E –51842

Register No. | | | | | | | |

BTech Degree Examination November 2021

Fifth Semester

Information Technology

18ITT53 – SOFTWARE ENGINEERING

(Regulations 2018)

Time: Three hours　　　　Maximum: 100 marks

Answer all Questions

Part – A (10 × 2 = 20 marks)

1. Write the IEEE definition of software engineering. [CO1,K1]
2. Give the reasons for developing prototype. [CO1,K2]
3. Classify the following as functional / non – functional requirements for a banking system [CO2,K3]
   i) verifying bank balance
   ii) withdrawing money from bank
   iii) completion of transactions less than one second
   iv) extending the system by providing more tellers for the customers
4. Differentiate between normal and exciting requirements. [CO2,K2]
5. With an example, find the difference between aggregation and composition. [CO3,K2]
6. Show the uses and extends relationship in any example use case diagram. [CO3,K2]
7. If a module has logical cohesion, what kind of coupling is this module likely to have? [CO4,K3]
8. Define patterns. Why it is required for designers? [CO4,K1]
9. Why does software fail after is has passed from acceptance testing? [CO5,K1]
10. When and why regressions test to be completed? [CO5,K1]

Part – B (5 × 16 = 80 marks)

11. a. Suppose you are asked to create an e – commerce web site. To satisfy the customer, you can develop some sample web page of the shopping site such as catalogue page, product order page etc., and present it to the customer for approval. If the customer approves it, requirements are stated again and the design of the web site is initiated. If the customer does not approve the web site, the development team revisits the sample pages and resubmits it to the customer's approval. This process continues until it is approved. Which model is suitable for doing this project? Explain the process model with proper reasons. (16) [CO1,K3]

(OR)

b. Assume that you are the technical manager of a software development organization. A client approached you for a software solution, the problems stated by the client have uncertainties, which lead to loss if not planned and solved properly. Which software model you will suggest for this project? Justify your answer with its pros and cons and neat sketch. (16) [CO1,K3]

12. a. Elaborate the seven distinct tasks which are used in managing the requirements. (16) [CO2,K2]

(OR)

b. Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirement engineering process. (16) [CO2,K2]

13. a. Develop an online railway reservation system, which allows the user to select route, book/cancel tickets using net banking/UPI/credit/debit cards. The site also maintains the history of the passengers for the above system. Model a use case diagram and class diagram. (16) [CO3,K3]

(OR)

b. Develop a sequence, collaboration and component diagram for a pizza ordering system. State the rules that you are taking consideration. Design the system with your own assumptions. (16) [CO3,K3]

14. a. With neat sketch, elaborate the architectural styles and describe the system category like set of components, connectors, constrains and semantic models. (16) [CO4,K2]

(OR)

b. With an example scenario, give brief note on Architectural considerations and Architectural Trad-off analysis. (16) [CO4,K2]

15. a. Given set of 'n' integer numbers, the function findprime(a[ ], n) prints a number if it is a prime number. Draw a control flow graph, calculate the cyclomate complexity and enumerate all the paths. How many test cases are needed to adequately cover the code in terms of branches, decisions and statement. (16) [CO5,K3]

(OR)

b. Confer the various activities involved in an SCM process. Explore how these requirements for SCM are satisfied. (16) [CO5,K3]

Bloom's Taxonomy Level	Remembering (K1)	Understanding (K2)	Applying (K3)	Analysing (K4)	Evaluating (K5)	Creating (K6)
Percentage	5	40	55	-	-	-

BTech Degree Examination November 2021

Fifth Semester

Information Technology

18ITT53 – SOFTWARE ENGINEERING

(Regulations 2018)

ANSWER KEY

PART A

1. IEEE definition of software engineering:

   The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

2. Reasons for developing prototype:
   - Adjust the design
   - Adjust colors, textures and shapes
   - Provides quality assurance
   - Measures acceptable tolerance level
   - Know the true cost of production

3. Classify the following as functional / non – functional requirements for a banking system

   i) verifying bank balance – Functional

   ii) withdrawing money from bank - Functional

   iii) completion of transactions less than one second – Non - Functional

   iv) extending the system by providing more tellers for the customers - Non - Functional

4. Differentiate between normal and exciting requirements:

   Normal requirements –

   In this the objective and goals of the proposed software are discussed with the customer.

   Example – normal requirements for a result management system may be entry of marks, calculation of results, etc.

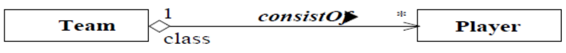
   Exciting requirements –

   It includes features that are beyond customer's expectations and prove to be very satisfying when present.

   Example – when unauthorized access is detected, it should backup and shutdown all processes.

