END TERM EXAMINATION

SEMESTER [B.TECH.]

Paper Code : ETCS	*Subject :*	*Software Engineering*

Time : 3 Hours *Maximum Marks : 75*

Note : Q.1 is compulsory. Rest of the paper contains Four units, attempt One question from each unit.

Q-1) Answer the following questions:

(a) What are the different activities in requirement analysis?

(b) What are the different dependencies present in Use Case Diagram?

(c) What are the different techniques to estimate size of program?

(d) What are the different activities of Risk Management?

(e) What are the different CMM levels? What does CMM level specifies?

Q-2) What do you understand by token count? Consider a program having

- Number of distinct operator: 12
- Number of operands: 5
- Total number of operator occurrences: 20
- Total number of operand occurrences: 15
Calculate different Halstead software metrics for above programs?

Q-3) What are information flow metrics?
How they are calculated for a given structure chart?
What is the purpose of data structure metrics?

Q-4) How function point analysis methodology is applied in estimation of software size?
Why FPA methodology is better than LOC methodology?

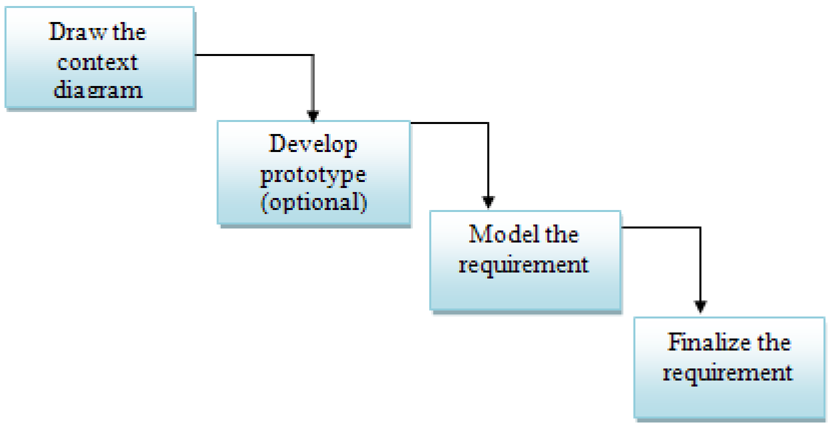
Q-5) What are the different types of COCOMO methods? Explain in detail?

Q-6) Discuss in brief about risk management in software development process?

Q 7) Discuss in detail the various methods to compute Cyclomatic complexity?

Solution

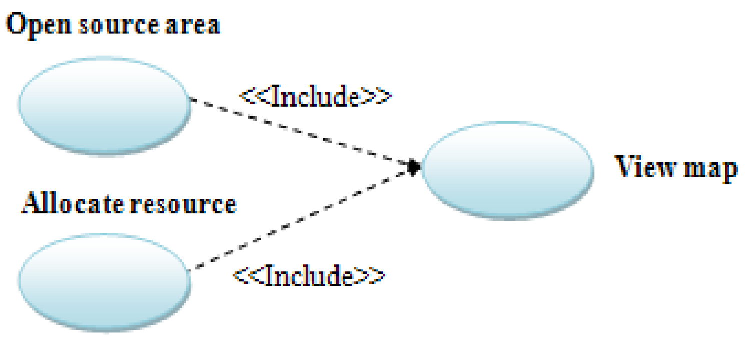
Ans. 1-a) Activities of requirement analysis:



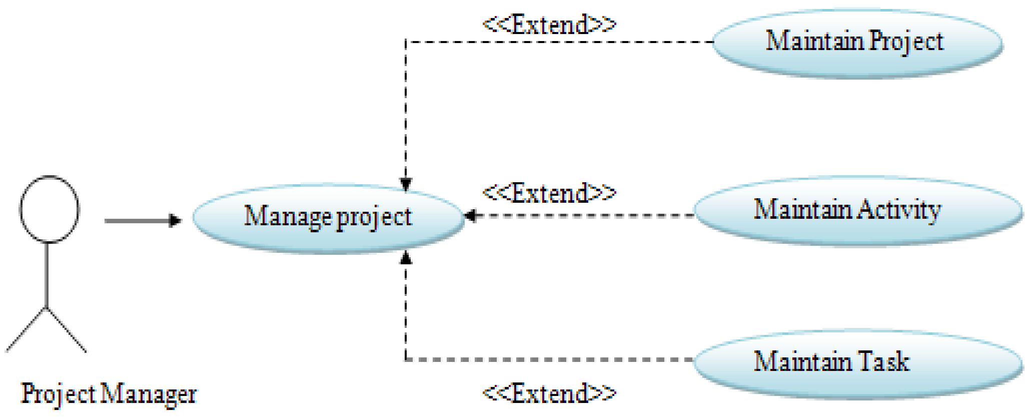
- ***Draw the context diagram***: It is a simple model that defines boundaries and interfaces of the proposed system with the external world. It identifies the entities outside the proposed system that interact with system.
- ***Development of a prototype***: One effective way to find out that the customer really wants is to construct a prototype, something that's looks preferably acts like a part of the system they say they want.
- ***Model the requirements***: This process usually consist of various graphical representations of the functions, data entities, external entities and the relationship between them.
- ***Finalize the requirements***: After modeling the requirements will have better under-standing of the system behavior. Inconsistencies and ambiguities have been identified and corrected we finish the analyzed requirements and next step is to document these requirement in a prescribed format.

Ans. 1-b) Dependencies in Use Case Diagrams:

An **include** dependency from one use case to another use case indicates that the base use case will include or call the inclusion use case. An include dependency is shown as a dashed arrow from the base use case to the inclusion use case market with the include keyword. The base use case is responsible for identifying where in its behavior sequence or at which step to include the inclusion use case. This identification is not done in the UML diagram, but rather in the textual description of the best use case.



An **Extended** Dependency from one use case indicates that the extension use case will extend and argument the base use case. A use case may extend multiple use cases, and a use case may be extended by multiple use cases. An extended dependency is shown as a dashed arrow from the extension use case to the base use case marked with the extended keyword. The base case is responsible for identifying at which steps in its behavior sequence the extending use cases may be inserted.

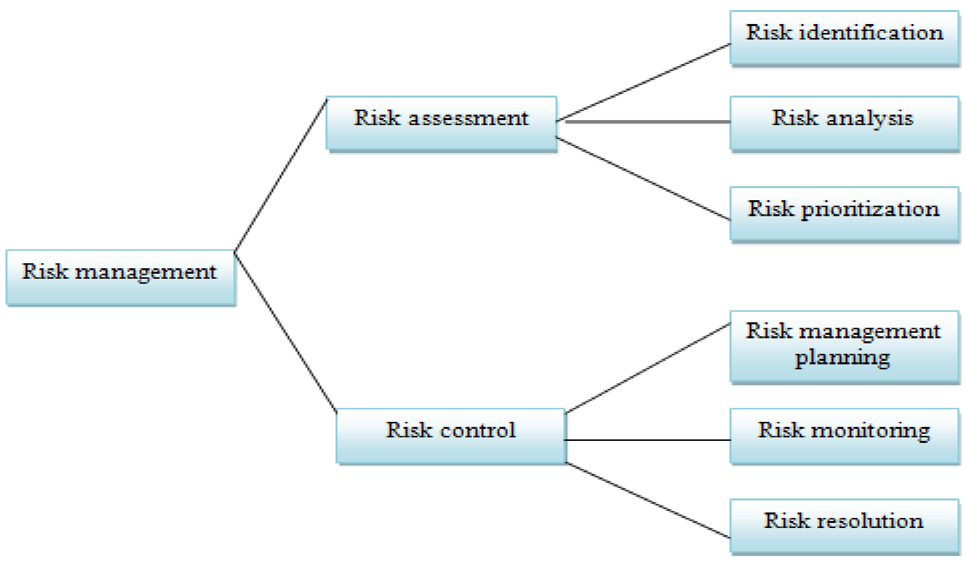


Ans. 1-c) Two techniques to estimate size of program**:-**

LOC (Lines Of Code) -- A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all line containing program header, declarations, executable and non-executable statements.

