SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V – Software Engineering – SCS1305

COST ESTIMATION & MAINTENENCE

Software cost estimation - COCOMO model - Quality management - Quality concepts- SQA - Software reviews - Formal technical reviews - Formal approaches of SQA and software reliability - Software maintenance - SCM - Need for SCM - Version control - Introduction to SCM process - Software configuration items. Re-Engineering - Software reengineering - Reverse engineering - Restructuring - Forward engineering.

Software Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually. To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.

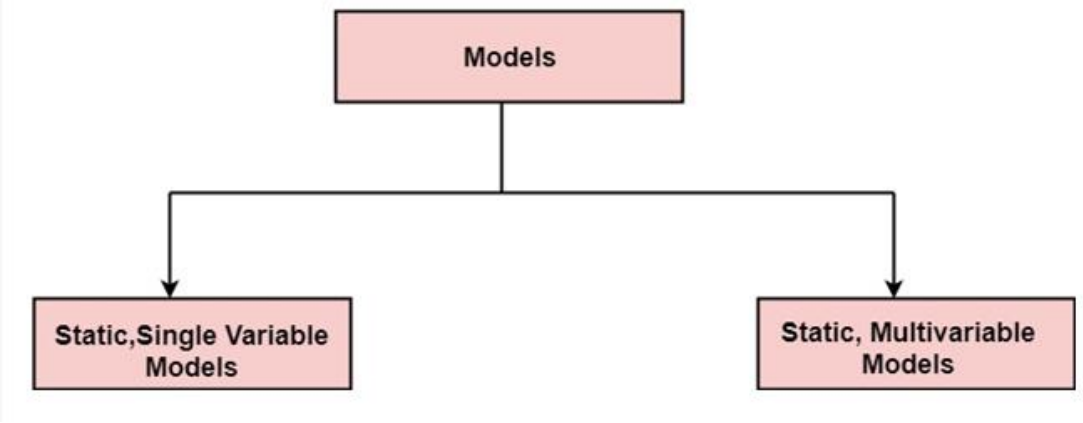


Fig 5.1 Models

Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

Where C = Costs
L= size
a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

$$E=1.4L^{0.93}$$
$$DOC=30.4L^{0.90}$$
$$D=4.6L^{0.26}$$

Where E= Efforts (Person Per Month)
DOC=Documentation (Number of Pages)
D = Duration (D, in months)
L = Number of Lines per code

Static, Multivariable Models: These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

$$E=5.2L^{0.91}$$

In the same manner duration of development is given by

$$\mathbf{D=4.1L^{0.36}}$$

The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

Where $\mathbf{W_i}$ is the weight factor for the $\mathbf{i^{th}}$variable and $\mathbf{X_i}$**={-1,0,+1}** the estimator gives $\mathbf{X_i}$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

Example: Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

a. Calculate the number of lines of source code that can be produced.
b. Calculate the duration of the development.
c. Calculate the productivity in LOC/PY
d. Calculate the average manning

Solution:

The amount of manpower involved = 8PY=96persons-months

(a)Number of lines of source code can be obtained by reversing equation to give:

Then

$$L\ (SEL) = (96/1.4)1/0.93=94264\ LOC$$
$$L\ (SEL) = (96/5.2)1/0.91=24632\ LOC$$

(b)Duration in months can be calculated by means of equation

$$D\ (SEL) = 4.6\ (L)\ 0.26$$
$$= 4.6\ (94.264)0.26 = 15\ months$$
$$D\ (W\text{-}F) = 4.1\ L^{0.36}$$
$$= 4.1\ (24.632)0.36 = 13\ months$$

(c) Productivity is the lines of code produced per persons/month (year)

(d)Average manning is the average number of persons required per month in the project

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

E_i=a*(KDLOC)b

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

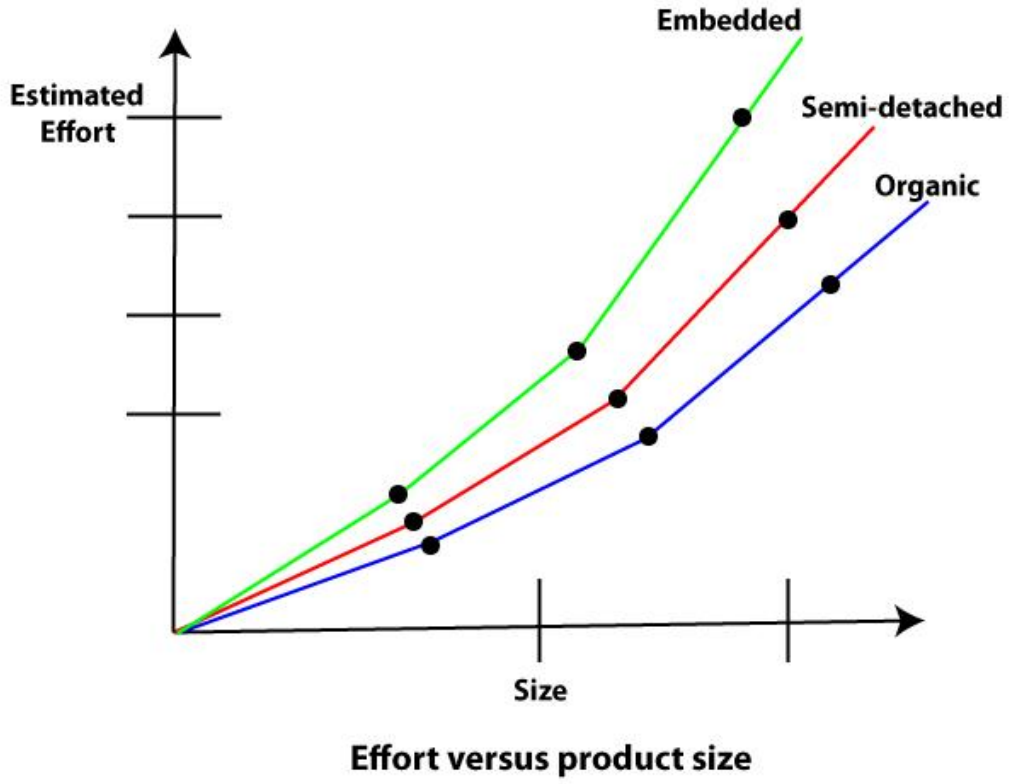


Fig 5.2 Effort vs size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project.

Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.

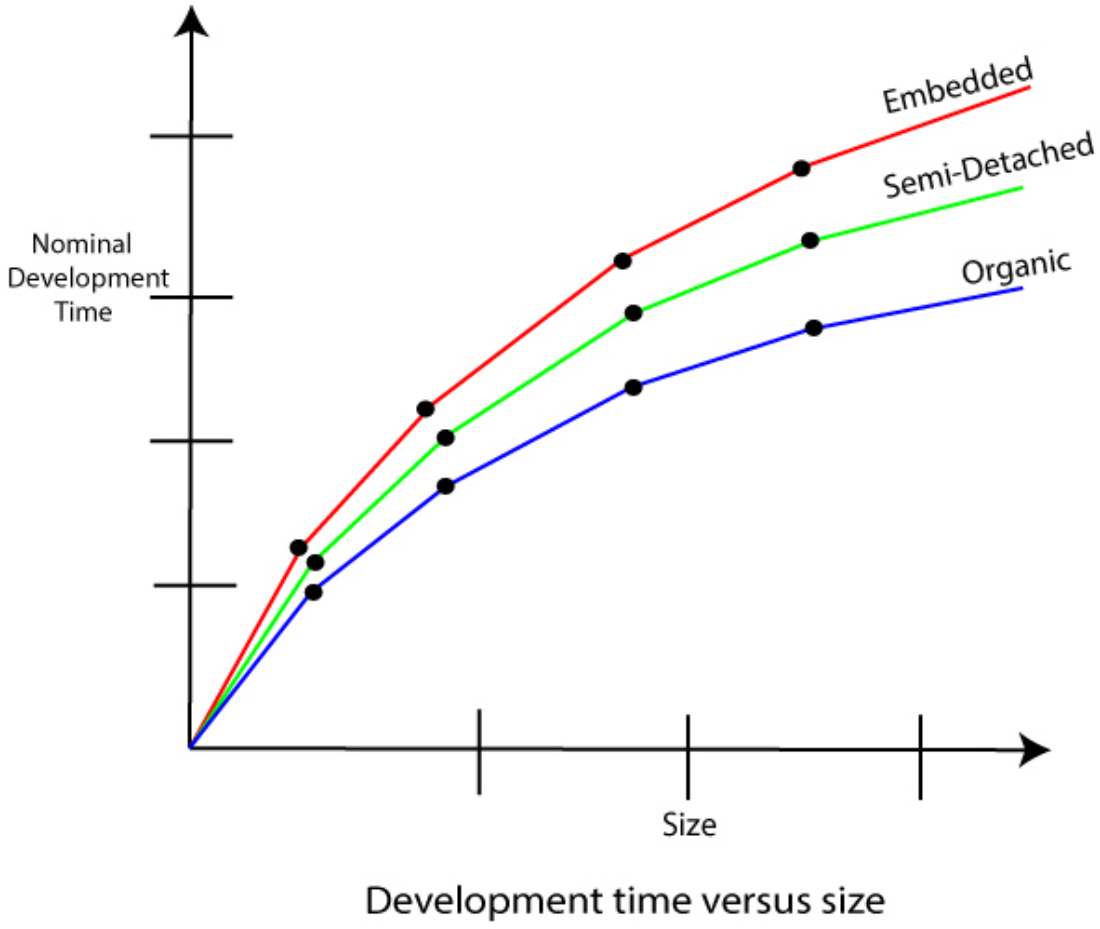


Fig 5.3 Time vs Size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) a_2 PM
Tdev=b_1*(efforts)b_2 Months
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM
D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM
D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM
D = 2.5 * (4772.8)0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM
D=2.5(1133.12)0.35=29.3PM

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	..	..	1.00	1.11	1.30	1.66
STOR	..	..	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..

Table 5.1 Ratings

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

Intermediate COCOMO equation:

$$E=a_i\ (KLOC)\ b_i*EAF$$

$$D=c_i\ (E)d_i$$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the softwareengineering

process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

Software Quality Management

Software Quality Management – abbreviated SQM – is a term used to describe the management aspects of developing quality software. SQM begins with an idea for a product and continues through the design, testing and launch phases.

A management process that is made up of a few different steps, SQM can be broken down most simply into three phases: Quality planning, assurance and control.

Quality Planning

Before software development begins, quality planning must take place. Quality planning involves the creation of goals and objectives for your software, as well as the creation of a strategic plan that will help you to successfully meet the objectives you lay out.

Quality planning is often considered the most important aspect of SQM, as it develops a strong blueprint for the rest of the process to follow – leading to the best-possible end product.

Quality Assurance

The quality assurance phase of SQM involves the actual building of the software program. With good SQM in place, product performance will be checked along the way to ensure that all standards are being followed. Audits may be performed and data will be collected throughout the entirety of the process.

Quality Control

The step of SQM where testing finally comes into play, quality control is in place to discover bugs, evaluate functionality and more. Depending on the results of the quality control phase, you may need to go back to development to iron out kinks and make some small final adjustments. Having a software quality management plan in place can guarantee that all industry standards are being followed and that your end-user will receive a well-developed, high quality product.