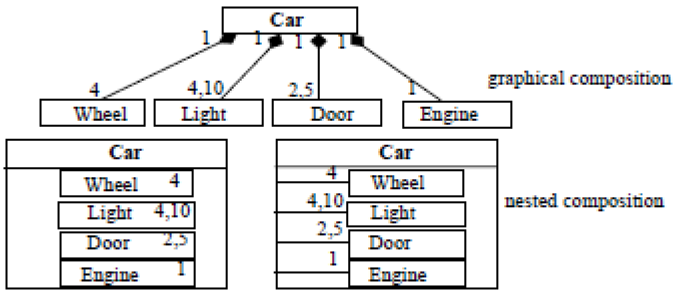
5. Difference between aggregation and composition:

   Aggregation:
   - A part-of relationship. A form of association. Ex:

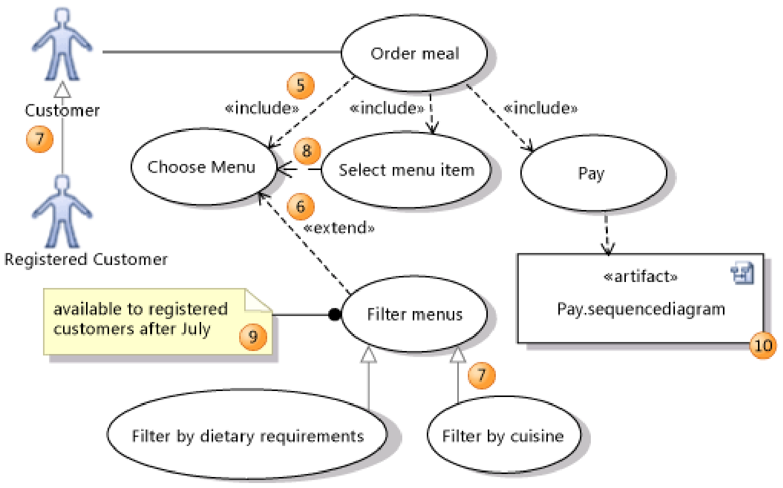


Composition:

- A part-whole relationship.
- Strong form of aggregation. Ex:



6. The uses and extends relationship in Use Case Diagram:



7. If a module has logical cohesion, what kind of coupling is this module likely to have?

Cohesion:

Def: Elements of component are related logically and not functionally.

Several logically related elements are in the same component and one of the elements is selected by the client component.

- Loosely coupled data formats or External coupling is suitable for this kind of cohesion.

8. Define patterns. Why it is required for designers?

A design pattern is an abstraction that prescribes a design solution to a specific, well-bounded design problem.

9. Why does software fail after is has passed from acceptance testing?

An acceptance test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end user. Client accepts the software only when all the features and functionalities work as expected.

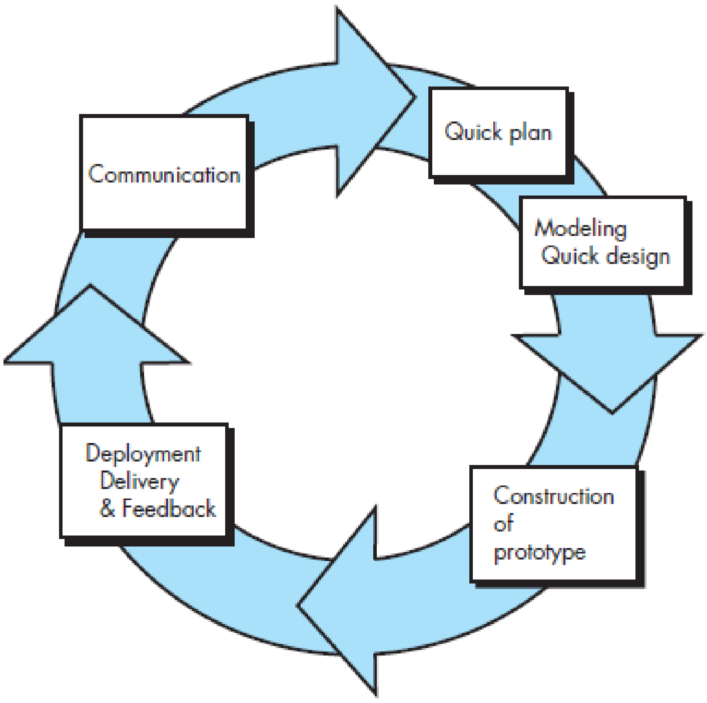
In 90% of cases where acceptance testing fails, it is because of what's been done before even starting it.

10. When and why regressions test to be completed?

Regression testing plays a key role in putting the existing and updated functionality in order, so it is important to know when to do regression testing in Agile. Regression testing is performed at the end of every sprint to make sure your software is stable and sustainable.

PART B

11.**a.** Suitable process model for the given scenario is **Prototyping Model.** **(3)**



(3)

(7)

- The prototyping, serves as a mechanism for identifying software requirements.
- Begins with communication, identify, whatever requirements are known,

1. Requirements gathering and analysis: A prototyping model begins with requirements analysis and the requirements of the system are defined in detail. The user is interviewed in order to know the requirements of the system.

2. Quick design: When requirements are known, a preliminary design or quick design for the system is created. It is not a detailed design and includes only the important aspects of the system, which gives an idea of the system to the user. A quick design helps in developing the prototype.
3. Build prototype: Information gathered from quick design is modified to form the first prototype, which represents the working model of the required system.

