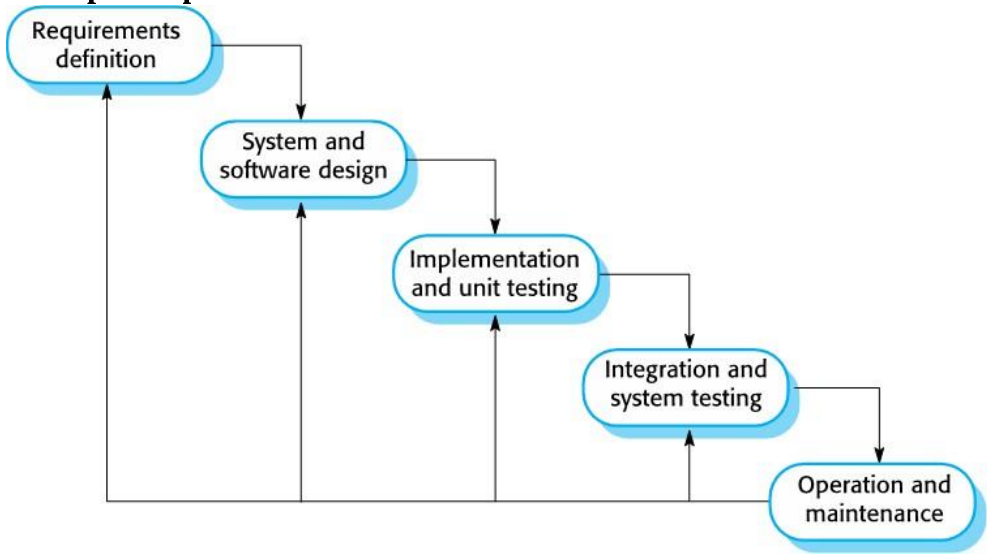
SOFTWARE ENGINEERING & PROJECT MANGEMENT- BCS501
VTU Question Paper Solution

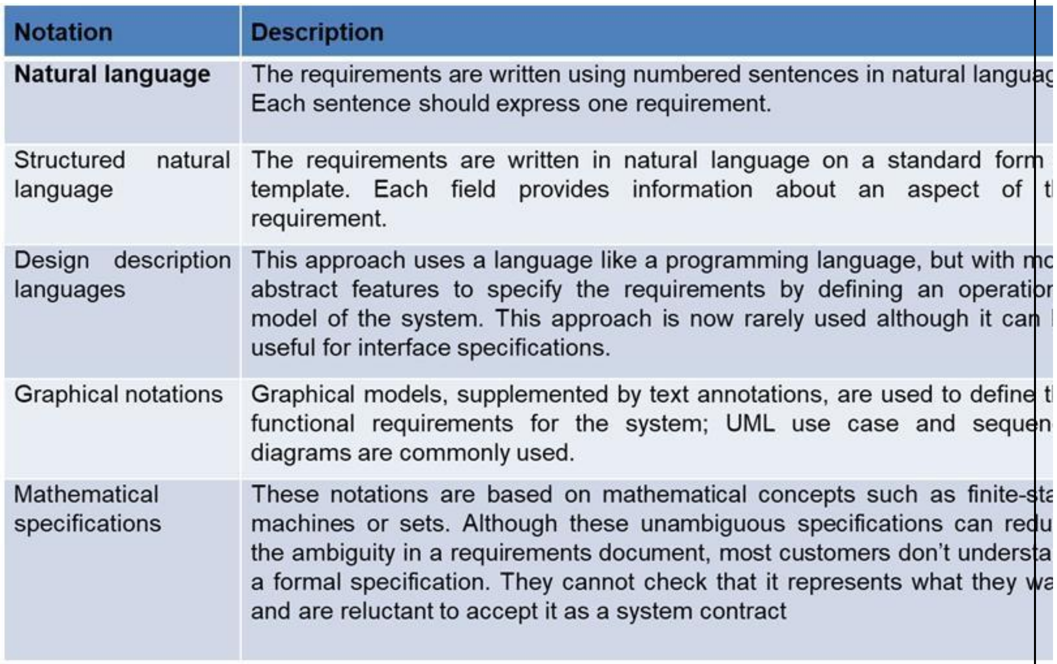
NO	Question	MARKS
1	**What are attributes of good software? Explain Key Challenges Faced in Software Engineering.** **Maintainability** Software should be written in such a way so that it can evolve to meet the **changing needs of customers**. This is a critical attribute because software change is an inevitable requirement of a changing business environment. **Dependability and security** Software dependability includes a range of characteristics including **reliability, security and safety**. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system. **Efficiency** Software should not make wasteful use of system resources such as **memory and processor cycles.** Efficiency therefore includes responsiveness, processing time, memory utilisation, etc. **Acceptability** Software must be acceptable to the type of users for which it is designed. This means that it must be **understandable, usable and compatible** with other systems that they use. Key challenges: ²Heterogeneity §Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. ²Business and social change §Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software. ²Security and trust §As software is intertwined with all aspects of our lives, it is essential that we can trust that software. ²Scale §Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-	08

	scale, cloud-based systems that serve a global community.	
b	**With neat diagram explain waterfall development model of software development process**  Requirements definition System and software design Implementation and unit testing Integration and system testing Operation and maintenance ²There are separate identified phases in the waterfall model: §Requirements analysis and definition §System and software design §Implementation and unit testing §Integration and system testing §Operation and maintenance ²The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase. ²Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. §Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. §Few business systems have stable requirements. ²The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work	06

c	Explain general model of software design process ²A structured set of activities required to develop a software system. ²Many different software processes but all involve: §Specification – defining what the system should do; §Design and implementation – defining the organization of the system and implementing the system; §Validation – checking that it does what the customer wants; §Evolution – changing the system in response to changing customer needs. ²A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. ²When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities. ²Process descriptions may also include: §Products, which are the outcomes of a process activity; §Roles, which reflect the responsibilities of the people involved in the process; §Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.	06
2a	Define and differentiate functional and nonfunctional requirements. ²Functional requirements §Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. §May state what the system should not do. ²Non-functional requirements §Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. §Often apply to the system as a whole rather than individual features or services.	06
b	What is the requirement specification?Explain ways of system requirements. ²Developed in a project studying the air traffic control process ²Combines ethnography with prototyping ²Prototype development results in unanswered questions which focus the	08

ethnographic analysis.

²The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.



