CHAPTER: 4

SOFTWARE PROJECT MANAGEMENT

(16 Marks)

4.1 The management spectrum: 4 p's

What is SPM? And It's need:

- SPM is to be done in scientific ways to ensure the successful development of software product.
- SPM involves the knowledge, technique & tools necessary to manage the development of software products.
- In SPM are project scheduling, tracking, risk management, life cycle models, development team organization, management of technical people.
- Building computer software is a complicated process, and hence SPM is very much required.
- The process of SPM includes thousands of activities which linked with each other. Without SPM approach it is impossible to develop a successful software product.
- Software Project Managements is an umbrella activity within Software Engineering
- Project Management involves the Planning, Monitoring and Control of People, Process and Events that occur as Software evolves from preliminary concept to an operational implementation.
- Project Management begins before any technical activity and continues throughout the Definition, Development and Support of Computer Software.
- Effective Project Management focuses on 4 P's

- The People
- The Product
- The Process
- The Project

4p's of Management Spectrum:

- Effective software project management focuses on these items (in this order)
  - The people
    - Deals with the cultivation of motivated, highly skilled people
    - Consists of the stakeholders, the team leaders, and the software team
  - The product
    - Product objectives and scope should be established before a project can be planned
  - The process
    - The software process provides the framework from which a comprehensive plan for software development can be established
  - The project
    - Planning and controlling a software project is done for one primary reason…it is the only known way to manage complexity.

1. **The people:**

- People management capability maturity model(PM-CMM) has been developed by the s/w Engg. Institute.
- PM-CMM is developed for motivating organization to undertake more complicated applications. It helps organization to retain, attract, motivate & grow the talent that enhances the overall development capability of an organization.
- PM-CMM defines the following key practice areas for s/w people:

1. recruiting, 2. selection, 3. performance management,
4. training, 5. compensation, 6. career development
7. Organization & work design 8. Team/Culture development.

- The various groups are involved in s/w project management.
  1. Stake holders
  2. Team leaders
  3. Software team
  4. Agile Teams

2. The Product (Software Application):

- Before a Product can be Planned: -
- Product Objectives and Scope should be Planned
- Alternative solutions should be considered
- Technical and Management Constraints should be identified.
- The scope of the software development must be established and bounded
  - Context – How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?
  - Information objectives – What customer-visible data objects are produced as output from the software? What data objects are required for input?
  - Function and performance – What functions does the software perform to transform input data into output? Are there any special performance characteristics to be addressed?
- Software project scope must be unambiguous and understandable at both the managerial and technical levels
- Problem decomposition to be considered
- Two major areas of problem decomposition
  - The functionality that must be delivered
  - The process that will be used to deliver it

3. The Process:

- **Getting started:**
  - The project manager must decide which process model is most appropriate based on
    - The customers who have requested the product and the people who will do the work
    - The characteristics of the product itself
    - The project environment in which the software team works

  - Once a process model is selected, a preliminary project plan is established based on the process framework activities
  - Process decomposition then begins
  - The result is a complete plan reflecting the work tasks required to populate the framework activities
- Project planning begins as a melding of the product and the process based on the various framework activities

4. The Project:

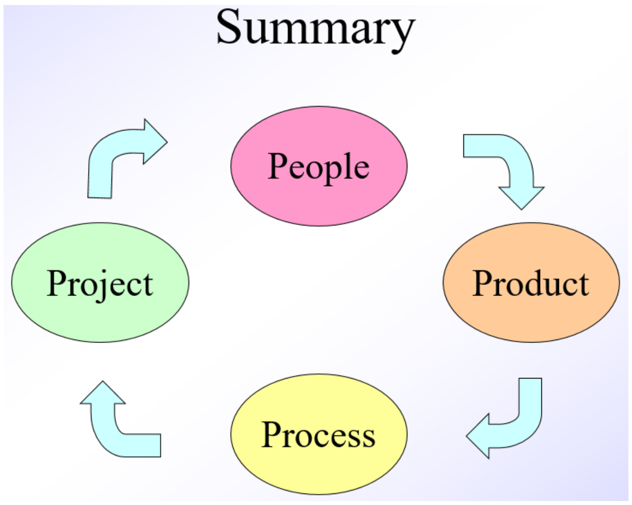
- In order to manage a Successful Software Project, we must understand what can go wrong so that the problem can be avoided,
- John Reel defines 10 Signs that indicate an Information Systems Project

is in Jeopardy: -

1. Software People don't understand their Customers' needs
2. The Product Scope is poorly defined
3. Changes are managed poorly
4. The chosen Technology changed
5. Business needs change or ill defined
6. Project Deadlines are unrealistic
7. Users are resistant
8. Sponsorship is lost (or never obtained)
9. The Project Team Lacks People with appropriate skills
10. Managers / Practitioners avoid best practice and Lessons learned.

