Chapter 5 Software RISK, Configuration Management and Quality assurance

1.1 Risk and Risk Management

■ Introduction about Risk

■ **Risk : A risk is a potential problem – it might happen and it might not**

- Conceptual definition of risk
  - Risk concerns future happenings
  - **Risk involves change in mind, opinion, actions, places, etc.**
  - Risk involves choice and the uncertainty that choice entails

■ Two characteristics of risk

- **Uncertainty** – the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)
- **Loss** – the risk becomes a reality and unwanted consequences or losses occur

■ Types of Risk

- **Project risks**
  - They threaten the project plan
  - If they become real, it is likely that the project schedule will slip and that costs will increase
  - Project risks **identify potential problems inbudgetary, schedule, personnel** (staffing and organization), **resource, customer, and requirements and their impact** on a software project.

- **Project complexity, size, and the degree of structural uncertainty**

- **Technical risks**
  - They threaten the quality and timeliness of the software to be produced
  - If they become real, implementation may become difficult or impossible
  - If a technical risk becomes a reality, **implementation** may become difficult or impossible.
  - Technical risks identify potential problems in **design, implementation, interface, verification, and maintenance problems.**
  - In addition, **specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology** are also risk factors.
  - Technical risks occur because the problem is harder to solve than we thought it would be.

- **Business risks**
  - They threaten the viability of the software to be built
  - Sub-categories of Business risks
    - **Market risk** – building an excellent product or system that no one really wants
    - **Strategic risk** – building a product that no longer fits into the overall business strategy for the company
    - **Sales risk** – building a product that the sales force doesn't understand how to sell
    - **Management risk** – losing the support of senior management due to a change in focus or a change in people
    - **Budget risk** – losing budgetary or personnel commitment

- **Known risks**

- ***Known risks*** are those that can be uncovered after careful **evaluation of the project plan**, the **business and technical environment** in which the project is being developed, and other reliable **information sources** (e.g., **unrealistic delivery date**, lack of **documented requirements** or **software scope**, poor **development environment**).
-

- **Predictable risks**
  - Those risks that are extrapolated from past project experience (e.g., past turnover)
- **Unpredictable risks**
  - Those risks that can and do occur, but are extremely difficult to identify in advance

■ risk strategies

- **Reactive risk strategies**
  - "Don't worry, I'll think of something"
  - ***The majority of software teams and managers rely on this approach***
  - ***Nothing is done about risks until something goes wrong***
    - The team then flies into action in an attempt to correct the problem rapidly (fire fighting)
  - Crisis management is the choice of management techniques

- **Proactive risk strategies**
  - Primary objective is to **avoid risk** and ***to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner***
  - **Potential risk**s are identified,
  - Their **probability and impact** are assessed, and
  - They are **ranked** by importance.

- Then, the software team **establishes a plan** for managing risk.
- The **primary objective** is **to avoid risk**, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner.

■ Risk Management

Steps:

- **Identify** possible risks; recognize what can go wrong
- **Analyze** each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur
- **Rank** the risks by probability and impact
  - Impact may be negligible, marginal, critical, and catastrophic
- **Develop** a contingency plan to manage those risks having high probability and high impact

■ Risk Analysis

- Risk analysis includes series of steps that gives software development team an opportunity to identify, understand and manage involved risks.
- Risk analysis refers to the identification and understanding of such problems and attempt to find out steps that need to be taken in case risk arries during the development process.
- **The process of risk analysis includes the following steps:**
  1. **Risk identification**
  2. **Risk Assessment**
  3. **Risk Management**
     a) Avoiding the risk
     b) Reducing damage caused by the risk
     c) Accepting the risk

1) RISK IDENTIFICATION:

Risk identification is a systematic attempt to **specify threats** to the project plan (**estimates**, **schedule, resource loading**, etc.).

By identifying known and predictable risks, the project manager takes a **first step** toward **avoiding them** when possible and c**ontrolling** them when necessary.

- ***Generic risks*** are a potential threat to every software project.
- ***Product-specific risks*** can be identified only by those with a clear understanding of the **technology**, the **people, and the environment** that is specific to the project at hand.
  - ➢ To identify product-specific risks, the **project plan** and the software **statement of scope** are examined and an answer to the following question is developed:
    - ✓ "What special characteristics of this product may threaten our project plan?"

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

  The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic **subcategories:**

  – ***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.

1. **Risk projection (or Assessment) attempts to rate each risk in two ways**
   - The probability that the risk is real
   - The consequence of the problems associated with the risk, should it occur
   - The project planner, managers, and technical staff perform four risk projection steps

   a. Establish a scale that reflects the likelihood of a risk (e.g., 1-low, 10-high)

   b. define the consequences of the risk

   c. Estimate the impact of the risk on the project and product

   d. Note the overall accuracy of the risk projection

   - **Developing Risk Table: A risk table** provides a project manager with a simple technique for risk projection

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:
1—catastrophic
2—critical
3—marginal
4—negligible

-
- It consists of five columns
  - Risk Summary – short description of the risk
  - Risk Category – one of seven risk categories
  - Probability – estimation of risk occurrence based on group input
  - Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
  - RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

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

3) Risk Management:

- Refers to the steps that need to be applied after identifying and analyzing a potential risk.
- When risk is identified and its probability of occurrence and impact is known, we can find out ways to manage the involved risk.
- Risk management can be carried out in 3 ways:
  - ✓ Avoiding the risk: implementing the ways to avoid the risk completely. The risk can be avoided by enhancing and improving the existing resources and methods

- ✓ Reducing damage caused by the risk: In case if risk is possible, some ways can be developed so that the impact of the risk can be minimized.
- ✓ Accepting the risk: refers to accepting the risk involved in the software development. New resources are sometimes brought to the project to manage the potential risks.

■ Risk Mitigation, Monitoring and Management (RMMM Plan):

- The RMMM plan may be a part of the software development plan or may be a separate document.
  - Risk mitigation is a problem **avoidance activity**
  - Risk monitoring is a project **tracking activity**

  Risk monitoring has three objectives
    - ✓ To assess whether predicted risks do, in fact, occur
    - ✓ To ensure that risk aversion steps defined for the risk are being properly applied
    - ✓ To collect information that can be used for future risk analysis
- The findings from risk monitoring may allow the project manager to ascertain what risks caused which problems **throughout the project**
- **Risk mitigation (avoidance) is the primary strategy and is achieved through a plan**
- **Example: Risk of high staff turnover**

  That is, high turnover will have **a critical impact on project cost and schedule.**

  1) **To mitigate this risk, project management must develop a strategy for reducing turnover.**

     Among the possible steps to be taken is Meeting with current staff to determine causes for turnover (e.g., poor working conditions, low pay, and competitive job market). Mitigate those causes that are under our control before the project starts.

Once the project commences, assume turnover will occur and **develop techniques to ensure continuity when people leave**. **Organize project teams so that information about each development activity is widely dispersed. Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner. Conduct peer reviews of all work** (so that more than one person is "up to speed"). Assign a backup staff member for every critical technologist.

2) **As the project proceeds**, **risk monitoring activities commence**.

- ✓ **The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely.**
- ✓ In the case of high staff turnover, the following factors can be monitored:
  - General attitude of team members based on project pressures. Interpersonal relationships among team members.
  - Potential problems with compensation and benefits.
  - The availability of jobs within the company and outside it.
- ✓ In addition to monitoring these factors, the project manager should monitor the effectiveness of risk mitigation steps.