Notation	Description
Natural language	The requirements are written using numbered sentences in natural languag Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form template. Each field provides information about an aspect of t requirement.
Design description languages	This approach uses a language like a programming language, but with mo abstract features to specify the requirements by defining an operatio model of the system. This approach is now rarely used although it can useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define t functional requirements for the system; UML use case and sequen diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-sta machines or sets. Although these unambiguous specifications can redu the ambiguity in a requirements document, most customers don't understa a formal specification. They cannot check that it represents what they wa and are reluctant to accept it as a system contract

c What is ethnography? How is it effective in requirement discovery? 06

²A social scientist spends a considerable time observing and analysing how people actually work.

²People do not have to explain or articulate their work.

²Social and organisational factors of importance may be observed.

²Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

²Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.

²Requirements that are derived from cooperation and awareness of other people‟s activities.

§Awareness of what other people are doing leads to changes in the ways in which we do things.

Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

²Developed in a project studying the air traffic control process

²Combines ethnography with prototyping

²Prototype development results in unanswered questions which focus the ethnographic analysis. ²The problem with ethnography is that it studies existing

	practices which may have some historical basis which is no longer relevant.	
3a	**What is object oriented development (OOD)?Explain OOD briefly.** The process for OO development and graphical notation for representing OO concepts consists of building a model of an application and then adding details to it during design. The methodology has the following stages: System conception : Software development begins with business analysis or users conceiving an application and formulating tentative requirements Analysis : The analyst must work with the requestor to understand the problem, because problem statements are rarely complete or correct. The analysis model is a precise abstraction of what the desired system must do, not how it will be done. It should not contain implementation decisions.The analysis model has 2 parts: • Domain model - a description of the real-world objects reflected within the system Eg: Domain objects for a stock broker • Application – model - a description of the parts of the application system itself that are visible to the user. tion might include stock, bond, trade and commission. objects might control the execution of trades and present the results. System design: The development teams devise a high – level strategy – the system architecture for solving the application problem. Class design : The class designer adds details to the analysis model in accordance with the system design strategy. The focus of class design is the data structures and algorithms needed to implement each class. Implementation : Implementers translate the classes and relationships developed during class design into particular programming language, database or hardware. During implementation, it is important to follow good software engineering practice so that traceability to the design is apparent and so that the system remains flexible and extensible.	
b	**Explain Links , associations. Explain UML notation for same with example.** • Associations are the means for establishing relationships among classes.An association is a description of a group of links with common structure and common semantics.E.g. a person WorksFor a company. If two classes in a model need to communicate with each other, there must be **link** between them, and that can be represented by an association (connector). • Associations are inherently bi-directional. The association name is usually read in a particular direction but the binary association may be traversed in either direction. Association can be represented by a line between these classes with an arrow indicating the navigation direction. In case arrow is on the both sides, association has bidirectional association.	

Association connects related classes and is also denoted by a line.Show association names in italics.

• Association end name Associations have ends. They are called „Association Ends‟. They may have names (which often appear in problem descriptions). Use of association end names is optional. But association end names are useful for traversing associations.

II. Qualified association

• A qualified association is an association in which an attribute called Qualifier the objects for a „many‟ association‟ end. A qualifier selects among the target objects, reducing the effective multiplicity from „many‟ to „one‟.Both below models are acceptable but the qualified model adds information.

Adding a qualifier clarifies the class diagram and increases the conveyed information. In this case, the model including the qualification denotes that the name of a file is unique

within a directory. Example of how a qualified association reduces multiplicity (UML class diagram).

III. Association classes

An association class is an association that is also a class.Like the links of an association, the instances of an association class derive identity from instances of the constituent classes. Like a class, an association class can have attributes and operations and participate in associations.

c **What is generalisation and association?**

Generalization

Deriving a class out of a parent class having some inherited property(from the parent class) and some new property of the derived class.

The term generalization is for the inheritance in the bottom to the up direction i.e. from derived class to the parent class. Generalization is the relationship between a class (superclass) and one or more variations of the class (subclasses).

A superclass holds common attributes, attributes and associations.The subclasses adds specific attributes, operations, and associations. They inherit the features of their superclass.

Generalization is called a “IS A” relationship

A *generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.A solid line with a hollow arrowhead that point from the child to the parent class.

Associations

- Associations are the means for establishing relationships among classes.An association is a description of a group of links with common structure and common semantics.E.g. a person WorksFor a company. If two classes in a model need to communicate with each other, there must be link between them, and that can be represented by an association (connector).Associations are inherently bi-directional. The association name is usually read in a particular direction but the binary association may be traversed in either direction. Association can be represented by a line between these classes **with an arrow indicating the navigation direction**. In case arrow is on the both sides, association has bidirectional association

4a	**What is object orientation? What are important characteristics of OO approach? Explain.**	8

In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.

In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and maintenance. It identifies the objects in problem domain, classifying them in terms of data and behavior.

The OO model is beneficial in the following ways −

- It facilitates changes in the system at low cost.
- It promotes the reuse of components.
- It simplifies the problem of integrating components to configure large system.
- It simplifies the design of distributed systems.

Elements of Object-Oriented System

Let us go through the characteristics of OO System −

- Objects − An object is something that is exists within problem domain and can be identified by data (attribute) or behavior. All tangible entities (student, patient) and some intangible entities (bank account) are modeled as object.
- Attributes − They describe information about the object.
- Behavior − It specifies what the object can do. It defines the operation performed on objects.
- Class − A class encapsulates the data and its behavior. Objects with similar meaning and purpose grouped together as class.
- Methods − Methods determine the behavior of a class. They are nothing more than an action that an object can perform.
- Message − A message is a function or procedure call from one object to another. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another.

Features of Object-Oriented System

An object-oriented system comes with several great features which are discussed below.