Function count – It measures the functionality from the user's point of view that is on the basis of what the user request and receives in return. Therefore it deals with the functionality being delivered and not with the line of code, source modules, files etc.

Ans. 1-d) Risk management involves several important steps; those are illustrated in figure which is given below:



Risk Assessment

It is process of examining a project and identifying areas of potential risk .Risk identification can be facilitated with the help of a checklist of common risk areas of software project .Risk analysis involves examining how project outcomes might change with modification of risk input variables .Risk prioritization helps the project focus on its most severe risks by assessing the risk explore .This prioritization can be done in a quantitative way , by estimating the probability (0.1-1.0) and relative loss , on a scale of 1 to 10. The higher the exposure, the more aggressively the risk should be tackled.

Risk Control

It is the process of managing risks to achieve the desired outcomes. Risk management planning produces a plan for dealing with each significant risk. It is useful to record decisions in the plan , so that both customer and developer can review how problems are to be avoided , as well as how they are to be handled when they arise .We should also monitor the project as development progresses, periodically revaluating the risks ,their probability ,and likely impact. Risk resolution is the execution of the plans for dealing with each risk.

Ans. 1-e) The CMM is used to judge the maturity of a software process. It has the following maturity levels.

- **Initial** (Maturity level 1) : There is no sound software engg. management. It's on adhoc basis. Success of project relies on competent manager and good development team. However, usual pattern is time and cost overrun due to lack of management.
- **Repeatable** (Maturity level 2): At this level policies for managing a project and procedures to implement them are established. Planning and managing is based upon past experience with similar project. Regular measurements are done to identify problems and immediate action taken to prevent problem from becoming crisis.
- **Defined** (Maturity level 3): In this level sets of defined and documented standard processes are established and subject to some degree of improvement over time. These standard processes are in place and used to establish consistency of process performance across the organization.
- **Managed** (Maturity level 4): It's the characteristic of process at this level that using process metrics, management can effectively control for software development. In

particular Management can identify ways to adjust and adapt the process to particular projects without measurable loses of quality or deviation from specifications. Processcapability is established from this level.

- **Optimizing** (Maturity level 5): In this level the focus is on continually improving process performance through both incremental and innovative technological changes improvement

Ans. 2) Operands and operators of any computer language are called tokens. All software science measures are function of the counts of these tokens.

Number of distinct operator (η^1) = 123

Number of distinct operands (η2) = 05

Total No. of operator occurence (N1) = 20

Total No. of operand occurence (N2) = 15

Program length (N)

$$N = N1 + N2 = 20+15$$

$$N = 35$$

Program vocabulary (η)

$$\eta = \eta^1 + \eta 2$$

$$\eta = 12 + 05$$

$$\eta = 17$$

Volume of Program (v) $V = N*\log2\ \eta$

$$V = 25*\log2\ 17 = 24 \times 4.08 = 102$$

Potential volume of Program (V*):

$$V^* = (2 + \eta 2\ ^*) \log2\ (2 + \eta 2\ ^*)$$

$$= (2 + 12) \log2\ (2 + 12)$$

$$= 14 \log2\ 14 = 14 \times 3.80 = 53.2$$

Program level (L) $L = V^*/V = 53.2/102 = 0.54$

Ans 3) **Information Flow Metrics**: Program consists of modules and as the information flow between the Modules increased, the Modules are called low cohesive and high coupled, and as a result, more complex software and requires more effort and time. Therefore effort and time also depend on the Information Flow Metric (IF).

The Basic Information Flow Modules:

IF (A) (Information Flow of Component A) = [FAN IN (A) * FAN OUT (A)]2

Here,

FAN IN (A) = the number of components calling Component (Module) A.

