20CB503/22CM504/PE1/20CS503/20DS503/20IT503

Hall Ticket Number:

III B.Tech (Regular\Supplementary) DEGREE EXAMINATION

December, 2024 **Common to CB, CM, CS, DS, & IT**

Fifth Semester **Software Engineering**

Time: Three Hours **Maximum:** 70 Marks

Answer question 1 compulsorily. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56Marks)**

			CO	BL	M
1	a)	Outline the characteristics of software engineering.	CO1	L3	1M
	b)	Incremental model is the combination of two different process models. What are those two models?	CO1	L3	1M
	c)	Define CMMI.	CO1	L2	1M
	d)	List the advantages of using Scrum process model.	CO1	L2	1M
	e)	Differentiate between functional requirements and non-functional requirements?	CO2	L3	1M
	f)	What is the purpose of use case diagram?	CO2	L2	1M
	g)	What is meant by refactoring?	CO2	L2	1M
	h)	Write short notes on CRC cards?	CO3	L2	1M
	i)	Define Cohesion and Coupling.	CO3	L2	1M
	j)	Differentiate Process and Product	CO3	L3	1M
	k)	Compare project risk and business risk.	CO3	L3	1M
	l)	What are basic principles of software testing?	CO4	L2	1M
	m)	Differentiate error, bug and defect.	CO4	L3	1M
	n)	Give the importance of Software Quality Assurance.	CO4	L3	1M
		Unit-I			
2	a)	Explain software engineering as a layered technology.	CO1	L2	4M
	b)	Illustrate spiral model with neat diagram.	CO1	L3	10M
		(OR)			
3	a)	What is software? Explain the essential attributes of good software.	CO1	L3	4M
	b)	Explain water fall model.	CO1	L4	10M
		Unit-II			
4	a)	What is Agility in context of software engineering? Explain extreme programming (XP) with suitable diagram?	CO2	L3	7M
	b)	What is the purpose of requirements elicitation? Who are the different stakeholders involved in requirements elicitation?	CO2	L4	7M
		(OR)			
5	a)	How to perform the process of requirement analysis and negotiation? Explain requirements specification.	CO2	L3	7M
	b)	Draw the complete DFD at least up to two levels (Level 0, Level 1) of library system.	CO2	L2	7M
		Unit-III			
6	a)	Explain in detail about the various design concepts.	CO3	L2	7M
	b)	List out the golden rules for User interface design.	CO3	L3	7M
		(OR)			
7	a)	What is object oriented design process? why it is so important for software development.	CO3	L3	7M
	b)	Explain about taxonomy of Architecture styles and patterns?	CO3	L4	7M
		Unit-IV			
8	a)	Discuss different types of integration testing.	CO4	L3	7M
	b)	What are the various testing strategies (White box and Black box) for software testing? Discuss them briefly.	CO4	L2	7M
		(OR)			
9	a)	What is cyclomatic complexity? Explain with an example of how to construct a flow graph for a program (Fibonacci series) and compute cyclomatic complexity.	CO4	L4	7M
	b)	Explain the Test Strategies for Object-Oriented Software.	CO4	L3	7M

20CB503/22CM504/PE1/20CS503/20DS503/20IT503

Hall Ticket Number:

III B.Tech (Regular\Supplementary) DEGREE EXAMINATION

December, 2024 — **Common to CB, CM, CS, DS, & IT**

Fifth Semester — **Software Engineering**

Time: Three Hours — **Maximum:**70 Marks

Answer question 1 compulsorily. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56Marks)**

