Module: 5

Risk Management

Lecture 1

1. Motivation:

The following are the motivational factors in risk management

- The prior knowledge of risks should be understandable to the developers and managers.
- The planning model should be developed such that risk cannot disturb the project.

2. Learning Objective:

This module explains that people involved in Software Risk Management will be able to understand the factors involved in risk management.

1. Students will be able to understand the Software Risk factors during the development of the software.
2. Students will be able to understand Risk Identification, Risk assessment, Risk projection.
3. Students will be able to understand the RMMM model.

3. Objective:

Risks are potential problems that might affect the successful completion of a software project. Risks involve uncertainty and potential losses. Risk analysis and management are intended to help a software team understand and manage uncertainty during the development process. The important thing is to remember that things can go wrong and to make plans to minimize their impact when they do. The work product is called a Risk Mitigation, Monitoring, and Management Plan (RMMM).

3. Prerequisite:

Knowledge of planning and estimation of software.

4. Syllabus:

Module	Content	Duration	Self Study Time
3.1	Risk Identification	1 lecture	1 hour
3.1	Risk assessment	1 lecture	1 hour
3.2	Risk projection	1 lecture	1 hour
3.3	RMMM	1 lecture	1 hour

5. Learning Outcomes:

Risks involve uncertainty and potential losses. Risk analysis and management are intended to help a software team understand and manage uncertainty during the development process. The important thing is to remember that things can go wrong and to make plans to minimize their impact when they do. The work product is called a Risk Mitigation, Monitoring and Management Plan (RMMM).

6. Weightage: Marks

7. Abbreviations:

8. Key Definitions:

risk: Risk is an undesired event or circumstance that occur while a project is underway

Uncertainty—the risk may or may not happen; that is, there are no 100% probable risks.

Loss—if the risk becomes a reality, unwanted consequences or losses will occur.

9. Theory

3.1 Risk management:

- It is necessary for the project manager to anticipate and identify different risks that a project may be susceptible to.

Risk Management- It aims at reducing the impact of all kinds of risk that may effect a project by identifying, analyzing and managing them. Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty.

Types of risks

- **Project risks** which threaten the project plan. if project risks become real, it is likely that the project schedule will slip and that costs will increase
- **Technical risks** are the risks that threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible.
- **Business risks** threaten the viability of the software to be built and often jeopardize the project or the product.

1.Risk identification

Risk identification is a attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks , the project manager can avoid these risks.

One method for identifying risks is to create a **risk item checklist.**

- ***Product size*—risks associated with the overall size of the software to be built or modified.**
- ***Business impact*—risks associated with constraints imposed by management or the marketplace.**
- ***Customer characteristics*—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.**
- ***Process definition*—risks associated with the degree to which the software process has been defined and is followed by the development organization.**

- ***Development environment*—risks associated with the availability and quality of the tools to be used to build the product.**
- ***Technology to be built*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.**
- ***Staff size and experience*—risks associated with the overall technical and project experience of the software engineers who will do the work.**

Risk Components and Drivers

The risk components are defined in the following manner:

Performance risk—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.

Cost risk—the degree of uncertainty that the project budget will be maintained.

Support risk—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.

Schedule risk—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic.

2. **Risk Projection**

The risk project has following steps

Impact Assessment:

Category \ Components		Performance	Support	Cost	Schedule
Catastrophic	1	Failure to meet the requirement would result in mission failure		Failure results in increased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable IOC
Critical	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in IOC
Marginal	1	Failure to meet the requirement would result in degradation of secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K	
	2	Minimal to small reduction in technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
Negligible	1	Failure to meet the requirement would create inconvenience or nonoperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in technical performance	Easily supportable software	Possible budget underrun	Early achievable IOC

Creating a Risk Table

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
•				
•				
•				

Impact values

The RMMM plan

The RMMM plan consists of risk avoidance, risk monitoring, risk management and contingency planning

risk mitigation is a problem avoidance activity.

Risk monitoring is a project tracking activity with three primary objectives:

(1) to assess whether predicted risks do, in fact, occur;

(2) to ensure that risk aversion steps defined for the risk are being properly applied; and

(3) to collect information that can be used for future risk analysis.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/09.			
Current status: 5/12/09: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

10. Objective Questions:

1. The most important feature of spiral model is
   (A) requirement analysis. (B) **risk management**. (C) quality management.
   (D) configuration management.

2. The RMMM plan consists of
(A) risk avoidance,
(B) risk monitoring,
(C) risk management and contingency planning
(D) **all of above**

11. Subjective Questions:

1. Describe the difference between "known risks" and "predictable risks.

2. Explain RMMM plan

12. Learning Resources:

1. Roger Pressman, "Software Engineering", sixth edition, Tata McGraw Hill.
2. Bernd Bruegge, "Object oriented software engineering", Second Edition, Pearson Education.

Software Configuration Management

1. Motivation:

Presented chapter is the foundation of basic software engineering and therefore the motivation of this course is to understand the software engineering process involved with SCM.

2. Learning Objective:

This module explains that people involved in the Software Configuration Management. will be able to understand the factors involved in the configuration management.

1. Students will be able to understand the Software configuration management process, Identification of objects in software configuration.
2. Students will be able to understand Managing and controlling changes, Managing and controlling versions.
3. Students will be able to understand the configuration audit, status reporting, SCM standards and SCM issues.

3. Objective:

This module explains set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them,

defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made.

