


CMR INSTITUTE OF TECHNOLOGY		USN <input type="text"/>							
Internal Assessment Test – III, November 2018									
Sub:	SOFTWARE ENGINEERING						Code:	17MCA34	
Date:	20-11-2018	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA
Answer ONE FULL QUESTION from each part								Marks	OBE
								CO	RBT
Part - I									
1	Explain in detail about different levels of cohesion.						10	CO5	L2
(OR)									
2	Discuss about Cyclomatic complexity with example.						10	CO5	L2
Part – II									
3	Explain in detail about the issues related to the distributed system design.						10	CO5	L2
(OR)									
4	Explain in detail about five different architectural styles of distributed systems.						10	CO5	L2
Part – III									
5	Explain in detail about software as a service.						10	CO5	L2
(OR)									
6	Describe in detail about COCOMO effort estimation model.						10	CO5	L2
PART - IV									
7	Discuss in detail about project scheduling and staffing.						10	CO5	L2
(OR)									
8	Briefly explain the steps involved in the risk management process.						10	CO5	L2
Part – V									
9	Write in detail about white box testing techniques.						10	CO4	L2
(OR)									
10	Write in detail about any two black box testing techniques.						10	CO4	L2

1. Explain in detail about different levels of cohesion (10)

i. Coincidental cohesion occurs when there is no meaningful relationship among the elements of a module

- tasks that have no meaningful relationship to one another
- Example: Fix Car, Bake Cake, Walk Dog, Fill out Application Form, Go to the Movie

ii. 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

- elements contribute to activities of the same general category in which the activity or activities to be executed are selected from outside the module.
- Example: contemplating a journey might compile the following list: Go by car, Go by Train, Go by Plane

iii. Temporal cohesion is the same as logical cohesion, except that the elements are also related in time and are executed together

- tasks that are all related in time
- Example: “initialization”, “termination”, “Do All Startup Activities”

iv. A procedurally cohesive module contains elements that belong to a common procedural unit

- possibly unrelated activities, in which control passes from one activity to the next
- Example: a loop or a sequence of decision statements

v. Sequentially cohesive modules bear a close resemblance to the problem structure.

- a sequentially bound module may contain several functions or parts of different functions
- elements are involved in activities such that output data from one activity serves as input data to the next (data passes from one activity to another)

iii. Temporal cohesion is the same as logical cohesion, except that the elements are also related in time and are executed together

- tasks that are all related in time
- Example: “initialization”, “termination”, “Do All Startup Activities”

iv. A procedurally cohesive module contains elements that belong to a common procedural unit

- possibly unrelated activities, in which control passes from one activity to the next
- Example: a loop or a sequence of decision statements

v. Sequentially cohesive modules bear a close resemblance to the problem structure.

- a sequentially bound module may contain several functions or parts of different functions
- elements are involved in activities such that output data from one activity serves as input data to the next (data passes from one activity to another)

(OR)

2. Discuss about Cyclomatic complexity with example (10)

- A more refined measure is the cyclomatic complexity measure
- For a graph G, the cyclomatic number $V(G)$ is defined as $V(G) = e - n + p$ where n – no. of nodes, e - no. of edges, and p - connected components (a subgraph in which any two vertices are connected to each other by paths) or p is number of nodes that have exit points
- To use this to define the cyclomatic complexity of a module, the control flow graph (CFG) G of the module is first drawn.
- To construct a control flow graph of a program module
 - break the module into blocks (nodes) delimited by statements that affect the control flow, like if, while, repeat, and goto
 - If the control from a block i can branch to a block j , then draw an arc from node i to node j in the graph
- The control flow of a program can be constructed mechanically
- If the Cyclomatic number is between 1 and 10 then it represents the Structured and well written code, High Testability and Cost and Effort is less

Note: the students can compute the Cyclomatic number based on the graph and connected components.

3. Explain in detail about the issues related to the distributed system design (10)

- Transparency
 - To what extent should the distributed system appear to the user as a single system?

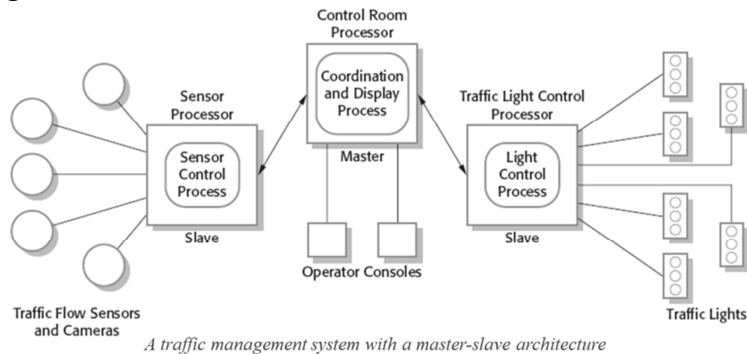
- Openness
 - Should a system be designed using standard protocols that support interoperability?
- Scalability
 - How can the system be constructed so that it is scalable?
- Security
 - How can usable security policies be defined and implemented?
- Quality of service (QoS)
 - How should the quality of service be specified?
- Failure management
 - How can system failures be detected, contained and repaired?