			CO	BL	M
1	a)	Outline the characteristics of software engineering. Functionality, usability, efficiency, flexibility, reliability, maintainability, portability, and integrity are key characteristics that software engineers should consider throughout the development lifecycle	CO1	L3	1M
	b)	Incremental model is the combination of two different process models. What are those two models? The incremental model is a combination of the Waterfall model and the RAD model:	CO1	L3	1M
	c)	Define CMMI. CMMI, or Capability Maturity Model Integration, is a framework that helps organizations improve their performance and capabilities	CO1	L2	1M
	d)	List the advantages of using Scrum process model. Scrum can help teams complete project deliverables quickly and efficiently. Scrum ensures effective use of time and money. Large projects are divided into easily manageable sprints. Developments are coded and tested during the sprint review. Works well for fast-moving development projects.	CO1	L2	1M
	e)	Differentiate between functional requirements and non-functional requirements? functional requirements define the specific behavior or functions of a system—what the system should do to meet user needs. These include features like data processing, authentication, and user interactions. In contrast, non-functional requirements specify how the system performs its tasks, focusing on attributes like performance, security, scalability, and usability.	CO2	L3	1M
	f)	What is the purpose of use case diagram? The purpose of a use case diagram is to provide a clear overview of how a system works and how users interact with it. Use case diagrams are a part of Unified Modeling Language (UML) and are often used in the early stages of a project.	CO2	L2	1M
	g)	What is meant by refactoring? Refactoring is a software design and computer programming technique that improves the structure of existing code without changing how the software behaves.	CO2	L2	1M
	h)	Write short notes on CRC cards? A Class-Responsibility-Collaboration (CRC) card is a structured tool used in object-oriented software design.	CO3	L2	1M
	i)	Define Cohesion and Coupling. Coupling refers to the interdependencies between modules, while cohesion describes how related the functions within a single module are.	CO3	L2	1M
	j)	Differentiate Process and Product The main difference between a process and a product is that the process is a set of steps hat guide the project to achieve a convenient product. while on the other hand, the roduct is the result of a project that is manufactured by a wide variety of people.	CO3	L3	1M
	k)	Compare project risk and business risk. Project risks are specific to a project's outcomes, while business risks can affect an entire organization	CO3	L3	1M
	l)	What are basic principles of software testing? 1) Exhaustive testing is not possible 2) Defect Clustering 3) Pesticide Paradox 4) Testing shows presence of defects 5) Absence of Error – fallacy 6) Early Testing 7) Testing is context dependent	CO4	L2	1M
	m)	Differentiate error, bug and defect. In software development, an error is a mistake made by a programmer, a bug is an error found during testing, and a defect is a mismatch between expected and actual results	CO4	L3	1M
	n)	Give the importance of Software Quality Assurance. Software Quality Assurance (SQA) is the practice of monitoring all software engineering processes, activities, and methods used in a project to ensure proper quality of the softwa conformance against the defined standards.	CO4	L3	1M

Unit-I

2 a) Explain software engineering as a layered technology. CO1 L2 4M

Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

Just as software engineering requires progressing through interconnected layers to build robust software, advancing your skills in software testing also involves a step-by-step approach. To effectively move from basic testing to more complex automation, consider exploring the Complete Guide to Software Testing & Automation by GeeksforGeeks . This course will help you build on each layer of your knowledge, ensuring you master the intricacies of testing and automation to create reliable, high-quality software.

Layered technology is divided into four parts:
1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.
Process activities are listed below:-

Communication: It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
Planning: It basically means drawing a map for reduced the complication of development.
Modeling: In this process, a model is created according to the client for better understanding.
Construction: It includes the coding and testing of the problem.
Deployment:- It includes the delivery of software to the client for evaluation and feedback.
3. Method: During the process of software development the answers to all “how-to-do” questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

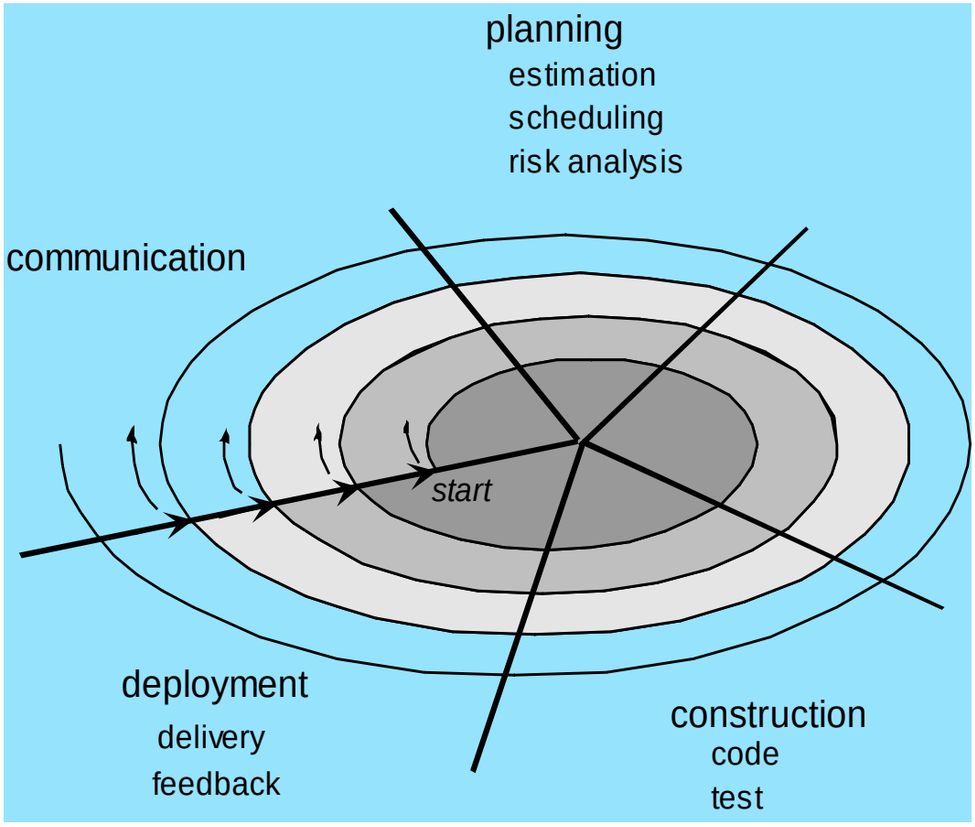
4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