4. User evaluation: Next, the proposed system is presented to the user for thorough evaluation of the prototype to recognize its strengths and weaknesses such as what is to be added or removed. Comments and suggestions are collected from the users and provided to the developer.
5. Refining prototype: Once the user evaluates the prototype and if he is not satisfied, the current prototype is refined according to the requirements. That is, a new prototype is developed with the additional information provided by the user. The new prototype is evaluated just like the previous prototype. This process continues until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed on the basis of the final prototype.

6. Engineer product: Once the requirements are completely met, the user accepts the final prototype. The final system is evaluated thoroughly followed by the routine maintenance on regular basis for preventing large-scale failures and minimizing downtime.

Disadvantages: **(3)**

- ▶ Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together
- ▶ As a software engineer, you often make implementation compromises in order to get a prototype working quickly
- ▶ An inappropriate operating system or programming language may be used simply

11.b. **For the given scenario, Agile Model will be suitable**. **(2)**

- Agility process is driven by customer descriptions of what is required (scenarios). Some assumptions: **(4)**
  - ➢ Recognizes that plans are short-lived (some requirements will persist, some will change. Customer priorities will change)
  - ➢ Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created.)!
  - ➢ Analysis, design, construction and testing are not predictable. !
- Thus has to Adapt as changes occur due to unpredictability!
- Delivers multiple 'software increments', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

Agility Principles: **(10)**

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

12.a. **The seven distinct tasks which are used in managing the requirements are:**

- **Inception**—ask a set of questions that establish … **(3)**
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- **Elicitation**—elicit requirements from all stakeholders **(2)**
  - Problem of scope
  - Problem of understanding
  - Problem of volatility
- **Elaboration**—create an analysis model that identifies data, function and behavioral requirements **(2)**
- **Negotiation**—agree on a deliverable system that is realistic for developers and customers **(2)**
- **Specification**—can be any one (or more) of the following: **(2)**
  - A written document
  - A set of models
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A prototype **(3)**
- **Validation**—a review mechanism that looks for
  - Errors in content or interpretation
  - Areas where clarification may be required
  - Missing information
  - Inconsistencies (a major problem when large products or systems are engineered)
  - Conflicting or unrealistic (unachievable) requirements.
- **Requirements management** **(2)**

12.b. **Distinction between developing the user requirements and developing system requirements:**

(6)

User Requirements and System Requirements

Generally the requirements engineering process follows below activities:

• Feasibility Study

• Requirements elicitation and analysis

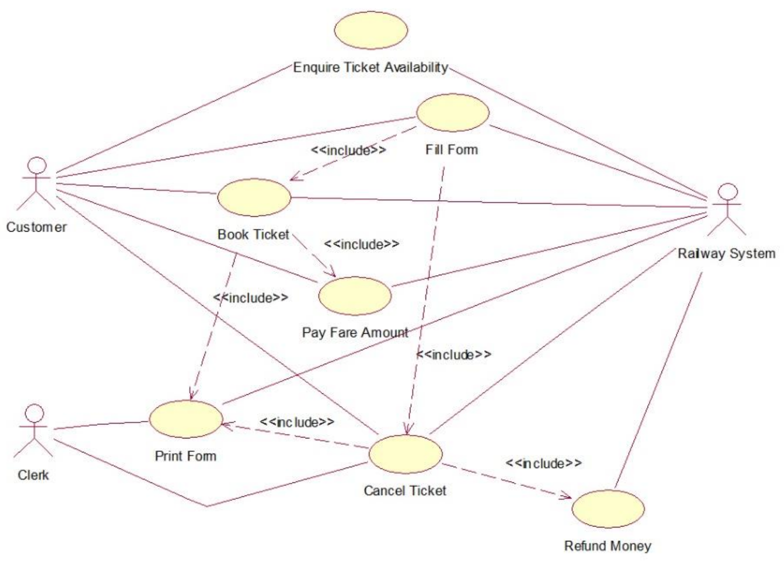
• Requirements specification

• Requirements validation

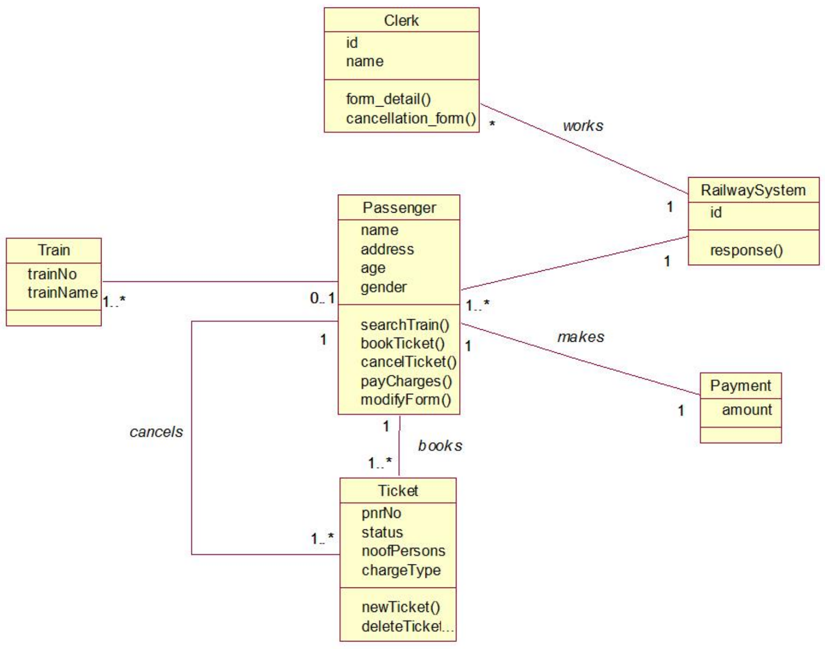
In requirements specification contains user requirements and system requirements. The distinction between developing the user requirements and system requirements in the requirement engineering process as follows: **(6)**

