level of risk represents the probability of failure.

Who performs the tests? is the next question to be answered.

Unit Testing – done by the programmer and/or development team.

Integration Testing – can be the development team or a testing unit.

System Testing – usually done by an independent testing team (internal or external (consultants) team. For small companies, another testing team from another development team can be used and swapped. Unit and integration testing are naturally carried out at the software developer's site.

If system testing is to be performed by external testing consultants, a third option arises: the consultant's site.

The stage at which software testing should be terminated is with respect to system tests. Five alternative routes are available, each chosen on the different criteria:

- The completed implementation route.
- The mathematical models application route
- The error seeding route
- The dual independent testing teams route
- Termination after resources have petered out

The Mathematical Models Application Route are applied to estimate the percentage of undetected errors. Testing would be terminated once the error detection rate declines below the rate that corresponds to a predetermined level of undetected errors. The disadvantage of this route is that the mathematical model chosen may not fully represent the project's characteristics. Error seeding route are seeded in the tested software prior to the outset of testing. The percentage of discovered seeded errors will correspond to the percentage of real errors detected. Testing will terminate once the residual percentage of undetected seeded errors reaches a predefined level considered acceptable for "passing" the system.

Test design is the planning stage of the software system tests is commonly documented in a "**Software Test Plan" (STP)**. The test design is carried out on the basis of the software test plan as documented by STP. In software test plan the scope of the tests is considered as the document that provide the basis for the planned tests. In testing environment it considers the testing sites and required hardware and firmware configuration. It also considers the preparation and training required of the test team.

In software test description the test procedures and the test case database/file may be documented in a “Software Test Procedure” document and “Test Case File” document or in a single document called the “Software Test Description”. Testing implementation phase activities consist of a series of tests, corrections of detected errors and re-tests. The tests are carried out by running the test cases according to the test procedures. In manual software tests, only a portion of the original test procedure is re-tested to save testing resources and shorten re-testing duration.

A test case is a documented set of the data inputs and operating conditions required to run a test item together with the expected results of the run. The tester is expected to run the program for the test item according to the test case documentation. Actual Results are compared with the expected results noted in the documents. When some or all of the results do not agree with the expected results, a potential error is identified.

In code auditing the test performs automated qualification testing. The code auditor checks the compliance of code to specified standards and procedures of coding. The auditor’s report includes a list of the deviations from the standards and a statistical summary of the findings. In coverage monitoring Produce reports about the line coverage achieved when implementing a given test case file. The output includes the percentage of lines covered by the test cases as well as listings of uncovered lines. These features make coverage monitoring a vital tool for white-box tests.

In functional tests, automated regression tests performed for the whole program verify that the error corrections have been performed satisfactorily. An automated testing tool that supports functional tests, the output comparator, will help in the regression test stage. The automated comparison of outputs of successive tests enables testers to prepare an improved analysis of the regression test.

For load tests to be performed, the maximal load environment must be first created. Load tests are based on scenarios of the maximal load situations. Virtual users and virtual events are generated and operated in a hardware and communication environment defined for load tests. Automated test management software provides features applicable for manual as well as automated testing and for automated tests only. The inputs together with the software package’s capabilities, determine the application’s scope. Alpha site and beta site tests are employed to obtain comments about quality from the package’s potential users.

Alpha site tests are tests of a new software package that are performed at the developer’s site. The errors identified by alpha site tests are expected to include the errors that only use by a real user can reveal, and thus should be reported to the developer. Once an advanced version of the software package is available, the developer offers it free of charge to one or more potential users. The users install the package in their sites (usually called the “beta sites”), with the understanding that they will inform the developer of all the errors revealed during trials or regular usage.

Assuring the quality of software maintenance in SQA involves systematic, proactive activities—such as code reviews, regression testing, and configuration management—to ensure that modifications, bug fixes, and updates meet predefined standards. It focuses on enhancing maintainability, reliability, and efficiency while preventing the introduction of new defects.

Key Aspects of Assuring Maintenance Component Quality:

- **Objectives:** The primary goals are ensuring conformity to technical requirements, adherence to budgetary and scheduling constraints, and increasing overall efficiency in maintenance activities.
- **Key SQA Activities for Maintenance:**
  - **Regression Testing:** Ensuring that new changes (corrective, adaptive, perfective) do not negatively impact existing, functional software components.
  - **Configuration Management:** Managing, tracking, and controlling changes to the software's codebase and documentation.
  - **Code Reviews & Audits:** Reviewing modified code for compliance with standards, readability, and potential bugs.
  - **Risk Management:** Proactively identifying potential issues that could arise from changes.
- **Impact on Quality:** Quality assurance during maintenance reduces the total cost of ownership, prevents the degradation of software quality over time (software aging), and ensures long-term user satisfaction.

