Chapter – 6

Management of Software Engineering

Responsibilities of a software project manager:

Proper project management is essential for the successful completion of a software project and the person who is responsible for it is called project manager. To do his job effectively, the project manager must have certain set of skills.

➢ Job Responsibilities of a Software Project Manager:

1. Involves with the senior managers in 'the process of appointing team members
2. Builds the project team and assigns tasks to various team members
3. Responsible for effective project planning and scheduling, project monitoring and control activities in order to achieve the project objectives
4. Acts as a communicator between the senior management and the other persons involved in the project like the development team and internal and external stakeholders
5. Effectively resolves issues (if any) that arise between the team members by changing their roles and responsibilities
6. Modifies the project plan (if required) to deal with the situation.

➢ Skills Necessary for Software Project Management:

Although the actual skills for effective project management develop with experience, every project manager must exhibit some basic skills that are listed below.

1. Must have the knowledge of different project management techniques like risk management, configuration management, cost estimation techniques, etc.
2. Must have the ability to make judgment, since project management frequently requires making decisions.
3. Must have good grasping power to learn the latest technologies to adapt to project requirements.
4. Should be open-minded enough to accept new ideas from the project members. In addition, he should be creative enough to come up with new ideas.
5. Should have good interpersonal, communication, and leadership qualities in order to get work done from the team members.

Project Planning:

Before a project begins, the manager and the software development team must make a proper plan. The project planning involves estimation of the work to be done, the resources that will be required, the effort and money to be invested and the time to be complete (i.e. from start to the completion of the project).

The software planning begins with determining the software scope. Before the software scope could be established, the function and performance of the software should be well understood

and well defined. The software scope describes the data and control to be processed function, performance, constraints and reliability.

Once the project scope has been well defined, it is necessary to study the feasibility of the software or project. By software feasibility, it means that if we can build the software to meet the defined scope, the software feasibility can be defined for the technology used, financial support, total time required and resources needed for the development of the software.

The success of any software or project is based on proper planning. After planning, we will be able to track or monitor the project in course of time.

- **THE SPMP DOCUMENT:**

SPMP stands for Software Project Management Plan. Document used for proper project management includes:

1. **Introduction:**
   a. Project Overview
   b. Project Deliverables
      - Preliminary Project Plan
      - Requirement Specification
      - Analysis [Object Model, Dynamic Model and User Interface]
      - Architectural Specification
      - Component/object Specification
      - Source code
      - Test Plan
      - Final Product/Demo

2. **Project Organization:**
   a. Process Model [Waterfall, Spiral, Prototype, etc.]
   b. Organization Structure
      - Team Members
      - WBS [Work Breakdown Structure]
      - Deliverable per week/month

3. **Managerial Process:**
   a. Management Objectives and Priorities
   b. Assumptions, Dependencies and Constraints
   c. Risk Management
      - Risk Identification
      - Risk Mitigation
   d. Monitoring and Controlling Mechanisms

4. **Technical Process:**
   a. Methods/tools and techniques
   b. Software Documentations, etc.

5. **Work Elements, Schedule and Budget:**

METRICS FOR PROJECT SIZE ESTIMATION:

Concerned with the cost associated with the project, the software considered to constitute very small cost than overall computer based system cost. But as the system become more advance, the cost of software has contributed a lot to the overall cost of the project.

The inappropriate estimation might lead to conditions that are disastrous and the developer has to bear a loss. Since, any project has its association with human, technical, environment, political aspects, remarkable estimation can be done only considering these aspects.

A general project estimation can be done by:

1. Delaying estimation until late in the project.
2. Estimating projects based upon similar past projects.
3. Using relatively simple decomposition technique.
4. Using one or more empirical models for cost and effort estimations.

The first two techniques '1' and '2' are not efficient because estimation should be done as early as possible and that not all past experiences resemble the same criterion and might not fulfill the requirement of the current project.

The technique '3' is a conventional estimation technique because it follows the "divide and conquer".

The technique '4' is the empirical estimation model, it can be used to complement the decomposition technique.

- **APPROACHES FOR PROJECT SIZE ESTIMATION:**

1. LOC Approach:

Consider a software package to be developed for a CAD application for mechanical components. The software needs to have interface with various computer graphics peripherals including a mouse, digitizer, high-resolution color display and laser printer. It will need to accept 2 or 3 dimensional data. CAD database need to be maintain. Design and analysis module need to be developed to produce the required output, which will be displayed on display devices. The software will be designed to control and interact with peripheral devices.