Encapsulation

	Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system. Abstraction It is a process of taking or selecting necessary method and attributes to specify the object. It focuses on essential characteristics of an object relative to perspective of user. Relationships All the classes in the system are related with each other. The objects do not exist in isolation, they exist in relationship with other objects.	
b	**Describe model. Define the relationship between models.** **Intention of object oriented modeling and design is to learn how to apply object -oriented concepts to all the stages of the software development life cycle.Object-oriented modeling and design is a way of thinking about problems using models organized around real world concepts. The fundamental construct is the object, which combines both data structure and behavior.** **Purpose of Models:** 1. **Testing a physical entity before building it** 2. **Communication with customers** 3. **Visualization** 4. **Reduction of complexity** **Types of Models:** There are 3 types of models in the object oriented modeling and design are: Class Model, State Model, and Interaction Model. These are explained as following below. 1. **Class Model:** The class model shows all the classes present in the system. The class model shows the attributes and the behavior associated with the objects. The class diagram is used to show the class model.The class diagram shows the class name followed by the attributes followed by the functions or the methods that are associated with the object of the class.Goal in constructing class model is to capture those	8

	concepts from the real world that are important to an application. 2. **State Model:** State model describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, states that define the context for events, and the organization of events and states.Actions and events in a state diagram become operations on objects in the class model. State diagram describes the state model. 3. **Interaction Model:** Interaction model is used to show the various interactions between objects, how the objects collaborate to achieve the behavior of the system as a whole. The following diagrams are used to show the interaction model: o Use Case Diagram o Sequence Diagram o Activity Diagram	
c	**With help of class diagram define multiplicity,association and names.** • Associations are the means for establishing relationships among classes.An association is a description of a group of links with common structure and common semantics.E.g. a person WorksFor a company. If two classes in a model need to communicate with each other, there must be link between them, and that can be represented by an association (connector). • Associations are inherently bi-directional. The association name is usually read in a particular direction but the binary association may be traversed in either direction.	4

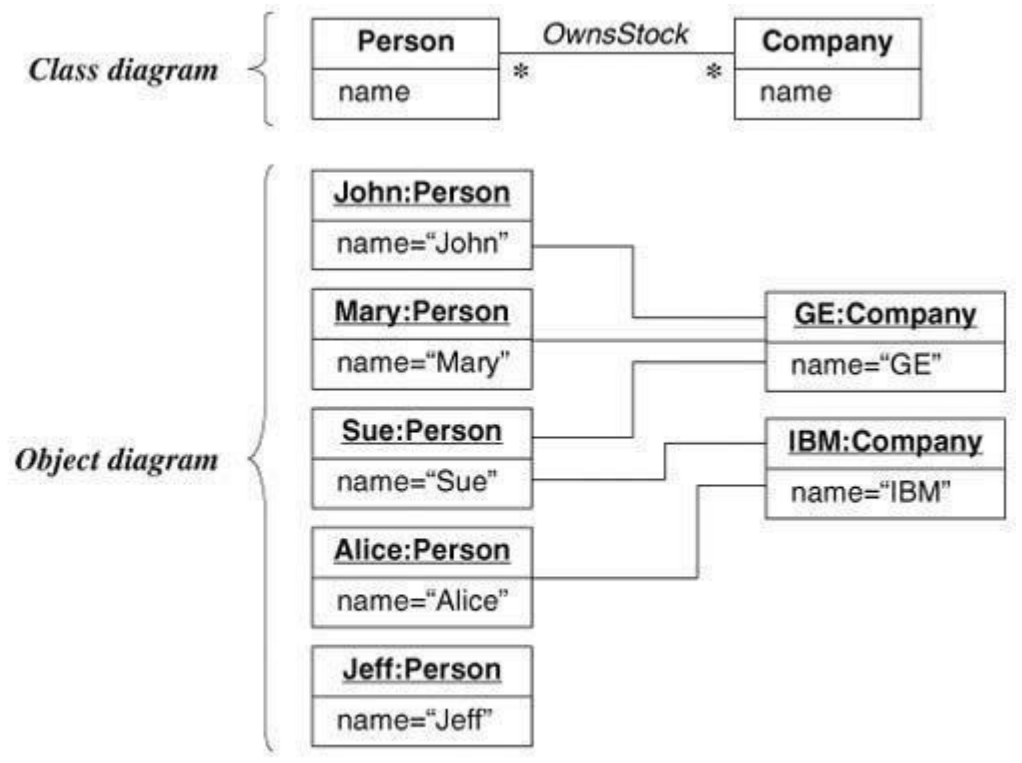


Figure 3.7 Many-to-many association. An association describes a set of potential links in the same way that a class describes a set of potential objects.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

- Association can be represented by a line between these classes **with an arrow indicating the navigation direction**. In case arrow is on the both sides, association has bidirectional association.

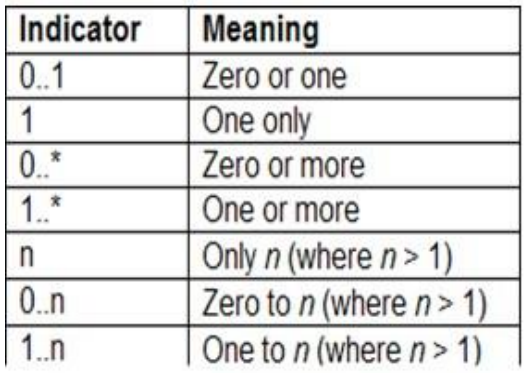
Association connects related classes and is also denoted by a line.Show association names in italics.

- **Association end name** Associations have ends. They are called „Association Ends". They may have names (which often appear in problem descriptions). Use of association end names is optional. But association end names are useful for traversing associations.

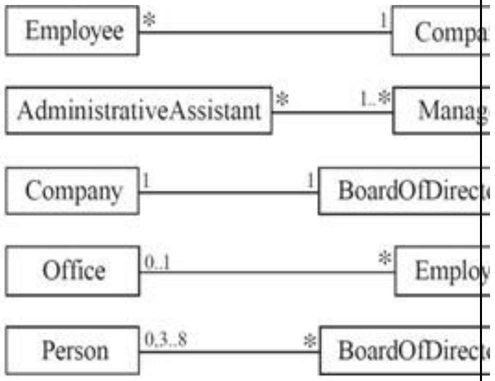
II. Multiplicity

Multiplicity defines the number of objects associated with an instance of the association.