- Developing the user requirements are abstract statements. It's described the system's functions and features of customer needs.
- While system requirements are provided a more detailed explanation of the procedure.
- The user requirements must be develop in accepted language and might not be communicated in depth detail to tolerate certain execution flexibility.
- While the development of system requirements are much more detailed than the user requirements.
- In user requirements, the starting level of the project, development team may put interest on gathering good information, identifying the important things and understanding them correctly.
- In system requirements, are implemented after user requirements have been established. **(4)**
- The development of system requirement is important from the beginning of a project to end and beyond programmers must have the knowledge and skill to write program applications.

13.a. Use case diagram and Class diagram for **online railway reservation system:**



(7)



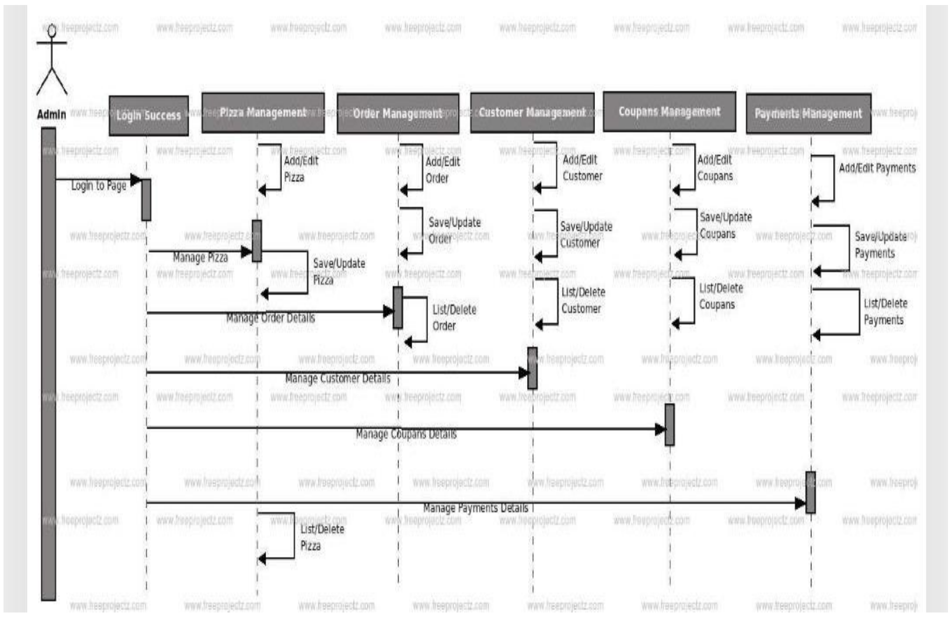
(9)

(Diagram may vary. Students can draw the diagram with their own perspectives) *

13.b. A sequence, collaboration and component diagram for a **pizza ordering system.**

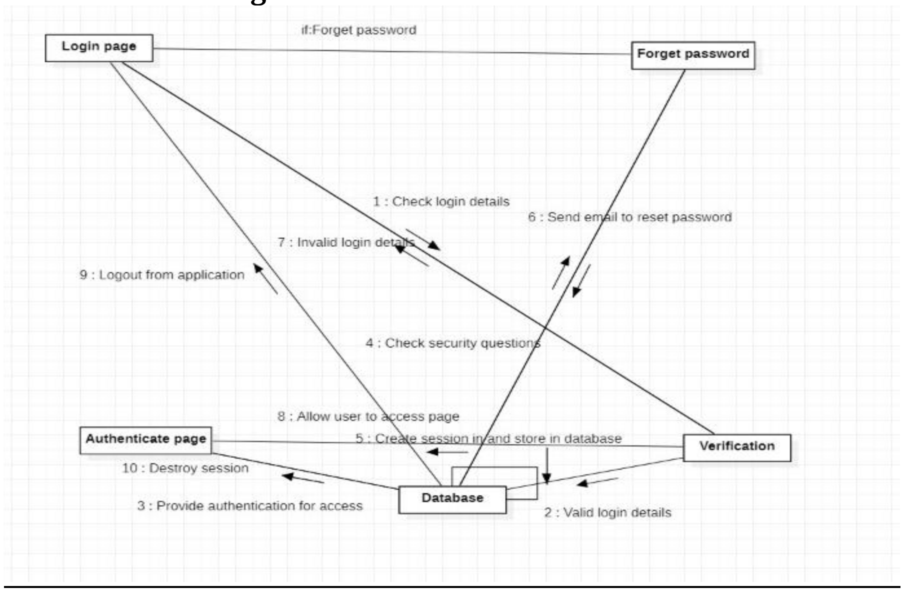
- User can frame their own rules based on their assumptions.

