MODILE-1

1. a Describe software engineering code of ethics and professional practice as defined by IEEE/ACM (8)

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. ***Public***: Software engineers shall act consistently with the public interest.
2. ***Client and Employer***: Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. ***Product***: Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. ***Judgment***: Software engineers shall maintain integrity and independence in their professional judgment.
5. ***Management***: Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. ***Profession***: Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. ***Colleagues***: Software engineers shall be fair to and supportive of their colleagues.
8. ***Self***: Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

1. b Why software engineering is important? Explain the attributes of good software. (8)

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

Software engineering is important for two reasons:

(i) More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

(ii) It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

Essential attributes of good software:

(i) ***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.

(ii) ***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.

(iii) ***Efficiency***: Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.

(iv) ***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.

OR

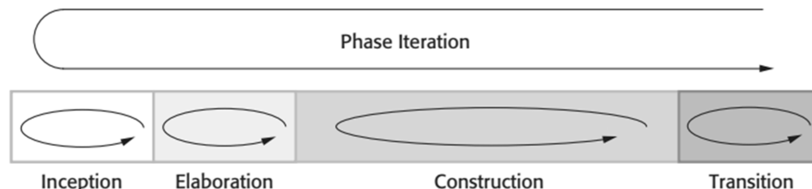
2. a Explain the phases in Rational Unified Process (8)

The Rational Unified Process (RUP) (Krutchen, 2003) is an example of a modern process model that has been derived from work on the UML and the associated Unified Software Development Process. The RUP recognizes that conventional process models present a single view of the process. In contrast, the RUP is normally described from **three perspectives**:

1. A dynamic perspective: which shows the phases of the model over time.

2. A static perspective: which shows the process activities that are enacted.

3. A practice perspective: which suggests good practices to be used during the process.



The RUP is a phased model that identifies **four discrete phases** in the software process.

1. Inception: The goal of the inception phase is to establish a business case for the system. You should identify all external entities (people and systems) that will interact with the system and define these interactions. You then use this information to assess the contribution that the system makes to the business. If this contribution is minor, then the project may be cancelled after this phase.

2. Elaboration: The goals of the elaboration phase are to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan, and identify key project risks. On completion of this phase you should have a requirements model for the system, which may be a set of UML use-cases, an architectural description, and a development plan for the software.

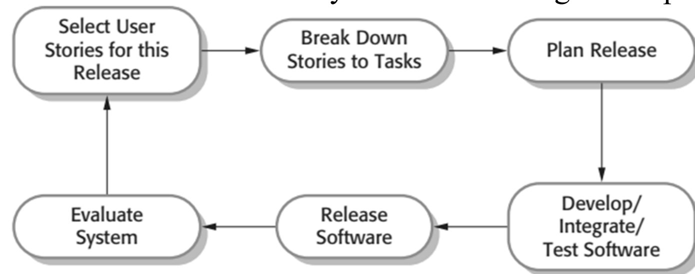
3. Construction: The construction phase involves system design, programming, and testing. Parts of the system are developed in parallel and integrated during this phase. On completion of this phase, you should have a working software system and associated documentation that is ready for delivery to users.

4. Transition: The final phase of the RUP is concerned with moving the system from the development community to the user community and making it work in a real environment. This is something that is ignored in most software process models but is, in fact, an expensive and sometimes problematic activity. On completion of this phase, you should have a documented software system that is working correctly in its operational environment.

2. b List and explain the Extreme Programming practices. (8)

Extreme programming (XP) is perhaps the best known and most widely used of the agile methods. In extreme programming, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task

before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system. The XP process to produce an increment of the system that is being developed.



Extreme programming involves a number of practices, which reflect the principles of agile methods:

1. Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
2. Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
3. People, not process, are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.
4. Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration of new functionality.
5. Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system.

1.	Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
2.	Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
3.	Simple design	Enough design is carried out to meet the current requirements.
4.	Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
5.	Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found to keep the code simple and maintainable.
6.	Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.