- **The Project: A Common Sense Approach:**
- **Start on the right foot**
  - Understand the problem; set realistic objectives and expectations; form a good team
- **Maintain momentum**
  - Provide incentives to reduce turnover of people; emphasize quality in every task; have senior management stay out of the team's way
- **Track progress**
  - Track the completion of work products; collect software process and project measures; assess progress against expected averages
- **Make smart decisions**
  - Keep it simple; use COTS or existing software before writing new code; follow standard approaches; identify and avoid risks; always allocate more time than you think you need to do complex or risky tasks
- **Conduct a post mortem analysis**
  - Track lessons learned for each project; compare planned and actual schedules; collect and analyze software project metrics; get feedback from team's members and customers; record findings in written form

4p's of Management Spectrum Summary cycle:-



4.2 Metrics for size estimation:

- Estimation of the size of software is an essential part of Software Project Management.
- It helps the project manager to further predict the effort and time which will be needed to build the project.
- The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
- Two metrics are popularly being used widely to estimate size:
  1. Lines of code (LOC)
  2. Function point (FP).

1. Function Point (FP): -

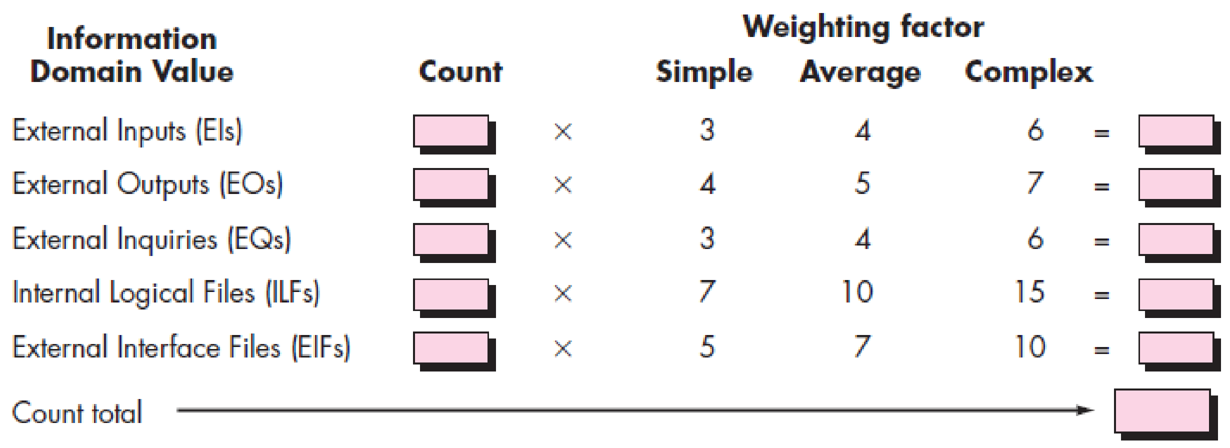
- The *function point* (FP) *metric* can be used effectively as a means for measuring the functionality delivered by a system.
- The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different Functions or features it supports.
- A software product supporting many features would certainly be of larger size than a product with less number of features.
- In simplest form in this method, the number and type of functions supported by the software are utilized to find FPC (function point count).
- Using historical data, the FP metric can then be used to
  (1) Estimate the cost or effort required to design, code, and test the software.

(2) Predict the number of errors that will be encountered during testing.

(3) Forecast the number of components and/or the number of projected source lines in the implemented system.

- Function points are derived from software's information domain and qualitative assessments of software complexity. Information domain values are defined in the following manner:
  - **Number of external inputs (EIs).** Each *external input* originates from a user or is transmitted from another application.
  - **Number of external outputs (EOs).** Each *external output* is derived data within the application that provides information to the user. In this context external output refers to reports, screens, error messages, etc.
  - **Number of external inquiries (EQs).** An *external inquiry* is defined as an online input that results in the generation of some immediate software response in the form of an online output.
  - **Number of internal logical files (ILFs).** Each *internal logical file* is a logical grouping of data (Files) that resides within the application's boundary and is maintained via external inputs.
  - **Number of external interface files (EIFs).** Each *external interface file* is a logical grouping of data (files) that resides external to the application but provides information that may be of use to the application.