(OR)

4. Explain in detail about five different architectural styles of distributed systems (10)

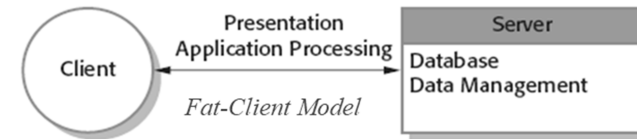
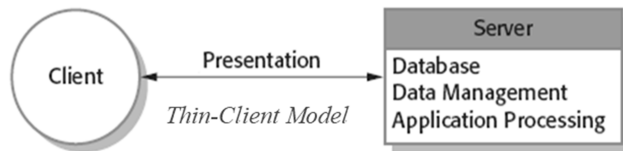
- When designing a distributed application, you should choose an architectural style (pattern) that supports the critical non-functional requirements of your system.
- Five architectural styles:
 1. Master-slave architecture: which is used in real-time systems in which guaranteed interaction response times are required.
- commonly used in real-time systems where there may be separate processors associated with data acquisition from the system's

environment, data processing, and computation and actuator management



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.

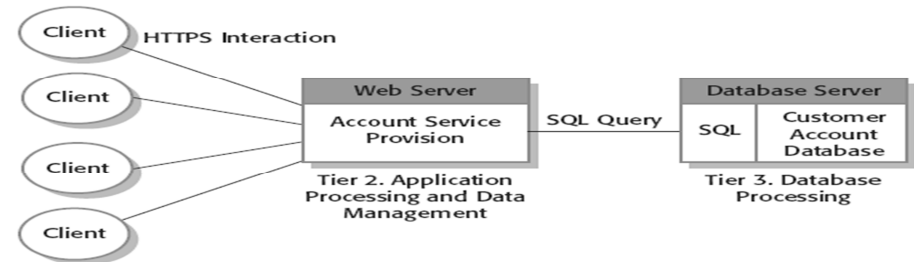
- The system is implemented as a single logical server plus an indefinite number of clients that use that server
- *Thin-Client model*: the presentation layer is implemented on the client and all other layers (data management, application processing, and database) are implemented on a server.
- *Fat-client model*: some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.



3. Multi-tier client–server architecture: which is used when there is a high volume of transactions to be processed by the server.

- Improves the problems identified in the previous models – scalability, performance
- the different layers of the system are separate processes that may execute on different processors.
- Example: An Internet banking system
- The three-tier client–server model can be extended to a multi-tier variant, where additional servers are added to the system

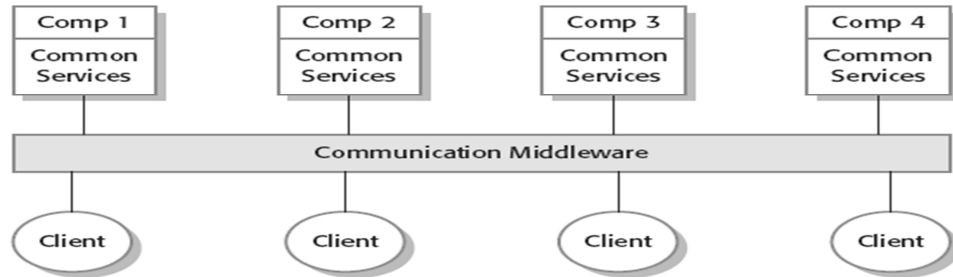
Tier 1. Presentation



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.

- There is no distinction in distributed component architectures between clients and servers.
- Each distributable entity is an object that provides services to other components and receives services from other components.

- Component communication is through a middleware system.
- However, distributed component architectures are more complex to design than client server systems.

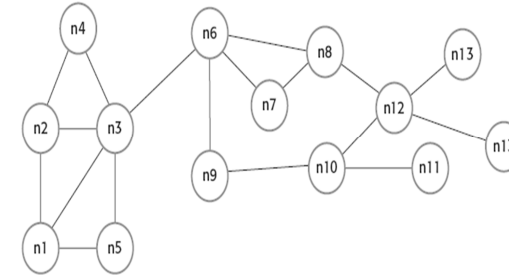


5. Peer-to-peer (P2P) architecture: which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other. It may also be used when a large number of independent computations may have to be made.