7.	Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
8.	Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
9.	Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
10.	On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

MODULE-2

3. a Explain the terms "user-requirements" and "system requirements". List different readers of user requirements and system requirements. (8)

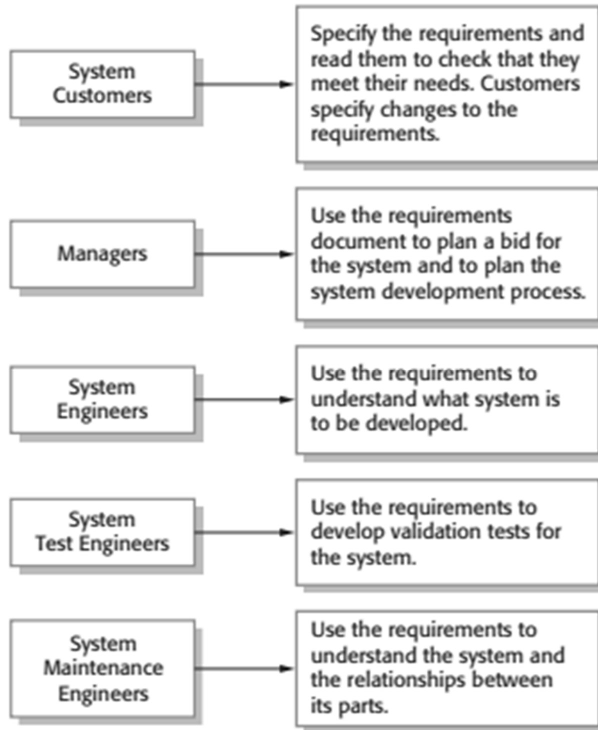
User requirements to mean the high-level abstract requirements and ***system requirements*** to mean the detailed description of what the system should do. User requirements and system requirements may be defined as follows:

1. User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

2. System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software.

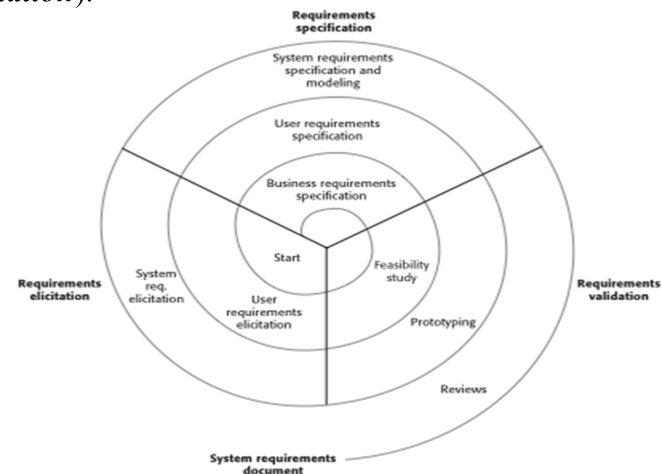
The diversity of possible users means that the requirements document has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for developers and testers, and including information about possible system evolution. Information on anticipated changes can help system designers avoid restrictive design decisions and help system maintenance engineers who have to adapt the system to new requirements.



The level of detail that you should include in a requirements document depends on the type of system that is being developed and the development process used. Critical systems need to have detailed requirements because safety and security have to be analyzed in detail. When the system is to be developed by a separate company (e.g., through outsourcing), the system specifications need to be detailed and precise. If an in-house, iterative development process is used, the requirements document can be much less detailed and any ambiguities can be resolved during development of the system.

3. b ***With a neat diagram explain the different types of activities that are performed in the requirement engineering process.*** ***(8)***

Requirements engineering processes may include **four high-level activities**. These focus on assessing if the system is useful to the business (***feasibility study***), discovering requirements (***elicitation and analysis***), converting these requirements into some standard form (***specification***), and checking that the requirements actually define the system that the customer wants (***validation***).



This spiral model accommodates approaches to development where the requirements are developed to different levels of detail. The number of iterations around the spiral can vary so the spiral can be exited after some or all of the user requirements have been elicited. Agile development can be used instead of prototyping so that the requirements and the system implementation are developed together.

Although structured methods have a role to play in the requirements engineering process, there is much more to requirements engineering than is covered by these methods. Requirements elicitation is a human-centered activity and people dislike the constraints imposed on it by rigid system models.