Sequence Diagram:



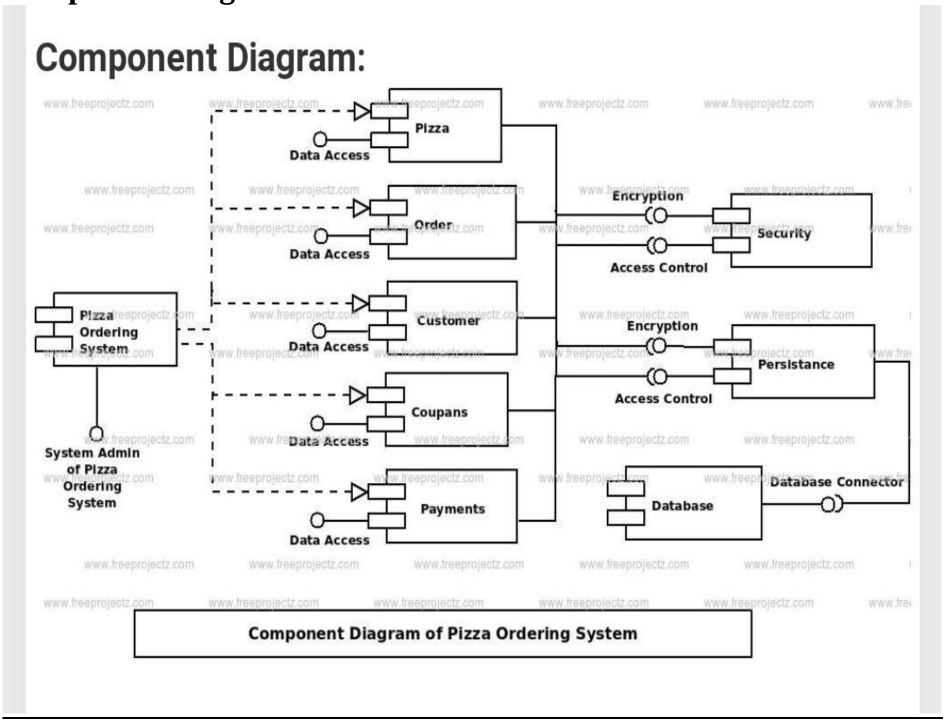
(6)

Collaboration Diagram:


if:Forget password
Login page
Forget password
1 : Check login details
6 : Send email to reset password
7 : Invalid login details
9 : Logout from application
4 : Check security questions
8 : Allow user to access page
Authenticate page
5 : Create session in and store in database
Verification
10 : Destroy session
Database
3 : Provide authentication for access
2 : Valid login details

(4)

Component Diagram:



(6)

(Diagram may vary. Students can draw the diagram with their own perspectives) *

14.a. **Architectural styles:**

(4)

Each style describes a system category that encompasses:

(1) A set of components (e.g., a database, computational modules) that perform a function required by a system,

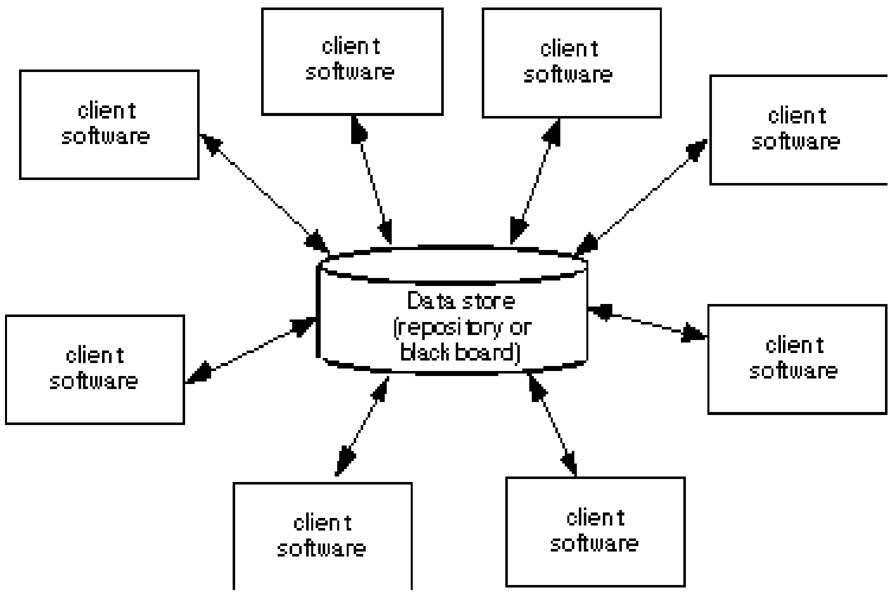
(2) A set of connectors that enable "communication, coordination and cooperation" among components,