Once these data have been collected, the following table is completed to get count total.



Information Domain Value	Count		Weighting factor				
			Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total						→	

- To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

where count total is the sum of all FP entries obtained from above table.

The *Fi* (*i* = 1 to 14) are *value adjustment factors* (VAF) based on responses to the following questions:

1. Does the system require reliable backup and recovery?

2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user? Each of these questions is answered using a scale that ranges from 0 (not important) to 5 (most important).

Advantages:

- It can be easily used in the early stages of project planning.
- It is independent on the programming language.
- It can be used to compare different projects even if they use different technologies (database, language etc.)

Disadvantages:

- It is not good for real time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

2. Lines of Code (LOC): -

- LOC is the simplest among all metrics available to estimate project size.
- This metric is very popular because it is the simplest to use.
- LOC is any line of program text that is not comments or blank line regardless of number of statement on the line.
- It includes all lines containing program headers, declarations, executable & non executable statements.
- Determining the LOC count at the end of a project is a very simple job.
- However, accurate estimation of the LOC count at the beginning of a project is very difficult.
- In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted.

- To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.
- **Measurement methods**

- There are two major types of SLOC (Source lines of code) measures: physical SLOC (LOC) and logical SLOC (LLOC).
- In this case blank lines in excess of 25% are not counted toward lines of code.
- **Consider following C code as an example** of the ambiguity encountered when determining SLOC:

  ```
  for (i = 0; i < 100; i += 1) printf("hello"); /* How many lines of code is this? */
  ```

  In this example we have:

  - 1 Physical Lines of Code (LOC)
  - 2 Logical Line of Code (LLOC) (for statement and printf statement)
  - 1 comment line
- Depending on the programmer and/or coding standards, the above "line of code" could be written on many separate lines:

  ```
  for (i = 0; i < 100; i += 1)
  {
     printf("hello");
  }/* Now how many lines of code is this? */
  ```

  In this example we have:

  - 4 Physical Lines of Code (LOC): is placing braces work to be estimated?
  - 2 Logical Line of Code (LLOC): what about all the work writing non-statement lines?
  - 1 comment line: tools must account for all code and comments regardless of comment placement.

Advantages:
- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

Disadvantages:
- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

4.3 Project cost estimation approaches:

- Estimation of various project parameters is a basic project planning activity.
- The important project parameters that are estimated include: project size, effort required to develop the software, project duration, and cost.
- These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling.
- There are three broad categories of estimation techniques:
    1. Empirical estimation techniques
    2. Heuristic techniques
    3. Analytical estimation techniques

1. <u>Empirical techniques:</u>

- Empirical estimation techniques are based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful.
- Two popular empirical estimation techniques are:
    1. Expert judgment technique and
    2. Delphi cost estimation.

1. **Expert Judgement:**

- o Expert judgment is one of the most widely used estimation techniques.
- o In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly.
- o Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate.
- o However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently.
- o Further, an expert making an estimate may not have experience and knowledge of all aspects of a project.
- o For example, he may be conversant with the database and user interface parts but may not be very knowledgeable about the computer communication part. A more refined form of expert judgment is the estimation made by group of experts. Estimation by a group of experts minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias.

2. Delphi Estimation:

- Delphi cost estimation approach tries to overcome some of the shortcomings of the expert judgment approach.
- Delphi estimation is carried out by a team comprising of a group of experts and a coordinator.
- In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate.
- Estimators complete their individual estimates anonymously and submit to the coordinator.
- In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation.
- The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators.
- Based on this summary, the estimators re-estimate.
- This process is iterated for several rounds.
- However, no discussion among the estimators is allowed during the entire estimation process. After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.

2. Heuristic techniques:

- 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:
  1. Single variable model
  2. Multi variable model.