b) Illustrate spiral model with neat diagram. CO1 L3 10M

The Spiral Model is one of the most important Software Development Life Cycle models. The Spiral Model is a combination of the waterfall model and the iterative model. It provides support for Risk Handling. The Spiral Model was first proposed by Barry Boehm.
The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a phase of the software development process.

Some Key Points regarding the phase of a Spiral Model:

The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
As the project manager dynamically determines the number of phases, the project manager has an important role in developing a product using the spiral model.
It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.



(OR)

3	a)	What is software? Explain the essential attributes of good software.	CO1	L3	4M

Software is a collection of instructions, data, and programs that allow a computer to perform specific tasks. Some essential attributes of good software include:
Functionality: The software's ability to perform as designed
Usability: How easy the software is to use, including its effectiveness, efficiency, and user satisfaction
Maintainability: How easy it is to modify, debug, and enhance the software throughout its lifecycle
Reliability: The software's ability to perform well under specified conditions for a specified time
Scalability: The software's ability to scale
Security: The software's security
Compatibility: The software's compatibility
Correctness: The software's correctness
Flexibility: The software's flexibility
Testability: The software's testability
Software is the opposite of hardware, which refers to the physical aspects of a computer.

	b)	Explain water fall model.	CO1	L4	10M

It is a simplest model, which states that the phases are organized in a linear order.
The model was originally proposed by Royce.
The various phases in this model are
Feasibility Study –
The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product.
The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behavior of the system.
Requirement Analysis
The aim of the requirement analysis is to understand the exact requirements of the customer and to document them properly.
The requirement analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions.
During this activity, the user requirements are systematically organized into a software requirement specification (SRS) document.
The important components of this document are the functional requirements, the nonfunctional requirements, and the goals of implementation.
The SRS document is written in end-user terminology. This makes the SRS document understandable by the customer.
After all, it is important that the SRS document be reviewed and approved by the customer.
Design This phase is concerned with
Identifying software components like functions, data streams and data stores.
Specifying software structure.
Maintaining a record of design decisions and providing blue prints for the implementation phase .
There are two categories of Design

Architectural Design – It involves
Identifying the software components.
Decoupling and decomposing the software components into modules and conceptual data structures.
Specifying the interconnection between the various components.
Detailed Design – It is concerned with details of the implementation procedures to process the algorithms, data structures and interaction between the modules and data structures.
The various activities that this phase includes are
Adaptation of existing code.
Modification of existing algorithms.
Design of data representation
Packaging of the software product.
Implementation – It involves the translation of the design specifications into source code.
It also involves activities like debugging, documentation and unit testing of the source code.
In this stage various styles of programming can be followed like built-in and user defined data types, secure type checking, flexible scope rules, exception handling, concurrency control etc.
Testing – It involves two kinds of activities
Integration Testing – It involves testing of integrating various modules and testing there overall performance due to their integration.
Acceptance Testing – It involves planning and execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements stated in the requirements document.
Maintenance – In this phase the activities include
Corrective Maintenance – Correcting errors that were not discovered during the product development phase.
Perfective Maintenance – Improving the implementation of the system, and enhancing the functionalities of the system according to customer's requirements.
Adaptive Maintenance – Adaptation of software to new processing environments.