- UICF = User Interface and Control Facilities
- 2DGA = Two Dimension Geometric Analysis
- 3DGA = Three Dimension Geometric Analysis
- DBM = Database Management
- CGDF = Computer Graphics Display Facilities
- PCF = Peripheral Control Function
- DAM = Design Analysis Module

Estimation Table for LOC Method

Function	Estimated LOC (Lines of Code)
UICF	2300
2DGA	5300
3DGA	6800

DBM	3350
CGDF	4950
PCF	2100
DAM	8400
Total LOC	**33200**

Total LOC for given CAD application = 33,200

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC per month, based on the labor rate of $8000 per month and the cost per line of code is approximately (8000/620 = $12.9 = $13). So, based on LOC estimate the total estimated project cost is ($13 * 33200 = $4,33,000) and the estimated effort is (33200/620 = 53.54 = 54 persons)

2. Function Point Metrics Approach:

Function Point Metric is indirect measure to estimate size. This approach can be used effectively as a means for measuring the functionality delivered by a system. Using historical data, the Function Point Metrics Approach can be used to:

a. Estimate the cost or effort required to design, code and test the software.
b. Predict the number of errors that will be encountered during the testing.
c. Forecasts the number of components and/or the number of projected sources lines in the implemented system.

Function Point Metrics are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessment of software complexity. Information domain values are defined in the following manner:

- Number of external inputs (EIs) from user
- Number of external outputs (EOs) by system
- Number of external inquires (EQs) for the system
- Number of internal logical files (ILFs) used in system
- Number of external interface files (EIFs) used in system

Once these data have been collected, we use this approach. To compute Function Point Metrics we use the following relationship:

- FP = Count *(sum of all FP entries)* total * (0.65+0.01*E(Fi))
- Fi = 0 – 14

Example:

Measurement Parameters	Count	Weight	Count * Weight
Number of user inputs	40	4	160
Number of user outputs	25	5	125
Number of user inquires	12	4	48
Number of logical files	4	7	28
Number of external interfaces	4	7	28
Count Total			389

Now, FP can be calculated as:

FP = count total * (0.65 + 0.01 * ∑ (Fi))

Where, Fi is the complexity adjustment factor based on response to various questions like:

- Does the system require reliable backup?
- Are data communication required?
- Is performance critical?
- Are the master's file updated online?

PROJECT ESTIMATION TECHNIQUES:

We discussed various parameters involving project estimation such as size, effort, time and cost. Project manager can estimate the listed factors using some techniques such as:

1. Empirical Estimation Technique:

It is based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are: Expert judgment technique and Delphi cost estimation.

This technique uses empirically derived formulae to make estimation. These formulae are based on LOC or FPs. The general form is:

Effort = Tuning Coefficient * (size) $^{\text{Exponent}}$

Here,

- Effort = Usually derived as person month of effort required
- Tuning Coefficient = Either a constant or a number derived based on complexity of project
- Size = usually LOC or FP
- Exponent = empirically derived
- i.e. $E = A + B\ (ev)^{C}$,
  A, B, C are constant, empirically derived

Some LOC oriented Approach are:

a. Walston Felix Model
   $E = 5.2 * (KLOC)^{0.91}$

b. Bailey Basili Model
   $E = 5.5 + 0.73 + (KLOC)^{1.16}$

Some Function Point oriented Approach are:

a. Albrecht and Gattney Model
   $E = 13.39 + 0.545 * FP$

b. Matson Barnett and Mellichamp Model

$$E = 585.7 + 150.12 * FP$$

- **Expert Judgment Technique:**

This technique is widely used for cost estimation. It is top-down estimation technique i.e. it first focuses on the system level costs such as computing resources and personnel required to develop the system as well as the costs of configuration management, quality assurance, system integration, training and publications. Personnel costs are estimated by examining the cost of similar past projects.

Expert judgment relies on the experience, background and business sense of one or more key people (experts) in the organization. An expert might arrive at a cost estimate in the following measures:

Example:

Consider a system to be developed is a process control system similar to one that was developed last 10 years in 10 months of a cost of $ 1 million. The new system has similar control functions, but has 25% more activities to control, thus we will increase our time and cost estimates by 25%. The previous system was the first of its type that we develop however we will use the same computer and external sensing/controlling devices and many of the same people are available to develop the new system. So, we can reduce our estimate by 20%.