4. Prerequisite:

Workflow of different phases of software life cycle

5. Syllabus:

Prerequisites	Syllabus	Duration	Self Study
Basic Project Knowledge	4.1 Software configuration management ,process, Identification of objects in software configuration	2 Hrs	2 Hrs
	4.2 Managing and controlling changes, Managing and controlling versions		
	4.3 configuration audit, status reporting, SCM standards and SCM issues.	2 Hrs	2 Hrs

5. Learning Outcomes:

- Students will be able to do the software configuration and also able to Identify the objects in software configuration.
- Students will be able to Manage and control changes during the configuring process.
- Students will be able to understand the configuration audits and prepare reports of the status.
- Students will be able to understand software engineering process involved with SCM.

6. Weightage: 10 marks

7. Abbreviation

(1) SCM – Software Configuration Management
(2) SCI - Software Configuration Management Item
(3) CCA - Change Control Authority
(4) ECO - Engineering Change Order

8. Key Definition:

Baselines: A baseline is a software configuration management concept that helps us to control change without seriously impeding justifiable change.

Software Configuration Objects: To control and manage configuration items, each must be named and managed using an object-oriented approach. Basic objects are created by software engineers during analysis, design, coding, or testing.

Version Control: Version control is to identify and manage project elements as they change over time.

Change Control: Change control refers to the policy, rules, procedures, information, activities, roles, authorization levels and states relating to creation, updates, approvals, tracking and archiving of items involved with the implementation of a change process.

9. Theory

5.1 Software Configuration Management

General Points

- Changes are inevitable when software is built.
- A primary goal of software engineering is to improve the ease with which changes can be made to software.
- Configuration management is all about change control.
- Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project.
- To ensure that quality is maintained the change process must be audited.
- Software configuration management is an important element of software quality assurance.
- Its primary responsibility is the control of change.
- Software configuration management (SCM) is an umbrella activity that is applied throughout the software process
- Five SCM tasks:

    - Identification (tracking multiple versions to enable efficient changes)
    - Version control (control changes before and after release to customer)
    - Change control (authority to approve and prioritize changes)
    - Configuration auditing (ensure changes made properly)
    - Reporting (tell others about changes made)

Difference between SCM and Software Support (Maintenance)

- Support is a set of software engineering activities that occur after software has been delivered to the customer and put into operation.
- Software configuration management is a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

Software Configuration Management Items

- The output of the software process is information that may be divided into three broad categories:
  - Computer programs (both source level and executable forms);
  - Documents that describe the computer programs (targeted at both technical practitioners and users),
  - Data (contained within the program or external to it).

- The items that comprise all information produced as part of the software process are collectively called a software configuration.
- As the software process progresses, the number of software configuration items (SCIs) grows rapidly.
- A System Specification spawns a Software Project Plan and Software Requirements Specification (as well as hardware related documents).
- These in turn spawn other documents to create a hierarchy of information. If each SCI simply spawned other SCIs, little confusion would result. Unfortunately, another variable enters the process—change.
- In the extreme, a SCI could be considered to be a single section of a large specification or one test case in a large suite of tests.
- More realistically, an SCI is a document, a entire suite of test cases, or a named program component (e.g., a C++ function or an Ada package).
- In addition to the SCIs that are derived from software work products, many software engineering organizations also place software tools under configuration control.
- That is, specific versions of editors, compilers, and other CASE tools are "frozen" as part of the software configuration.
- Because these tools were used to produce documentation, source code, and data, they must be available when changes to the software configuration are to be made.
- Although problems are rare, it is possible that a new version of a tool (e.g., a compiler) might produce different results than the original version.
- For this reason, tools, like the software that they help to produce, can be baselined as part of a comprehensive configuration management process.
- In reality, SCIs are organized to form configuration objects that may be cataloged in the project database with a single name.

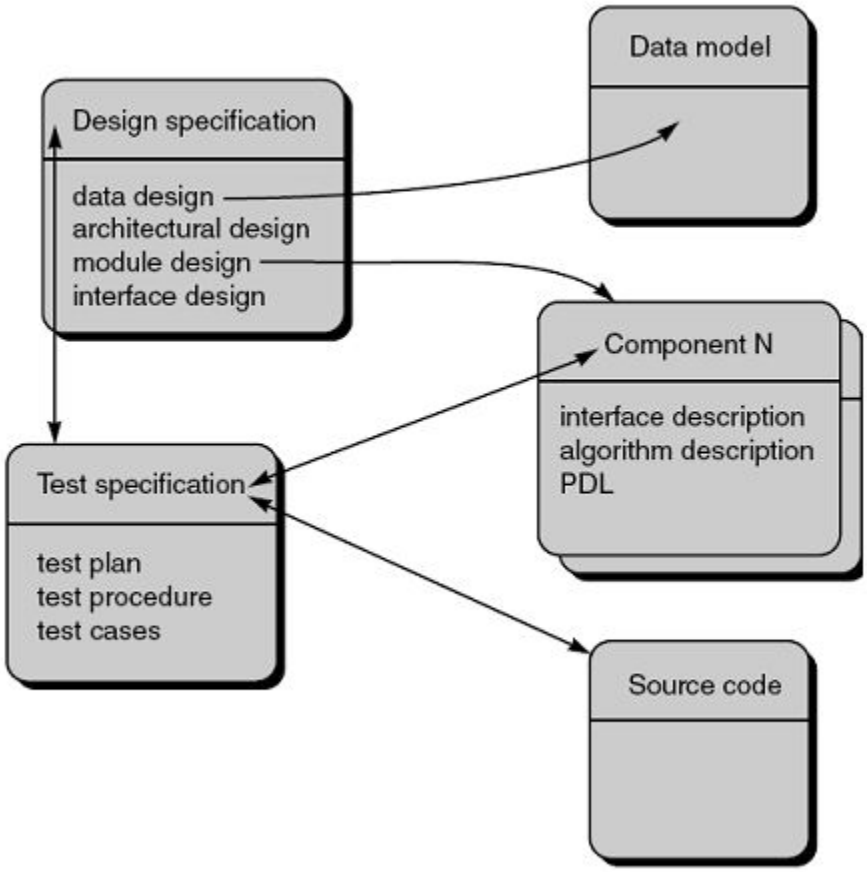

Figure 5.1 Configuration objects

- A configuration object has a name, attributes, and is "connected" to other objects by relationships.
- Referring to Figure, the configuration objects, Design Specification, data model, component N, source code and Test Specification are each defined separately.
- However, each of the objects is related to the others as shown by the arrows. A curved arrow indicates a compositional relation.
- That is, data model and component N are part of the object Design Specification.
- A double-headed straight arrow indicates an interrelationship.
- If a change were made to the source code object, the interrelationships enable a software engineer to determine what other objects (and SCIs) might be affected.