.

Assuring the quality of external participant contributions (subcontractors, vendors) in software engineering involves mitigating risks like low quality, delays, and integration issues through rigorous contract clauses, strict selection, and ongoing, structured monitoring. Key strategies include setting clear standards, using joint coordination committees, conducting thorough reviews (design/testing), and requiring regular progress reports to maintain control.
Key methods to assure the quality of external contributions include:

- **Contractual and Planning Controls:** Establishing clear, detailed requirements and acceptance criteria for deliverables before work begins.
- **Supplier Selection and Evaluation:** Choosing qualified subcontractors by assessing their competence and past performance, often requiring certification of their team leaders.
- **Active Monitoring and Reviews:** Implementing "coordination committees" to track progress and using formal reviews (design reviews, code walkthroughs) to detect issues early.
- **Integration and Testing:** Performing rigorous acceptance testing and independent verification of software components developed by external parties to ensure they meet quality standards.
- **Process Standardization:** Ensuring external participants adhere to the hiring organization's documentation, coding, and development standards.

CASE Tool and its Scope - Software Engineering

- A CASE (Computer power-assisted software package Engineering) tool could be a generic term accustomed to denote any type of machine-driven support for software package engineering. In a very additional restrictive sense, a CASE tool suggests that any tool accustomed to automatize some activity related to software package development.
  Several CASE tools square measure obtainable. A number of these CASE tools assist in part-connected tasks like specification, structured analysis, design, coding, testing, etc, and other non-phase activities like project management and configuration management.

Reasons for Using CASE Tools
The primary reasons for employing a CASE tool are:

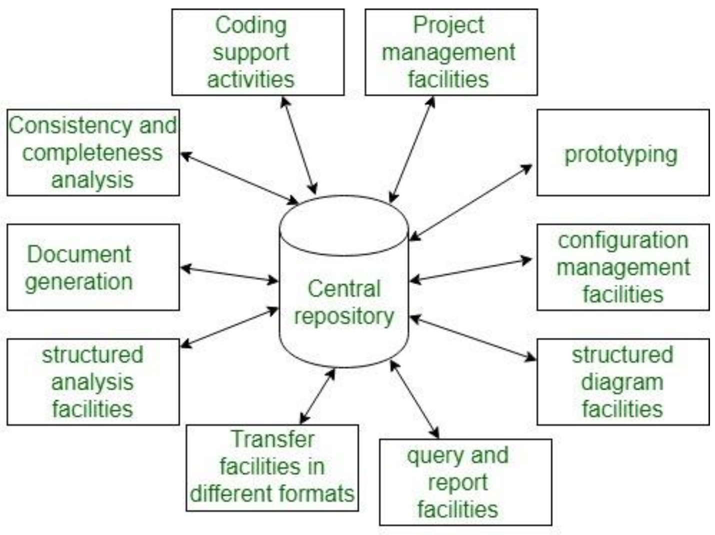
- To extend productivity
- To assist in turning out higher quality codes at a lower price

CASE Environment

Although individual CASE tools square measure helpful, the true power of a tool set is often completed only when this set of tools square measure integrated into a typical framework or setting.

1. CASE tools square measure characterized by the stage or stages of package development life cycle that they focus on.
2. Since different tools covering different stages share common data, it's needed that they integrate through some central repository to possess an even read of data related to the package development artifacts.
3. This central repository is sometimes information lexicon containing the definition of all composite and elementary data things.
4. Through the central repository, all the CASE tools in a very CASE setting share common data among themselves. therefore a CASE setting facilities the automation of the step-wise methodologies for package development.

A schematic illustration of a CASE setting is shown in the below diagram:


Coding support activities
Project management facilities
Consistency and completeness analysis
prototyping
Document generation
Central repository
configuration management facilities
structured analysis facilities
structured diagram facilities
Transfer facilities in different formats
query and report facilities

A CASE environment

Unit 4

SQA System – an SQA architecture

An SQA system combines a wide range of components to challenge the multitude of sources of software errors and to achieve an acceptable level of software quality. SQA system components can be classified into six classes:

The SQA Pre-project components improves the preparatory steps taken prior to initiating work on the project. The two main components are:

- Contract review.
- Development and quality plans.