Furthermore, we can reuse much of the low-level code from the previous product, which reduce the time and cost estimate by 25%. The net effect of these consideration is a time and test reduction of 20% which result is an estimate of $ 800,000 and 8 months development time. We know that the customer has budgeted $ 1 million and 1 year delivery time for system. Therefore, we add a small margin of safety and bid the system at $ 850,000 and 9 months development time.

- **Delphi Cost Estimation Technique:**

The Delphi estimation technique can be adapted to software cost estimation in the following manner:

- A coordinator provides each estimator with the system definition document and a form for recording a cost estimate.
- Estimators study the definition and complete their estimates anonymously. They may ask questions of the coordinator but they do not discuss their estimates with one another.
- The coordinator prepares and distributes a summary of the estimator's response and includes any usual rationales (reasons and principle on which a decision is based) noted by the estimators.
- Estimator complete another estimate using the results from the previous estimate
- Estimators whose estimates differ sharply from the group may be asked to provide justification for their estimates.
- The process is iterated for as many rounds as required. No group discussion is allowed during the entire process.

It is possible that several round of estimates will not lead to a consequence estimate. In this case, the coordinator must discuss the issues involved with each estimator to determine the reasons

for the differences. The coordinator may have to gather additional information and present it to the estimators in order to resolve the difference in view point.

2. Heuristic Technique:

Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: single variable model and the multi variable model.

Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size. A single variable estimation model takes the following form:

$$\textit{Estimated Parameter} = c_1 * e^{d_1}$$

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). *Estimated Parameter* is the dependent parameter to be estimated. The dependent parameter to be estimated could be effort, project duration, staff size, etc. c_1 and d_1 are constants. The values of the constants c_1 and d_1 are usually determined using data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.

A multivariable cost estimation model takes the following form:

$$\textit{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \ldots$$

Where e_1, e_2, … are the basic (independent) characteristics of the software already estimated, and c_1, c_2, d_1, d_2, … are constants. Multivariable estimation models are expected to give more accurate estimates compared to the single variable models, since a project parameter is typically influenced by several independent parameters. The independent parameters influence the dependent parameter to different extents. This is modeled by the constants c_1, c_2, d_1, d_2, … . Values of these constants are usually determined from historical data. The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.

- **COCOMO Model:**

It stands for Constructive Cost Model and is proposed by Barry Boehm. Barry Boehm postulates that any software development can be classified into three categories based on the development complexity.

1. **Organic:**

It deals with well understood application program, size of development team is reasonably small, and team members are experienced in developing similar type of projects.

2. **Semi-Detached:**

Project team consists of a mixture of experienced and inexperienced staff. Team members have limited experience on related systems and may be unfamiliar with some aspects of system being developed.

3. Embedded:

The software is strongly coupled to complex hardware, or real-time systems. They have strict procedure and operation with more than 300 KLOC.

☑ Basic COCOMO Model:

Gives only an approximate estimation:

- Effort = $a_1(KLOC)^{a2}$
- $T_{dev}=b_1 (Effort)^{b2}$
  - ✓ KLOC is theestimated kilolines of source code,
  - ✓ $a_1a_2b_1b_2$ are constants for different categories of software products,
  - ✓ T_{dev} is the estimated time to develop the software in months,
  - ✓ Effort estimation is obtained in terms of person months (PMs).

Development Effort Estimation

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

Development Time Estimation

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

3. Analytical Estimation technique:

It derives the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis, Halstead's software science is an example of an analytical technique.

It can be used to derive some interesting results starting with a few simple assumptions. It is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.

Halstead's Software Science – An Analytical Technique

Halstead's software science is an analytical technique to measure size, development effort, and development cost of software products. Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time.

For a given program, let:

- ☑ η_1 be the number of unique operators used in the program,
- ☑ η_2 be the number of unique operands used in the program,
- ☑ N_1 be the total number of operators used in the program,
- ☑ N_2 be the total number of operands used in the program.