5.1.1 Managing and controlling changes

Fundamental Sources of Change

- New business or market conditions dictate changes in product requirements or business rules.
- New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
- Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
- Budgetary or scheduling constraints cause a redefinition of the system or product.

Baselines

- A baseline is a software configuration management concept that helps us to control change without seriously impeding justifiable change.
- A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

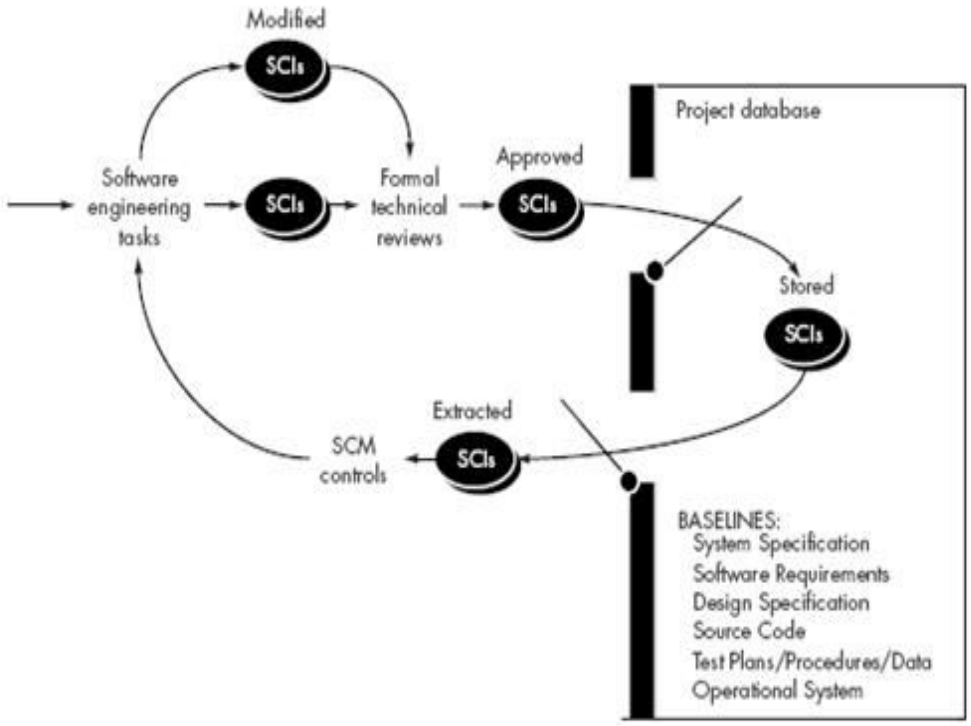

Figure 5.2 Baselined SCIs and the project database

- Before a software configuration item becomes a baseline, change may be made quickly and informally.

- However, once a baseline is established, we figuratively pass through a swinging one way door.
- Changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.
- A work product becomes a baseline only after it is reviewed and approved.
- A baseline is a milestone in software development that is marked by the delivery of one or more configuration items.
- Once a baseline is established each change request must be evaluated and verified by a formal procedure before it is processed.
- The progression of events that lead to a baseline is also illustrated in Figure.

- Software engineering tasks produce one or more SCIs.
- After SCIs are reviewed and approved, they are placed in a project database (also called a project library or software repository).
- When a member of a software engineering team wants to make a modification to a baselined SCI, it is copied from the project database into the engineer's private work space.
- However, this extracted SCI can be modified only if SCM controls (discussed later in this chapter) are followed.
- The arrows in Figure 5.2 illustrate the modification path for a base lined SCI.

Description of main tasks in detail

1. Software Configuration Objects

- To control and manage configuration items, each must be named and managed using an object-oriented approach.
- **Basic objects** are created by software engineers during analysis, design, coding, or testing.
- For example, a basic object might be a section of a requirements specification, a source listing for a component, or a suite of test cases that are used to exercise the code.
- **Aggregate objects** are collections of basic objects and other aggregate objects. Design Specification is an aggregate object.
- Configuration object attributes:

  - unique name,
  - description,
  - list of resources,
  - realization (a pointer to a work product for a basic object or null for an aggregate object)

- An entity-relationship (E-R) diagram can be used to show the interrelationships among the objects