FAN OUT (A) = the number of components that are called by (A)

Data Structure Metrics: Line of Code, Function Point, and Token Count are important metrics to find out the effort and time required to complete the project. There are some Data Structure metrics to compute the effort and time required to complete the project. There metrics are:

- The Amount of Data.
- The Usage of data within a Module.
- Program weakness.
- The sharing of Data among Modules.

The Amount of Data: To measure Amount of Data, there are further many different metrics and these are:

Number of variable (VARS): In this metric, Number of variables used in the program are counted.

Number of Operands (η2): In this metric, Number of operand used in the program are counted.

$\eta 2 = VARS + Constants + Labels$

Total number of occurrence of variable (N2): In this metric, total number of occurrence of variables are computed

The Usage of data within a Module:

The measure this metric, average number of live variables is computed. A variable is live from its first to its last references with in procedure.

$$\text{Average no of Live variables (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

Program weakness: Program weakness depends on its Modules weakness. If Modules are weak(less Cohesive), then it increase the effort and time metrics required to complete the project.

$$\text{Average life of variables } (\gamma) = \frac{\text{Sum of count of live variables}}{\text{No. of unique variables}}$$

Module Weakness (WM) = LV* γ

Here, LV: average no. of live variables.

γ: average life of variables.

Program Weakness (WP) = $(\sum WM)/m$

Here, WM: weakness of module.

m: number of modules in the program.

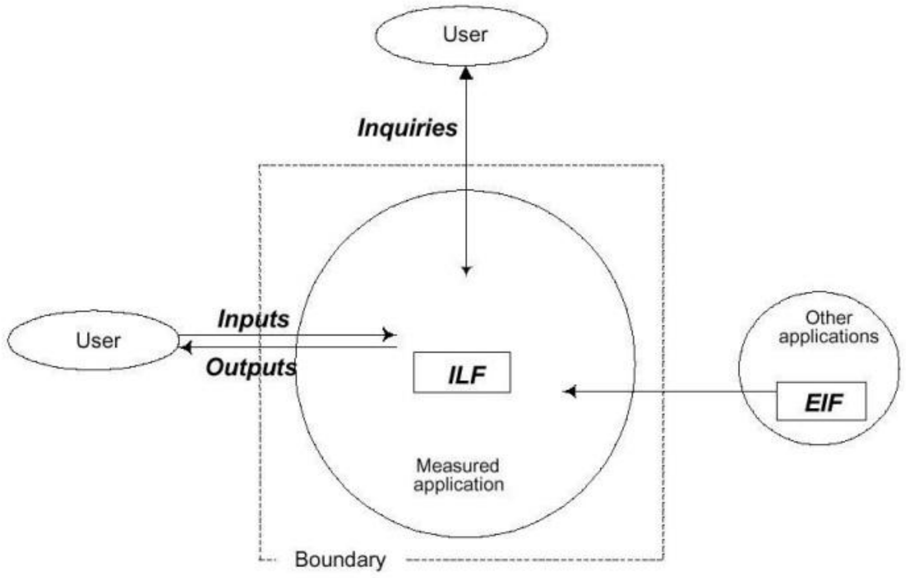
Sharing of Data among Module: As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased, as a result more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate effort and time.

Ans 4) It measures functionality from the user's point of view, that is, on the basis of what the user requests and receives in return. Therefore, it deals with the functionality being delivered, and not the lines of code, source modules, files, etc. Measuring size in this way has the advantage that size measure is independent of the technology used to deliver the functions. In other words, two identical counting systems, one written in 4GL and the other in assembler, would have the same function count. This makes sense to the user, because the object is to buy an accounting system, not lines of assembler and it makes sense to the IT department, because they can measure the performance differences between the assembler and 4GL environments.

Function point measures functionality from the user's point of view, that is, on the basis of what the user requests and receives in return from the system. The principle of the function point analysis (FPA) is that a system is decomposed into functional units.

- Inputs : Information entering the system.
- Outputs : Information leaving the system.
- Enquiries : Requests for instant access to information.
- Internal logical files : Information held within the system.
- External interface files : Information held by other systems that is used by the system being analysed.