3) ***Risk management and contingency planning*** **assumes** that mitigation efforts have failed and that the risk has become a reality.

Examples of Risks:

1. Estimation and scheduling

The unique nature of individual software projects creates problems for developers and managers in estimating and scheduling development time. Always monitor existing projects so that you apply lessons learnt in the future.

2. Sudden growth in requirements

As a project progresses, issues that are not identified earlier can create a last-minute hurdle to meeting deadlines. Try to think big early on in the project, and anticipate the worst-case or heaviest-use scenario.

1. Employee turnover

Every project has a number of developers working on it. When a developers leaves, he or she may take critical information with him/her. This can delay, and sometimes derail an entire project. Ensure you have resources where team members can collaborate and share knowledge.

4. Breakdown of specification

During the initial phases of integration and coding, requirements might conflict. Moreover, developers may find that even the specification is unclear or incomplete.

5. Productivity issues

On projects involving long timelines, developers tend to take things easy to begin with. As a result, sometimes, they lose significant time to complete the project. Set a realistic schedule, and stick to it.

2. Compromising on designs

In order to get stuck into the next 'real' tasks, developers tend to rush the design-process. This is a waste of programming hours, as designing is the most critical part of software development.

7. Gold plating

Developers sometimes like to show off their skills by adding unnecessary features. For instance, a developer might add Flash to a basic login module to make it look 'stylish'. Again, this is a waste of programming hours.

3. Tight Schedules

Often, project managers face the pressure of having to deliver the project earlier than anticipated. This can happen due to many different reasons – lack of resources, poor planning, or even technical glitches.

What can you do?

Sometimes, all you really need is more resources. Project managers call this strategy 'crashing the schedule' where more team members are tagged to the critical paths of the project. This reduces the time required significantly, but on the flip side, it can also create chaos if not handled properly. However, finding the right engineers can be quite a task. This is where some companies outsource their software development to external vendors for the duration of the project. **Alternatively, going offshore and building your own development team and scaling it quickly, can also prove to be beneficial, especially in the long run.** That being said, one must also be aware of the risks in offshore software development, especially when engaging with the wrong partner.

4. Budget Changes

Budget changes are a common occurrence, especially in software development projects. **A common reason why this happens is 'scope creep'.** Scope creep is when you start off your project with a set of clear, well-defined requirements, but as you approach the finish line, so many requirements are added or deleted that all you're left with is one big mess.

Other times, your engineers may fail to understand the specific requirements of the project and add extra functionalities in an attempt to deliver an improvised version of the initial project. **Even if the client didn't ask for it, they are left with a better product but with a bigger price tag attached to it**.

What can you do?

Double-check that your team is aware of the exact requirements of the project. Talk to them about the change control process and how they should only work on what is approved. If the list of requirements keeps climbing, it's time to take a step back and analyse which changes will lead to a better outcome, and which ones won't. **The key is to stay as close to the original goals and objectives of the project, without compromising on output quality.**

1.2 Software configuration management (SCM)

- ***Software configuration management*** **(SCM)** is an **umbrella activity** that is applied throughout the software process. Because change can occur at any time, **SCM activities** are developed to
  (1) Identify change,
  (2) Control change,
  (3) Ensure that change is being properly implemented, and
  (4) Report changes to others who may have an interest.

- **The items that comprise all information produced as part of the software process are collectively called a *software configuration.***

- SCM is a process of tracking and maintaining changes in the software.

- The purpose of SCM is to record and report all crucial information regarding the status of development process.

- 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.***
- Another variable enters the process—***change***. Change may occur at any time, for any reason
- There are **four fundamental sources** of change: delivered by products, or services delivered by a computer-based system.
  - ✓ **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
  - ✓ **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.

- **Software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software.**

- SCM can be viewed as a **software quality assurance** activity that is applied throughout the software process.

■ Baselines:

- Change is a fact of life in software development.
- **Customers** want to modify requirements.
- **Developers** want to modify the technical approach.
- **Managers** want to modify the project strategy.

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

- **A baseline is 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.

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

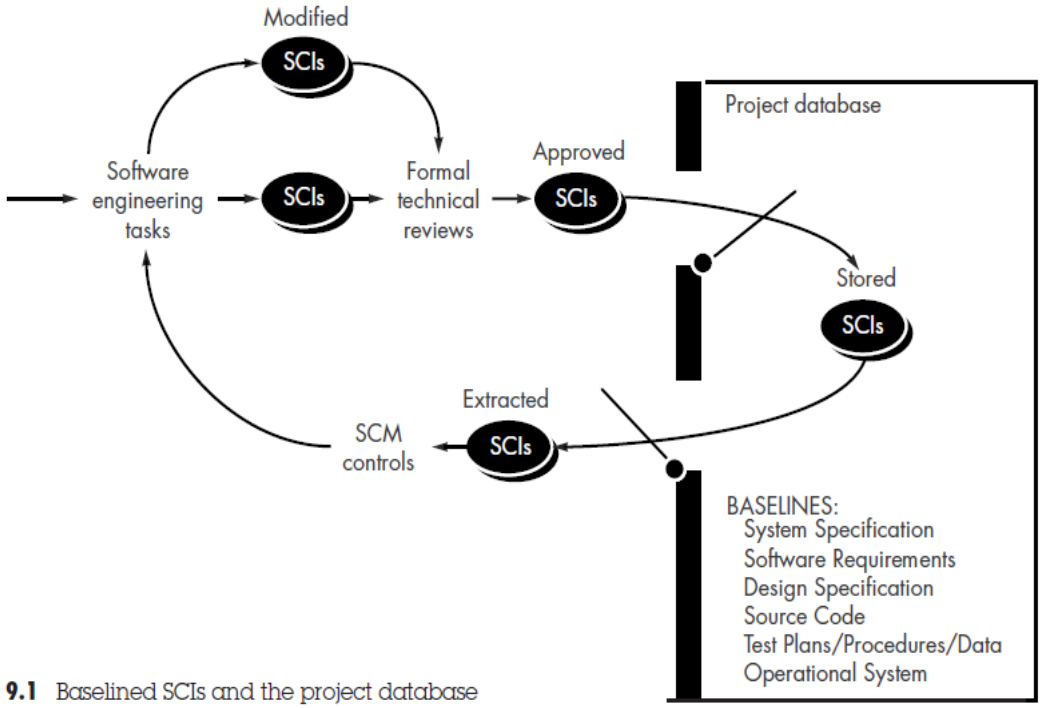


FIGURE 9.1 Baselined SCIs and the project database

- A **baseline** is a **milestone** in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review.

(**FTR:** is a SQA activity that is performed by software engineers, objectives of FTR are:

- ✓ Identify errors in function, logic or implementation
- ✓ To verify software under review meets its requirements.