2. Version Control

- Configuration management allows a user to specify alternative configurations of the software system through the selection of appropriate versions.
- This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes.
- These "attributes" mentioned can be as simple as a specific version number that is attached to each object or as complex as a string of Boolean variables (switches) that indicate specific types of functional changes that have been applied to the system
- One representation of the different versions of a system is the evolution graph presented in Figure 5.3.
- Each node on the graph is an aggregate object, that is, a complete version of the software.
- Each version of the software is a collection of SCIs (source code, documents, data), and each version may be composed of different variants.
- To illustrate this concept, consider a version of a simple program that is composed of entities 1, 2, 3, 4, and 5
- Entity 4 is used only when the software is implemented using color displays. Entity 5 is implemented when monochrome displays are available. Therefore, two variants of the version can be defined: (1) entities 1, 2, 3, and 4; (2) entities 1, 2, 3, and 5.
- To construct the appropriate variant of a given version of a program, each entity can be assigned an "attribute-tuple"—a list of features that will define whether the entity should be used when a particular variant of a software version is to be constructed.
- One or more attributes is assigned for each variant. For example, a color attribute could be used to define which entity should be included when color displays are to be supported.
- Another way to conceptualize the relationship between entities, variants and versions (revisions) is to represent them as an object pool. Referring to Figure 5.4, the relationship between configuration objects and entities, variants and versions can be represented in a three-dimensional space.
- An entity is composed of a collection of objects at the same revision level. A variant is a different collection of objects at the same revision level and therefore coexists in parallel with other variants.
- A new version is defined when major changes are made to one or more objects.
- A number of different automated approaches to version control have been proposed over the past decade. The primary difference in approaches is the sophistication of the attributes that are used to construct specific versions and variants of a system and the mechanics of the process for construction.

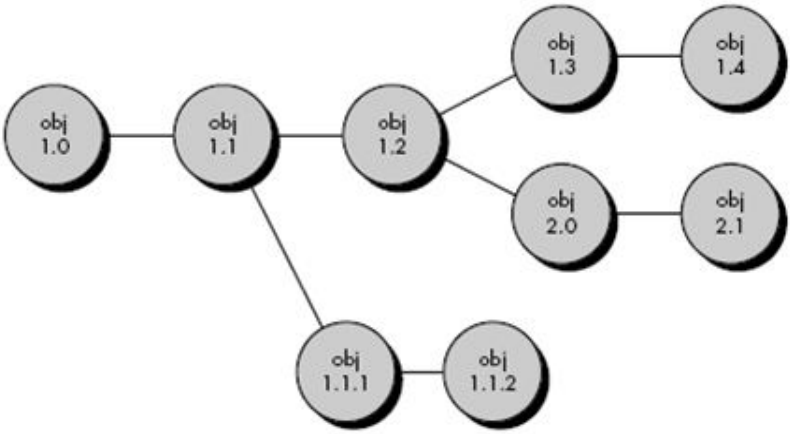

Figure 5.3 Evolution graph

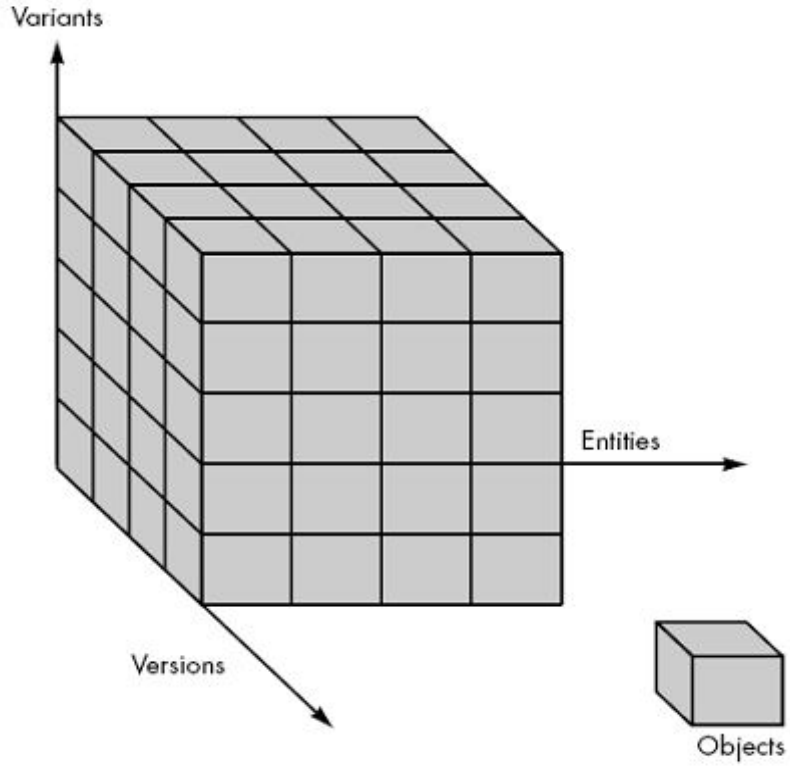

Figure 5.4 Object pool representation of components, variants and versions

3. Change Control

- Change request is submitted and evaluated to assess technical merit and impact on the other configuration objects and budget

- Change report contains the results of the evaluation
- Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report
- Engineering change order (ECO) is generated for each change approved (describes change, lists the constraints, and criteria for review and audit)
- Object to be changed is checked-out of the project database subject to access control parameters for the object
- Modified object is subjected to appropriate SQA and testing procedures
- Modified object is checked-in to the project database and version control mechanisms are used to create the next version of the software
- Synchronization control is used to ensure that parallel changes made by different people don't overwrite one another