The FPA functional units are shown in Fig.



The five functional units are divide in two categories:

(i) Data function types

- *Internal Logical Files (ILF):* A user identifiable group of logically related data or control information maintained within system.
- *External Interface Files (EIF):* A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

(ii) Transactional Function Types

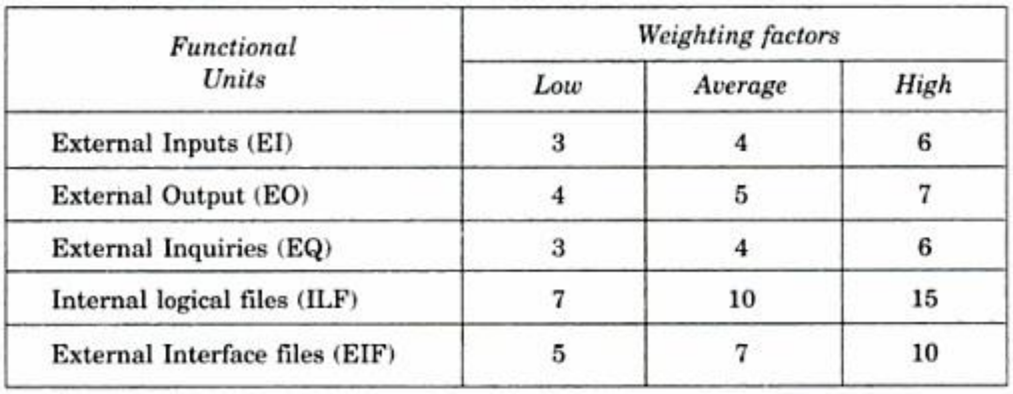
- *External Input (EI):* An EI process data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- *External Output (EO):* An EO is an elementary process that generates data or control information to be sent outside the system.
- *External Inquiry (EQ):* An EQ is an elementary process that is made up of an input-output combination that results in data retrieval.

Counting function points

The five functional units are ranked according to their complexity i.e. Low, Average, or High, using a set of prescriptive standards. Organizations that use FP methods develop criteria for determining whether a particular entry is Low, Average, or High. Nonetheless, the determination of complexity is somewhat subjective.