Contract review activities include:

- Clarification of the customer's requirements
- Review of the project's schedule and resource requirement estimates
- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfill his obligations
- Evaluation of development risks.

Once a software development contract has been signed or a commitment made to undertake an internal project for the benefit of another department of the organization, a plan is prepared of the project ("development plan") and its integrated quality assurance activities ("quality plan"). These plans include additional details and needed revisions based on prior plans that provided the basis for the current proposal and contract.

The main issues treated in the project development plan are:

- Schedulesb Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans.

The main issues treated in the project's quality plan are:

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities.

The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage. Several SQA components enter the software development project life cycle at different points.

The main components are:

- Reviews
- Expert opinions

- Software testing
- Software maintenance
- Assurance of the quality of the
- subcontractors' work and the customer supplied parts.

In Formal Design Reviews (DRs),the design phase of the development process produces a variety of documents.The developer continues to the next phase of the development process only on receipt of formal approval of these documents. Adhoc committees examine the documents presented by development teams and carry out formal design reviews.Expert opinions support quality assessment efforts by introducing additional external capabilities into the organization's in-house development process.Turning to outside experts may be particularly useful in the following situations:

- Insufficient in-house professional capabilities in a given area.
- In small organizations, in many cases it is difficult to find enough suitable candidates to participate in the design review teams. In such situations, outside experts may join a DR committee or, alternatively, their expert opinions may replace a DR.
- In small organizations or in situations characterized by extreme work pressures, an outside expert's opinion can replace an inspection.
- Temporary inaccessibility of in-house professionals (waiting will cause substantial delays in the project completion schedule).
- In cases of major disagreement among the organization's senior professionals, an outside expert may support a decision.

Software tests are formal SQA components that are targeted towards review of the actual running of the software. Software testing programs are constructed from a variety of tests, some manual and some automated. The test report includes a detailed list of the faults detected and recommendations about the performance of partial following a subsequent round of corrections based on the test findings.

Software maintenance services vary in range and are provided for extensive periods, often several years. Maintenance services fall into the following categories:

- **Corrective maintenance** – User's support services and correction of software code and documentation failures.
- **Adaptive maintenance** – Adaptation of current software to new circumstances and customers without changing the basic software product

- **Functionality improvement maintenance** – The functional and performance-related improvement of existing software, carried out with respect to limited issues.

The motivation for turning to external participants lies in any number of factors, ranging from the economic to the technical to personnel related interests. Special software assurance efforts are required to establish effective controls over the external participant's work.
The main goal of Infrastructure Components for Error Prevention and Improvement is the prevention of software faults, the lowering of software fault rates, and improvement of productivity. This class of SQA components includes:

- Procedures and work instructions
- Supporting Quality devices
- Staff training, retraining, and certification
- Preventive and corrective actions
- Configuration management
- Documentation control

Corrective and Preventive Actions

Systematic activities that implement organization-wide improvements of effectiveness and operational efficiency fall under the heading of **Corrective and Preventive actions (CAPA)**. By promoting continuous improvement of effectiveness and efficiency, the CAPA process has become one of the main tools used to achieve the performance oriented objective of SQA. In corrective actions, a regularly applied feedback process that includes Collection of information on quality non-conformities, identification and analysis of sources of irregularities. Development and assimilation of improved practices and procedures, together with control of their implementation and measurement of their outcomes. In preventive actions a regularly applied feedback process that includes the collection of information on potential quality problems and analysis of departures from quality standards, development and assimilation of improved practices and procedures together with control of their implementation and measurement of their outcomes.

Successful operation of a CAPA process includes the following activities:

- Information collection
- Analysis of information
- Development of solutions and improved methods
- Implementation of improved methods

- Follow-up.

  The process is regularly fed by the flow of information from a variety of sources. In order to estimate the success of the process, a closed feedback loop is applied to control the flow of information, implementation of the resulting changes in practices and procedures together with measurement of the outcomes.

The variety of information sources, internal and external, that serve the CAPA process is quite remarkable. Following this internal/external dichotomy, the four main internal sources of information are the (1) Software development process, (2) Software maintenance, (3) SQA infrastructure and (4) Software quality management procedures. The initiation of inquiries into major project failures is almost instinctive. The conclusions reached by these inquiries affect a project's immediate environment; in many cases they also contribute to improved practices and procedures through the application of CAPA.