Quality Concepts

Qulaity:

Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

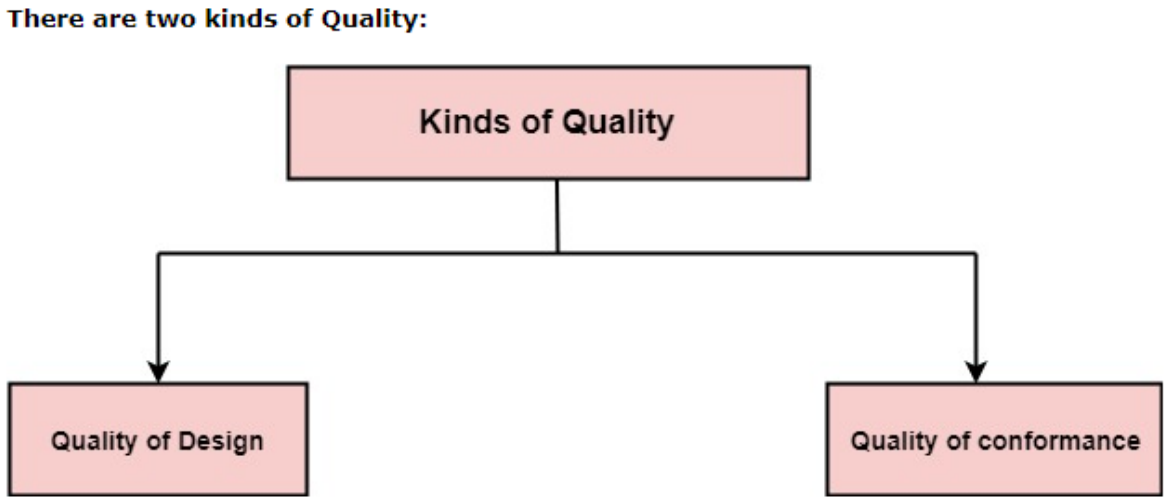


Fig 5.5 Types of quality

Quality of Design: Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

Quality of conformance: Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

Software Quality: Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

Quality Assurance focuses on how the engineering and management activity will be done

As anyone is interested in the quality of the final product, it should be assured that we are building the right product.

It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

Importance of Quality

We would expect the quality to be a concern of all producers of goods and services. However, the distinctive characteristics of software and in particular its intangibility and complexity, make special demands.

Increasing criticality of software: The final customer or user is naturally concerned about the general quality of software, especially its reliability. This is increasing in the case as organizations become more dependent on their computer systems and software is used more and more in safety-critical areas. For example, to control aircraft.

The intangibility of software: This makes it challenging to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce 'deliverables' that can be examined for quality.

Accumulating errors during software development: As computer system development is made up of several steps where the output from one level is input to the next, the errors in the earlier ?deliverables? will be added to those in the later stages leading to accumulated determinable effects. In general the later in a project that an error is found, the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly challenging to control.

Software Quality Assurance

Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

SQA Activities

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

Following activities are performed by an independent SQA group:

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **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.
3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.
5. **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 method, process description, applicable standards, or technical work products.

6. **Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

Quality Assurance	Quality Control
Quality Assurance (QA) is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use.	**Quality Control (QC)** is described as the processes and methods used to compare product quality to requirements and applicable standards, and the actions are taken when a nonconformance is detected.
QA is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA.	**QC** is an activity that demonstrates whether or not the product produced met standards.
QA helps establish process	**QC** relates to a particular product or service
QA sets up a measurement program to evaluate processes	**QC** verified whether particular attributes exist, or do not exist, in a explicit product or service.
QA identifies weakness in processes and improves them	**QC** identifies defects for the primary goals of correcting errors.
Quality Assurance is a managerial tool.	Quality Control is a corrective tool.
Verification is an example of QA.	Validation is an example of QC.

Table 5.5 QA vs QC

Software Reviews:

A software review is a process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval.

Formal Review: Formal reviews follow a formal process. It is well structured and regulated. A formal review process consists of six main steps:

1. Planning
2. Kick-off
3. Preparation
4. Review meeting
5. Rework
6. Follow-up

1. Planning: The first phase of the formal review is the Planning phase. In this phase thereview process begins with a request for review by the author to the moderator (or inspection leader). A moderator has to take care of the scheduling like date, time, place and invitation of the review.

For the formal reviews the moderator performs the entry check and also defines the formal exit criteria.

The **entry check** is done to ensure that the reviewer's time is not wasted on a document that is not ready for review. After doing the entry check if the document is found to have very little defects then it's ready to go for the reviews. So, the **entry criteria** are to check that whether the document is ready to enter the formal review process or not. Hence the entry criteria for any document to go for the reviews are:

1.
    1.
    - The documents should not reveal a large number of major defects.
    - The documents to be reviewed should be with line numbers.
    - The documents should be cleaned up by running any automated checks that apply.
    - The author should feel confident about the quality of the document so that he can join the review team with that document.

Once, the document clear the entry check the moderator and author decides that which part of the document is to be reviewed. Since the human mind can understand only a limited set of pages at one time so in a review the maximum size is between 10 and 20 pages. Hence checking the documents improves the moderator ability to lead the meeting because it ensures the better understanding.

2. Kick-off: This kick-off meeting is an optional step in a review procedure. The goal of this step is to give a short introduction on the objectives of the review and the documents to everyone in the meeting. The relationships between the document under review and the other documents are also explained, especially if the numbers of related documents are high. At customer sites, we have measured results up to 70% more major defects found per page as a result of performing a kick-off, [van Veenendaal and van der Zwan, 2000].

3. Preparation: In this step the reviewers review the document individually using the related documents, procedures, rules and checklists provided. Each participant while reviewing individually identifies the defects, questions and comments according to their understanding of the document and role. After that all issues are recorded using a logging form. The success factor for a thorough preparation is the number of pages checked per hour. This is called the **checking rate.** Usually the checking rate is in the range of 5 to 10 pages per hour.

4. Review meeting: The review meeting consists of three phases:

- **Logging phase:** In this phase the issues and the defects that have been identified during the preparation step are logged page by page. The logging is basically done by the author or by a **scribe.** Scribe is a separate person to do the logging and is especially useful for the formal review types such as an inspection.