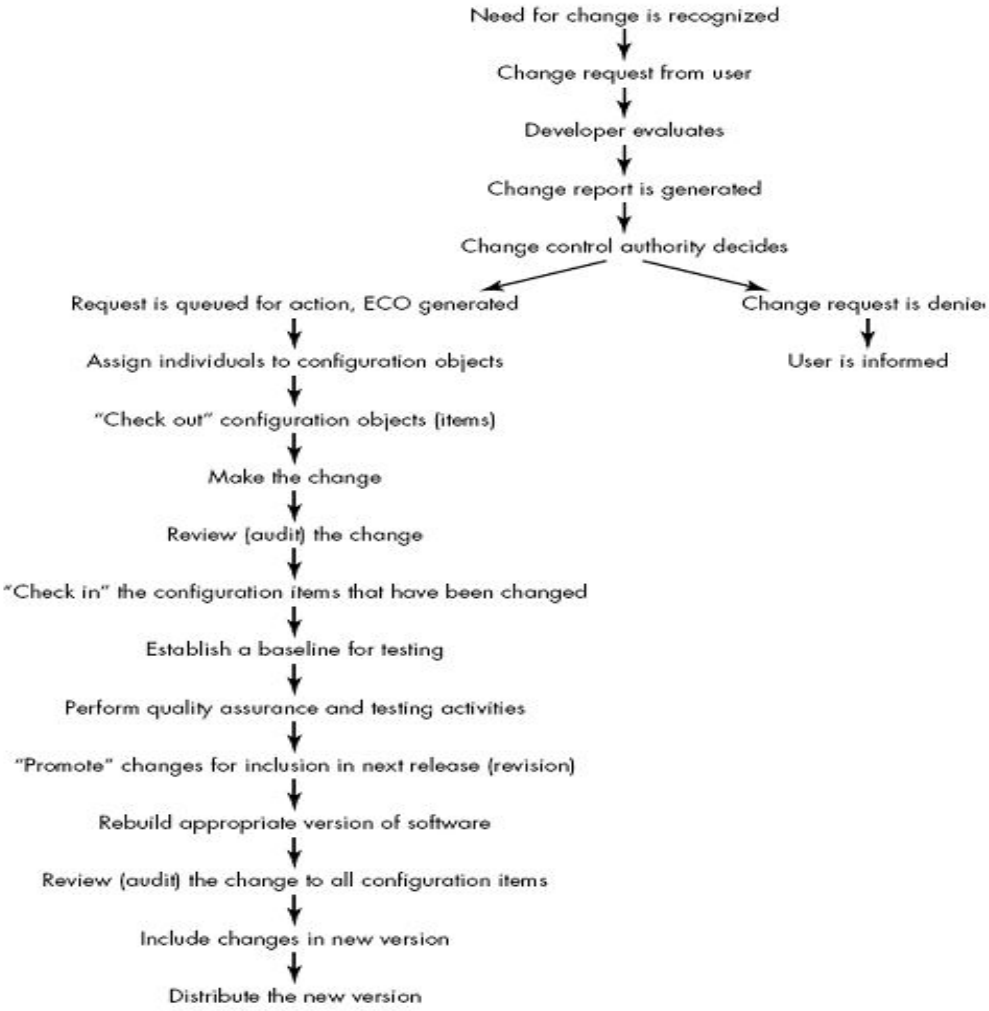

Figure 5.5: The change control process

4. Configuration Audit

- Identification, version control, and change control help the software developer to maintain order in what would otherwise be a chaotic and fluid situation.
- To ensure that the change has been properly implemented we conduct :
  - formal technical reviews and
  - the software configuration audit.
- The **formal technical review** focuses on the technical correctness of the configuration object that has been modified.
- The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects.
- A formal technical review should be conducted for all but the most trivial changes.
- A **software configuration audit** complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review.
- The audit asks and answers the following questions:
  - Has the change specified in the ECO (Engineering change order) been made? Have any additional modifications been incorporated?
  - Has a formal technical review been conducted to assess technical correctness?
  - Has the software process been followed and have software engineering standards been properly applied?
  - Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
  - Have SCM procedures for noting the change, recording it, and reporting it been followed?
  - Have all related SCIs been properly updated?
- In some cases, the audit questions are asked as part of a formal technical review.
- However, when SCM is a formal activity, the SCM audit is conducted separately by the quality assurance group.

5. Status Reporting

- Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions:
  - What happened?
  - Who did it?
  - When did it happen?
  - What else will be affected?
- Each time an SCI is assigned new or updated identification, a CSR entry is made.
- Each time a configuration audit is conducted, the results are reported as part of the CSR task.
- Output from CSR may be placed in an on-line database, so that software developers or maintainers can access change information by keyword category.
- In addition, a CSR report is generated on a regular basis and is intended to keep management and practitioners appraised of important changes.
- Configuration status reporting plays a vital role in the success of a large software development project.
- When many people are involved, it is likely that "the left hand not knowing what the right hand is doing" syndrome will occur.
- Two developers may attempt to modify the same SCI with different and conflicting intents.
- A software engineering team may spend months of effort building software to an obsolete hardware specification.

Importance of version and change control

- Version control is used to combine procedures and tools to manage different versions of configurations objects that are made during software development.
- Change control adds human procedure with the automated tools so that effective control of change will be made.

Difference

Version Control	Change Control
Version control is to identify and manage project elements as they change over time.	Change control refers to the policy, rules, procedures, information, activities, roles, authorization levels and states relating to creation, updates, approvals, tracking and archiving of items involved with the implementation of a change process.

5.1.2 Managing and controlling version

Refer page no.195-197

10. References:

1) Roger Pressman, Software Engineering: A Practitioners Approach, (6th Edition), McGraw Hill, 1997.

2) I. Sommerville, Software Engineering, 7th edition, Adison Wesley, 1996.