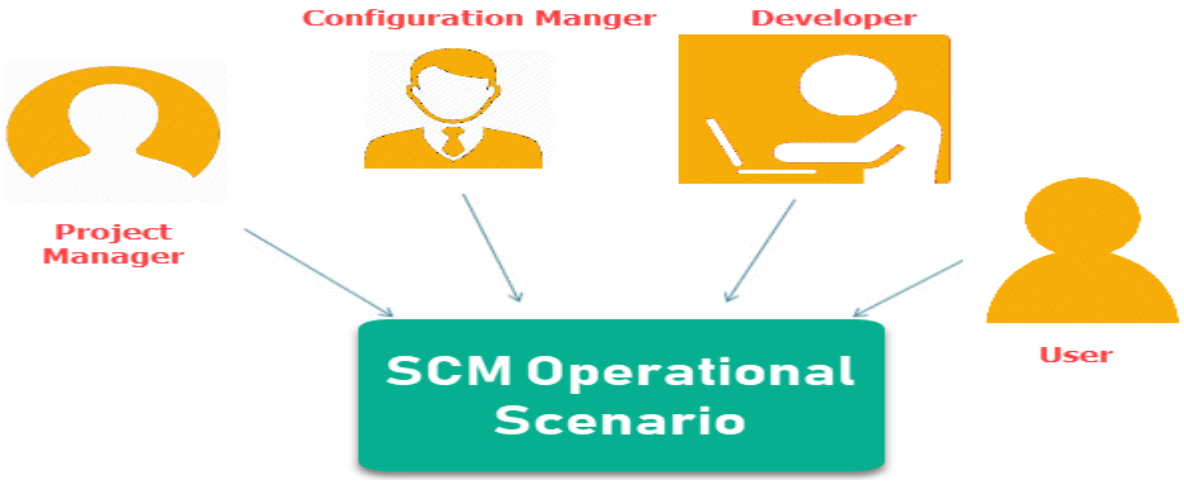
- ✓ To ensure that software has been represented according to predefined standards
- ✓ To achieve software that is developed in uniform manner.

)

- 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 are followed

■ Participant of SCM process:

Following are the key participants in SCM



1. Configuration Manager

- Configuration Manager is the head who is Responsible for identifying configuration items.
- CM ensures team follows the SCM process
- He/She needs to approve or reject change requests

2. Developer

- The developer needs to change the code as per standard development activities or change requests. He is responsible for maintaining configuration of code.
- The developer should check the changes and resolves conflicts

3. Auditor

- The auditor is responsible for SCM audits and reviews.
- Need to ensure the consistency and completeness of release.

4. Project Manager:

- Ensure that the product is developed within a certain time frame
- Monitors the progress of development and recognizes issues in the SCM process
- Generate reports about the status of the software system
- Make sure that processes and policies are followed for creating, changing, and testing

5. User

The end user should understand the key SCM terms to ensure he has the latest version of the software

THE SCM PROCESS:

- **Software configuration management** is an important element of **software quality assurance**. Its primary responsibility is the **control of change**.
- SCM is also responsible for the **identification of individual SCIs** and various **versions** of the software, the **auditing** of the software configuration to **ensure** that it has been properly **developed**, and the **reporting of all changes** applied to the configuration.

- **Any discussion of SCM introduces a set of complex questions:**

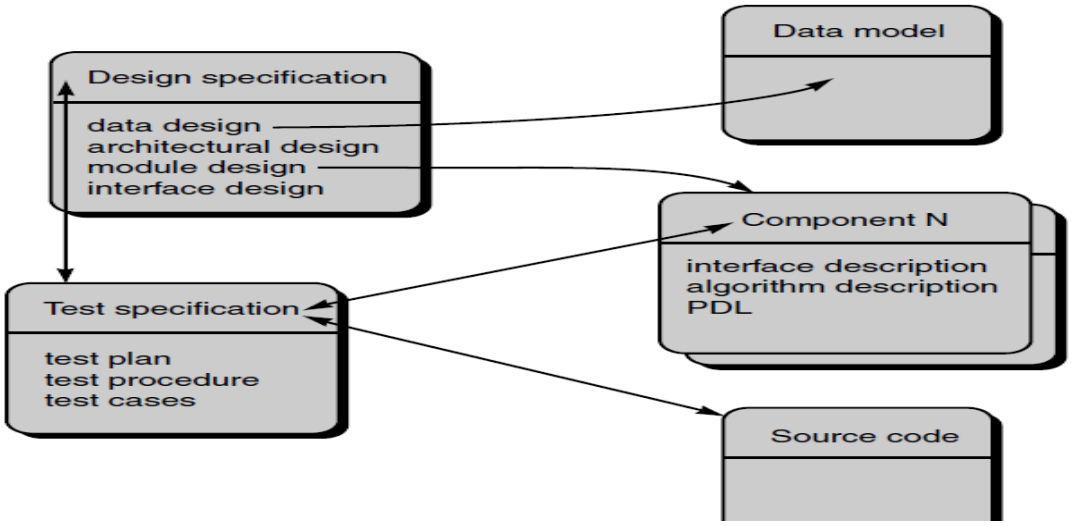
  - ✓ How does an organization **identify and manage the many existing versions of a program** (and its documentation) in a manner that will enable change to be accommodated efficiently?
  - ✓ How an organization control changes does **before and after software is released** to a customer?
  - ✓ Who has responsibility for **approving and ranking changes**?
  - ✓ How can we ensure that **changes have been made properly**?
  - ✓ What mechanism is used to **apprise others of changes** that are made?

- **These questions lead us to the definition of five SCM tasks:**
  1. ***Identification***
  2. ***version control***
  3. ***change control***
  4. ***configuration auditing***
  5. ***Reporting.***

1. IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

- Identify all configuration items (SCI's), check documentation of each and every SCI. The status of SCI at a given point of time is called as a baseline.
- To control and manage software configuration items, each must be separately named and then organized using an object-oriented approach.
- SCIs are organized to form ***configuration objects*** that may be cataloged in the project database with a single name.

  - ✓ A **configuration object** has a name, attributes, and is "connected" to other objects by relationships. The configuration objects like **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.



Two types of objects can be identified:

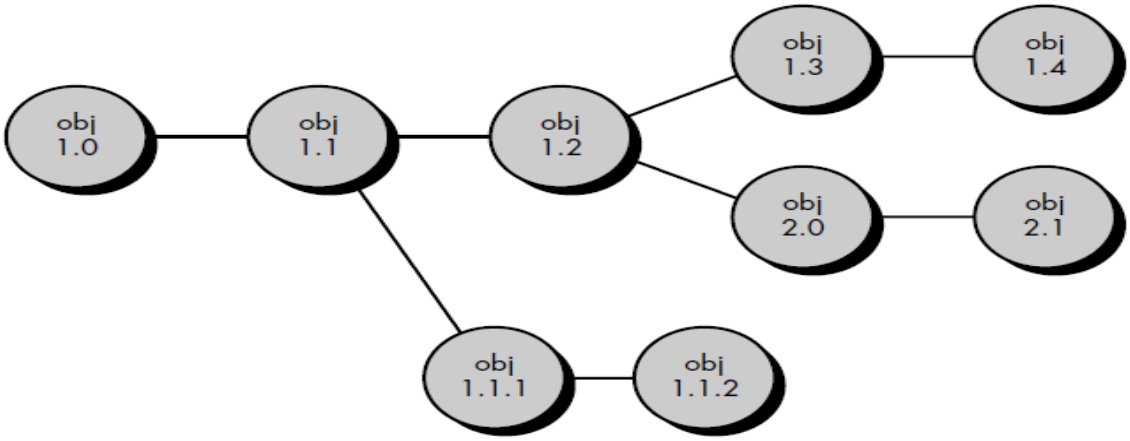
Basic object* and *aggregate object

A **basic object** is a "**unit of text**" that has been created by a software engineer during analysis, design, code, or test. 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.