OR

4. a Explain the CBSE process with a neat diagram. (8)

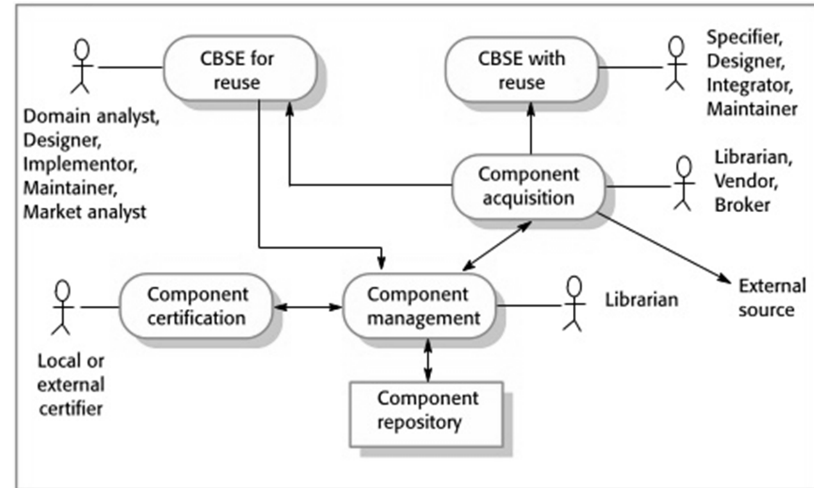
CBSE processes are software processes that support component-based software engineering. They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.

At the highest level, there are two types of CBSE processes:

1. Development for reuse: This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.

2. Development with reuse: This is the process of developing new applications using existing components and services.

These processes have different objectives and therefore, include different activities. In the development for reuse process, the objective is to produce one or more reusable components. You know the components that you will be working with and you have access to their source code to generalize them. In development with reuse, you don't know what components are available, so you need to discover these components and design your system to make the most effective use of them. You may not have access to the component source code.



The basic processes of CBSE with and for reuse have supporting processes that are concerned with component acquisition, component management, and component certification:

1. Component acquisition is the process of acquiring components for reuse or development into a reusable component. It may involve accessing locally developed components or services or finding these components from an external source.
2. Component management is concerned with managing a company's reusable components, ensuring that they are properly cataloged, stored, and made available for reuse.
3. Component certification is the process of checking a component and certifying that it meets its specification.

Components maintained by an organization may be stored in a component repository that includes both the components and information about their use.

(i) CBSE for reuse

CBSE for reuse is the process of developing reusable components and making them available for reuse through a component management system. There would be specialist component providers and component vendors who would organize the sale of components from different developers. Software developers would buy components to include in a system or pay for services as they were used. CBSE for reuse is most likely to take place within an organization that has made a commitment to reuse-driven software engineering. Changes that you may make to a component to make it more reusable include:

• removing application-specific methods;
• changing names to make them more general;
• adding methods to provide more complete functional coverage;
• making exception handling consistent for all methods;
• adding a 'configuration' interface to allow the component to be adapted to different situations of use;
• integrating required components to increase independence.

(ii) CBSE with reuse

The CBSE with reuse process has to include activities that find and integrate reusable components. Some of the activities within this process, such as the initial discovery of user requirements, are carried out in the same way as in other software processes.

The essential differences between CBSE with reuse and software processes for original software development are:

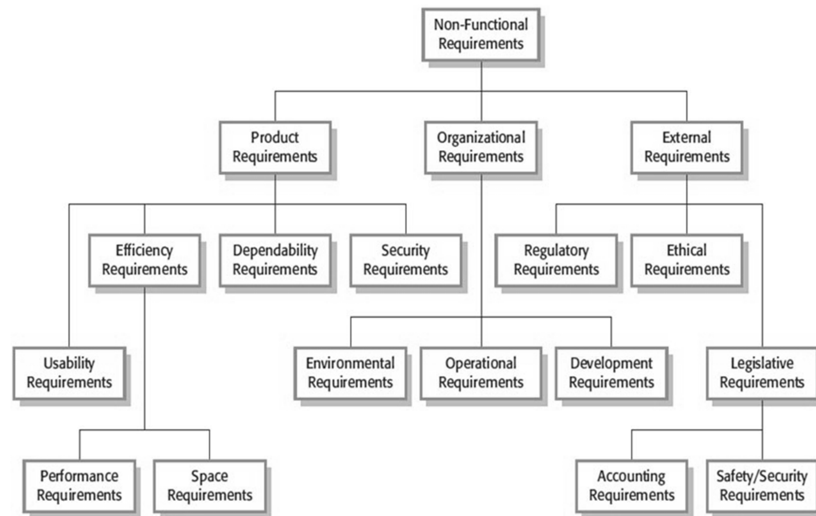
1. The user requirements are initially developed in outline rather than in detail, and stakeholders are encouraged to be as flexible as possible in defining their requirements. Requirements that are too specific limit the number of components that could meet these requirements. However, unlike incremental development, you need a complete set of requirements so that you can identify as many components as possible for reuse.
2. Requirements are refined and modified early in the process depending on the components available. If the user requirements cannot be satisfied from available components, you should discuss the related requirements that can be supported. Users may be willing to change their minds if this means cheaper or quicker system delivery.
3. There is a further component search and design refinement activity after the system architecture has been designed. Some apparently usable components may turn out to be unsuitable or do not work properly with other chosen components.
4. Development is a composition process where the discovered components are integrated. This involves integrating the components with the component model infrastructure and, often, developing adaptors that reconcile the interfaces of incompatible components. Of course, additional functionality may also be required over and above that provided by reused components.

4. b With a neat diagram, explain the different types of non-functional requirements. (8)

The requirements for a system are the descriptions of what the system should do. The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users. They may relate to emergent system properties such as reliability, response time, and store occupancy. Alternatively, they may define constraints on the system implementation such as the capabilities of I/O

devices or the data representations used in interfaces with other systems. Non-functional requirements, such as performance, security, or availability, usually specify or constrain characteristics of the system as a whole. The non-functional requirements may come from required characteristics of the software (product requirements), the organization developing the software (organizational requirements), or from external sources:



1. Product requirements: These requirements specify or constrain the behavior of the software. Examples include performance requirements on how fast the system must execute and how much memory it requires, reliability requirements that set out the acceptable failure rate, security requirements, and usability requirements. The product requirement is an availability requirement that defines when the system has to be available and the allowed down time each day.

2. Organizational requirements: These requirements are broad system requirements derived from policies and procedures in the customer's and developer's organization. Examples include operational process requirements that define how the system will be used, development process requirements that specify the programming language, the development environment or process standards to be used, and environmental requirements that specify the operating environment of the system. The organizational requirement specifies how users authenticate themselves to the system.

3. External requirements: This broad heading covers all requirements that are derived from factors external to the system and its development process. These may include regulatory requirements that set out what must be done for the system to be approved for use by a regulator, such as a central bank; legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements that ensure that the system will be acceptable to its users and the general public.

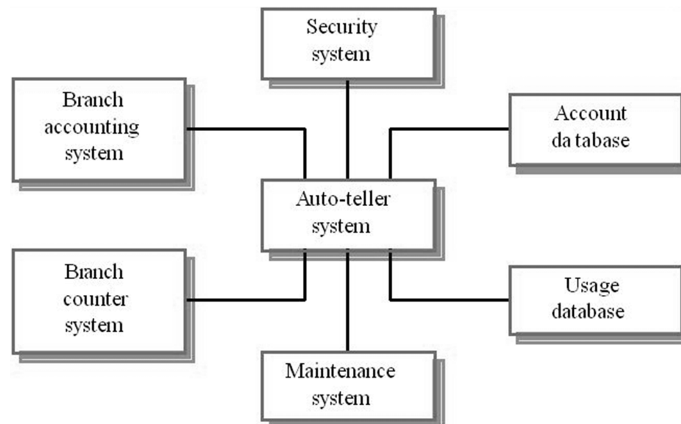
Non-functional requirements often conflict and interact with other functional or non-functional requirements. Non-functional requirements such as reliability, safety, and confidentiality requirements are particularly important for critical systems.