Regular operation of the CAPA process is expected to create a massive flow of documents related to a wide range of information. Analysis involves**:** **Screening the information and identifying potential improvements**: Documents received from the various sources of information are reviewed by professionals in order to identify potential opportunities for CAPA. **Analysis of potential improvements**: Efforts are directed to determine: – Expected types and levels of damage resulting from the identified fault. – Causes for faults. Typical causes are non-compliance with work instructions and procedures, insufficient technical knowledge, extreme time and/or budget pressures due to unrealistic estimates, and lack of experience with new development tools. **Generating feedback** on the content and regularity of information received from the designated information sources.

In the phase of development of solutions, solutions to identified causes of recurrent software systems faults are required to:

- Eliminate recurrence of the types of faults detected
- Contribute to improved efficiency by enabling higher productivity and shorter schedules.
- Changes in practices, including updating of relevant work instructions (if any exist).
- Shifting to a development tool that is more effective and less prone to the detected faults.
- Improvement of reporting methods, including changes in report content, frequency of reporting and reporting tasks.

- Initiatives for training, retraining or updating staff. This direction is taken only in cases when the same training deficiencies are found in several teams.

  Implementation of CAPA solutions relies on proper instructions and often training but most of all on the cooperation of the relevant units and individuals. Successful implementation requires that targeted staff members be convinced of the appropriateness of the proposed solution. Three main follow-up tasks are necessary for the proper functioning of a CAPA process in any organization:

- Follow-up of the flow of development and maintenance CAPA records
- Follow-up of implementation
- Follow-up of outcomes

  Follow-up of the flow of development and maintenance CAPA records from the various sources of information which enables feedback that reveals cases of no reporting as well as low-quality reporting, where important details are missing or inaccurate. Conducted mainly through analysis of long-term activity information, which generates feedback to the CAPA information sources. This activity is intended to indicate whether the designated actions – training activities, replacement of development tools, procedural changes (after approval) have been performed in practice. Adequate feedback is delivered to the bodies responsible for implementation of the corrective and preventive actions. Follow-up of the improved methods' actual outcomes enables assessment of the degree to which corrective or preventive actions have achieved the expected results. Feedback on the outcomes is delivered to the improved methods' developers. In cases of low performance, formulation of a revised or new corrective action is needed.

  Proper performance of these CAPA activities depends on the existence of a permanent core organizational unit generally known as Corrective Action Board (CAB). Members of the SQA unit, top-level professionals and development and maintenance department managers are the natural candidates for membership in a CAB committee. The various tasks of Corrective Action Board (CAB) includes:

- Collecting CAPA records from the various sources Screening the collected information
- Nominating entire adhoc CAPA teams to attend to given subjects, or heading some of the teams Promoting implementation of CAPA in units, projects, etc.

  Proper performance of these CAPA activities depends on the existence of a permanent core organizational unit as well as many ad hoc team participants. This nucleus, generally known as the Corrective Action Board (CAB) committee, although it may have other titles in different organizations, promotes the CAPA cause within the organization. Its tasks include:

Documentation control

Software documentation is written content that accompanies a software product to help developers, testers, users, and maintainers understand how the system works. It can include anything from API references and architecture notes to installation guides and user manuals.

Documentation is a critical part of the software development process because it supports

- Development
- Testing
- Maintenance
- Debugging
- Onboarding
- Long-Term Knowledge Transfer.

For example, before the development of any software product, requirements are documented, which is called the Software Requirement Specification (SRS). Requirement gathering is considered a stage of the Software Development Life Cycle (SDLC).

Another example can be a user manual that a user refers to for installing, using, and providing maintenance to the software application/product.

Types of Software Documentation

- **Requirement Documentation** : Describes what the software should do, expected behaviors, functional and non-functional requirements, and the environment in which it should operate. These documents guide developers and testers during development.
- **Architectural Documentation** : Focuses on the system's structure, components, data flow, interfaces, and interactions. It contains minimal code but provides a high-level view of how the system is designed.
- **Technical Documentation** : Prepared for developers and engineers. Includes API docs, algorithms, internal logic, database structures, configuration details, and code-level explanations.
- **End-user Documentation** : Targets end-users. Includes installation guides, FAQs, troubleshooting instructions, and user manuals.