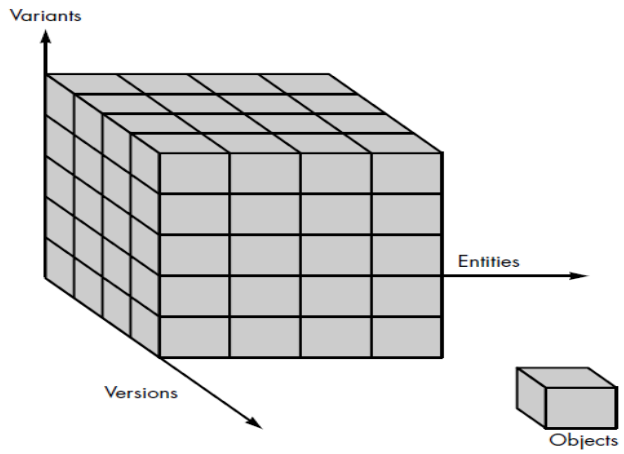
An **aggregate object** is a **collection** of basic objects and other aggregate objects. **Design Specification** is an aggregate object. Conceptually, it can be viewed as a named (identified) list of pointers that specify basic objects such as **data model** and **component N**.

2. VERSION CONTROL:

- ***Version control*** combines **procedures and tools** to **manage different versions** of **configuration objects** that are created during the software process.



- Software Configuration Management allows us to specify alternative configuration of software system, through the selection of appropriate versions. This is supported by associating attributes with each version.



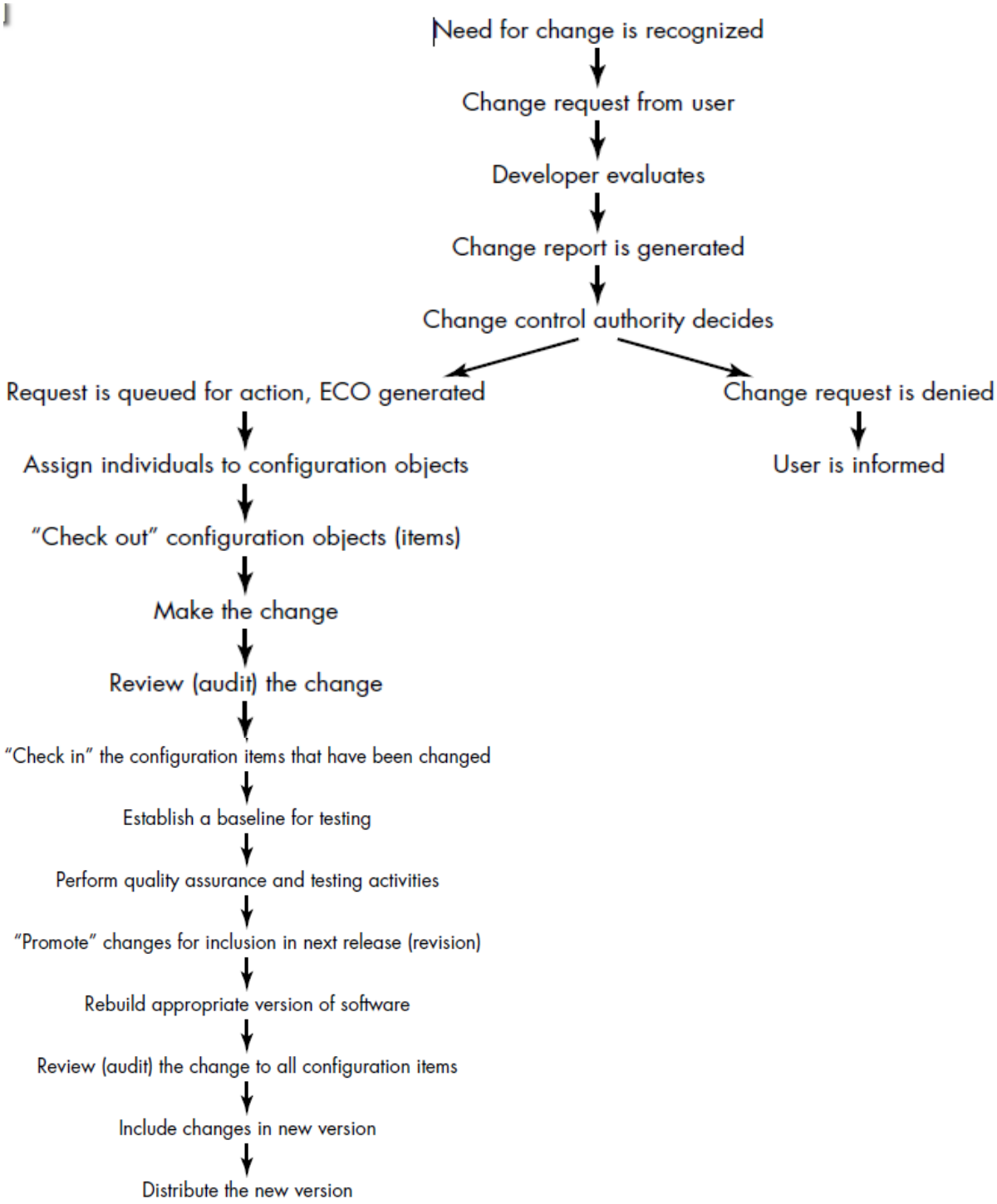
- **Example:** Single program composed of entities 1, 2, 3, 4, and 5.
  Entity 4 is used only when the software is implemented using color displays.
  Entity 5 is implemented when monochrome displays are available.
  Therefore, two variants of the version can be defined:
  (1) Entities 1, 2, 3, and 4 (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 0more 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.

3. CHANGE CONTROL:

- For a large software engineering project, uncontrolled change rapidly leads to difficulties. For such projects, **change control combines human procedures and automated tools to provide a mechanism for the control of change**.

- **A *change request* is submitted** and **evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.**
- **The results of the evaluation are presented as a *change report*,** which is used by a *change control authority* (CCA)—**a person or group who makes a final decision on the status and priority of the change.**
- **An *engineering change order* (ECO) is generated for each approved change. The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit.**
- **The object to be changed is "checked out" of the project database, the change is made, and appropriate SQA activities are applied.**
- **The object is then "checked in" to the database** and appropriate version control mechanisms are used to create the next version of the software.
- **The "check-in" and "check-out" process implements two important elements of change control—access control and synchronization** control.
  - ✓ *Access control* governs which software engineers have the authority to access and modify a particular configuration object.
  - ✓ *Synchronization control* helps to ensure that parallel changes, performed by two different people, don't overwrite one another.

- Based on an approved change request and ECO, a software engineer *checks out* a configuration object. An access control function ensures that the software engineer has authority to check out the object, and synchronization control *locks* the object in the project database so that no updates can be made to it until the currently checked out version has been replaced. Note that other copies can be checked-out, but other updates cannot be made. A copy of the baselined object, called the *extracted version,* is modified by the software engineer. After appropriate SQA and testing, the modified version of the object is *checked in* and the new baseline object is unlocked.