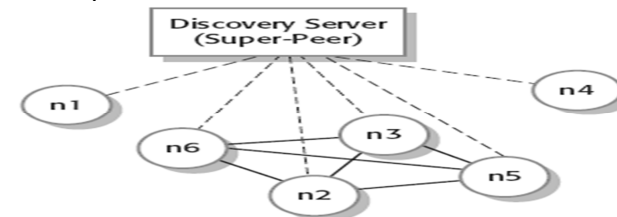
- In the typical client-server architectural models, servers do more work than clients
- P2P are decentralised systems where computations may be carried out by any node in the network.
- The overall system is designed to take advantage of the computational power and storage of a large number of networked computers
- The standards and protocols that enable communications across the nodes are embedded in the application itself and each node must run a copy of that application.

Example: BitTorrent, Instant messaging systems, Voice over IP (VOIP) phone services, such as Skype

- *Decentralized architecture*: the nodes in the network are not simply functional elements but are also communications switches that can route data and control signals from one node to another



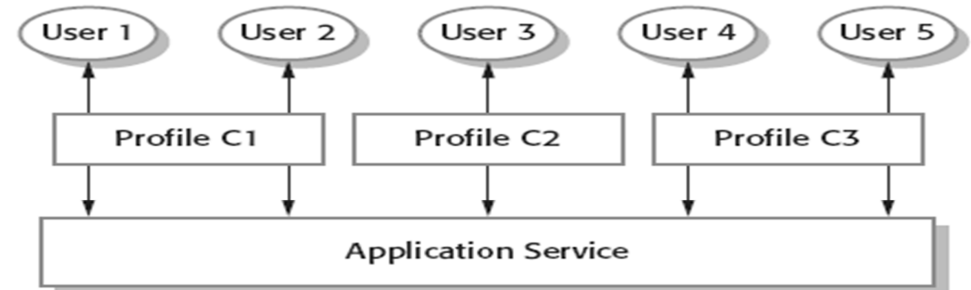
- Semi-centralized architecture: one or more nodes act as servers to facilitate node communications
 - This reduces the amount of traffic between nodes
 - the role of the server (sometimes called a super-peer) is to help establish contact between peers in the network, or to coordinate the results of a computation



5. Explain in detail about software as a service (5)

- the problems of server overload can be significantly reduced by using a modern browser as the client software.
- a browser can be configured and used as a client, with significant local processing.

- This notion of Software as a Service (SaaS) involves hosting the software remotely and providing access to it over the Internet
- The key elements of SaaS
 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.
 - Implementation factors for SaaS
 - *Configurability*: How do you configure the software for the specific requirements of each organization?
 - *Multi-tenancy*: How do you present each user of the software with the impression that they are working with their own copy of the system while, at the same time, making efficient use of system resources?
 - *Scalability*: How do you design the system so that it can be scaled to accommodate an unpredictably large number of users?



- *the benefit of SaaS*: 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 license management costs are zero and pay-per-use model
- *Disadvantages*:
 - the costs of data transfer to the remote service
 - lack of control over software evolution
 - problems with laws and regulations

(OR)

6. Describe in detail about COCOMO effort estimation model (10)
- estimates the total effort in terms of person-months.
 - The basic steps in this model are:

1. Obtain an initial estimate (also called nominal estimate) of the development effort from the estimate of thousands of delivered lines of source code (KLOC).
2. Determine a set of 15 multiplying factors from different attributes of the project.
3. Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors
 - To determine the initial effort E_i in person-months the equation used is of the type $E_i = a * (KLOC)^b$
 - The value of the constants a and b depend on three of the project types: organic, semidetached, and embedded
 - There are 15 different attributes, called cost driver attributes that determine the multiplying factors.
 - The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF).
 - The final effort estimate, E , is obtained by multiplying the initial estimate by the EAF. That is, $E = EAF * E_i$.

7. Discuss in detail about project scheduling and staffing (10)

- 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.

- Overall Scheduling
 - 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
 - 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.
 - 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
- input from every member is taken for major decisions
- Group leadership rotates among the group members

(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

(OR)

8. Briefly explain the steps involved in the risk management process (10)

- Risk is defined as an exposure to the chance of injury or loss
- Risk management is an attempt to minimize the chances of failure caused by unplanned events
- The aim of risk management is not to avoid getting into projects that have risks but to minimize the impact of risks in the projects that are undertaken
- Risk management is the area that tries to ensure that the impact of risks on cost, quality, and schedule is minimal
- risk management has to deal with:
 - identifying the undesirable events that can occur,
 - the probability of their occurring, and
 - the loss if an undesirable event does occur.
- the risk management revolves around risk assessment and risk control
- Risk Assessment: an activity that must be undertaken during project planning
- This involves:
 - identifying the risks
 - analyzing them and
 - prioritizing them on the basis of the analysis
- The goal of risk assessment is to prioritize the risks so that attention and resources can be focused on the more risky items