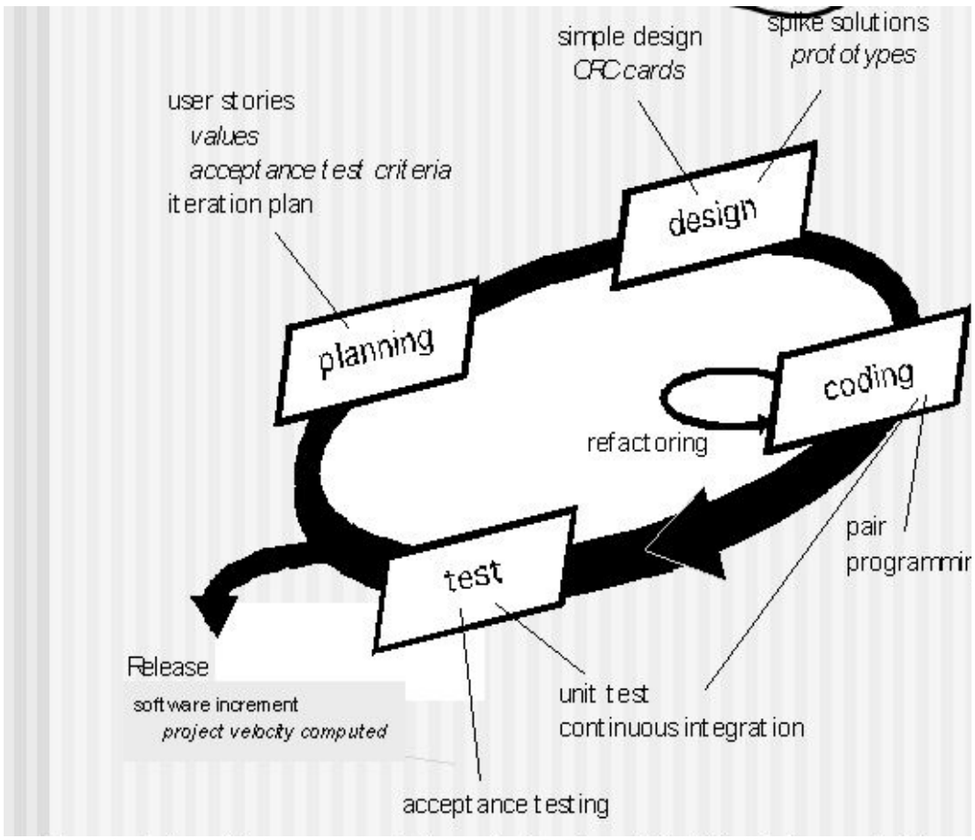
Unit-II

4	a)	What is Agility in context of software engineering? Explain extreme programming (XP) with suitable diagram?	CO2	L3	7M

Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods.
These methods:
Focus on the code rather than the design.
Are based on an iterative approach to software development.
Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.
What is "Agility"?
Effective (rapid and adaptive) response to change
Effective communication among all stakeholders
Drawing the customer onto the team
Organizing a team so that it is in control of the work performed
Yielding …
Rapid, incremental delivery of software

The most widely used agile process, originally proposed by Kent Beck
XP Planning
Begins with the creation of "user stories"
Agile team assesses each story and assigns a cost
Stories are grouped to for a deliverable increment
A commitment is made on delivery date
After the first increment "project velocity" is used to help define subsequent delivery dates for other increments
XP Design
Follows the KIS(Keep It Simple) principle
Encourage the use of CRC(class responsibility collaborator) cards
For difficult design problems, suggests the creation of "spike solutions"—a design prototype
Encourages "refactoring"—an iterative refinement of the internal program design

- XP Coding
  - Recommends the construction of a unit test for a store *before* coding commences
  - Encourages "pair programming"
- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and excuted to assess customer visible functionality



b) What is the purpose of requirements elicitation? Who are the different stakeholders involved in requirements elicitation? CO2 L4 7M

RE provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.

The requirements engineering process is accomplished through the execution of seven distinct functions :

Inception,
Elication,
Elaboration,
Specification,
Negotiation,
Validation,
Management
Elicitation

Christel and Kang identify a number of problems that help us understand why requirements elicitation is difficult:

Problems of scope. The boundary of the system is ill-defined.

Problems of understanding. The customers/users are not completely sure of what is needed, have poor understanding of the capabilities and limitations of their computing environment, that are ambiguous or unstable.

Problems of volatility. The requirements change over time.

Identifying the Stakeholders :

A stakeholder is anyone who benefits in a direct or indirect way from the system which is being developed.

stakeholders are: operations managers, product managers, marketing people, internal and external customers, end-users, consultants, product engineers, software engineers, support and maintenance engineers etc.