MODULE-3

5. a What is meant by system model? With an ATM model explain the context model. (8)

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. System modeling has generally come to mean representing the system using some kind of graphical

notation, which is now almost always based on notations in the Unified Modeling Language (UML).
Context Diagrams are used in systems design to represent the more important external actors that interact with the system. The objective of a system context diagram is to focus attention on external factors and events that should be considered in developing a complete set of system requirements and constraints.



The context model of the ATM describes the different functionalities of the machine in accordance with the banking business process. The core is the auto-teller system.

5. b ***Explain the role of software architecture.*** ***(8)***

Architecture is a design of a system which gives a very high level view of the parts of the system and how they are related to form the whole system. An architecture description of a system will therefore describe the different structures of the system.

The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Some of the important uses that software architecture descriptions play are:

1. Understanding and communication: An architecture description is primarily to communicate the architecture to its various stakeholders, which include the users who will use the system, the clients who commissioned the system, the builders who will build the system, and, of course, the architects.

2. Reuse: Architecture descriptions can help software reuse. Reuse is considered one of the main techniques by which productivity can be improved, thereby reducing the cost of software. The architecture has to be chosen in a manner such that the components which have to be reused can fit properly and together with other components that may be developed, they provide the features that are needed.

3. Construction and Evolution: Architecture can help decide what is the impact of changes to existing components on others A suitable partitioning in the architecture can provide the project with the parts that need to be built to build the system. During software evolution, architecture helps decide what needs to be changed to incorporate the new changes/features

4. Analysis: It is highly desirable if some important properties about the behavior of the system can be determined before the system is actually built. Software architecture provides possibilities for software to consider the alternatives during design to reach the desired performance levels like reliability

OR

---

6. a Explain architectural styles of C and C view (8)

- C &C view has two main elements—components and connectors.
- Components: computational elements or data stores that have some presence during the system execution.
- Connectors: define the means of interaction between these components
- A component and connector (C&C) view of the system defines the components, and which component is connected to which and through what connector.
- C&C view describes a runtime structure of the system
- Architectural styles: some structures and related constraints that have been observed in many systems and that seem to represent general structures that are useful for architecture of a class of problems

(i) Pipe and Filter

- well suited for systems that primarily do data transformation some input data is received and the goal of the system is to produce some output data by suitably transforming the input data
- A filter performs a data transformation, and sends the transformed data to other filters for further processing using the pipe connector

(ii) Shared-Data Style

- there are two types of components:

i. data repositories: Stores shared data (these could be file systems or databases)

- provide a reliable and permanent storage
- take care of any synchronization needs for concurrent access provide data access support

ii. data accessors

- access data from the repositories
- perform computation on the data obtained
- if they want to share the results with other components, put the results back in the depository
- Communication between data accessors is only through the repository

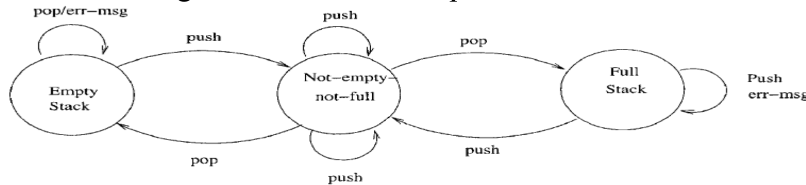
(iii) Client-Server Style

- Clients can only communicate with the server, but not with other clients
- Communication is initiated by a client which sends request and server responds
- A 3-tier structure is commonly used by many application and web systems
  - Client-tier contains the clients
  - Middle-tier contains the business rules
  - Database tier has the information

6. b Explain state machine models with an example. (8)

- a class is not a functional abstraction and cannot be viewed as an algorithm.
- A method of a class can be viewed as a functional module, and the methods can be used to specify the logic
- An object of a class has some state and many operations on it.
- A method to understand the behavior of a class is to view it as a finite state automata (FSA).
- An FSA consists of states and transitions between states (values), which take place when some events occur