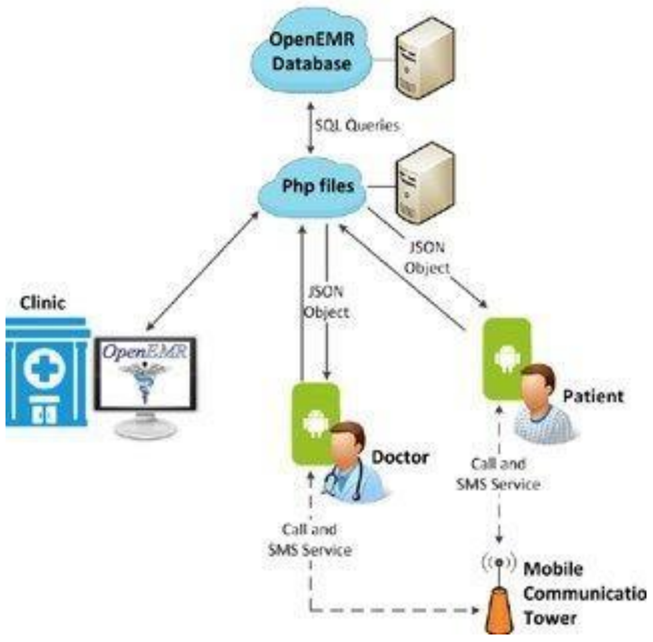
UML diagrams explicitly list multiplicity at the end of association lines.Intervals are used to express multiplicity:

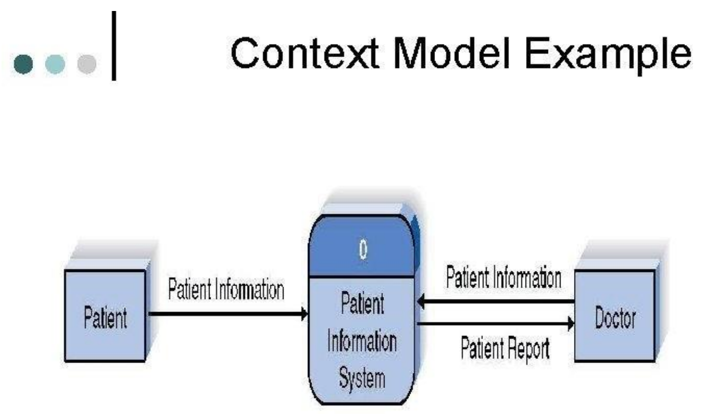


Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)



5 **Draw and Explain the Context Model for Patient Information System**





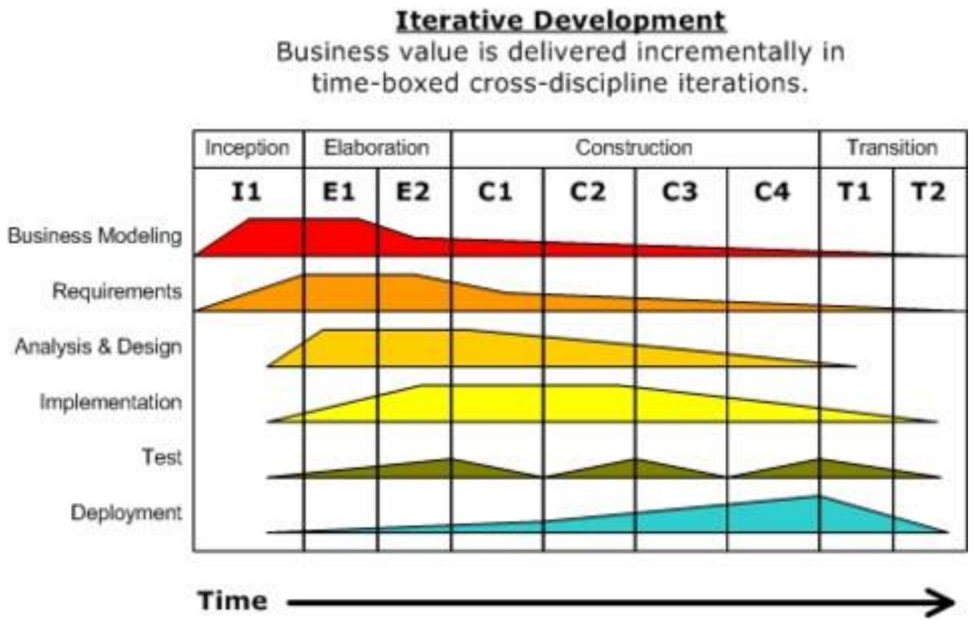
With a neat diagram explain the phases in the Rational Unified Process (RUP)

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003.

RUP and its Phases

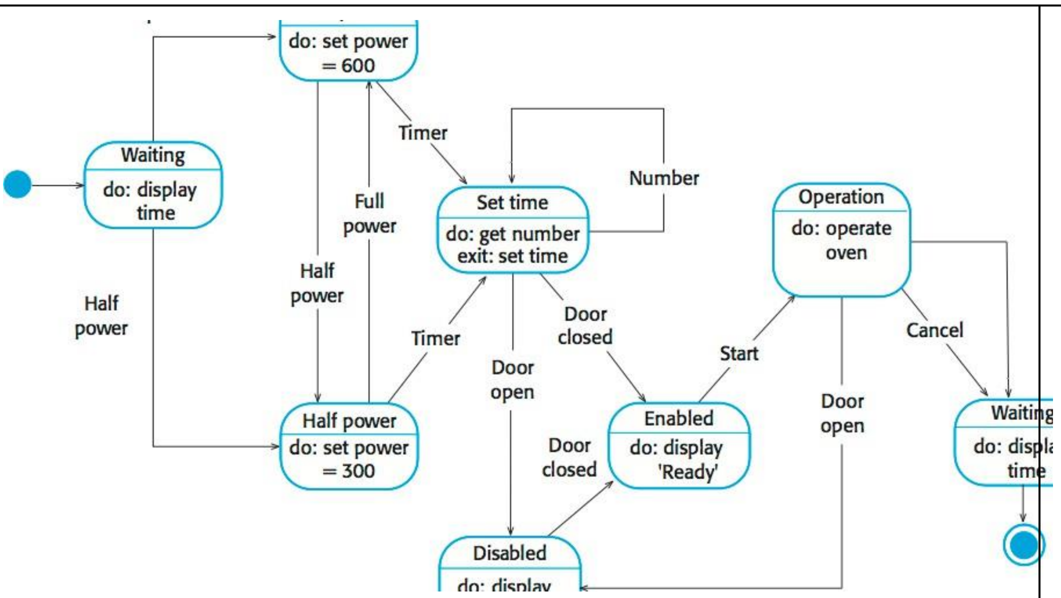
- **Inception** – Communication and planning are main**. ...**
- **Elaboration** – Planning and modeling are main. **...**
- **Construction** – Project is developed and completed. **...**
- **Transition** – Final project is released to public. **...**
- **Production** – Final phase of the model**.**

Stands for "Rational Unified Process." RUP is a software development process from Rational, a division of IBM. It divides the development process into four distinct phases that each involve business modeling, analysis and design, implementation, testing, and deployment. The four phases are:



1. Inception - The idea for the project is stated. The development team determines if the project is worth pursuing and what resources will be needed.
2. Elaboration - The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development.
3. Construction - The project is developed and completed. The software is designed, written, and tested.
4. Transition - The software is released to the public. Final adjustments or updates are made based on feedback from end users.