(3) Constraints that define how components can be integrated to form the system, and

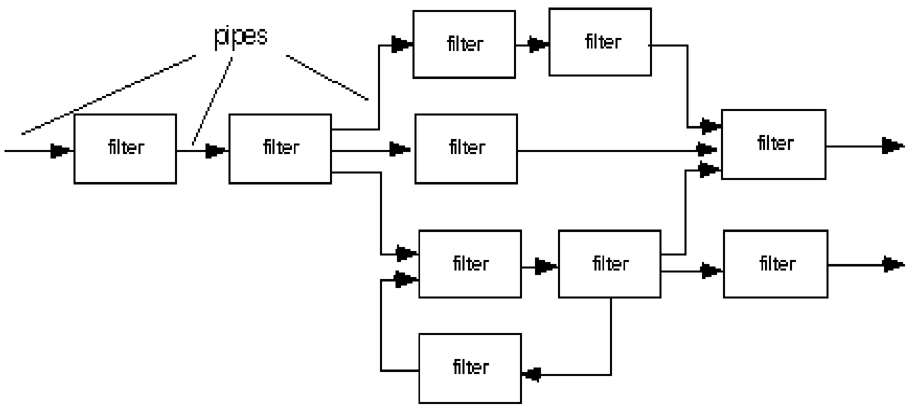
(4) Semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

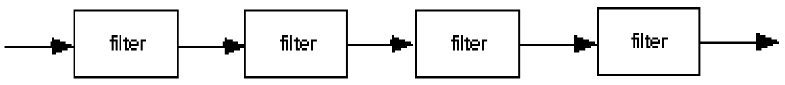
Data-centered architectures: (diagram with explanation) **(3)**



Data flow architectures: (diagram with explanation) **(3)**

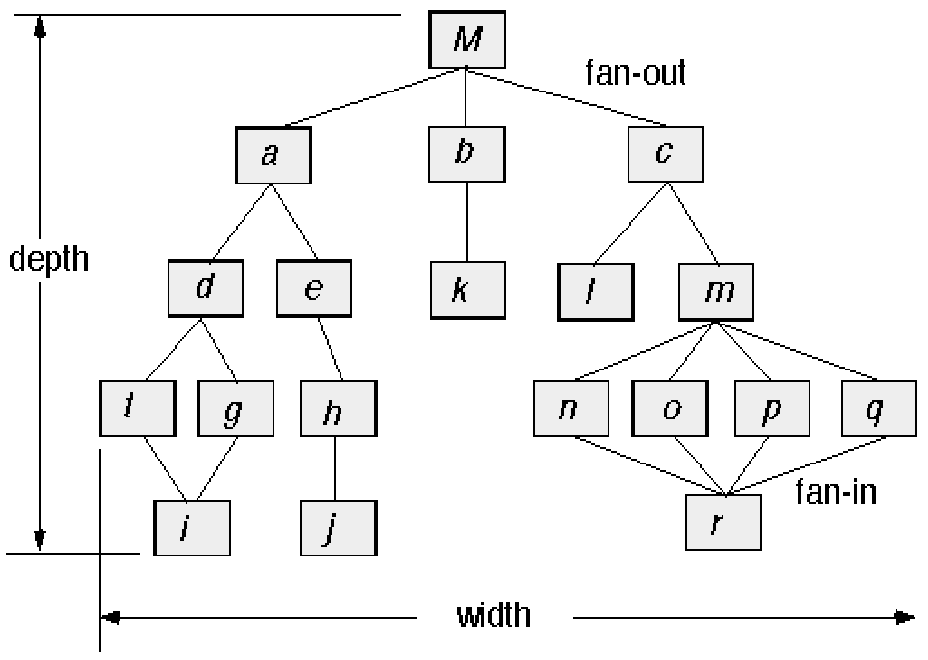


(a) pipes and filters



(b) batch sequential

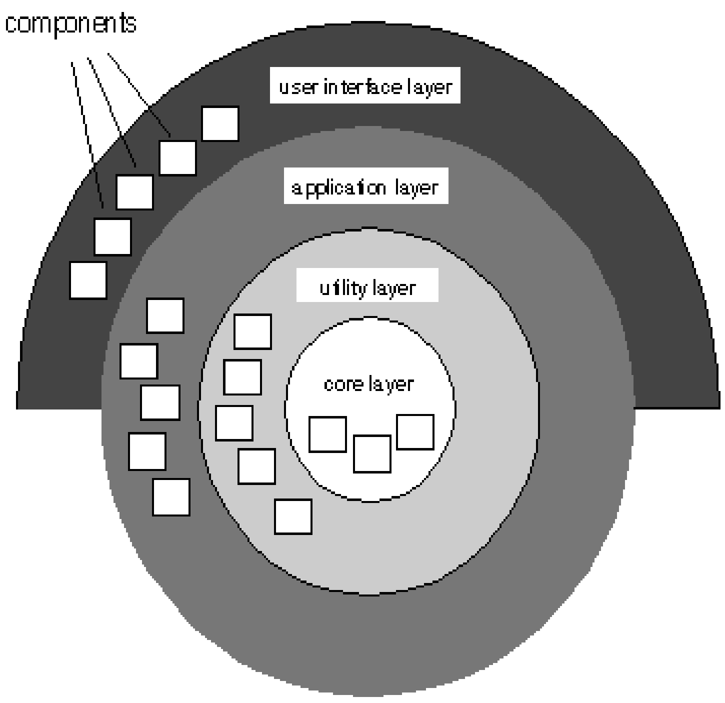
Call and Return Architecture: (diagram with explanation)