1. **Single variable estimation**: These 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:

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

Where, e is already estimated (independent variable), c1 and d1 are constants. The values of the constants c1 and d1 are usually determined using data collected from past projects. **The basic COCOMO model is an example of single variable cost estimation model.**

2. **A multivariable estimation:** This cost estimation model takes the following form:

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

Where e1, e2, … are the basic (independent) characteristics of the software already estimated, and c1, c2, d1, d2, … are constants. Multivariable estimation models are expected to give more accurate estimates compared to the single variable models. **The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.**

- Different Heuristic **estimation** techniques are:

- Price-to-win
- Top-down
- Bottom up
- Expert's opinion
- Parkinson's Law
- Analogy

3. **Analytical techniques:**

- Analytical estimation techniques derive 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.
- **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

Example: Let us consider the following C program:

```
main ()
  {
     int a, b, c, avg;
     scanf (“%d %d %d”, &a, &b, &c);
     avg = (a+b+c)/3;
     printf (“avg = %d”, avg);
  }
```

The unique operators are: main, (), {}, int, scanf, &, “,”, “;”, =, +, /, printf

The unique operands are: a, b, c, &a, &b, &c, a+b+c, avg, 3, “%d %d %d”, “avg = %d”

Therefore, n1 = 12, n2 = 11

Estimated Length = (12*log12 + 11*log11)
= (12*3.58 + 11*3.45)
= (43+38)
= 81

Volume = Length*log (23)
= 81*4.52
= 366

4.4 COCOMO, COCOMO-II

4.4.1 COCOMO: COCOMO (Constructive Cost Estimation Model) was proposed by Boehm [1981].

- This is also called as COCOMO 81model. It is a software cost estimation model.
- It provides following three stages/levels of models:
  1. Basic COCOMO
  2. Intermediate COCOMO
  3. Complete COCOMO

1. **Basic COCOMO Model**

- The basic COCOMO model gives an approximate estimate of the project parameters.
- The basic COCOMO estimation model is given by the following expressions:

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2}\ \text{PM}$$

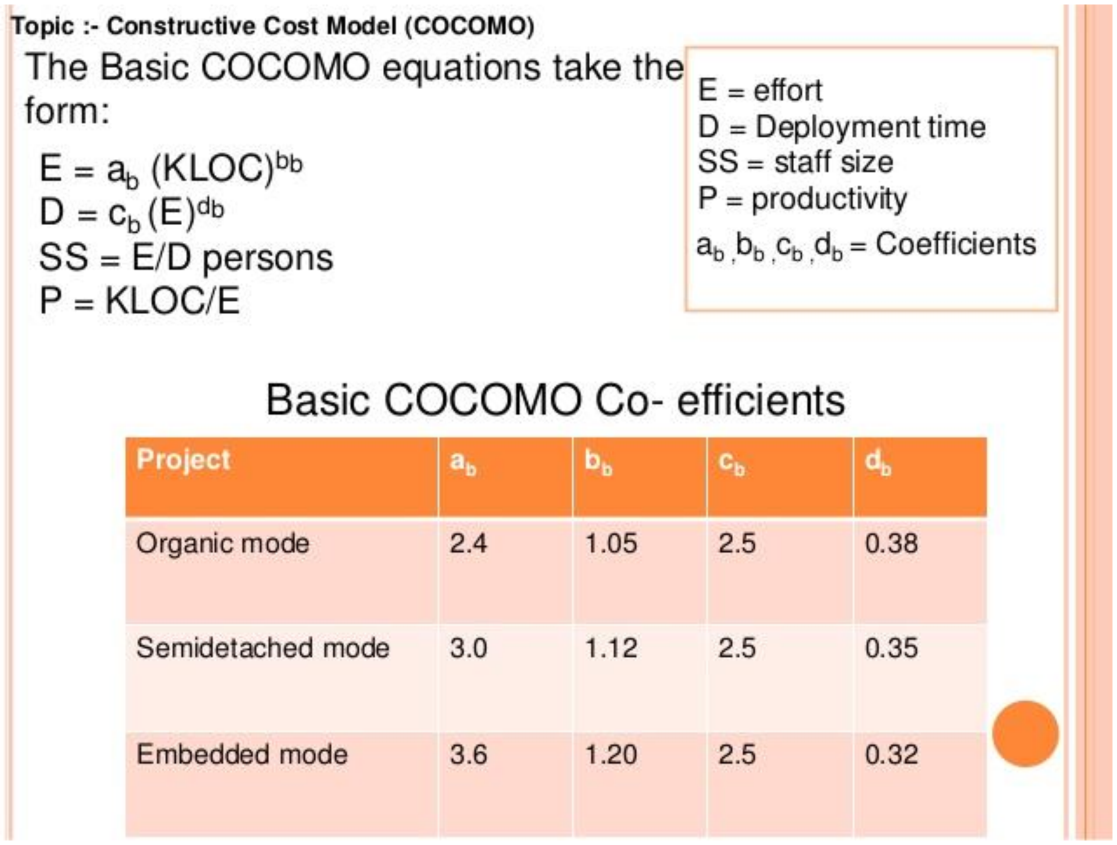
$$\text{Tdev} = b_1 \times (\text{Effort})^{b_2}\ \text{Months}$$

Where
- KLOC is the estimated size of the software product expressed in Kilo Lines of Code,
- a_1, a_2, b_1, b_2 are constants for each category of software products,
- Tdev is the estimated time to develop the software, expressed in months,
- Effort is the total effort required to develop the software product, expressed in person months (PMs).