With the help of a neat state diagram , Illustrate the working of a microwave oven

	 do: set power = 600 Timer Number Waiting do: display time Full power Set time do: get number exit: set time Operation do: operate oven Half power Half power Door closed Start Cancel Timer Door open Door open Half power do: set power = 300 Enabled do: display 'Ready' Door closed Waiting do: displa time Disabled do: display	
6	**a. What is model driven Engineering? State the three types of abstract system model produced with a neat diagram** Model-driven engineering (MDE) is a software development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem. 1. A computation independent model (CIM) that models the important domain abstractions used in the system. CIMs are sometimes called domain models. You may develop several different CIMs, reflecting different views of the system. 2. A platform independent model (PIM) that models the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events. 3. Platform specific models (PSM) which are transformations of the platform independent model with a separate PSM for each application platform. **b. What are the activities to be carried out in object design process of a system? How the objects are identified?** **Object Oriented Design** (OOD), the technology independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and the interfaces are designed, which results in a model for the	

solution domain.
Object design activities include:

1. Reuse: Identification of existing solutions
• Use of inheritance
• Off-the-shelf components and
additional solution objects
• Design patterns
2. Interface specification
• Describes precisely each class interface
3. Object model restructuring
• Transforms the object design model to
improve its understandability and extensibility
4. Object model optimization
• Transforms the object design model to address
performance criteria such as response
time or memory utilization.

c. What is open / source development? Explain general models of open source licensing.

An open source development model is the process used by an open source community project to develop open source software. The software is then released under an open source license, so anyone can view or modify the source code

Copyleft licenses include the **GPL license (GPL v2 and GPL v3)** and the Mozilla Public License 2.0. Since licenses are legally binding, you'll have to carefully consider which one to choose when writing your open source code.

The following OSI-approved licenses are popular, widely used, or have strong communities:

- Apache License 2.0.
- BSD 3-Clause "New" or "Revised" license.
- BSD 2-Clause "Simplified" or "FreeBSD" license.
- GNU General Public License (GPL)
- GNU Library or "Lesser" General Public License (LGPL)
- MIT license.
- Mozilla Public License 2.0

7 **a. What is list driven development? With a neat diagram explain the**

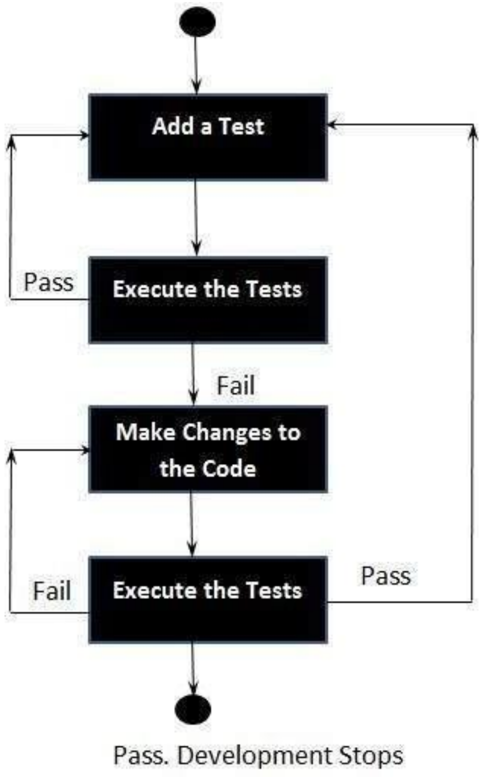
test driven development process

Test-driven development starts with developing test for each one of the features. The test might fail as the tests are developed even before the development. Development team then develops and refactors the code to pass the test.

Test-driven development is related to the test-first programming evolved as part of extreme programming concepts.

Test-Driven Development Process:

- Add a Test
- Run all tests and see if the new one fails
- Write some code
- Run tests and Refactor code
- Repeat



Context of Testing:

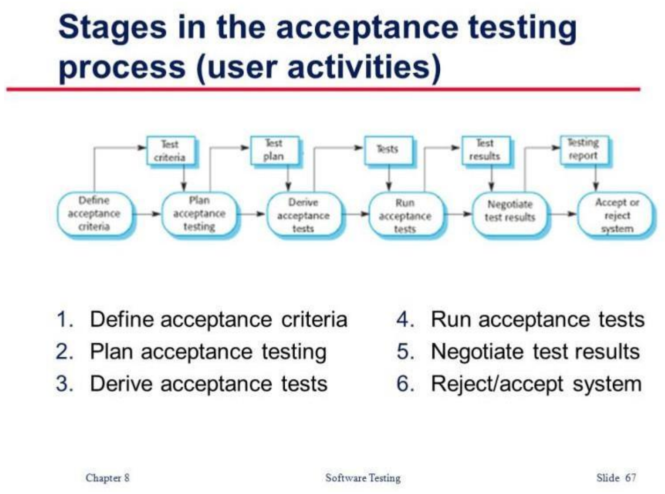
- Valid inputs
- Invalid inputs
- Errors, exceptions, and events

- Boundary conditions
- Everything that might break

Benefits of TDD:

- Much less debug time
- Code proven to meet requirements
- Tests become Safety Net
- Near zero defects
- Shorter development cycles

b. With neat diagram, explain six stages of acceptance testing process



Stages in the acceptance testing process (user activities)

1. Define acceptance criteria
2. Plan acceptance testing
3. Derive acceptance tests
4. Run acceptance tests
5. Negotiate test results
6. Reject/accept system

Chapter 8 Software Testing Slide 67

There are six stages in the acceptance testing process, as shown in Figure They are:
1. Define acceptance criteria This stage should, ideally, take place early in the process before the contract for the system is signed. The acceptance criteria should be part of the system contract and be agreed between the customer and the developer. In practice, however, it can be difficult to define criteria so early in the process. Detailed requirements may not be available and there may be sig- nificant requirements change during the development process.
2. Plan acceptance testing This involves deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule. The accep- tance test plan should also discuss the required coverage of the requirements and the order in which system features are tested. It should define risks to the testing process, such as system crashes and inadequate performance, and discuss how these risks can be mitigated.
3. Derive acceptance tests Once acceptance criteria have been established, tests

have to be designed to check whether or not a system is acceptable. Acceptance tests should aim to test both the functional and non- functional characteristics (e.g., performance) of the system. They should, ideally, provide complete cover- age of the system requirements. In practice, it is difficult to establish completely objective acceptance criteria. There is often scope for argument about whether or not a test shows that a criterion has definitely been met.