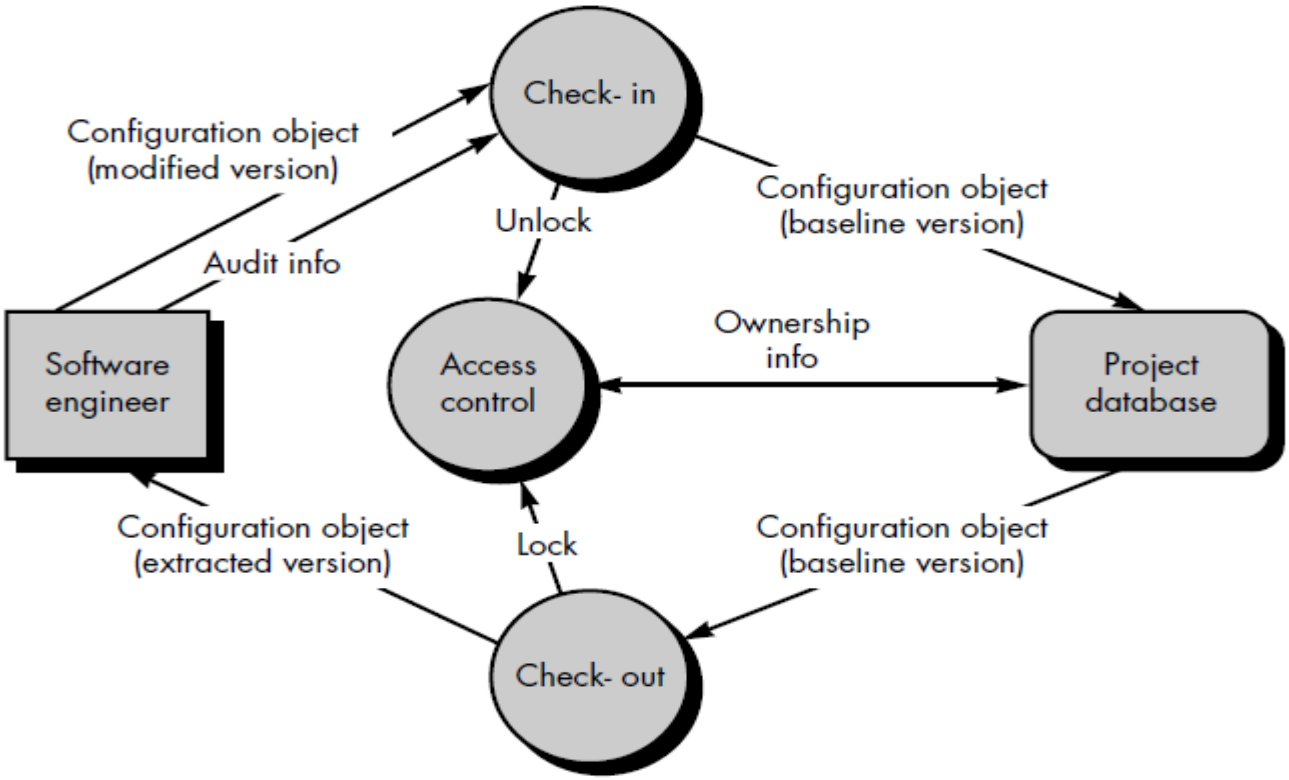


Prior **to an SCI becoming a baseline, only *informal change control* need be applied**. The developer of the configuration object (SCI) in question may make whatever changes

are justified by project and technical requirements. Once the object has **undergone formal technical review and has been approved**, a baseline is created. Once an **SCI becomes a baseline**, ***project level change control*** is implemented. Now, to make a change, the developer must gain approval from the project manager or from the CCA if the change affects other SCIs.

When the software product **is released to** customers, ***formal change control*** is instituted. The formal change control procedure has been outlined in Figure.
The change control authority plays an active role in the second and third layers control. Depending on the size and character of a software project, the **CCA** may be composed of one person—the **project manager—or a number of people** (e.g., representatives from software, hardware, database engineering, support, marketing).

The **role of the CCA** is to take a global view, that is, to assess the impact of change beyond the SCI in question.
How will the change affect hardware?
How will the change affect performance?
How will the change modify customer's perception of the product?
How will the change affect product quality and reliability?



4. CONFIGURATION AUDIT:

Configuration Audit is a quality assurance function which evaluates level of compliance achieved with specification and standards.

Identification, version control, and change control help the software developer to maintain order. However, even the most successful control mechanisms track a change only until an ECO is generated. How can we ensure that the change has been properly implemented?

The answer is twofold: (1) formal technical reviews and (2) 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:

1. Has the change specified in the ECO been made? Have any additional modifications been incorporated?

2. Has a formal technical review been conducted to assess technical correctness?

3. Has the software process been followed and have software engineering standards been properly applied?

4. 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?

5. Have SCM procedures for noting the change, recording it, and reporting it been followed?

6. 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-CSR):

Configuration status reporting (sometimes called ***status accounting***) is an SCM task that answers the following questions:

(1) What happened?
(2) Who did it?
(3) When did it happen?
(4) What else will be affected?

The flow of information for configuration status reporting (CSR) is shown in diagram. Each time an SCI is assigned new or updated identification, a CSR entry is made. Each time a change is approved by the CCA (i.e., an ECO is issued), 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. **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.

The person who would recognize serious side effects for a proposed change is not aware that the change is being made.

CSR helps to eliminate these problems by improving communication among all people involved.

5.3 Software Quality Assurance (SQA)

The **SQA** organizational **role is to review the product(s) at specific times during product implementation.** Upon reviewing, the **SQA** team's **duties** will be to evaluate the software at its current development stage and recognize any defects in the subsequent stage (design or implementation).

Software Quality Assurance (**SQA**) consists of the means **to ensure the quality of the released software by monitoring the software engineering methods and processes. SQA spans across the entire software development lifecycle that includes requirements management, software design, coding, testing, and release management.**

Software Quality Assurance (SQA) is simply a way to assure quality in the software. **It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.**

Software Quality Assurance is a process which works parallel to development of a software. **It focuses on improving the process of development of software so that problems can be prevented before they become a major issue**.

Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

OR

SQA is planned and systematic method for evaluation of the quality of

- ✓ Software Product
- ✓ Software Process
- ✓ Software standards
- ✓ Software Procedures

The goal of SQA is to provide management with data necessary to inform about the quality of products.

There may be separate SQA groups who evaluate quality of product, process etc.

Software Quality Assurance has:

1 A quality management approach
2 Formal technical reviews
3 Multi testing strategy
4 Effective software engineering technology
5 Measurement and reporting mechanism

Major Software Quality Assurance Activities:

1. **SQA Management Plan:** Make a plan how you will carry out the sqa throughout the project. **Think which set of software engineering activities are the best for project.** Check level of sqa team skills.

2. **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.

3. **Multi testing Strategy:** Do not depend on single testing approach. When you have lot of testing approaches available use them.

4. **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 change check the compatibility of this fix with whole project.

5. **Manage Good Relations:** In the working environment managing the good relation with other teams involved in the project development is mandatory. **Bad relation of sqa team with programmer's team will impact directly and badly on project. Don't play politics.**

Benefits of Software Quality Assurance (SQA):

1 SQA produce high quality software.

2 High quality application saves time and cost.

3 SQA is beneficial for better reliability.

4 SQA is beneficial in the condition of no maintenance for long time.

5 High quality commercial software increase market share of company.

6 Improving the process of creating software.

7 Improves the quality of the software.

Disadvantage of SQA:

There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

Quality attributes of a software: Or Quality Metrics

A. Goal of software engineering or SQA is to produce high quality

- System
- Product
- Application

Within a timeframe that meets/satisfies a market needs.

B. To achieve this software engineers need to apply effective methods and tools within software process

C. Quality Metrics: Is criteria for measuring quality of product, process etc.

D. Software can be judged for quality on the basis of the following six characteristics(or quality metrics):

1) Functionality (Correctness)

The functionality of the software is the set of functions that the software provides. The software must provide appropriate functions as per requirements and these functions must be implemented correctly.