- A state diagram relates events and states by showing how the state changes when an event is performed



MODULE-4

7. a Explain function oriented design. (8)

- Design Principles
  - Correctness: the system is built precisely according to the design which satisfies the requirements of that system
  - Verifiable: in accordance with the requirements document
  - Complete: implements all the specifications
  - Traceable: all design elements can be traced to some requirements
  - Efficiency: is concerned with the proper use of scarce resources by the system
  - Simplicity: is perhaps the most important quality criteria for software systems.
- ▪ Abstraction
- The basic goal of system design is to specify the modules in a system and their abstractions
- An abstraction of a component describes the external behavior of that component without bothering with the internal details
- Abstraction is used for existing components as well as components that are being designed
- There are two common abstraction mechanisms for software systems:
  - functional abstraction: a module is specified by the function it performs
- It is the basis of partitioning in function-oriented approaches
  - Data abstraction supports certain operations are required from a data object, depending on the object and the environment in which it is used
- ▪ Modularity:
- two of the most important quality criteria for software design are: simplicity and understandability
- Modularity supports independence of models
- Modularity enhances design clarity, eases implementation
- Reduces cost of testing, debugging and maintenance
- Design hierarchy: two possible approaches

(i) Top-down approach: starts from the highest-level component of the hierarchy and proceeds through to lower levels.

- starts by identifying the major components of the system, decomposing them into their lower- level components and iterating until the desired level of detail is achieved.

(ii) Bottom-up approach: starts with the lowest-level component of the hierarchy and proceeds through progressively higher levels to the top-level component.

- Bottom-up methods work with layers of abstraction.
- Starting from the very bottom, operations that provide a layer of abstraction are implemented.
- The operations of this layer are then used to implement more powerful operations and a still higher layer of abstraction, until the stage is reached where the operations supported by the layer are those desired by the system

---

7. b Write a note on cohesion and coupling. (8)

- A module is a logically separable part of a program
- Modularization criteria: (i) Coupling and (ii) Cohesion

(i) Cohesion (intra-module criteria)

- It identifies the relationship between elements of the same module
- determining how closely the elements of a module are related to each other
- Cohesion of a module represents how tightly bound the internal elements of the module are to one another
- the greater the cohesion of each module in the system, the lower the coupling between modules is. (i.e) max(cohesion) and min(coupling)
- There are several levels of cohesion:
  - ***Coincidental cohesion*** occurs when there is no meaningful relationship among the elements of a module
  - A module has ***logical cohesion*** if there is some logical relationship between the elements of a module, and the elements perform functions that fall in the same logical class
  - ***Temporal cohesion*** is the same as logical cohesion, except that the elements are also related in time and are executed together
  - A ***procedurally cohesive module*** contains elements that belong to a common procedural unit
  - ***Sequentially cohesive modules*** bear a close resemblance to the problem structure
  - ***Functional cohesion***: In a functionally bound module, all the elements of the module are related to performing a single function. It is the strongest cohesion
  - A module with ***communicational cohesion*** has elements that are related by a reference to the same input or output data

(ii) Coupling (inter-module criteria)

- The notion of coupling attempts to capture this concept of "how strongly" different modules are interconnected
- Coupling between modules is the strength of interconnections between modules or a measure of interdependence among modules
- "Highly coupled" modules are joined by strong interconnections, while "loosely coupled" modules have weak interconnections.
- Factors affecting coupling:
  - ***Complexity*** of the interface
  - ***obscurity*** of the interface between modules
- coupling is reduced when elements in different modules have little or no relationship between them

OR

8. a Briefly explain the architectural patterns for distributed systems. (8)

Five architectural styles:

1. Master-slave architecture, which is used in real-time systems in which guaranteed interaction response times are required.

2. Two-tier client–server architecture, which is used for simple client–server systems, and in situations where it is important to centralize the system for security reasons. In such cases, communication between the client and server is normally encrypted.

---

3. Multitier client–server architecture, which is used when there is a high volume of transactions to be processed by the server.
4. Distributed component architecture, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client–server systems.
5. Peer-to-peer architecture, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other. It may also be used when a large number of independent computations may have to be made.