4. Run acceptance tests The agreed acceptance tests are executed on the system. Ideally, this should take place in the actual environment where the system will be used, but this may be disruptive and impractical. Therefore, a user testing environment may have to be set up to run these tests. It is difficult to automate this process as part of the acceptance tests may involve testing the interactions between end- users and the system. Some training of end-users may be required.

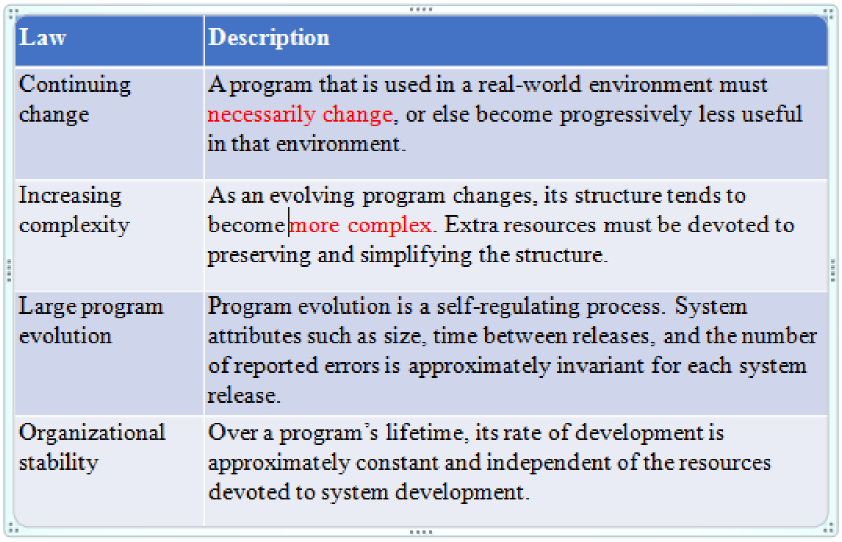
5. Negotiate test results It is very unlikely that all of the defined acceptance tests will pass and that there will be no problems with the system. If this is the case, then acceptance testing is complete and the system can be handed over. More com- monly, some problems will be discovered. In such cases, the developer and the customer have to negotiate to decide if the system is good enough to be put into use. They must also agree on the developer‟s response to identified problems.

6. Reject/accept system This stage involves a meeting between the developers and the customer to decide on whether or not the system should be accepted. If the system is not good enough for use, then further development is required to fix the identified problems. Once complete, the acceptance testing phase is repeated

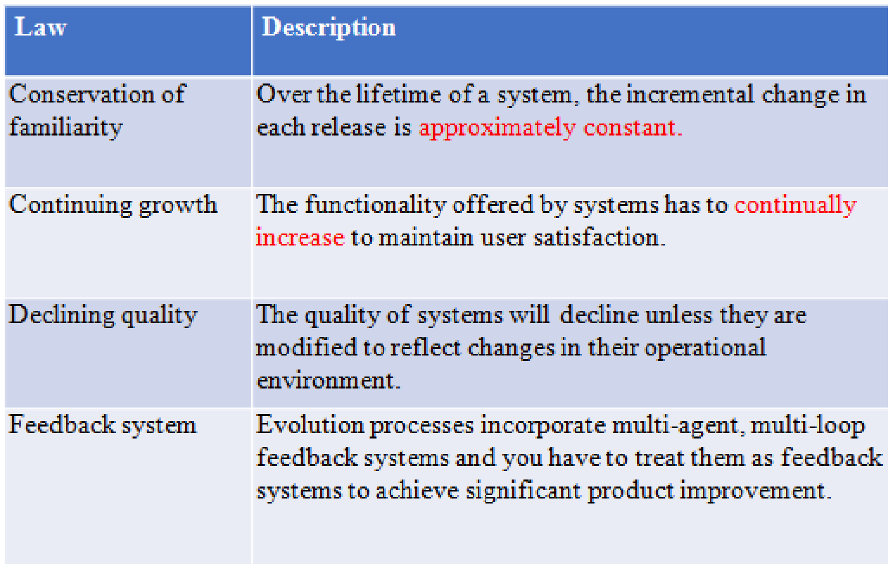
c. What are the different types of interfaces to be tested during component testing ? Explain.

Component testing, also known as program or module testing, is done after unit testing. In this type of testing those test objects can be tested independently as a component without integrating with other components e.g. modules, classes, objects, and programs. This testing is done by the development team.

Tests are often generated using a components interface.
Interface itself forms a part of the components requirements and hence this form of testing is black box
testing. However, the focus on the interface leads us to consider interface testing in its own right.
Techniques such as
o --->pairwise testing

	o --->interface mutation Pairwise testing: Set of values for each input is obtained from the components requirement. Interface mutation: The interface itself, such as function coded in /c orCORBA component written in an IDL,serves to extract the information needed to perform interface mutation. o pairwise testing:is a black box testing o interface mutation:is a white box testing	
8. a)	Write and Explain Lehman"s laws related to system change Ans: **Lehman's laws (8)** 	08

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.



Law	Description
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

8 b) What is Software Maintenance? Draw the general model of Reengineering process and explain. 08

ANS:

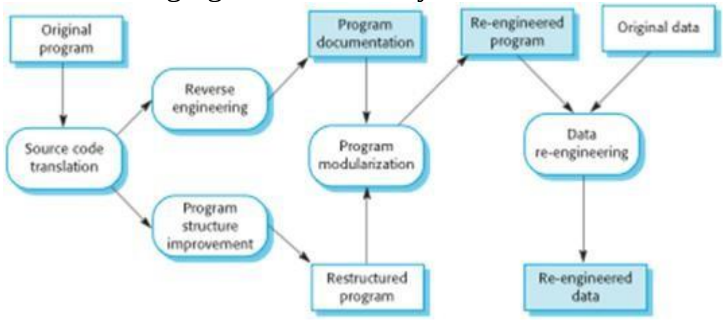
Software Maintenance is Modifying a program after it has been put into use.

•The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.

•Maintenance does not normally involve major changes to the system‟s architecture.