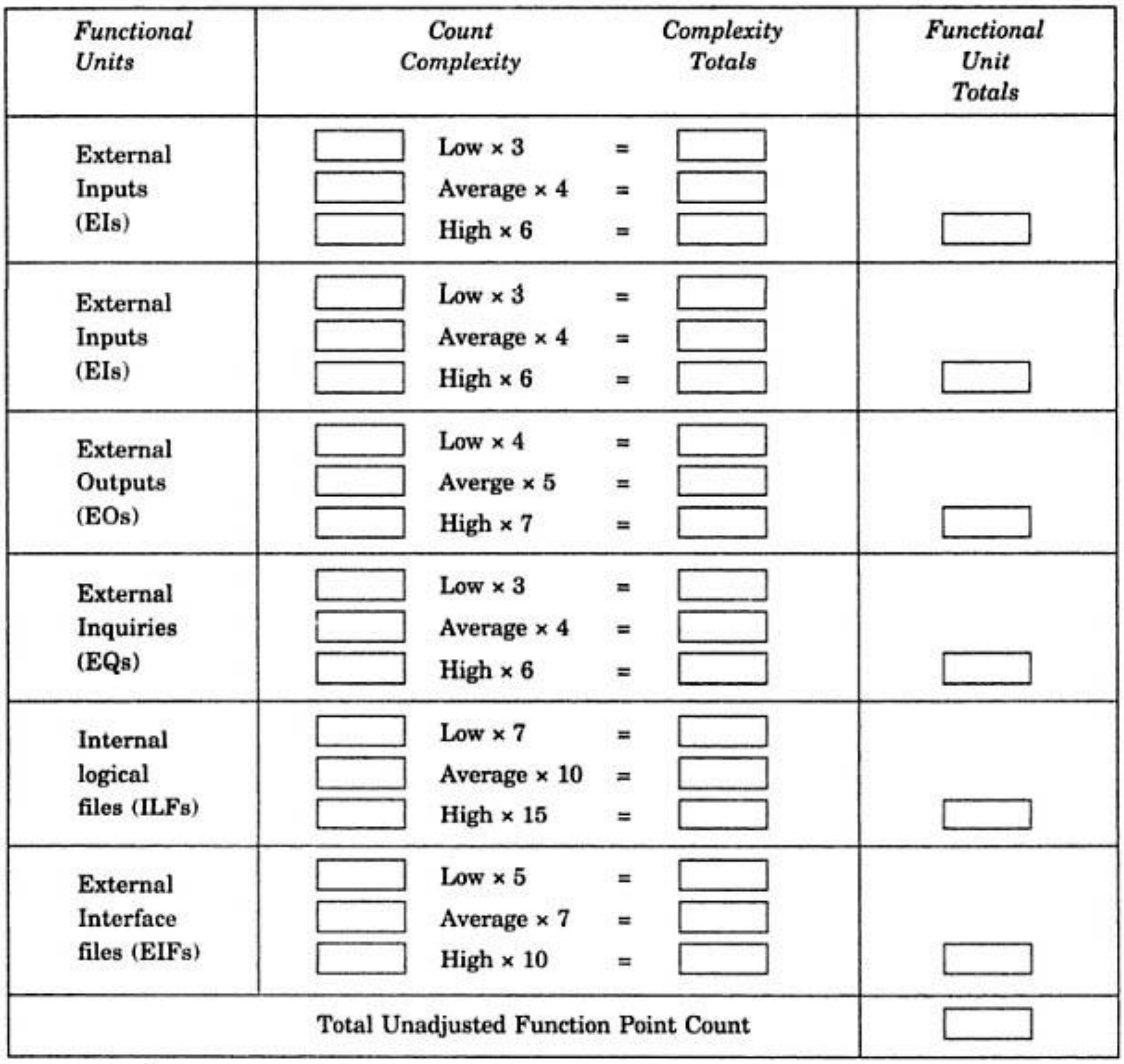
After classifying each of the five function types, the Unadjusted Function Points (UFP) are calculated using predefined weights for each function types as given in Table(1)

Table 1: Functional units with weighting factors



Functional Units	*Weighting factors*		
	Low	*Average*	*High*
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 2:



Functional Units	*Count Complexity*		*Complexity Totals*	*Functional Unit Totals*
External Inputs (EIs)	☐	Low × 3 =	☐	
	☐	Average × 4 =	☐	
	☐	High × 6 =	☐	☐
External Inputs (EIs)	☐	Low × 3 =	☐	
	☐	Average × 4 =	☐	
	☐	High × 6 =	☐	☐
External Outputs (EOs)	☐	Low × 4 =	☐	
	☐	Averge × 5 =	☐	
	☐	High × 7 =	☐	☐
External Inquiries (EQs)	☐	Low × 3 =	☐	
	☐	Average × 4 =	☐	
	☐	High × 6 =	☐	☐
Internal logical files (ILFs)	☐	Low × 7 =	☐	
	☐	Average × 10 =	☐	
	☐	High × 15 =	☐	☐
External Interface files (EIFs)	☐	Low × 5 =	☐	
	☐	Average × 7 =	☐	
	☐	High × 10 =	☐	☐
Total Unadjusted Function Point Count				☐

The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted functional Point (UFP) is given in Table (2).

The procedure for the calculation of UFP in mathematical form is given below:

$$\mathbf{UFP} = {}_{i=1}^{3} \cdot \ {}_{j=1}^{3} z_{ij} w_{ij}$$

Where i indicated the row and j indicated the column of Table 4.1

W_{ij} : It is the entry of the i^{th} and j^{th} column of the Table 4.1

$Z_{ij:}$ It is the count of the number of functional units of type i that has been classified as having the complexity corresponding to column j.