8. b Discuss the complexity matrix for function oriented design. (4)

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht , who suggested a measure called the function point. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.
Function points are computed by completing the table as shown below. Five information domain characteristics are determined and counts are provided in the appropriate table location. Information domain values are defined in the following manner:
Number of user inputs. Each user input that provides distinct applicationoriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

- Number of user outputs. Each user output that provides application oriented information to the user is counted. Example: reports, screens, error messages, etc.
- Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.
- Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.
- Number of external interfaces. All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

Measurement parameter	Count		Weighting factor				
			Simple	Average	Complex		
No. of user inputs		X	3	4	6	=	
No. of user outputs		X	4	5	7	=	
No. of user inquiries		X	3	4	6	=	
No. of files		X	7	10	15	=	
No. of external interfaces		X	5	7	10	=	
Count total	→						

To compute function points (FP), the following relationship is used:
FP = count total [0.65 + 0.01 Σ(Fi)] where count total is the sum of all FP entries .

8. c Write a note on Software as a Service. (4)

This notion of SaaS involves hosting the software remotely and providing access to it over the Internet. The key elements of SaaS are the following:
1. Software is deployed on a server (or more commonly a number of servers) and is accessed through a web browser. It is not deployed on a local PC.
2. The software is owned and managed by a software provider, rather than the organizations using the software.
3. Users may pay for the software according to the amount of use they make of it or through an annual or monthly subscription. Sometimes, the software is free for anyone to use but users must then agree to accept advertisements, which fund the software service.
For software users, the benefit of SaaS is that the costs of management of software are transferred to the provider. The provider is responsible for fixing bugs and installing software upgrades, dealing with changes to the operating system platform, and ensuring that hardware capacity can meet demand. Software licence management costs are zero. The software may be accessed from mobile devices, such as smart phones, from anywhere in the world.
Disadvantages:

- the costs of data transfer to the remote service
- lack of control over software evolution
- problems with laws and regulations

MODULE-5

9. a Explain project schedule and staffing. (8)

- Once the effort is estimated, various schedules (or project duration) are possible, depending on the number of resources (people) put on the project
- A schedule cannot be simply obtained from the overall effort estimate by deciding on average staff size and then determining the total time requirement by dividing the total effort by the average staff size.
- In a project, the scheduling activity can be broken into two sub-activities:
  - determining the overall schedule (the project duration) with major milestones, and
  - developing the detailed schedule of the various tasks.
- One method to determine the normal (or nominal) overall schedule is to determine it as a function of effort
- One approach: fitting a regression curve through the scatter plot obtained by plotting the effort and schedule of past projects
- the total duration, M, in calendar months can be estimated by $M = 4.1*E^{0.36}$ by IBM Federal Systems Division
- In COCOMO, the equation for schedule for an organic type of software is $M = 2.5*E^{0.38}$
- Square root check (rule of thumb), is sometimes used to check the schedule of medium-sized projects
- the proposed schedule can be around the square root of the total effort in person-months
- Detailed Scheduling
- Once the milestones and the resources are fixed, it is time to set the detailed scheduling

- For detailed schedules, the major tasks fixed while planning the milestones are broken into small schedulable activities in a hierarchical manner.
- At each level of refinement, the project manager determines the effort for the overall task from the detailed schedule and checks it against the effort estimates.
- Activities related to tasks such as project management, coordination, database management, and configuration management may also be listed in the schedule, even though these activities have less direct effect on determining the schedule

▪ Team Structure

- Detailed scheduling is done only after actual assignment of people has been done

(i) Hierarchical (Chief Programmer Team) organization:

- the project manager is responsible for all major technical decisions of the project
- The team typically consists of programmers, testers, a configuration controller, and possibly a librarian for documentation.