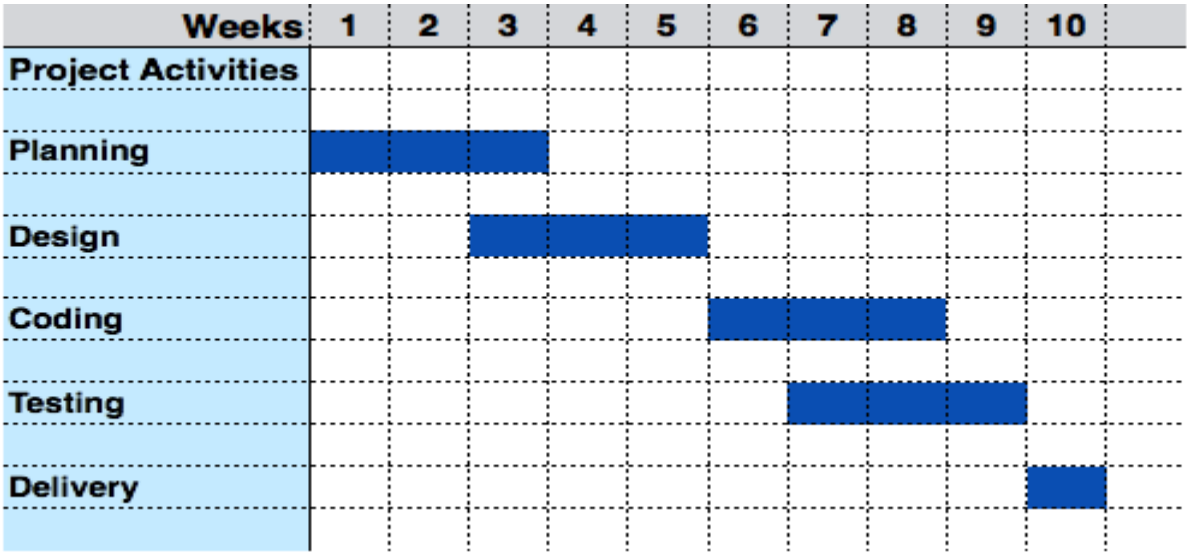
PROJECT SCHEDULING:

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and they arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to:

- ☑ Break down the project tasks into smaller, manageable form
- ☑ Find out various tasks and correlate them
- ☑ Estimate time frame required for each task
- ☑ Divide time into work-units
- ☑ Assign adequate number of work-units for each task
- ☑ Calculate total time required for the project from start to finish

For project scheduling, creating timeline chart is an efficient option.



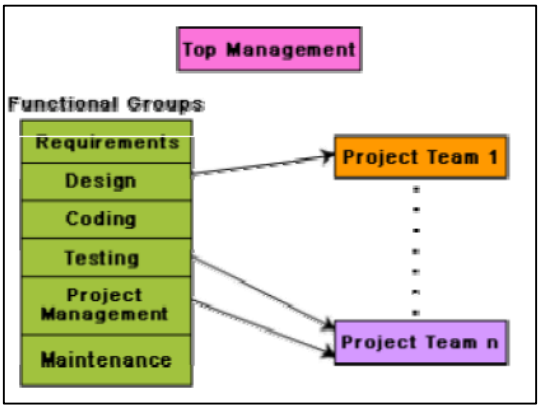
ORGANIZATION AND TEAM STRUCTURE:

1. Organization Structure:

Usually every software development organization handles several projects at any time. Software organizations assign different teams of engineers to handle different software projects. Each type of organization structure has its own advantages and disadvantages so the issue "how is the organization as a whole structured?" must be taken into consideration so that each software project can be finished before its deadline.

There are essentially two broad ways in which a software development organization can be structured:

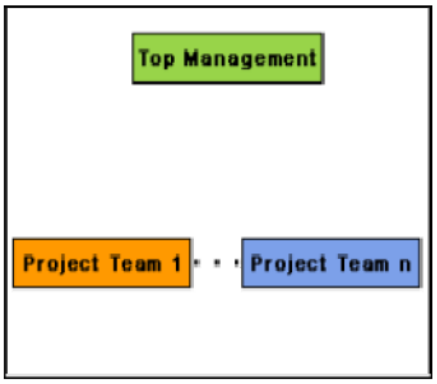
a. Functional Format



In the functional format, the development staff are divided based on the functional group to which they belong. The different projects borrow engineers from the required functional groups for specific phases to be undertaken in the project and return them to the functional group upon the completion of the phase.