- Every defects and it's severity should be logged in any of the three severity
    - **Critical,Major Minor**

During the logging phase the moderator focuses on logging as many defects as possible within a certain time frame and tries to keep a good logging rate (number of defects logged per minute). In formal review meeting the good logging rate should be between one and two defects logged per minute.

- **Discussion phase:** If any issue needs discussion then the item is logged and then handled in the discussion phase. As chairman of the discussion meeting, the moderator takes care of the people issues and prevents discussion from getting too personal and calls for a break to cool down the heated discussion. The outcome of the discussions is documented for the future reference.

- **Decision phase:** At the end of the meeting a decision on the document under review has to be made by the participants, sometimes based on formal **exit criteria. Exit criteria** are the average number of critical and/or major defects found per page (for example no more than three critical/major defects per page). If the number of defects found per page is more than a certain level then the document must be reviewed again, after it has been reworked.

5. Rework: In this step if the number of defects found per page exceeds the certain level then the document has to be reworked. Not every defect that is found leads to rework. It is the author's responsibility to judge whether the defect has to be fixed. If nothing can be done about an issue then at least it should be indicated that the author has considered the issue.

6. Follow-up: In this step the moderator check to make sure that the author has taken action on all known defects. If it is decided that all participants will check the updated documents then the moderator takes care of the distribution and collects the feedback. It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

Informal Reviews:

Informal reviews are applied many times during the early stages of the life cycle of the document. A two person team can conduct an informal review. In later stages these reviews often involve more people and a meeting. The goal is to keep the author and to improve the quality of the document. The most important thing to keep in mind about the informal reviews is that they are not documented.

Technical Review:

- It is less formal review
- It is led by the trained moderator but can also be led by a technical expert
- It is often performed as a peer review without management participation

- Defects are found by the experts (such as architects, designers, key users) who focus on the content of the document.
- In practice, technical reviews vary from quite informal to very formal

The goals of the technical review are:

1. To ensure that an early stage the technical concepts are used correctly
2. To access the value of technical concepts and alternatives in the product
3. To have consistency in the use and representation of technical concepts
4. To inform participants about the technical content of the document

Walkthroughs :

Walkthroughs are represented by the below characteristics:

- 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

Quality:

Quality is extremely hard to define, and it is simply stated: "Fit for use or purpose." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

Assurance:

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

Quality Assurance:

Quality Assurance (QA) is defined as an activity to ensure that an organization is providing the best possible product or service to customers. QA focuses on improving the processes to deliver Quality Products to the customer. An organization has to ensure, that processes are efficient and effective as per the quality standards defined for software products. Quality Assurance is popularly known as QA Testing.

Complete Process

Quality assurance has a defined cycle called PDCA cycle or Deming cycle. The phases of this cycle are:

- Plan
- Do
- Check
- Act

Fig 5.5 QA

These above steps are repeated to ensure that processes followed in the organization are evaluated and improved on a periodic basis. Let's look into the above steps in detail -

- Plan - Organization should plan and establish the process related objectives and determine the processes that are required to deliver a high-Quality end product.
- Do - Development and testing of Processes and also "do" changes in the processes
- Check - Monitoring of processes, modify the processes, and check whether it meets the predetermined objectives
- Act - Implement actions that are necessary to achieve improvements in the processes

An organization must use Quality Assurance to ensure that the product is designed and implemented with correct procedures. This helps reduce problems and errors, in the final product.

Quality Control

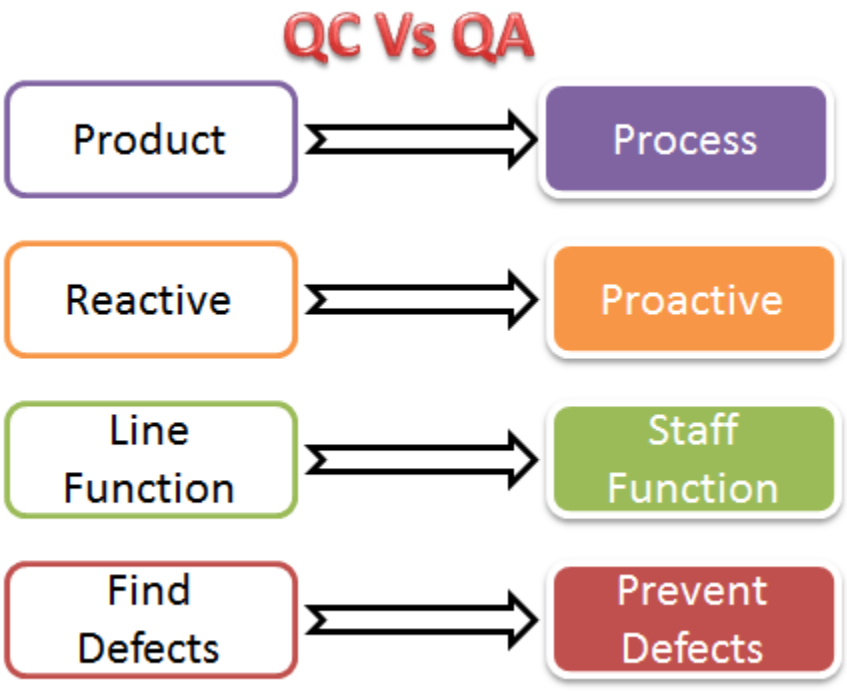
Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.

QC also evaluates people on their quality level skill sets and imparts training and certifications. This evaluation is required for the service based organization and helps provide "perfect" service to the customers.

Quality Control and Quality Assurance

Sometimes, QC is confused with the QA. Quality control is to examine the product or service and check for the result. Quality assurance is to examine the processes and make changes to the processes which led to the end-product.