(3)

(3)

Layered Architecture: (diagram with explanation)



14.b. **Architectural considerations and Architectural Trade-off analysis:**

Architectural considerations: (6)

Economy – The best software is uncluttered and relies on abstraction to reduce unnecessary detail.
Visibility – Architectural decisions and the reasons for them should be obvious to software engineers who examine the model at a later time.
Spacing – Separation of concerns in a design without introducing hidden dependencies.

Symmetry – Architectural symmetry implies that a system is consistent and balanced in its attributes.
Emergence – Emergent, self-organized behavior and control.

Architectural Trade-off analysis:

(6)

1. Collect scenarios.
2. Elicit requirements, constraints, and environment description.
3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements:
    - module view
    - process view
    - data flow view
4. Evaluate quality attributes by considered each attribute in isolation.
5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.
6. Critique candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

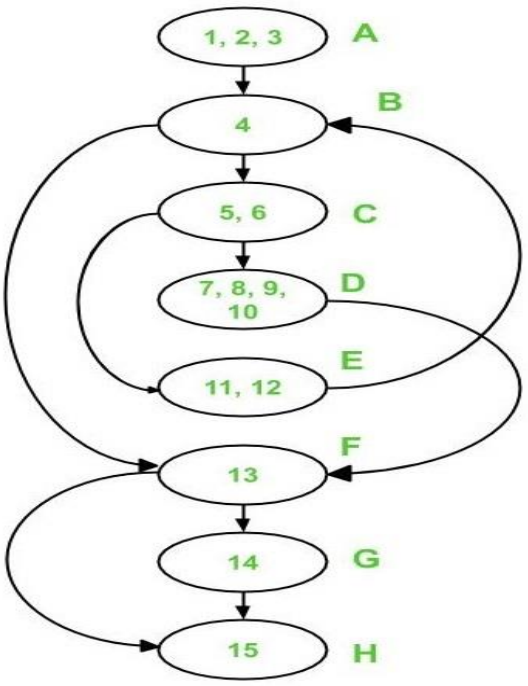
For example scenario and its explanation - **(4 Marks)** **(4)**

15.a. **Control Flow Graph:**

(2)

```
int main()
{
   int n, index;
1    cout << "Enter a number: " <> n;
3    index = 2;
4    while (index <= n - 1)
5    {
6       if (n % index == 0)
7       {
8          cout << "It is not a prime number" << endl;
9          break;
10      }
11      index++;
12   }
13   if (index == n)
14      cout << "It is a prime number" << endl;
15 } // end main
```

(5)



(6)

Cyclomatic Complexity Calculation:
Method-1:
V(G) = e - n + 2*p
In the above control flow graph,
where, e = 10, n = 8 and p = 1

Therefore,
Cyclomatic Complexity V(G)
= 10 - 8 + 2 * 1
= 4
Method-2:
V(G) = d + p
In the above control flow graph,
where, d = 3 (Node B, C and F) and p = 1

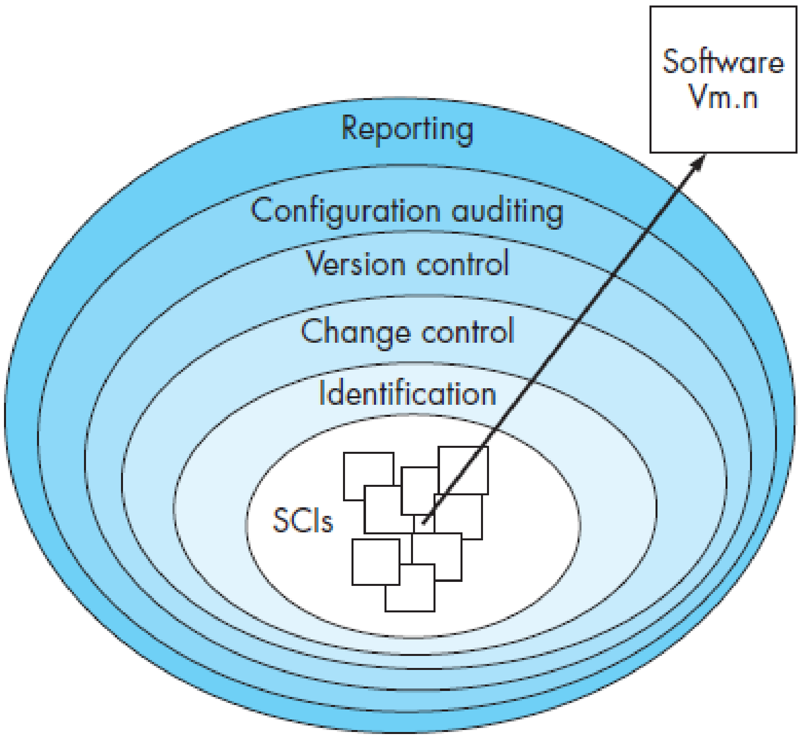
Therefore,
Cyclomatic Complexity V(G)
= 3 + 1
= 4
Method-3:
V(G) = Number of Regions
In the above control flow graph, there are 4 regions : R1, R2, R3 and R4