In the functional format, different teams of programmers perform different phases of a project. For example, one team might do the requirements specification, another do the design, and so on. The partially completed product passes from one team to another as the project evolves. Therefore, the functional format requires considerable communication among the different teams because the work of one team must be clearly understood by the subsequent teams working on the project. This requires good quality documentation to be produced after every activity.

b. Project Format:



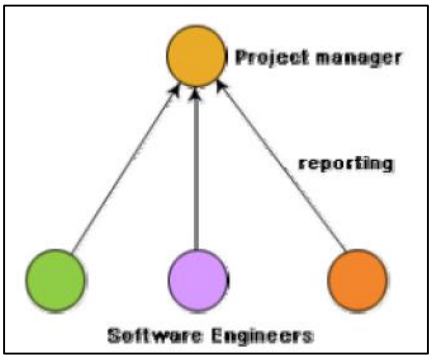
In the project format, the project development staff are divided based on the project for which they work.

In the project format, a set of engineers is assigned to the project at the start of the project and they remain with the project till the completion of the project. Thus, the same team carries out all the life cycle activities. Obviously, the functional format requires more communication among teams than the project format, because one team must understand the work done by the previous teams.

2. Team Structure:

Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized. There are mainly three formal team structures: chief programmer, democratic, and the mixed team organizations although several other variations to these structures are possible. Problems of different complexities and sizes often require different team structures for chief solution.

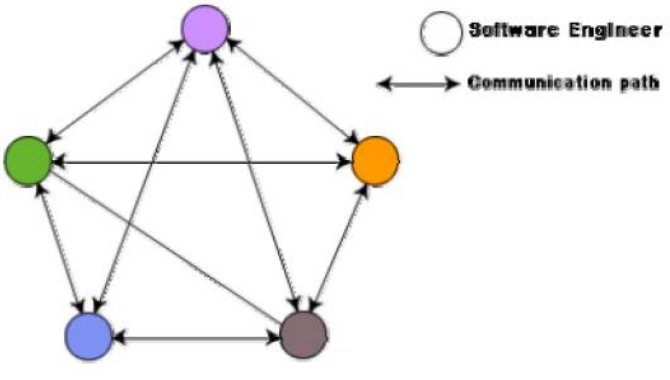
a. Chief Programmer Team:



In this team organization, a senior engineer provides the technical leadership and is designated as the chief programmer. The chief programmer partitions the task into small activities and assigns them to the team members. He also verifies and integrates the products developed by different team members. The chief programmer provides an authority, and this structure is arguably more efficient than the democratic team for well-understood problems. However, the chief programmer team leads to lower team morale, since team-members work under the constant supervision of the chief programmer. This also inhibits their original thinking. The chief programmer team is subject to single point failure since too much responsibility and authority is assigned to the chief programmer.

The chief programmer team is probably the most efficient way of completing simple and small projects since the chief programmer can work out a satisfactory design and ask the programmers to code different modules of his design solution. For example, suppose an organization has successfully completed many simple MIS projects. Then, for a similar MIS project, chief programmer team structure can be adopted. The chief programmer team structure works well when the task is within the intellectual grasp of a single individual. However, even for simple and well-understood problems, an organization must be selective in adopting the chief programmer structure. The chief programmer team structure should not be used unless the importance of early project completion outweighs other factors such as team morale, personal developments, life-cycle cost etc.

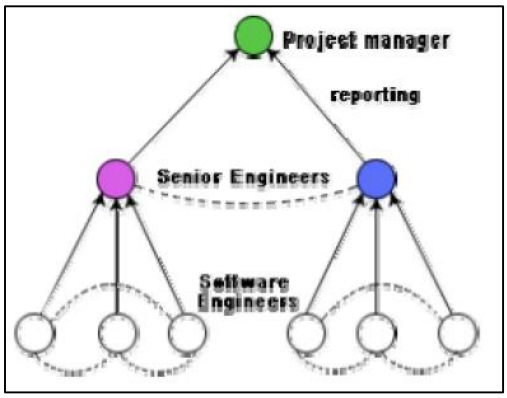
b. Democratic Team:



The democratic team structure, as the name implies, does not enforce any formal team hierarchy. Typically, a manager provides the administrative leadership. At different times, different members of the group provide technical leadership.