The software should have interoperability which means how effectively the software interacts with other components of the system. It must be compliant with the laws and guidelines. The software should handle data related transactions securely.

2) Reliability

The reliability of the software is its capability to perform under specific conditions for a defined duration. It also implicates the ability of the software to withstand failures of its components.

The software is reliable on the basis of its **Maturity** that is the frequency of the failures and **Recoverability** which is the ability of the software to get fully operational after a failure.

3) Usability

The usability of the software is its ease of use. It also refers to how easily a user can understand the functions of the software and how much efforts are required by the users to understand the functions.

4) Efficiency

The efficiency of the software is dependent on its architecture and coding practices followed during development.

5) Maintainability

The maintainability of the software depends upon the code complexity and readability. It also refers to the ability to identify and fix a fault in the software.

The software is analyzed on the basis of ease of identifying the main cause of failure and the ease of modification of code to remove a fault. It should be stable in its performance when the changes are made.

Maintainability also depends on its testability which means how much efforts are required for testing the system.

6) Portability

The portability of the software is its ability to adapt to the changes in its environment. It defines how easily a system adapts to any changes made in the specifications. It also includes how easy it is to install the software and how easy is it to replace a component of the system in a given environment.

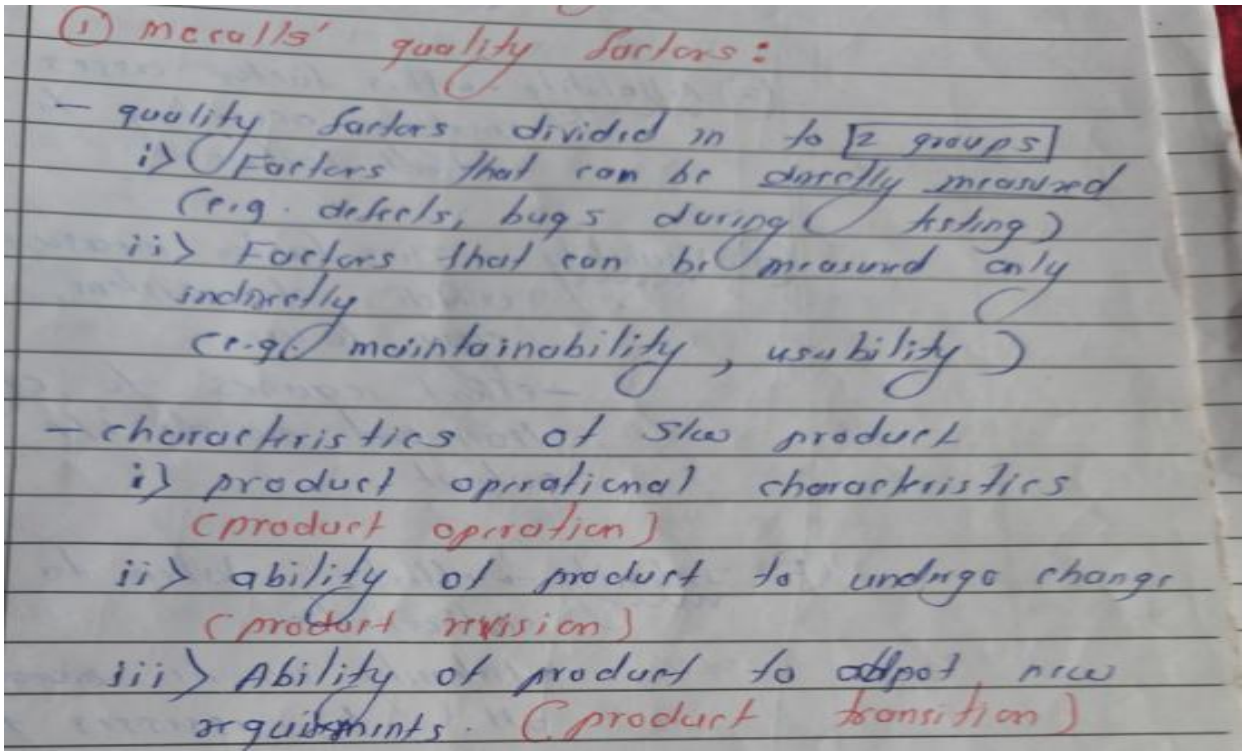
To ensure a software scores well on these quality attributes, we need the following software quality assurance components.