Purpose of Documentation:

Software requirements and documentation must be created efficiently to achieve project goals. Documentation evolves over time because:

- Technology changes rapidly
- User needs change
- System environments evolve

Documentation also supports **verification, testing, prototyping**, team communication, and knowledge sharing.
For software engineers, reliable documentation is essential. It helps track application behavior, supports maintenance, improves software quality, and simplifies onboarding for new developers. Good documentation provides easy access to information, guides users, and reduces overall project costs.

Principles of Software Documentation:
While writing or contributing into any software documentation, one must keep in mind the following set of 7-principles :

1. Write from reader's point of view:

Understand who will read the documentation, developers, testers, stakeholders, or users. Use clear language, domain-specific terms where necessary, and organize content logically.

Best practices:

- Keep structure clean and easy to navigate
- Use a glossary for complex terms
- Avoid jargon unless necessary

2. Avoid unnecessary repetition:

Duplicate information leads to inconsistencies. Instead:

- Use hyperlinks or references
- Keep each piece of information in one place
- Ensure content remains maintainable

3. Avoid ambiguity:

All descriptions must be clear, precise, and consistent. If a term has multiple meanings, define it clearly to avoid misunderstanding among developers and stakeholders.

4. Follow a certain standard organization:

Stick to a recognized format or style guide. This makes the document professional, consistent, and easier to maintain. Use examples from existing industry-standard software documentation.

5. Record a Rationale

Rationale explain why certain design or implementation decisions were made. This helps future teams understand the reasoning behind choices, especially during debugging or system upgrades.

6. Keep the documentation updated but to an extent

Add new information only when relevant. Avoid cluttering documents with outdate or unnecessary details. Updates may reflect:

- Bug fixes
- New features
- Modified behaviour

7. Review documentation

Documentation can become large and complex. Regular reviews by architects, senior developers, and stakeholders ensure accuracy and alignment with the system's intended use.

Advantages of software documentation

- The presence of documentation helps in keeping the track of all aspects of an application and also improves the quality of the software product.
- The main focus is based on the development, maintenance, and knowledge transfer to other developers.
- Helps development teams during development.
- Helps end-users in using the product.
- Improves overall quality of software product
- It cuts down duplicative work.
- Makes easier to understand code.
- Helps in establishing internal coordination in work.

Software Quality metrics

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.

In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. A set of activities in SAQ is continuously applied throughout the software process. Software Quality is measured based on some software quality metrics.

There is a number of metrics available based on which software quality is measured. But among them, there are a few most useful metrics which are essential in software quality measurement. They are -

1. Code Quality
2. Reliability
3. Performance
4. Usability
5. Correctness
6. Maintainability
7. Integrity
8. Security

Now let's understand each quality metric **in detail -**

1. 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.

2. 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).

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

There is cost of activity in every project, it should have business value and software testing is no exception. To optimize its business value, test managers should optimize testing appropriately. There should not be unnecessary testing otherwise it causes unnecessary delays and ends up incurring more costs. Also, there should not be incomplete or too less testing otherwise, there may be a chance of defective products to be handed to the end users. Therefore, software testing should be done appropriately.

Cost of Quality : It is the most established, effective measure of quantifying and calculating the business value of testing. There are four categories to measure cost of quality: Prevention costs, Detection costs, Internal failure costs, and External failure costs. These are explained as follows below.

1. **Prevention costs** include cost of training developers on writing secure and easily maintainable code
2. **Detection costs** include the cost of creating test cases, setting up testing environments, revisiting testing requirements.

3. **Internal failure costs** include costs incurred in fixing defects just before delivery.
4. **External failure costs** include product support costs incurred by delivering poor quality software.

Major parts of total cost are detecting defects and internal failure cost. But, these costs less than external failure costs. That's why testing provides good business value. **Business value of software testing :** Test manager has responsibility to identify the business value and provide communications between teams and senior management. It concerns the cost of quality. To deliver business value, these are some of the measurable ways, like

1. Detecting defects
2. Documenting defects
3. Status reports and test metrics on project progress
4. Status reports on process implementation and product development
5. Increasing confidence in product quality
6. Legal liabilities
7. Decreasing risks
8. Ensuring predictable product
9. Enhancing reputation for product
10. Enhancing reputation for process quality
11. Testing can prevent mission failure