11. Objective Questions

1. Which of these are valid software configuration items?
   a) software tools
   b) documentation
   c) executable programs
   d) test data
   e) all of the above
2. Which of the following is not considered one of the four important elements that should exist when a configuration management system is developed?
   a) Component elements
   b) human elements
   c) process elements
   d) validation elements
3. Once a software engineering work product becomes a baseline it cannot be changed again

   a) True b) False

4. Which configuration objects would not typically be found in the project database?

   a) design specification
   b) marketing data
   c) organizational structure description
   d) test plans
   e) both b and c
5. Modern software engineering practice suggests that a software team maintain SCI's in a project database or repository
   a) True b) False
6. A data repository meta model is used to determine how
   a) information is stored in the repository

b) data integrity can be maintained
c) the existing model can be extended
d) All of the above

7. Many data repository requirements are the same as those for a typical database application.
   a) True b) False

8. The ability to track relationships and changes to configuration objects is one of the most important features of the SCM repository.

   a) True b) False

9. Which of the following tasks is not part of software configuration management?
   a) change control
   b) reporting
   c) statistical quality control
   d) version control

10. A basic configuration object is a _____ created by a software engineer during some phase of the software development process.
    a) program data structure
    b) a software component
    c) unit of information
    d) all of the above

11. A new _____ is defined when major changes have been made to one or more Configuration objects
    a) Entity
    b) Item
    c) Variant
    d) Version

12. Change control is not necessary if a development group is making use of an automated project database tool
    a) True b) False

13. When software configuration management is a formal activity, the software configuration audit is conducted by the
    a) development team
    b) quality assurance group
    c) senior managers
    d) testing specialists

14. The primary purpose of configuration status reporting is to
    a) allow revision of project schedules and cost estimates by project managers
    b) evaluate the performance of software developers and organizations
    c) make sure that change information is communicated to all affected parties

d) none of the above

15. Configuration issues that need to be considered when developing Web Apps include:
    a) Content
    b) Cost
    c) People
    d) Politics
    e) a, b, and c

(Ans : 1-E,2-B,3-B,4-E,5-A,6-B,7-B,8-A,9-C,10-D,11-D,12-B,13-B,14-C,15-E)

12. Subjective questions

1. What is SCM? Explain in detail?
Ans: Refer page no.190-193
2. Explain in detail about change control?
Ans: Refer page no.193-195, 198
3. Explain in detail about version control?
Ans: Refer page no.195-197

13. University Questions

1. Write short note on Software Configuration Management. [May 2010] [10 marks]
Ans:

It is an umbrella activity that is applied throughout the software process.

- Software Configuration Objects
- Software Configuration Management Items

2. Explain Software Configuration Management and Change Control Management in detail. [Nov. 2010] [10 marks]

Software Quality Assurance

1.1 . Motivation:

Presented chapter is the foundation of basic software engineering and therefore motivation of this course is to understand software quality assurance process is involved with software.

1.2 Learning Objective:

This module explains the meaning of Software Quality Assurance, with the help of concepts that define the quality of a software product.

Student will be able to understand importance of Quality Assurance during the development of the software.

1. Students will be able to understand Software Quality Assurance Concepts and the various Software standards.
2. Students will be able to understand the Quality metrics and how to improve Software Reliability.
3. Students will be able to understand the Quality Measurement and Metrics.

1.3 Objective:

The main objective is to introduce to the students about the product that is to be engineered and the process that provides a framework for the engineering technology.

1. To provide knowledge of software engineering discipline.
2. To analyze risk in software design and quality.
3. To introduce the concept of advance software methodology.

1.4 Prerequisite:

Workflow of different phases of software life cycle

1.5 Learning Outcomes:

1. The Student will be able to learn Software Quality Assurance Concepts and Software standards.
2. The Student will be able to learn Quality Measurement and Metrics.
3. The Student will be able to learn to improve Software Reliability.

6.6 Syllabus:

Prerequisites	Syllabus	Duration	Self Study
Basic Knowledge of Quality	6.1 Software Quality Assurance – Software standards	2 Hrs	2 Hrs
	6.2 Quality metrics Software Reliability		
	6.3 Quality Measurement and Metrics	2 Hrs	2 Hrs

Software Quality

Software quality conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

The definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.

2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. There is a set of implicit requirements that often goes unmentioned (e.g., the desire for ease of use). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

Software quality is a complex mix of factors that will vary across different applications and the customers who request them.

McCall's Quality Factors:

The factors that affect software quality can be categorized in two broad groups: (1) factors that can be directly measured (e.g., defects per function-point) and (2) factors that can be measured only indirectly (e.g., usability or maintainability). In each case measurement must occur. The software must be compared (documents, programs, data) to some datum and an indication of quality is arrived at.

McCall, Richards, and Walters propose a useful categorization of factors that affect software quality. These software quality factors, shown in the figure below, focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.

Referring to the factors noted in the figure below, McCall and his colleagues provide the following descriptions:

Correctness. The extent to which a program satisfies its specification and fulfills the customer's mission objectives.

Reliability. The extent to which a program can be expected to perform its intended function with required precision.

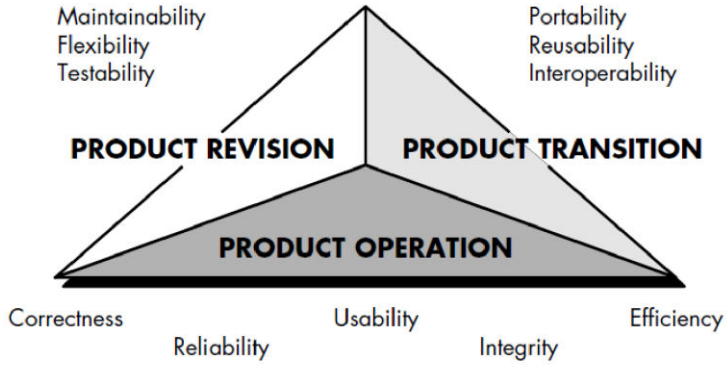

Figure 4.30: McCall's software quality factors

Efficiency. The amount of computing resources and code required by a program to perform its function.

Integrity. Extent to which access to software or data by unauthorized persons can be controlled.

Usability. Effort required to learn, operate, prepare input, and interpret output of a program.

Maintainability. Effort required to locate and fix an error in a program.

Flexibility. Effort required to modify an operational program.