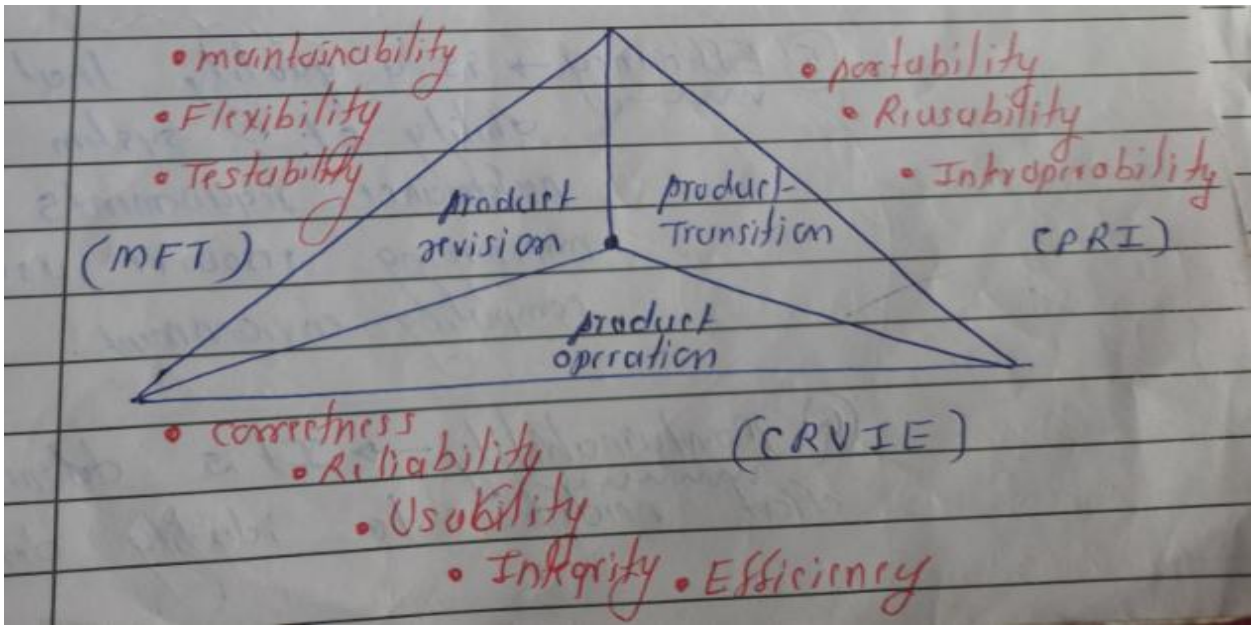
7) Integrity

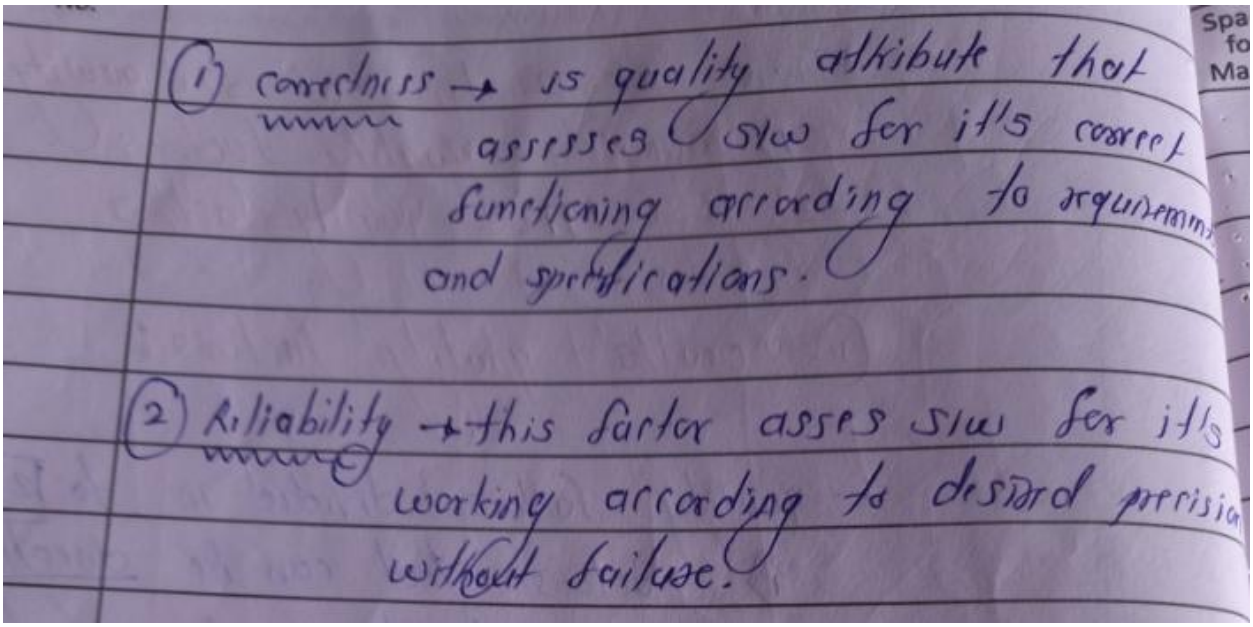
Efforts required managing access authorization. Integrity of a system can be ensured by controlling all the accesses to the system.

Two main types of Software quality factors

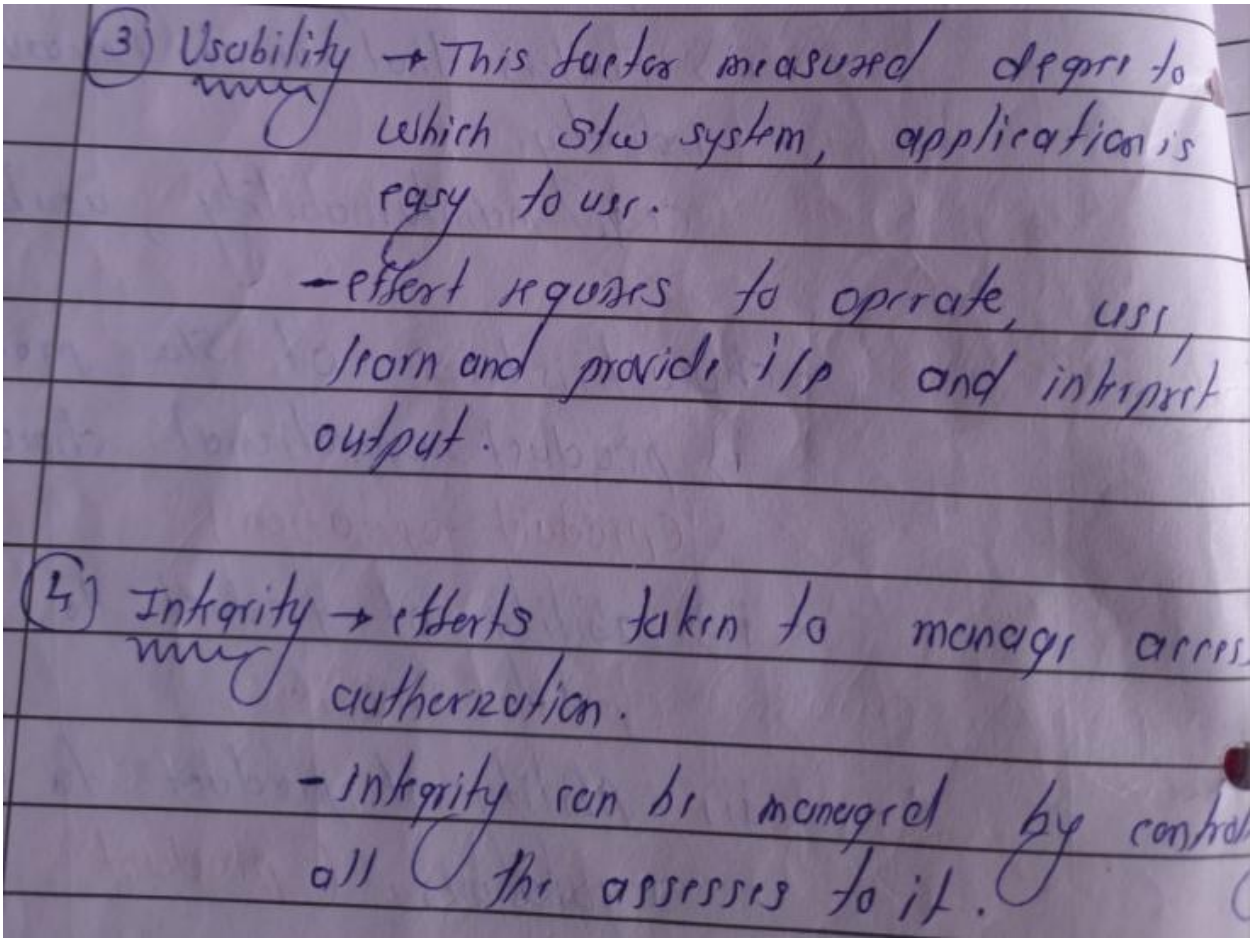
1) Mac all's quality factors
2) ISO 9126 quality factors