The democratic organization leads to higher morale and job satisfaction. Consequently, it suffers from less man-power turnover. Also, democratic team structure is appropriate for less understood problems, since a group of engineers can invent better solutions than a single individual as in a chief programmer team. A democratic team structure is suitable for projects requiring less than five or six engineers and for research-oriented projects. For large sized projects, a pure democratic organization tends to become chaotic. The democratic team organization encourages egoless programming as programmers can share and review one another's work.

c. Mixed Control Team Organization:



The mixed team organization, as the name implies, draws upon the ideas from both the democratic organization and the chief-programmer organization. This team organization incorporates both hierarchical reporting and democratic set up. In figure below, the democratic connections are shown as dashed lines and the reporting structure is shown using solid arrows. The mixed control team organization is suitable for large team sizes. The democratic arrangement at the senior engineer's level is used to decompose the problem into small parts. Each democratic setup at the programmer level attempts solution to a single part. Thus, this team organization is eminently suited to handle large and complex programs. This team structure is extremely popular and is being used in many software development companies.

SOFTWARE PROJECT STAFFING:

All the management activities that involve filling and keeping filled the positions that were established in the project organizational structure by well-qualified personnel.

MAJOR ISSUES IN STAFFING:

The major issues in staffing for a software engineering project are as follows:

- ☑ Project managers are frequently selected for their ability to program or perform engineering tasks rather than their ability to manage (few engineers make good managers).
- ☑ The productivity of programmers, analysts, and software engineers varies greatly from individual to individuals.
- ☑ There is a high turnover of staff on software projects especially those organized under a matrix organization.
- ☑ Universities are not producing a sufficient number of computer science graduates who understand the software engineering process or project management.
- ☑ Training plans for individual software developers are not developed or maintained.

QUALITY OF SOFTWARE ENGINEER:

- ❖ **Education:**

Does the candidate have the minimum level of education for the job? Does the candidate have the proper education for future growth in the company?

- ❖ **Experience:**

Does the candidate have an acceptable level of experience? Is it the right type and variety of experience?

- **Training:**

Is the candidate trained in the language, methodology, and equipment to be used, and the application area of the software system?

- **Motivation:**

Is the candidate motivated to do the job, work for the project, work for the company, and take on the assignment?

- **Commitment:**

Will the candidate demonstrate loyalty to the project, to the company, and to the decisions made?

- **Self-motivation:**

Is the candidate a self-starter, willing to carry a task through to the end without excessive direction?

- **Group affinity:**

Does the candidate fit in with the current staff? Are there potential conflicts that need to be resolved?

- **Intelligence:**

Does the candidate have the capability to learn, to take difficult assignments, and adapt to changing environments?

RISK MANAGEMENT:

Risk management is a series of steps that help a software team to understand and manage uncertainty. It's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan that 'should the problem actually occur'. Risk management is a part of umbrella activities.

Simplistically, we can think of a risk as something that we prefer not to have happen. Risks may threaten the project, the software that is being developed or the organization.

CATEGORIES OF RISKS:

There are, three related categories of risk:

1. **Project Risks:**

Risks which will affect the project schedule or resources. For example, stuff turnover that is an experience team member of a project left the organization.

2. **Product Risks:**

Risks that affect the quality or performance of the software being developed. For example a component isn't performing as expected.

3. **Business Risks:**

Risks that affect the organization developing or procuring the software. For example, a competitor is developing a similar product that will challenge the product being developed.

Process of risks management:

Risks has to be listed and recovery actions has to be predetermined to avoid bigger impacts when development is underway. Following are the stages of risk management.

1. **Risk Identification:**

It is a systematic attempt to specify threats to the project plan (i.e. estimates, schedule, resource loading, etc.). Risks can be removed or avoided only if they can be identified. Basically all the risks are classified in two categories:

a. **Generic Risks:**

Generic risks are those risks which are common to all. A risk item checklist is created based upon the known and predictable risks to identify generics risks are as follows:

- **Product Size:** Risk associated with the product size of the software to be built.
- **Business Impact:** Risk associated with market place and constraints in management.
- **Customer Characteristics:** Risk associated with intensions of customer and the developer ability to communicate with the customer in timely manner.
- **Process Definition:** Risks associated with the degree to which the software process have been defined and is followed by development organization.
- **Development Environment:** Risk associated with the availability and quality of the tools to be used to build the product.
- **Technology to be built:** Risk associated with the complexity of the system to be built and due to advance of technology.

b. **Product Specific Risks:**

Product specific risks are those that are associated or relevant only to certain products. These risks occurs depending upon the nature and characteristics of product like the technology implemented, the people involved and the environment conditions. The identification of such risks requires proper knowledge of the special characteristics of the product and may vary from products to products.