Testability. Effort required to test a program to ensure that it performs its intended function.

Portability. Effort required to transfer the program from one hardware and/or software system environment to another.

Reusability. Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.

Interoperability. Effort required to couple one system to another.

It is difficult, and in some cases impossible, to develop direct measures of these quality factors. In fact, many of the metrics defined by McCall et al. can be measured only subjectively. The metrics may be in the form of a checklist that is used to grade specific attributes of the software.

Quality Standards

A quality assurance system may be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications. These systems cover a wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process. ISO 9000 describes quality assurance elements in generic

terms that can be applied to any business regardless of the products or services offered.

The ISO 9000 standards have been adopted by many countries including all members of the European Community, Canada, Mexico, the United States, Australia, New Zealand, and the Pacific Rim. Countries in Latin and South America have also shown interest in the standards.

After adopting the standards, a country typically permits only ISO registered companies to supply goods and services to government agencies and public utilities. Telecommunication equipment and medical devices are examples of product categories that must be supplied by ISO registered companies. In turn, manufacturers of these products often require their suppliers to become registered. Private companies such as automobile and computer manufacturers frequently require their suppliers to be ISO registered as well.

To become registered to one of the quality assurance system models contained in ISO 9000, a company's quality system and operations are scrutinized by third party auditors for compliance to the standard and for effective operation. Upon successful registration, a company is issued a certificate from a registration body represented by the auditors. Semi-annual surveillance audits ensure continued compliance to the standard.

The ISO approach to software quality systems:

The ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes. For a quality system to be ISO compliant, these processes must address the areas identified in the standard and must be documented and practiced as described.

ISO 9000 describes the elements of a quality assurance system in general terms. These elements include the organizational structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance, and quality improvement. However, ISO 9000 does not describe how an organization should implement these quality system elements. Consequently, the challenge lies in designing and implementing a quality assurance system that meets the standard and fits the company's products, services, and culture.

The ISO 9001 Standard:

ISO 9001 is the quality assurance standard that applies to software engineering. The standard contains 20 requirements that must be present for an

effective quality assurance system. Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guidelines (ISO 9000-3) have been developed to help interpret the standard for use in the software process.

The requirements delineated by ISO 9001 address topics such as management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, corrective and preventive action, control of quality records, internal quality audits, training, servicing, and statistical techniques. In order for a software organization to become registered to ISO 9001, it must establish policies and procedures to address each of the requirements just noted (and others) and then be able to demonstrate that these policies and procedures are being followed.

ISO 9126 Quality Factors:

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

Functionality. The degree to which the software satisfies stated needs as indicated by the following subattributes: suitability, accuracy, interoperability, compliance, and security.

Reliability. The amount of time that the software is available for use as indicated by the following sub-attributes: maturity, fault tolerance, recoverability.

Usability. The degree to which the software is easy to use as indicated by the following subattributes: understandability, learnability, operability.

Efficiency. The degree to which the software makes optimal use of system resources as indicated by the following sub-attributes: time behavior, resource behavior.

Maintainability. The ease with which repair may be made to the software as indicated by the following subattributes: analyzability, changeability, stability, testability.

Portability. The ease with which the software can be transposed from one environment to another as indicated by the following subattributes: adaptability, installability, conformance, replaceability.

Like McCall's quality factors, the ISO 9126 factors do not necessarily lend themselves to direct measurement. However, they do provide a worthwhile basis for indirect measures and an excellent checklist for assessing the quality of a system.

Quality Metrics

The overriding goal of software engineering is to produce a high-quality system, application, or product. To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process. In addition, a good software engineer (and good software engineering managers) must measure if high quality is to be realized.

The quality of a system, application, or product is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program, and the tests that exercise the software to uncover errors. A good software engineer uses measurement to assess the quality of the analysis and design models, the source code, and the test cases that have been created as the software is engineered. To accomplish this real-time quality assessment, the engineer must use technical measures to evaluate quality in objective, rather than subjective ways.

The project manager must also evaluate quality as the project progresses. Private metrics collected by individual software engineers are assimilated to provide project level results. Although many quality measures can be collected, the primary thrust at the project level is to measure errors and defects. Metrics derived from these measures provide an indication of the effectiveness of individual and group software quality assurance and control activities.

Metrics such as work product (e.g., requirements or design) errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric. Error data can also be used to compute the *defect removal efficiency* (DRE) for each process framework activity.

Measuring quality:

Although there are many measures of software quality, correctness, maintainability, integrity, and usability provide useful indicators for the project team. Gilb suggests definitions and measures for each.

Correctness. A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function. The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements. When considering the overall quality of a software product, defects are those problems reported by a user of the program after the program has been released for general use. For quality assessment purposes, defects are counted over a standard period of time, typically one year.

Maintainability. Software maintenance accounts for more effort than any other software engineering activity. Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements. There is no way to measure maintainability directly; therefore, we must use indirect measures. A simple time-oriented metric is *mean-time-to-change* (MTTC), the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users. On average, programs that are maintainable will have a lower MTTC (for equivalent types of changes) than programs that are not maintainable.

Hitachi has used a cost-oriented metric for maintainability called *spoilage*—the cost to correct defects encountered after the software has been released to its end-users. When the ratio of spoilage to overall project cost (for many projects) is plotted as a function of time, a manager can determine whether the overall maintainability of software produced by a software development organization is improving. Actions can then be taken in response to the insight gained from this information.

Integrity. Software integrity has become increasingly important in the age of hackers and firewalls. This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security. Attacks can be made on all three components of software: programs, data, and documents.

To measure integrity, two additional attributes must be defined: threat and security. *Threat* is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time. *Security* is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled. The integrity of a system can then be defined as

integrity = summation [(1 – threat) × (1 – security)]

where threat and security are summed over each type of attack.