Organization that use function point methods develop a criterion for determining whether a particular entry is Low, Average, or High. Nonetheless, the determination of complexity is somewhat subjective.

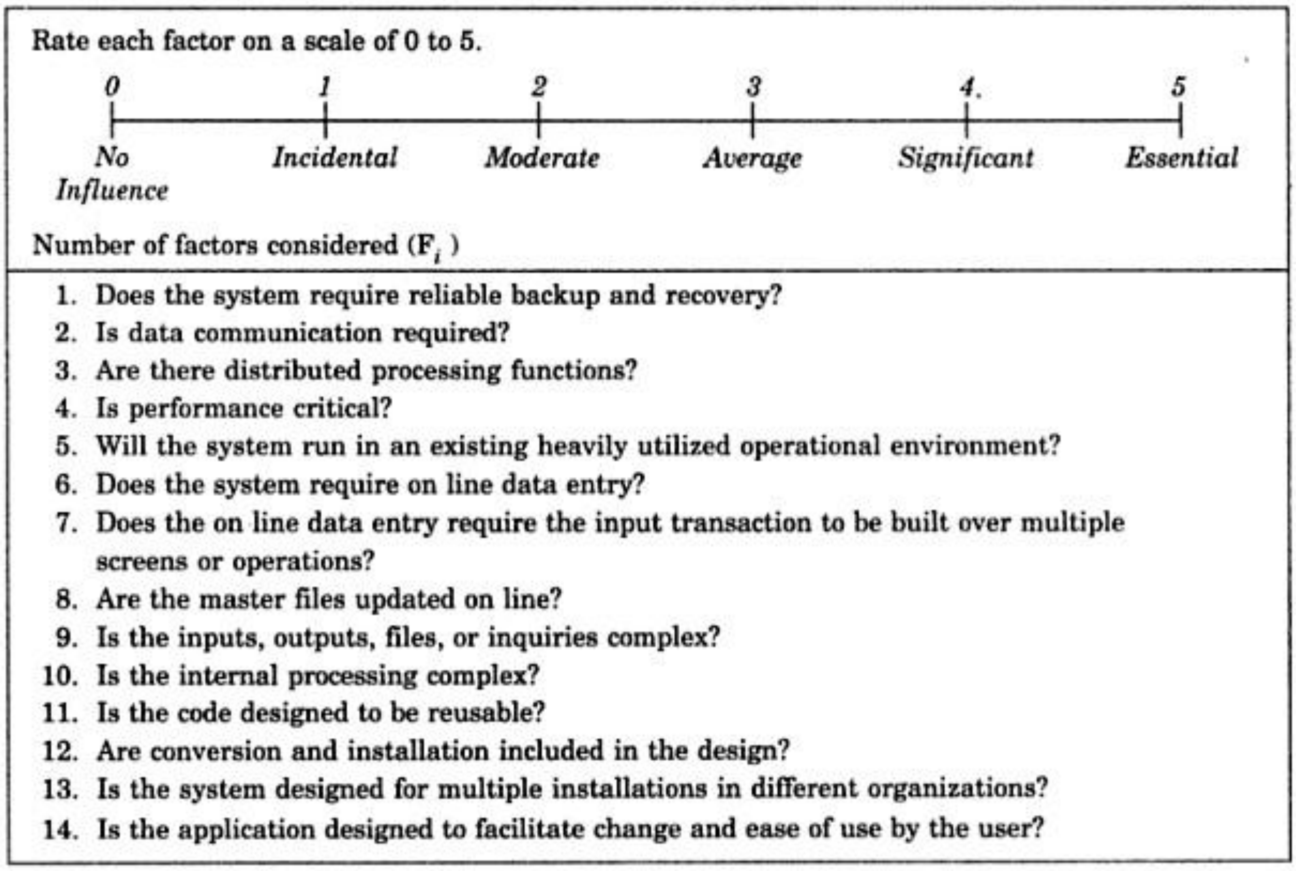
The final number of function points is arrived at by multiplying the UFP by the adjustment factor that is determined by considering 14 aspects of processing complexity which are given in Table 3. This adjustment factor allows the UFP count to be modified by at most 35%. The final adjusted FP count is obtained by using the following relationship.

$$\mathbf{FP = UFP \times CAF}$$

where CAF is complexity adjustment factor and is equal to $(0.65 + 0.01 \times \sum F_i)$.