•Changes are implemented by modifying existing components and adding new components to the system.

Reengineering is Re-structuring or re-writing part or all of a legacy system without changing its functionality.

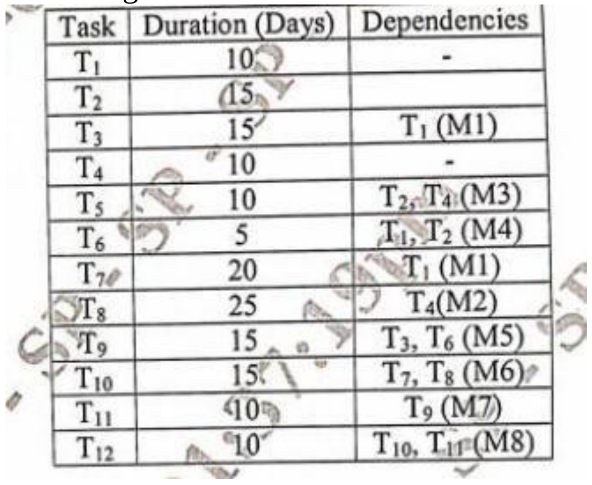


Details if reengineering process is

Source code translation: Convert code to a new language.

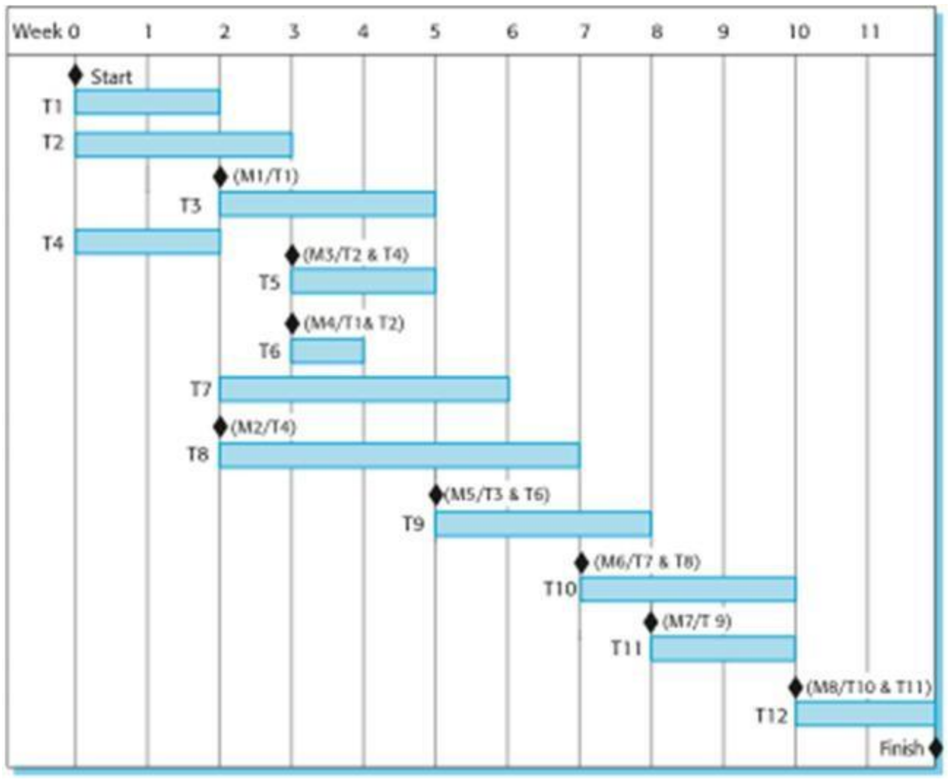
Reverse engineering: Analyse the program to understand it;

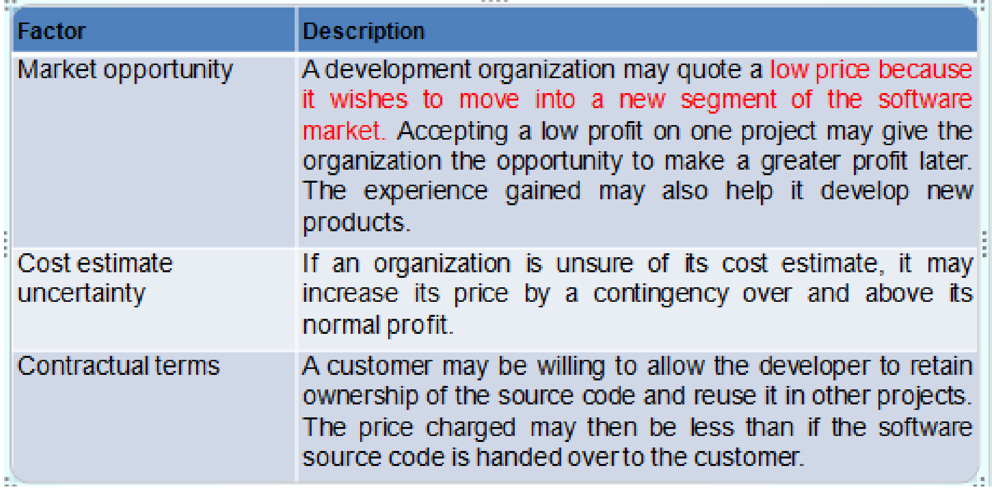
	Program structure improvement: Restructure automatically for understandability; Program modularisation: Reorganise the program structure; Data reengineering: Clean-up and restructure system data.	
8 c)	What are the strategic options involved in Legacy system management? Discuss ANS: Organisations that rely on legacy systems must choose a strategy for evolving these systems • Scrap the system completely and modify business processes so that it is no longer required; • Continue maintaining the system; • Transform the system by re-engineering to improve its maintainability; • Replace the system with a new system. The strategy chosen should depend on the system quality and its business value.	04
Module -5		
9. a)	For the set of Tasks shown below, draw the activity bar chart for the project scheduling.	08