Quality Assurance Functions:

There are 5 primary Quality Assurance Functions:

1. **Technology transfer:** This function involves getting a product design document as well as trial and error data and its evaluation. The documents are distributed, checked and approved
2. **Validation:** Here validation master plan for the entire system is prepared. Approval of test criteria for validating product and process is set. Resource planning for execution of a validation plan is done.
3. **Documentation:** This function controls the distribution and archiving of documents. Any change in a document is made by adopting the proper change control procedure. Approval of all types of documents.
4. **Assuring Quality of products**
5. **Quality improvement plans**

Software Reliability Models

A software reliability model indicates the form of a random process that defines the behavior of software failures to time.

Software reliability models have appeared as people try to understand the features of how and why software fails, and attempt to quantify software reliability.

There is no individual model that can be used in all situations. No model is complete or even representative.

Most software models contain the following parts:

- Assumptions
- Factors

A mathematical function that includes the reliability with the elements. The mathematical function is generally higher-order exponential or logarithmic.

Software Reliability Modeling Techniques:

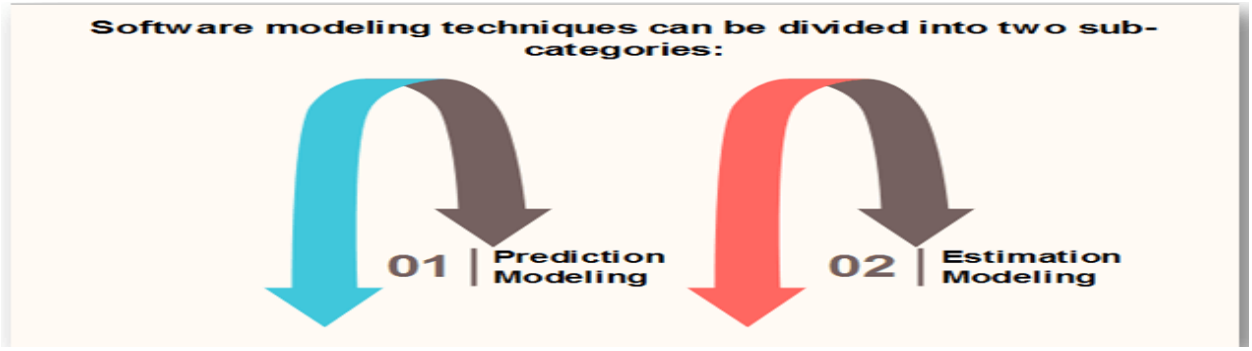


Fig 5.6 software modeling types

Both kinds of modeling methods are based on observing and accumulating failure data and analyzing with statistical inference.

Differentiate between software reliability prediction models and software reliability estimation models

Basics	Prediction Models	Estimation Models
Data Reference	Uses historical information	Uses data from the current software development effort.
When used in development cycle	Usually made before development or test phases; can be used as early as concept phase.	Usually made later in the life cycle (after some data have been collected); not typically used in concept or development phases.
Time Frame	Predict reliability at some future time.	Estimate reliability at either present or some next time.

Table 4.3 Prediction vs Estimation

Reliability Models

A reliability growth model is a numerical model of software reliability, which predicts how software reliability should improve over time as errors are discovered and repaired. These models help the manager in deciding how much efforts should be devoted to testing. The objective of the project manager is to test and debug the system until the required level of reliability is reached.

Following are the Software Reliability Models are:

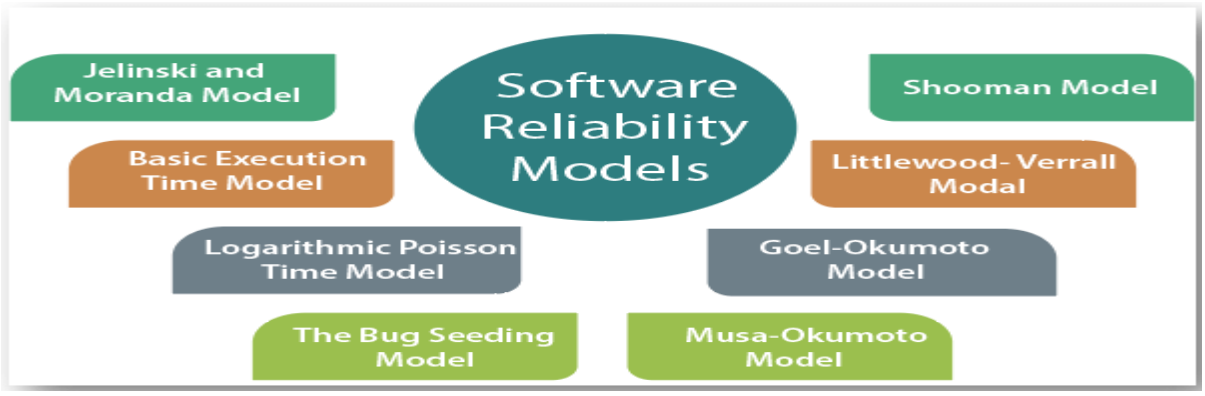


Fig 5.6 Software Reliability Models

Software Maintenance:

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

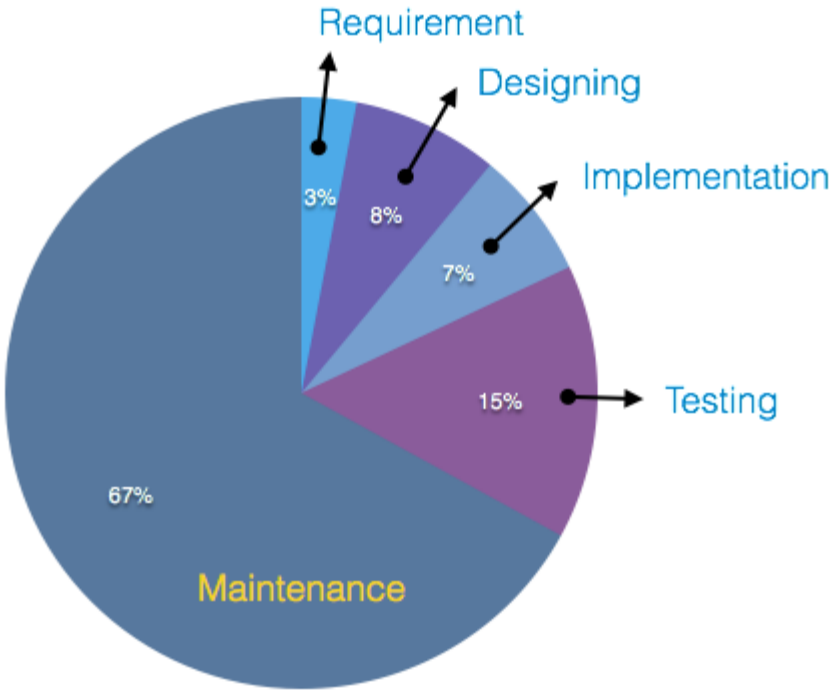
Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process .



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be include.

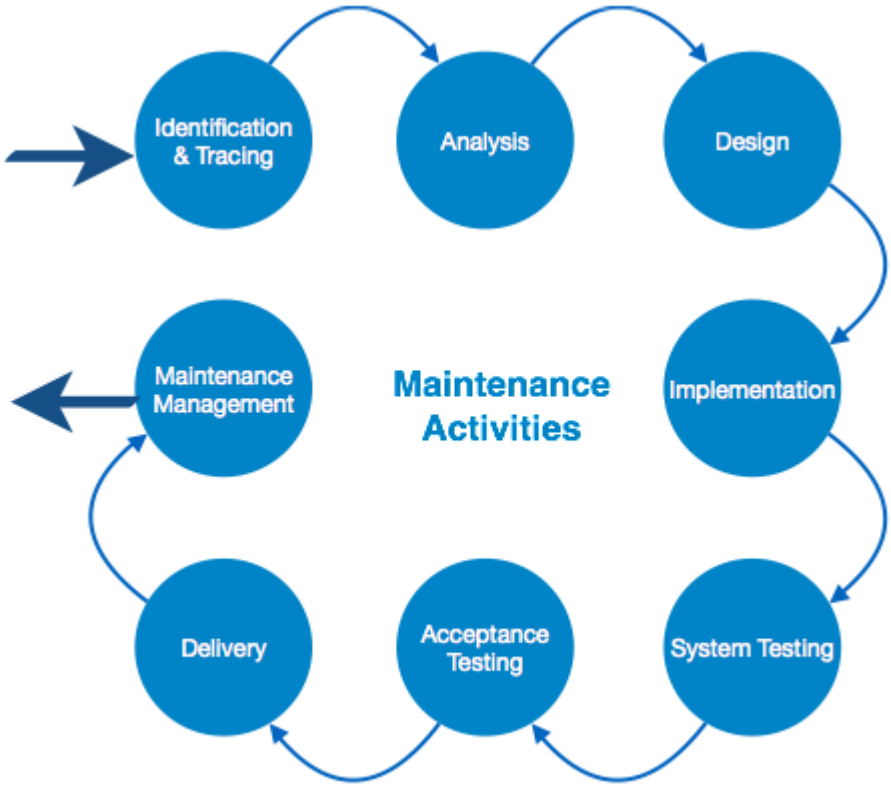


Fig 5.6-Maintanenece Activities

These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.

- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

  Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

Software Configuration Management:

Software Configuration Management is defined as a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. It is abbreviated as the SCM process in software engineering. The primary goal is to increase productivity with minimal mistakes.

Need of Configuration management:

The primary reasons for Implementing Software Configuration Management System are**:**

- There are multiple people working on software which is continually updating
- It may be a case where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently
- Changes in user requirement, policy, budget, schedule need to be accommodated.
- Software should able to run on various machines and Operating Systems
- Helps to develop coordination among stakeholders
- SCM process is also beneficial to control the costs involved in making changes to a system