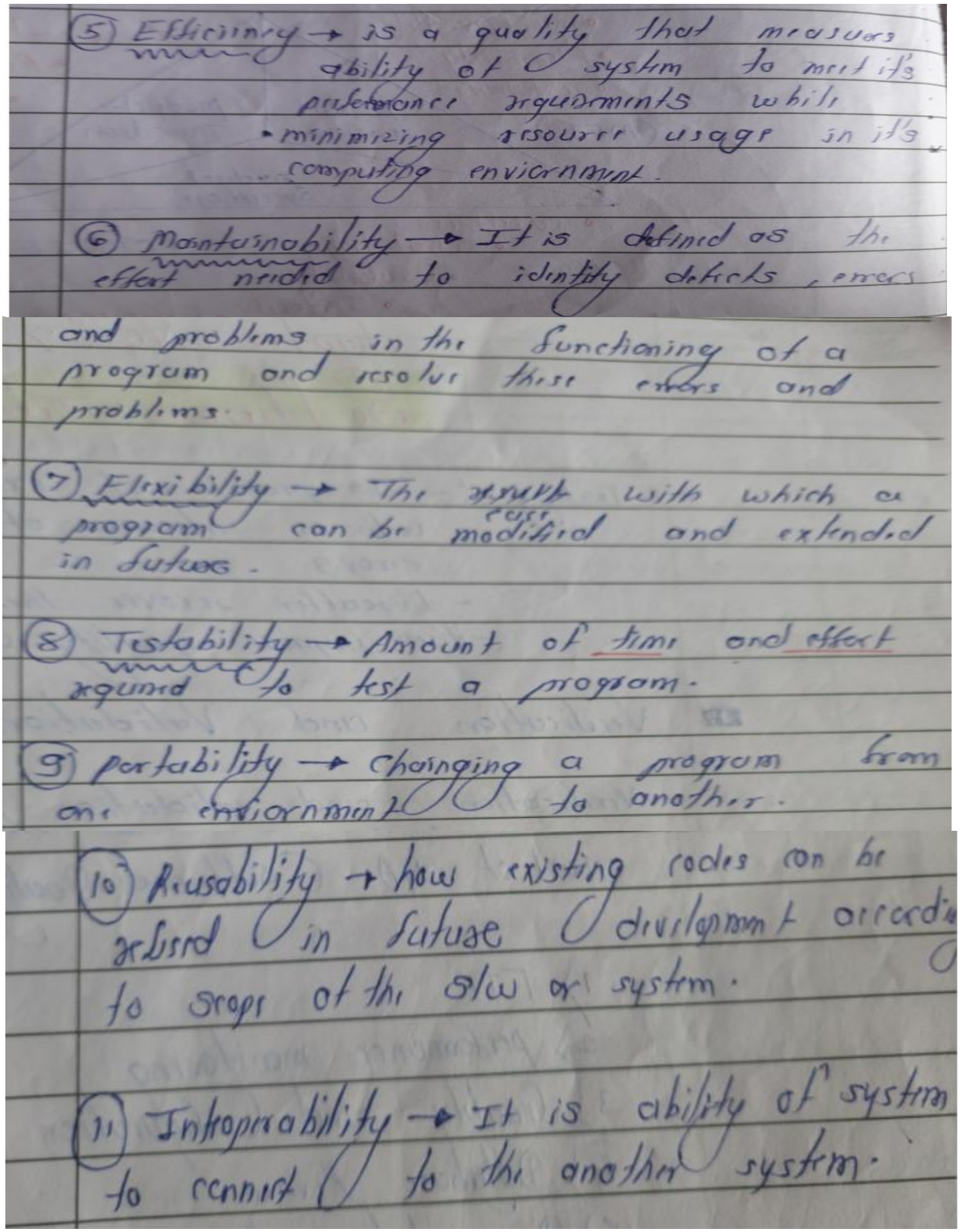



(1) correctness → is quality attribute that assesses sfw for it's correct functioning according to requirements and specifications.

(2) Reliability → this factor asses sfw for it's working according to desired precision without failure.


(3) Usability → This factor measured degree to which sfw system, application is easy to use.
- effort requires to operate, user learn and provide i/p and interpret output.

(4) Integrity → efforts taken to manage access authentication.
- integrity can be managed by controlling all the assesses to it.



(5) Efficiency → is a quality that measures ability of system to meet its performance requirements while minimizing resource usage in its computing environment.

(6) Maintainability → It is defined as the effort needed to identify defects, errors and problems in the functioning of a program and resolve these errors and problems.

(7) Flexibility → The ease with which a program can be modified and extended in future.

(8) Testability → Amount of time and effort required to test a program.

(9) Portability → Changing a program from one environment to another.

(10) Reusability → how existing codes can be reused in future development according to scope of the sw or system.

(11) Interoperability → It is ability of system to connect to the another system.

Software Quality Assurance Components

The software quality assurance has the following six classes of components:

3) Pre-project Components

The pre-project components ensure that the resources required for the project, the schedule, and the budget is clearly been defined. The plan for development and ensuring quality has been clearly determined. The components are as follows:

- Development plan
- Quality plan
- Schedules
- Required resources (Hardware & Human resources)
- Risk evaluations
- Project methodology

4) Project lifecycle components

A project lifecycle is usually comprised of two stages. The first one is the development stage and then comes the operation-maintenance stage. In the development stage, SQA components help to identify the design and programming errors.

The SQA components for the operation-maintenance stage include the development lifecycle components along with specialized maintenance components aimed to improve the maintenance tasks.

The project lifecycle components include:

- Reviews
- Expert opinions
- Software testing
- Software maintenance
- Sub-contractors quality assurance

5) Infrastructure error prevention and improvement components

The main goal of these components is the prevention of software faults and minimizes the rate of errors. These components include:

- Procedure & work instructions
- Templates & checklists
- Staff training, retaining & certification
- Preventive & corrective actions
- Configuration management
- Documentation control

6) Software quality management components

This class of components consists of controlling the development and maintenance activities. These components establish the managerial control of software development projects. The management control aims to prevent the project from going over budget and behind schedule.

The management control components include:

- Project progress control
- Software quality metrics
- Software quality costs

7) Standardization, certification, and SQA assessment components

The components aim to implement international managerial and professional standards within the organization. These components help to improve the coordination among the organizational quality systems and establish standards for the project process. The components include:

- Quality management standards

- Project process standards

8) Organizing for SQA – the human components

The main aim of this class of components is to initiate and support the implementation of SQA components, identify any deviations from the predefined SQA procedures & methods and recommend improvements. The SQA organizational team includes test managers, testers, SQA unit, SQA committee, and SQA forum members.

■ FTR(Formal Technical Review)

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR): Some of these are:

- ✓ Useful to uncover error in logic, function and implementation for any representation of the software.
- ✓ The purpose of FTR is to verify that the software meets specified requirements.
- ✓ To ensure that software is represented according to predefined standards.
- ✓ It helps to review the uniformity in software that is development in a uniform manner.
- ✓ To makes the project more manageable.

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely. FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise.

Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successfully only if it is properly planned, controlled and attended.

The review meeting: Each review meeting should be held considering the following constraints-

***Involvement of people*:**

1 Between 3, 4 and 5 people should be involve in the review.
2 Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3 The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1 Accept the product without any modification.
2 Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3 Accept the product provisional (minor errors are encountered and are should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

Review reporting and record keeping:-

1 During the FTR, the reviewer actively records all issues that have been raised.
2 At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3 Finally, a formal technical review summary report is prepared.

It answers three questions:-

1 What was reviewed?
2 Who reviewed it?
3 What were the findings and conclusions?

Review guidelines: - Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

1 Review the product, not the manufacture (producer).
2 Take written notes (record purpose)

3 Limit the number of participants and insists upon advance preparation.
4 Develop a checklist for each product that is likely to be reviewed.
5 Allocate resources and time schedule for FTRs in order to maintain time schedule.
6 Conduct meaningful training for all reviewers in order to make reviews effective.
7 Reviews earlier reviews which serve as the base for the current review being conducted.
8 Set an agenda and maintain it.
9 Separate the problem areas, but do not attempt to solve every problem notes.
10 Limit debate and rebuttal.

■ Walkthrough

Walkthrough in software testing is used to review documents with peers, managers, and fellow team members who are guided by the author of the document to gather feedback and reach a consensus. A walkthrough can be pre-planned or organised based on the needs. Generally people working on the same work product are involved in the walkthrough process.

- It is not a formal process/review
- It is led by the authors
- Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
- Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
- Is especially useful for higher level documents like requirement specification, etc.

The goals of a walkthrough:

1 To present the documents both within and outside the software discipline in order to gather the information regarding the topic under documentation.
2 To explain or do the knowledge transfer and evaluate the contents of the document
3 To achieve a common understanding and to gather feedback.
4 To examine and discuss the validity of the proposed solutions