Risk Projection and Risk Table:

Risk projection also known as risk estimation is the process of rating the risk based upon its likelihood or provability of occurrence and the consequence of the problems associated with the risks. It is carried out by project planner along with other managers and technical staff. The risk table is used in risk projection.

Risk Table

Risks	Category	Probability	Impact
Size estimate may be significantly low	PS	60%	2
Large numbers of inputs than planned	PS	30%	3

Less reuse than planned	PS	70%	2
End user resists system	BU	40%	3
Delivery deadline will be tight	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 turnover will be tight	ST	30%	2

Full Form	Impact Value
PS = Product Size	1 = Catastrophic
BU = Business Risk	2 = Critical
CU = Customer Characteristics	3 = Marginal
TE = Technology and Environment	4 = Negligible

2. **Risk Assessment (Refinement):**

Whenever we carryout project planning, at the previous stages, we state or specify the risk that occur in general, risks that are common, distinct and more likely to occurs. But as time passes by, we become more familiar and more involved in the project. As a result, we learn more and more about the project and the risk associated with it. Consequently, we will be able to refine the risk into more detailed and advance form, and finally we will be able to mitigate (avoid) monitor and manage (RMMM) the risk occurrence.

One way of risk assessment/ refinement is to represent the risk in condition-transition-consequence (CTC) format i.e. risk is stated in following form:

Given that <condition> then there is <consequence>

3. **Risk Containment:**

It assumes that mitigation effort have been failed and risk become a reality. Example: Number of people announces that they will be leaving. When the project is well under way if the mitigation strategy has been followed, backup is available information is documented and knowledge has been dispersed across the team.

Further, those individual who are leaving are asked to stop all work and spend their last week in "knowledge transfer mode" with the new comers who must be added to the team to get up to speed.

SOFTWARE CONFIGURATION MANAGEMENT:

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items".

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun. One of the concept used for SCM is baseline.

Baseline:

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Necessarily of Software Configuration Management:

- Reduced redundant work.
- Effective management of simultaneous updates.
- Avoids configuration-related problems.
- Facilitates team coordination.
- Helps in building management; managing tools used in builds.
- Defect tracking: It ensures that every defect has traceability back to its source.

Change Control/Activities of SCM:

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps:

- **Identification**: A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation**: Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis**: The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control**: If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- **Execution**: If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request**: The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

Source Code Control System:

Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored. It was originally developed at Bell Labs in 1972 by Marc Rochkind for an IBM System/370 computer running OS/360.

Software is typically upgraded to a new version by fixing bugs, optimizing algorithms and adding extra functions. Popular software has many versions such as Java. Changing software causes problems that require version control to solve.

- ☑ Source code takes up too much space because it is repeated in every version.
- ☑ Passing optimization from one version to other versions is difficult.
- ☑ It is hard to acquire information about when and where changes occurred.
- ☑ Finding the exact version which the client has problems with is difficult.

SCCS was built to solve these problems. SCCS has nine major versions which are designed to help programmers control changes in software. Two specific implementations using SCCS are: PDP 11 under UNIX and IBM 370 under the OS.

Revision Control System (RCS):

The **Revision Control System** (**RCS**) is a software implementation of revision control that automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, procedural graphics, papers, and form letters. RCS is also capable of handling binary files, though with reduced efficiency and efficacy. Revisions are stored with the aid of the diff utility.

RCS was initially developed in the 1980s by Walter F. Tichy while he was at Purdue University as a free and more evolved alternative to the then-popular Source Code Control System (SCCS). It is now part of the GNU Project but is still maintained by Purdue University.

RCS operates only on single files, has no way of working with an entire project. Although it provides branching for individual files, the version syntax is cumbersome. Instead of using branches, many teams just use the built-in locking mechanism and work on a single *head* branch.

Advantages

- ➢ Simple structure and easy to work with
- ➢ Revision saving is not dependent to the central repository

Disadvantages

- ➢ Poor security system when it comes to an intentional misconduct
- ➢ Only one user can work on a file at a time