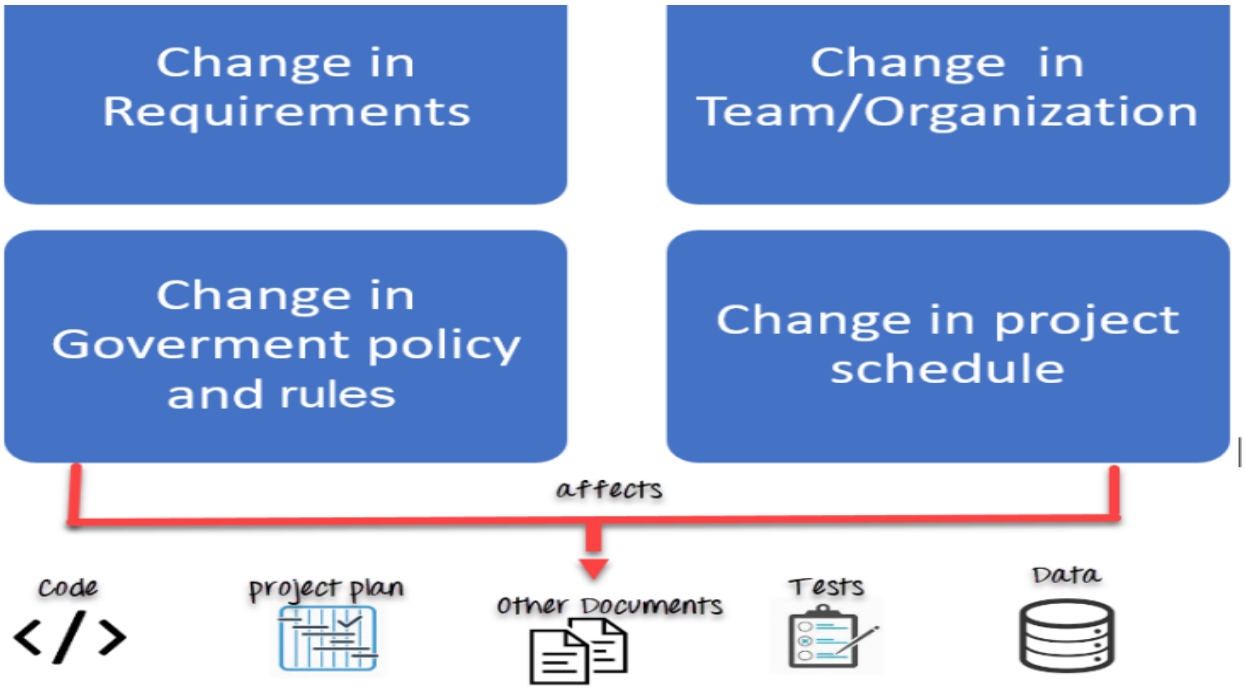


Fig 5.7 SCM

Any change in the software configuration Items will affect the final product. Therefore, changes to configuration items need to be controlled and managed.

Tasks in SCM process

Configuration Identification

Baselines

Change Control

Configuration Status Accounting

Configuration Audits and Reviews

Configuration Identification:

Configuration identification is a method of determining the scope of the software system. With the help of this step, you can manage or control something even if you don't know what it is. It is a description that contains the CSCI type (Computer Software Configuration Item), a project identifier and version information.

Activities during this process:

- Identification of configuration Items like source code modules, test case, and requirements specification.
- Identification of each CSCI in the SCM repository, by using an object-oriented approach
- The process starts with basic objects which are grouped into aggregate objects. Details of what, why, when and by whom changes in the test are made
- Every object has its own features that identify its name that is explicit to all other objects
- List of resources required such as the document, the file, tools, etc.

Example:

Instead of naming a File login.phpits should be named login_v1.2.php where v1.2 stands for the version number of the file

Instead of naming folder "Code" it should be named "Code_D" where D represents code should be backed up daily.

Baseline:

A baseline is a formally accepted version of a software configuration item. It is designated and fixed at a specific time while conducting the SCM process. It can only be changed through formal change control procedures.

Activities during this process:

- Facilitate construction of various versions of an application
- Defining and determining mechanisms for managing various versions of these work products
- The functional baseline corresponds to the reviewed system requirements
- Widely used baselines include functional, developmental, and product baselines

In simple words, baseline means ready for release.

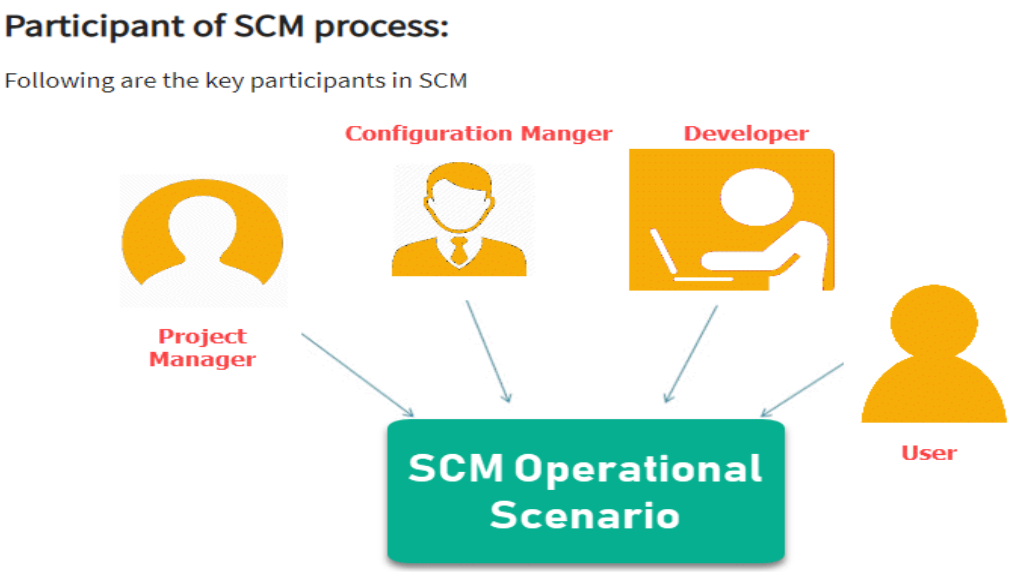


Fig 5.8 SCM operational scenario

Change Control:

Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object. In this step, the change request is submitted to software configuration manager.

Activities during this process:

- Control ad-hoc change to build stable software development environment. Changes are committed to the repository
- The request will be checked based on the technical merit, possible side effects and overall impact on other configuration objects.
- It manages changes and making configuration items available during the software lifecycle

Configuration Status Accounting:

Configuration status accounting tracks each release during the SCM process. This stage involves tracking what each version has and the changes that lead to this version.

Activities during this process:

- Keeps a record of all the changes made to the previous baseline to reach a new baseline
- Identify all items to define the software configuration
- Monitor status of change requests
- Complete listing of all changes since the last baseline

- Allows tracking of progress to next baseline
- Allows to check previous releases/versions to be extracted for testing

Configuration Audits and Reviews:

Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.

Activities during this process:

- Configuration auditing is conducted by auditors by checking that defined processes are being followed and ensuring that the SCM goals are satisfied.
- To verify compliance with configuration control standards. auditing and reporting the changes made
- SCM audits also ensure that traceability is maintained during the process.
- Ensures that changes made to a baseline comply with the configuration status reports
- Validation of completeness and consistency

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

Software Configuration Management Plan

The SCMP (Software Configuration management planning) process planning begins at the early phases of a project. The outcome of the planning phase is the SCM plan which might be stretched or revised during the project.

- The SCMP can follow a public standard like the IEEE 828 or organization specific standard
- It defines the types of documents to be management and a document naming. Example Test_v1
- SCMP defines the person who will be responsible for the entire SCM process and creation of baselines.
- Fix policies for version management & change control
- Define tools which can be used during the SCM process
- Configuration management database for recording configuration information.