Usability. The catch phrase "user-friendliness" has become ubiquitous in discussions of software products. If a program is not user-friendly, it is often doomed to failure, even if the functions that it performs are valuable. Usability is an attempt to quantify user-friendliness and can be measured in terms of four characteristics: (1) the physical and or intellectual skill required to learn the system, (2) the time required to become moderately efficient in the use of the system, (3) the net increase in productivity (over the approach that the system replaces) measured when the system is used by someone who is moderately efficient, and (4) a subjective assessment (sometimes obtained through a questionnaire) of users attitudes toward the system.

4.3.3 Testing & SQA

Different testing strategies have been discussed in the previous section. The following paragraphs explain Software Quality Assurance.
SQA (Software Quality Assurance):

Even the most jaded software developers will agree that high-quality software is an important goal. Many definitions have been proposed in the literature. One such definition is follows: "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." The definition serves to emphasize the following three important points:

- Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
- Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
- A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

SQA activities:

Software quality assurance is composed of a variety of tasks associated with two different constituencies—the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Software engineers address quality (and perform quality assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing.

The charter of the SQA group is to assist the software team in achieving a high quality end product. The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are performed (or facilitated) by an independent SQA group that:

*Prepares an SQA plan for a project***.** The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the

software engineering team and the SQA group are governed by the plan. The plan identifies:

- evaluations to be performed
- audits and reviews to be performed
- standards that are applicable to the project
- procedures for error reporting and tracking
- documents to be produced by the SQA group
- amount of feedback provided to the software project team

Participates in the development of the project's software process description. The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.
Reviews software engineering activities to verify compliance with the defined software process. The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
Audits designated software work products to verify compliance with those defined as part of the software process. The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.
Ensures that deviations in software work and work products are documented and handled according to a documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

Records any noncompliance and reports to senior management. Noncompliance items are tracked until they are resolved.

In addition to these activities, the SQA group coordinates the control and management of change and helps to collect and analyze software metrics.

Difference between testing and SQA:

Software quality is the degree to which a system, component, or process meets specified requirements and customer or user needs or expectations.

Quality assurance and quality control both contribute in delivering a high quality software product though the way they go about it is different. This can be illustrated by looking at the definitions of the two.

Software Quality Assurance:-

Software QA involves the entire software development process - monitoring and improving the process, making sure that any agreed-upon standards and

procedures are followed, and ensuring that problems are found and dealt with. It is oriented to 'prevention'.

This is a 'staff' function, and is responsible for establishing standards and procedures to prevent defects and breakdowns in the SDLC. The focus of QA is prevention, processes, and continuous improvement of these processes.

Software Quality Control:-

This is a department function, which compares the standards to the product, and takes action when non-conformance is detected for example testing.

This involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented to 'detection'.

Relationship between Quality Control and Quality Assurance:-

An application that meets its requirements totally can be said to exhibit quality. Quality is not based on a subjective assessment but rather on a clearly demonstrable, and measurable, basis.

- Quality Control is a process directed at validating that a specific deliverable meets standards, is error free, and is the best deliverable that can be produced. It is a responsibility internal to the team.

- Quality Assurance (QA), on the other hand, is a review with a goal of improving the process as well as the deliverable. QA is often an external process. QA is an effective approach to producing a high quality product.

One aspect is the process of objectively reviewing project deliverables and the processes that produce them (including testing), to identify defects, and then making recommendations for improvement based on the reviews. The end result is the assurance that the system and application is of high quality, and that the process is working. The achievement of quality goals is well within reach when organizational strategies are used in the testing process. From the client's perspective, an application's quality is high if it meets their expectations.

10. Objective Questions:

7. A key concept of quality control is that all work products
    a. are delivered on time and under budget
    b. have complete documentation
    c. have measurable specifications for process outputs
    d. are thoroughly tested before delivery to the customer
8. People who perform software quality assurance must look at the software from the customer's perspective.
    a. True
    b. False
9. Which of these activities is not one of the activities recommended to be performed by an independent SQA group?
    a. prepare SQA plan for the project
    b. review software engineering activities to verify process compliance
    c. report any evidence of noncompliance to senior management
    d. serve as the sole test team for any software produced
10. The purpose of software reviews is to uncover errors in work products so they can be removed before moving on to the next phase of development
    a. True
    b. False
11. In general the earlier a software error is discovered and corrected the less costly to the overall project budget.
    a. True
    b. False
12. Which of the following are objectives for formal technical reviews?
    a. allow senior staff members to correct errors
    b. assess programmer productivity
    c. determining who introduced an error into a program
    d. uncover errors in software work products
13. At the end of a formal technical review all attendees can decide to
    a. accept the work product without modification
    b. modify the work product and continue the review
    c. reject the product due to stylistic discrepancies
    d. reject the product due to severe errors
    e. both a and d
14. In any type of technical review, the focus of the review is on the product and not the producer.

a. True
b. False

15. The ISO quality assurance standard that applies to software engineering is
a. ISO 9000:2004
b. ISO 9001:2000
c. ISO 9002:2001
d. ISO 9003:2004

16. Which of the following is not a section in the standard for SQA plans recommended by IEEE?
a. budget
b. documentation
c. reviews and audits
d. test

Answers: 1. d 2. b 3. e 4. a 5. a 6. d 7. c 8. a 9. d 10. a 11. a 12. d 13. e 14. a 15. b 16. a

11. Subjective Questions:

2. Explain about software reviews? Differentiate inspection and walkthrough?
6. What is software quality? Explain Software Quality Assurance?
7. Write short notes on Quality Standards?

12. University Questions:

1. What are the five of the most important attributes of software quality? Explain them.
(May 2010) (10 marks)