(ii) Egoless team (democratic team) organization:

  - consist of ten or fewer programmers
  - Group leadership rotates among the group members
  - suited for long-term research-type projects

(iii) Emerging organization:

- recognizes that there are three main task categories in software development: (i) development related, (ii) testing related and (iii) management related
- recognizes that it is often desirable to have the test and development team be relatively independent, and also not to have the developers or tests report to a nontechnical manager
- there is an overall unit manager, under whom there are three small hierarchic organizations – (i) for development, (ii) for testing and (iii) for program management
  - The developers write code and they work under a development manager
  - The testers will test the code and they work under a test manager
  - The program managers provides the specifications for what is being built, and ensure that development and testing are properly coordinated

9. b Explain the activities of software configuration management plan. (8)

▪ Planning for configuration management involves:
  - identifying the configuration items and
  - specifying the procedures to be used for controlling and implementing changes to them

▪ The activities in this stage include:
  - Identify configuration items, including customer-supplied and purchased items.
  - Define a naming scheme for configuration items.
  - Define the directory structure needed for configuration management.
  - Define version management procedures, and methods for tracking changes to configuration items.
  - Define access restrictions.
  - Define change control procedures.

- Identify and define the responsibility of the configuration controller.
- Identify baseline points
- Define a backup procedure and a reconciliation procedure, if needed.
- Define a release procedure.

• The output of this phase is the configuration management plan

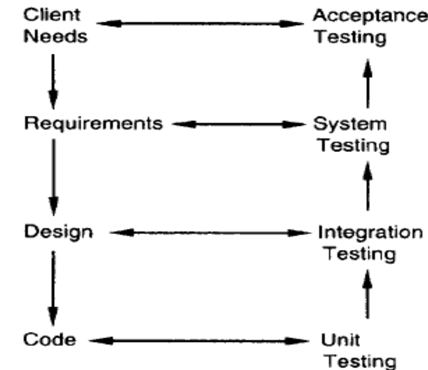
OR

10. a Describe in details the process of testing. (8)

- The basic goal of the software development process is to produce software that has no errors
- Most of the verification methods are based on human evaluation
- Regression testing: old test cases are executed with the expectation that the same old results will be produced (after incorporating changes)
- testing should not be done on-the-fly
- testing process focuses on how testing should proceed for a particular project

▪ Levels of Testing

- Testing is usually relied upon to detect the faults remaining from earlier stages
- different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system



- Unit testing: different modules are tested against the specifications produced during design for the modules
- integration testing: many unit tested modules are combined into subsystems, which are then tested
- System testing: complete and integrated software is tested
- Acceptance testing: a system is tested for acceptability

▪ Test Plan

- testing commences with a test plan and terminates with acceptance testing.
- a general document for the entire project that defines:
  - the scope,
  - approach to be taken, and
  - the schedule of testing
  - identifies the test items for the entire testing process and
  - the personnel responsible for the different activities of testing
- The inputs for forming the test plan are:

(1) project plan,
(2) requirements document, and
(3) system design document.

▪ Test Case Specifications

- to be done separately for each unit
- the features to be tested for this unit must be determined
- the test cases are specified for testing the unit
- Test case specification gives:
  - All test cases,
  - inputs to be used in the test cases,
  - conditions being tested by the test case, and
  - outputs expected for those test cases
- two basic reasons test cases are specified before they are used for testing:
  - Test case review (having the test case specification in the form of a document)
  - the process of specifying all the test cases will be used for testing helps the tester in selecting a good set of test cases

10. b Mention the difference between white box testing and block box testing. (8)

The following are the major differences:

Sl. No.	Black box testing	White box testing
1	It is used to test the software without knowing the internal structure of code or program.	The internal structure is being known to tester who is going to test the software.
2	It also knowns as data-driven, box testing, data-, and functional testing	It is also called structural testing, clear box testing, code-based testing, or glass box testing
3	The main objective of this testing is to check what functionality of the system under test	The main objective of White Box testing is done to check the quality of the code
4	This testing is carried out by testers.	this type of testing is carried out by software developers
5.	Implementation Knowledge is not required	Implementation Knowledge is required
6.	Black Box testing can be started based on Requirement Specifications documents	White Box testing can be started based on Detail Design documents
7	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing	Testing is best suited for a lower level of testing like Unit Testing, Integration testing
8	Granularity is low	Granularity is high