(3)

Cyclomatic Complexity V(G)
= 1 + 1 + 1 + 1
= 4

Test cases :
To derive test cases, we have to use the independent paths obtained previously. To design a test case, provide input to the program such that each independent path is executed.
For the given program, **4 test cases are required.**

15.b. **Various activities involved in an SCM process:**



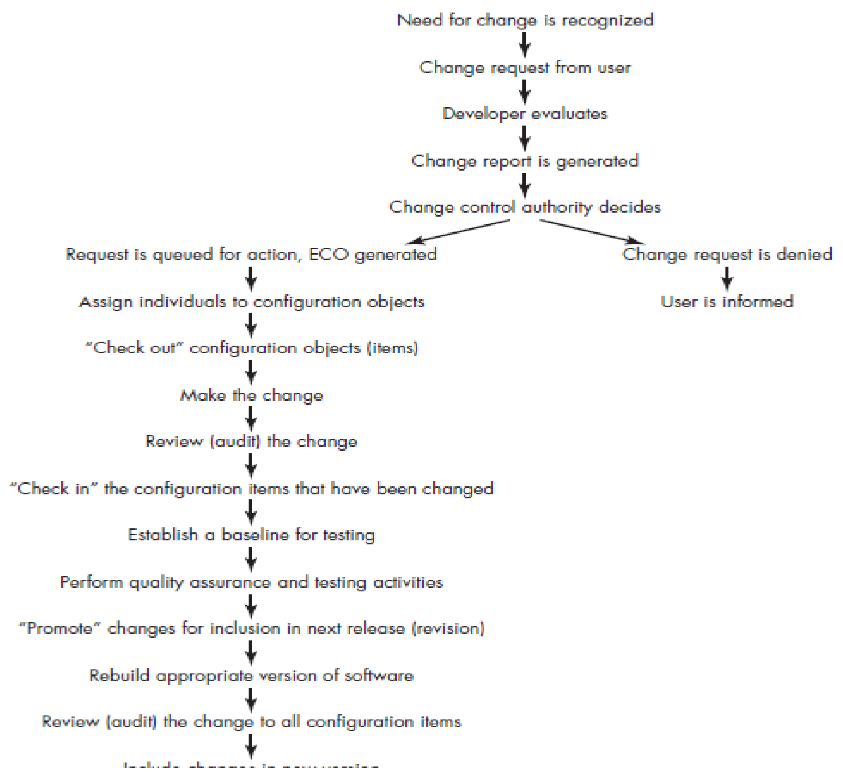
(3)

(4)

Version Control:

- Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process
- A version control system implements or is directly integrated with four major capabilities:
- a project database (repository) that stores all relevant configuration objects
- a version management capability that stores all versions of a configuration object (or enables any version to be constructed using differences from past versions);
- a make facility that enables the software engineer to collect all relevant configuration objects and construct a specific version of the software.
- an issues tracking (also called bug tracking) capability that enables the team to record and track the status of all outstanding issues associated with each configuration object.

Change Control:


Need for change is recognized
Change request from user
Developer evaluates
Change report is generated
Change control authority decides
Request is queued for action, ECO generated
Change request is denied
Assign individuals to configuration objects
User is informed
"Check out" configuration objects (items)
Make the change
Review (audit) the change
"Check in" the configuration items that have been changed
Establish a baseline for testing
Perform quality assurance and testing activities
"Promote" changes for inclusion in next release (revision)
Rebuild appropriate version of software
Review (audit) the change to all configuration items

(3)

Configuration Audit:

- A software configuration audit complements the technical review by assessing a configuration object for characteristics that are generally not considered during review.
- The audit asks and answers the following questions:
- Has the change specified in the ECO been made? Have any additional modifications been incorporated?
- Has a technical review been conducted to assess technical correctness?
- Has the software process been followed and have software engineering standards been properly applied?
- Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
- Have SCM procedures for noting the change, recording it, and reporting it been followed?
- Have all related SCIs been properly updated?

(3)

Status Reporting:

- *Configuration status reporting* (sometimes called *status accounting*) is an SCM task that answers the following questions:
  (1) What happened?
  (2) Who did it?
  (3) When did it happen?
  (4) What else will be affected?
- Each time a configuration audit is conducted, the results are reported as part of the CSR task
- Output from CSR may be placed in an online database or website, so that software developers or support staff can access change information by keyword category

(3)