-
- Risk identification identifies all the different risks for a particular project
 - Methods that can aid risk identification include:
 - checklists of possible risks
 - meetings and brainstorming
 - work products
 - surveys
 - reviews of plans
 - processes
 - Risk Analysis:
 - the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize
 - This is often done through discussion, using experience and understanding of the situation.
 - Cost estimation (like COCOMO) can be used to assess the cost and schedule risks
 - The other approaches for risk analysis include:
 - decision analysis: studying the probability and the outcome of possible decisions
 - network analysis: understanding the task dependencies to decide critical activities and the probability and cost of their not being completed on time
 - quality factor analysis: risks on the various quality factors like reliability and usability and
 - performance analysis: evaluating the performance early through simulation, etc., if there are strong performance constraints on the system

9. Write in detail about white box testing techniques (10)

- White box testing focuses on implementation
- Is also called structural testing
- Types of White-Box (structural) testing:
 - Control flow based criteria: looks at the coverage of the control flow graph
 - Data flow based testing: looks at the coverage in the definition-use graph
 - Mutation testing: looks at various mutants of the program
- Control flow based criteria
 - Considers the program as control flow graph
 - Nodes represent code blocks – i.e. set of statements always executed together
 - An edge (i,j) represents a possible transfer of control from i to j
 - Statement Coverage Criterion:
 - Criterion: Each statement is executed at least once during testing (i.e.) set of paths executed during testing should include all nodes
 - Limitation: does not require a decision to evaluate to false if no else clause
 - Branch coverage
 - Criterion: Each edge should be traversed at least once during testing i.e. each decision must evaluate to both true and false during testing
 - Branch coverage implies statement coverage

-
- If multiple conditions in a decision, then all conditions need not be evaluated to T and F
 - The trouble with branch coverage comes if a decision has many conditions in it
 - There are other criteria too - path coverage, predicate coverage, cyclomatic complexity
 - Data Flow-Based Testing
 - select the paths to be executed during testing based on data flow analysis
 - information about where the variables are defined and where the definitions are used is also used to specify the test cases
 - The basic idea: the definitions of variables and their subsequent use is tested
 - ensure some coverage of the definitions and uses of variables
 - A variable occurrence can be one of the following three types:
 - def represents the definition of a variable
 - c-use represents computational use of a variable
 - p-use represents predicate use
 - Mutation Testing
 - It takes the program and creates many mutants of it by making simple changes to the program
 - The goal: make sure that during the course of testing, each mutant produces an output different from the output of the original program

- faults of some pre-decided types are introduced in the program being tested.
- Testing then tries to identify those faults in the mutants
- this technique will be successful only if the changes introduced in the main program capture the most likely faults in some form
- competent programmer hypothesis: programmers are generally very competent
- a programmer will produce a program that is very "close" to a correct program
- coupling effect: the test cases that distinguish programs with minor differences with each other are so sensitive

(OR)

10. Write in detail about any two black box testing techniques (10)

- In black-box testing the structure of the program is not considered
- It is also called functional or behavioral testing
 - Equivalence Class Partitioning
- divide the input domain into a set of equivalence classes, so that if the program works correctly for a value then it will work correctly for all the other values in that class
- The equivalence class partitioning method tries to approximate without looking at the internal structure of the program

-
- An equivalence class is formed of the inputs for which the behavior of the system is specified or expected to be similar.
 - Example: $0 < \text{count} < \text{Max} = \text{count} < 0$ and $\text{count} > \text{Max}$
 - Boundary Value Analysis (BVA)
 - an equivalence class fail on some special values.
 - These values often lie on the boundary of the equivalence class
 - In boundary value analysis, choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes
 - Boundary value test cases are also called "extreme cases"
 - Cause-Effect Graphing
 - is a technique that aids in selecting combinations of input conditions in a systematic way, such that the number of test cases does not become unmanageably large
 - A cause is a distinct input condition, and an effect is a distinct output condition
 - Each condition forms a node in the cause-effect graph. The conditions should be stated such that they can be set to either true or false
 - Example: an input condition "file is empty" can be set to true by having an empty input file, and false by a nonempty file.
 - Pair-wise Testing
 - single-mode fault: parameters can take different values, and for some of them the software may not work correctly. (Example: a software not able to print for a particular type of printer)
 - multi-mode faults: there are n parameters for a system, and each one of them can take m different values, generate each test case one different value of each parameter (Example: a telephone billing software that does not compute correctly for night time calling)
 - most software faults are revealed on some special single values or by an interaction of pair of values
 - pair-wise testing: test all combinations of values for each pair of parameters
-