Software Configuration Management Tools

Any Change management software should have the following 3 Key features:

Concurrency Management:

When two or more tasks are happening at the same time, it is known as concurrent operation. Concurrency in context to SCM means that the same file being edited by multiple persons at the same time.

If concurrency is not managed correctly with SCM tools, then it may create many pressing issues.

Version Control:

SCM uses archiving method or saves every change made to file. With the help of archiving or save feature, it is possible to roll back to the previous version in case of issues.

Synchronization:

Users can checkout more than one files or an entire copy of the repository. The user then works on the needed file and checks in the changes back to the repository.They can synchronize their local copy to stay updated with the changes made by other team members.

Following are popular tools

1. Git: Git is a free and open source tool which helps version control. It is designed to handle all types of projects with speed and efficiency.

2. Team Foundation Server: Team Foundation is a group of tools and technologies that enable the team to collaborate and coordinate for building a product.

3. Ansible: It is an open source Software configuration management tool. Apart from configuration management it also offers application deployment & task automation.

Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.

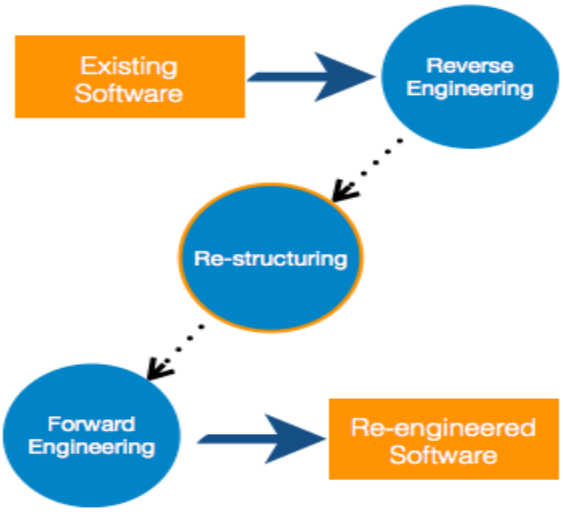


Fig 5.8 Re-Engineering

Re-Engineering Process

- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.

- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering:

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.

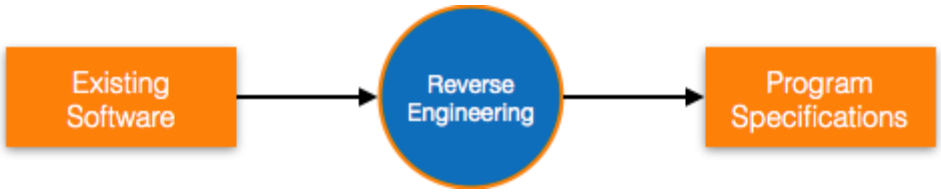


Fig 5.9 Reverse Engineering

Program Restructuring:

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.

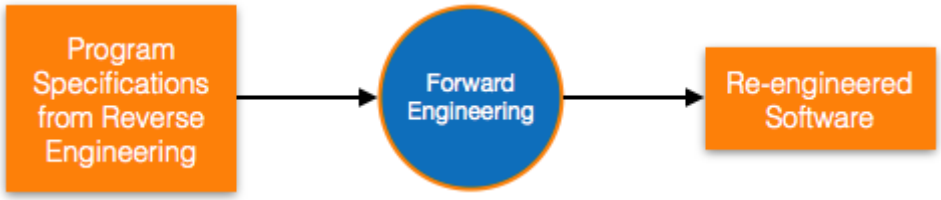


Fig 5.10 Forward Engineering

Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example:

The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.

In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.

There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).

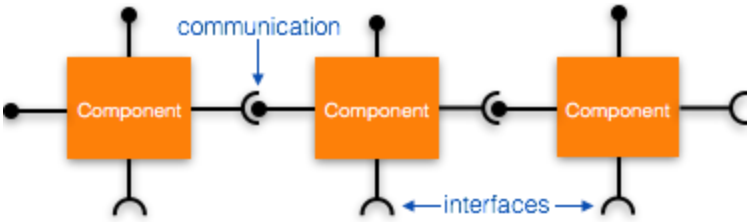


Fig 5.11- Component reusability

Re-use can be done at various levels

- **Application level** - Where an entire application is used as sub-system of new software.
- **Component level** - Where sub-system of an application is used.
- **Modules level** - Where functional modules are re-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process:

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.

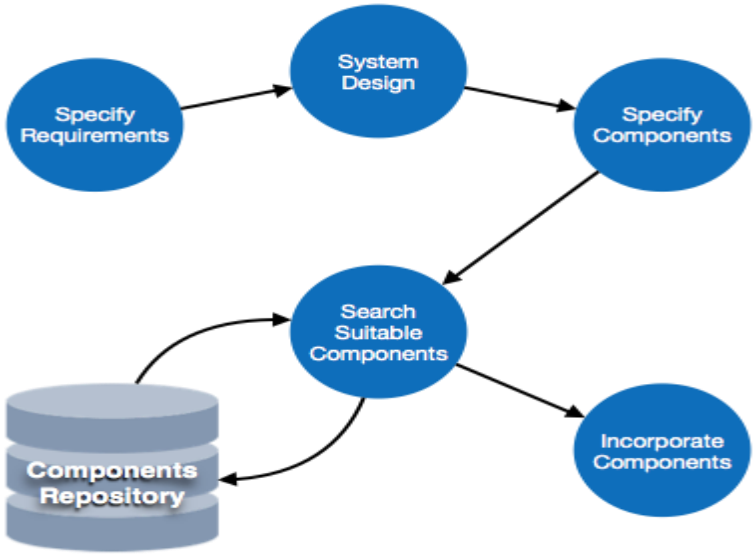


Fig 5.12 Reuse Process

- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems are created.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.