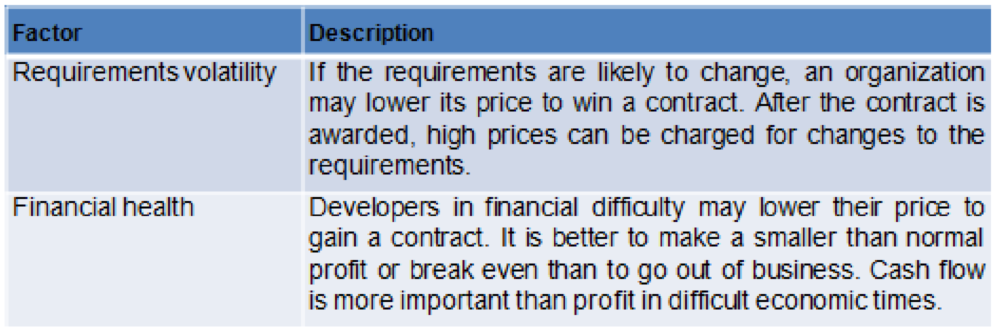
Task	Duration (Days)	Dependencies
T_1	10	-
T_2	15	
T_3	15	T_1 (M1)
T_4	10	-
T_5	10	T_2, T_4 (M3)
T_6	5	T_1, T_2 (M4)
T_7	20	T_1 (M1)
T_8	25	T_4(M2)
T_9	15	T_3, T_6 (M5)
T_{10}	15	T_7, T_8 (M6)
T_{11}	10	T_9 (M7)
T_{12}	10	T_{10}, T_{11} (M8)

ANS:

		
9. b)	Write and Explain the factors affecting Software Pricing ANS: Factors affecting software pricing	05

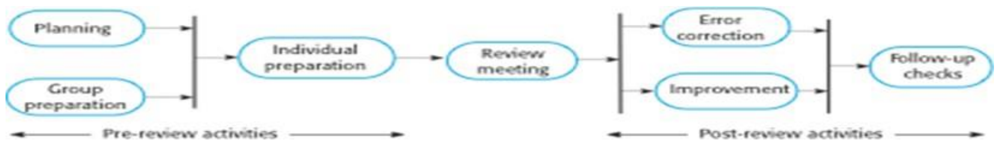
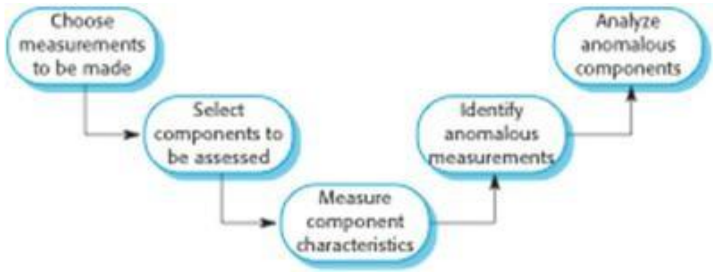


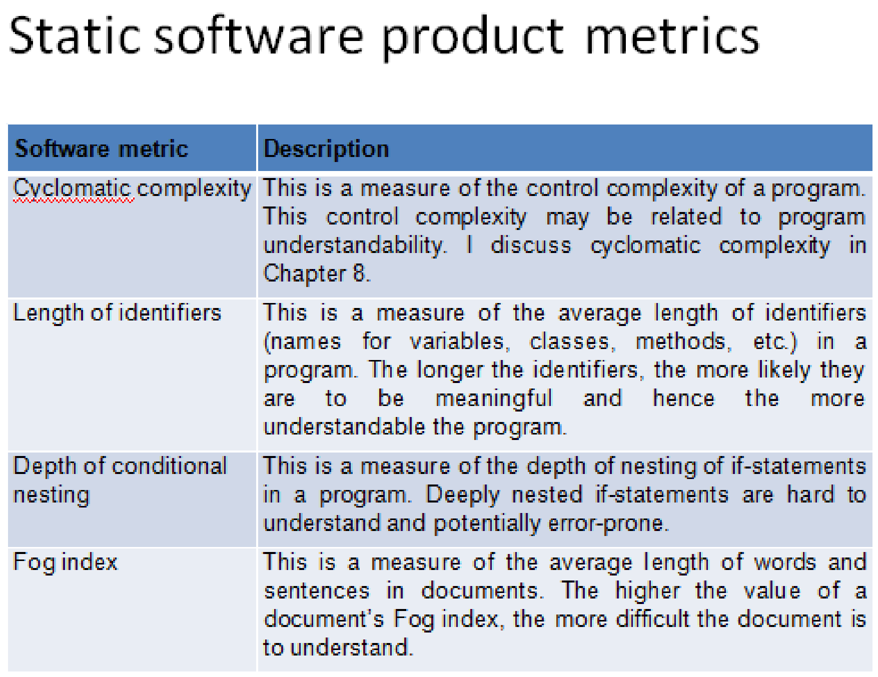
Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.



Factor	Description
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.

9. c	Explain briefly the algorithm cost modelling and write the difficulties. ANS: Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers: Effort = A ´Size^{B}´M –A is an organisation-dependent constant, –Size is code size of the software. –B reflects the disproportionate effort for large projects – M is a multiplier reflecting product, process and people attributes. The most commonly used product attribute for cost estimation is code size. Most models are similar but they use different values for A, B and M. Difficulties: The size of a software system can only be known accurately when it is finished. • Several factors influence the final size –Use of COTS and components; –Programming language; –Distribution of system. • As the development process progresses then the size estimate becomes more accurate. • The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.	07
10.a)	With a diagram , Explain the Phase involved in the software Review process. ANS: A group examines part or all of a process or system and its documentation to find potential problems.	08

	•Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management. •There are different types of review with different objectives –Inspections for defect removal (product); –Reviews for progress assessment (product and process); –Quality reviews (product and standards). A group of people carefully examine part or all of a software system and its associated documentation. Code, designs, specifications, test plans, standards, etc. can all be reviewed. Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.  Planning Group preparation Individual preparation Review meeting Error correction Improvement Follow-up checks Pre-review activities Post-review activities	
10. b)	Explain briefly the key stages in the process of product measurement. ANS:  Choose measurements to be made Select components to be assessed Measure component characteristics Identify anomalous measurements Analyze anomalous components A software measurement process as a part of the quality control process is shown in Figure. The steps of measurement process are the followings: 1. Select measurements to be made. Selection of measurements that are relevant to answer the questions to quality assessment. 2. Select components to be assessed. Selection of software components to	08

	be measured. 3. Measure component characteristics. The selected components are measured and the associated software metric values computed. 4. Identify anomalous measurements. If any metric exhibit high or low values it means that component has problems. 5. Analyze anomalous components. If anomalous values for particular metrics have been identified these components have to be examined to decide whether the anomalous metric values mean that the quality of the component is compromised.	
10.c)	Write any four product and process standards. ANS: 	04

Static software product metrics

Software metric	Description
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.