F_i(i=1 to 14) are the degrees of influence and are based on the responses of questions in Table 3.

Table 3: Computing Functional Points



Rate each factor on a scale of 0 to 5.

0	1	2	3	4	5
No Influence	*Incidental*	*Moderate*	*Average*	*Significant*	*Essential*

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery?
2. Is data communication required?
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 on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is 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?

Ans. 5) COCOMO model was given by Boehm in 1981. It is a hierarchy of software cost estimation models which includes basic, intermediate and detailed sub-models.

BASIC MODEL:-This model aims at estimating most of the small to medium sized software projects in a quick and rough fashion.

It has 3 modes of software development:-

- ***ORGANIC MODE***
- ***SEMIDETACHED MODE***
- ***EMBEDDED MODE***

ORGANIC MODE

The size of project is 2 to 50 KLOC. A small team of experienced developers develops software in a very familiar environment. The deadline of project is not tight.

SEMIDETACHED MODE

The size of project is 50 to 300 KLOC. A team having average previous experience on similar projects develops software in a medium environment. The deadline of project is medium.

EMBEDDED MODE

The size of project is over 300 KLOC. Real time systems, complex interfaces and a team of very little previous experienced developers are involved. The deadline of project is tight.

INTERMEDIATE MODEL:-

The basic model allowed for a quick and rough estimate but the result will lack accuracy. So, Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment. These cost drivers are used to adjust the nominal cost of a project to the actual project environment. Hence, the accuracy of the estimate will be increased. The cost drivers are grouped into 4 categories:-

Product attributes

- Required Software Reliability (RELY)
- Database size (DATA)
- Product Complexity (CPLX)

Computer attributes

- Execution Time Constraint (TIME)
- Main Storage Constraint (STOR)
- Virtual Machine Volatility (VIRT)
- Computer Turnaround Time (TURN)

Personal attributes

- Analyst Capability (ACAP)
- Application Experience (AEXP)
- Programmer Capability (PCAP)
- Virtual Machine Experience (VEXP)
- Programming Language Experience (LEXP)

Project attributes

- Modern Programming Practices (MODP)
- Use Of Software Tools (TOOL)
- Required Development Schedule (SCED)

DETAILED MODEL:-

It offers a means for processing all the project characteristics to construct a software estimate. The detailed model introduces two more capabilities:-

Phase-sensitive effort multipliers

Some phases are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase sensitive effort multipliers for each cost driver. This helps in determining the manpower allocation for each phase of the project.

Three-level product hierarchy

Three product levels are defined. These are module, subsystem and system levels.

Ans 6) Risk can be defined as a problem that could cause some loss or threaten the success of the project but which has not happened yet. These potential problems might have an adverse impact on cost, schedule or technical success of the project, the quality of software products or project team morale.

Risk management is a process of identifying, addressing and eliminating these problems before they can damage the project.

Main categories are:-

Dependencies

Availability of trained experienced people

Inter component or intergroup dependencies

Customer furnished items or information

Internal and external subcontractor relationships

Requirement Issues

Lack of clear product vision

Lack of agreement on product requirements

Unprioritized requirements

New market for uncertain needs

Rapidly changing requirements

Inadequate impact analysis of requirement changes

Management Issues

Poor communication

Staff personality conflicts

Managers or customers with unrealistic expectation

Unrealistic commitments made

Unclear project ownership and decision making

Inadequate visibility into actual project status

Inadequate planning and task identification

Lack of knowledge

Inadequate training

Poor understanding of methods, tools, techniques

Inadequate application domain experience

New technologies

Other risk categories

Unavailability of adequate testing facilities

Turnover of essential personnel