- The model is applied to the three classes of software projects:

1. **Organic:** Relatively small project with well understood application.
2. **Semidetached:** Intermediate project with mixture of experience and inexperienced staff.

3. **Embedded:** A project strongly coupled with hardware and real time systems.



Topic :- Constructive Cost Model (COCOMO)

The Basic COCOMO equations take the form:

$E = a_b\,(KLOC)^{b_b}$
$D = c_b\,(E)^{d_b}$
$SS = E/D$ persons
$P = KLOC/E$

E = effort
D = Deployment time
SS = staff size
P = productivity
a_b, b_b, c_b, d_b = Coefficients

Basic COCOMO Co- efficients

Project	a_b	b_b	c_b	d_b
Organic mode	2.4	1.05	2.5	0.38
Semidetached mode	3.0	1.12	2.5	0.35
Embedded mode	3.6	1.20	2.5	0.32

Table 4.1

Example1: Calculate using COCOMO model (Winter-19 6 Marks)

i)Effort

ii) Project duration

iii)Average staff size

If estimated size of project is 200 KLOC using organic mode.

Ans:- Given data: size=200 KLOC mode= organic

1. **Effort:**

E = a (KLOC) b

For organic a=2.4 and b= 1.05 (refer above table 4.1)

E = 2.4 (200) 1.05

= 626 staff members

2. Project duration:

TDEV= c (E) d

Where TDEV= time for development
c and d are constant to be determined
E = effort
For organic mode, c= 2.5 and d= 0.38

TDEV= 2.5 (626) 0.38
= 29 months

3. Average staff size:

SS = E/TDEV
SS = 626/29
= 22 staffs

Example 2:
Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time. From the basic COCOMO estimation formula for organic software:

Ans: - **Effort = 2.4 x $(32)^{1.05}$**
= 91 PM

Nominal development time = 2.5 x $(91)^{0.38}$
= 14 months

Cost required to develop the product = 14 x 15,000
= Rs. 210,000/-

4.4.2 COCOMO-II:

Q. Describe COCOMO II model for evaluating size of software project with any three parameters in detail. (Summer-19 6 Marks)

- COCOMO-II is the revised version of the original COCOMO (Constructive Cost Model) and is developed at University of Southern California.
- It was built because the first COCOMO model was not compatible with the more recent practices in software development.
- It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.
- It has following features:
  - The model is simple and well tested.
  - Provides about 20% cost and 70% time estimate accuracy.

- **In COCOMO II effort is expressed as Person Months (PM).** The inputs are the Size of software development, a constant, A, and a scale factor, B. The size is in units of thousands of source lines of code (KSLOC). The constant, A, is used to capture the multiplicative effects on effort with projects of increasing size.

- The parameters used in COCOMO II are described below:-
  a. Person month- A person month is the amount of time one person spends working on the software development project for one month. The nominal effort for a given size project and expressed as person months (PM) is given by Equation 1.

$$PM_{nominal} = A * (Size)^{B}$$

Where,

A- constant

B = 0.91 + 0.01 Σ (exponent driver ratings)

- B ranges from 0.91 to 1.23

- 5 drivers; 6 rating levels each

b. Maintenance size: - It is the amount of project code that is change. It is calculated as below:-

Size= [(BaseCodeSize) *MCF] *MAF

COCOMO II uses the reuse model for maintenance when the amount of added or changed base source code is less than or equal to 20% or the new code being developed. Base code is source code that already exists and is being changed for use in the current project. For maintenance projects that involve more than 20% change in the existing base code (relative to new code being developed) COCOMO II uses maintenance size.

c. Maintenance Change Factor (MCF):- The percentage of change to the base code is called the Maintenance Change Factor (MCF).

MCF= (SizeAdded +SizeModified)/BaseCodeSize

d. Maintenance effort (MAF): COCOMO II instead used the Software Understanding (SU) and Programmer Unfamiliarity (UNFM) factors from its reuse model to model the effects of well or poorly structured/understandable software on maintenance effort.

MAF=1+ (SU.01*UNFM).

4.5 Risk Management:

Q. What is software risk? Explain types of software risks. (4M)

Definition of Risk:-

- A risk is a potential problem – it might happen and it might not.
- Conceptual definition of risk
  - Risk concerns future happenings
  - Risk involves change in mind, opinion, actions, places, etc.
  - Risk involves choice and the uncertainty that choice entails.
- Two characteristics of risk
  - Uncertainty – the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)
  - Loss – the risk becomes a reality and unwanted consequences or losses occur

Types of Software Risk:-

- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.
- To accomplish this, different categories of risks are considered.

- **Project risks**
  - They threaten the project plan. (i.e.)
  - That is If they become real, it is likely that the project schedule will slip and that costs will increase
  - This type of risk arises in software development process & they affect the project plan, budget, schedule, staffing & resources.
- **Technical risks**
  - They threaten the quality and timeliness of the software to be produced.
  - If they become real, implementation may become difficult or impossible.
  - This types of risk identifies design, implementation, interface, verification & maintenance problem.
- **Business risks**
  - It affects the capability of the software product & often causes failure of project or product.
  - They threaten the viability of the software to be built.
  - If they become real, they jeopardize the project or the product
  - **Types of Business risks Risk**
    1. Market risk – building an excellent product or system that no customer for this product then it is called a market risk.
    2. Strategic risk – building a product that no longer fits into the overall business strategy for the company

3. Sales risk – building a product that the sales department doesn't understand how to sell.
4. Management risk – when senior manager or responsible staff leaves the organization then management risk occurs.
5. Budget risk – using the overall budget of the project is called budget risk.

Steps for Risk Management

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

Risk Strategies Reactive vs. Proactive

- There are two categories of risk strategies

  1. Reactive risk strategies
  - "Don't worry, I'll think of something“
  - Reactive risk mgmt. Is risk mgmt. strategy in which when project gets into trouble then only corrective action are taken.
  - Reactive strategy monitors the project for likely risk.
  - Resources are utilized to manage such risk. In this strategy no preventive care is taken about the risk.
  - They are handled only on their occurrences.
  - Software team do not consider about the risk until something goes wrong. When risk occurs then the teams do some actions to correct the problem.

  2. Proactive risk strategies
  - It is more beneficial that reactive strategy for risk management.
  - A Proactive strategy begins its work before the technical work is initiated.
  - Risks are identified, their probability & impact are access & ranking is done accordingly.
  - software team develops a plan for managing those risk.
  - Primary objective is to avoid risk and to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner

4.5.1 RMMM strategy

Q. Explain RMMM plan with example. (Winter-19, Summer-19)

- Risk mitigation, monitoring, and management (RMMM) plan.
- A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan.
- The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.
- **There are three important issues considered in developing an effective strategy:**
  1. Risk avoidance or mitigation - It is the primary strategy which is fulfilled through a plan.
  2. Risk monitoring - The project manager monitors the factors and gives an indication whether the risk is becoming more or less.
  3. Risk management and planning - It assumes that the mitigation effort failed and the risk is a reality.

1. <u>Risk avoidance or mitigation:-</u>

- Risk mitigation is a problem avoidance activity.
- It means preventing the risk to occur(risk avoidance).
- **Following are the steps to be taken for mitigating the risk**
  - Communication with concerned staff to find out probable risk.
  - Find out and eliminate all those causes that can create risk before the project starts.
  - Information about each activity must be shared among all team members.
  - Definition standards and establish mechanisms to ensure that all models and documentation completed within the time.
  - Conduct reviews of all activities.

2. Risk monitoring

- Risk monitoring is a project tracking activity.
- It monitors risk occurrences, consequence and exposure.
- Following things must be monitored by project manager in risk monitoring.
  - Types of co-operation among team members.
  - Types of problem that are occurring.
  - Degree to which team performs team work.
  - The objective of risk monitoring is to check whether.
    a) Predictable risk really occurs or not.
    b) To ensure the steps defined to avoid the risk are applied properly or not.

3. Risk management and planning

- Involve the activities to be performed if risk becomes real and reducing the risk impact to some acceptable levels.

- The lack of a stable-computing environment is extremely hazardous to a software development team.
- In the event that the computing environment is found unstable, the development team should cease work on that system until the environment is made stable again, or should move to a system that is stable and continue working there.
- Ex. If project development process is working in good condition at the same instance some people declare that they are going to leave the project and if sufficient additional staff is available and if current development activity is known to everybody in the team then new developer can easily understand current development activity.