At inception, the requirements engineer should create a list of people who will contribute input as requirements are elicited.

Recognizing Multiple Viewpoints :

Each of the stakeholders will contribute to the RE process.

As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.

The requirements engineer is to categorize all stakeholder information including inconsistencies and conflicting requirements in a way that will allow decision makers to choose an internally inconsistent set of requirements for the system.

(OR)

5 a) How to perform the process of requirement analysis and negotiation? Explain requirements specification. CO2 L3 7M

Requirement analysis is typically a procedure of analyzing, validating, and aligning the requirements documented during the phase of Requirement Elicitation. In other words, requirement analysis is a process of studying and understanding the requirements stated by the stakeholders. Requirement analysis requires frequent communication with the stakeholders and end-users in order to define the expectations, solve the conflicts, and finally, document the key requirements. The solutions may involve issues like:

Different kinds of set-ups for the workflow in the company

Setting up a new system that is to be used from now onwards, etc.
One thing to be kept in mind is that Requirement Elicitation and Requirement Analysis work together. They two feed each other. When we start gathering the requirements, we elicitate them and analyze them at the same time as well.
The first and foremost objective of requirement analysis is to understand the requirements and needs of the users
When we use different sources to gather the requirements, there may be some conflicts between them. Requirement Analysis is about finding those conflicts among the requirements stated by the users and resolving them.
Negotiate the requirements with the users and stakeholders. There is no way our system can meet all the requirements in the exact way they are explained by the stakeholders and users.
We will have to negotiate and prioritize the requirements. Some requirements may not be big for us but they can be pretty important for the end-users. To understand them, we have to analyze and prioritize the requirements of the stakeholders.
We must elaborate on the requirements stated by the users and system. This helps while documenting the requirements in the requirement specifications. Also, this helps the developers develop, design, and test better as they understand the requirements in an elaborated and better way.
We must classify the requirements into various different categories and sub-categories and further allocate those requirements to different sub-systems.
We must also evaluate the requirements for the quality that is desired by the organization.

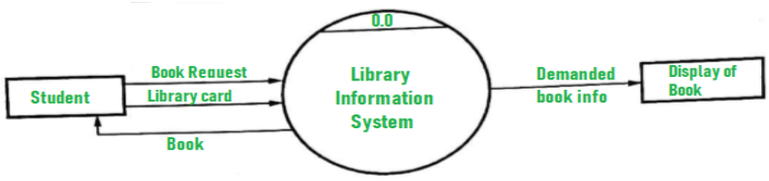
b) Draw the complete DFD at least up to two levels (Level 0, Level 1) of library system. CO2 L2 7M

Data Flow Diagram (DFD) depicts the flow of information and the transformation applied when data moves in and out of a system. The overall system is represented and described using input, processing, and output in the DFD. The inputs can be:

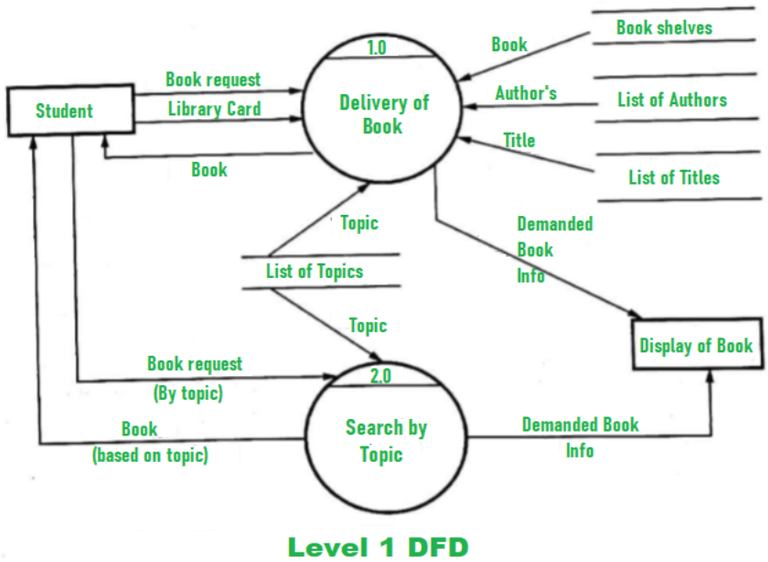
Book request when a student requests for a book.
Library card when the student has to show or submit his/her identity as proof.
The overall processing unit will contain the following output that a system will produce or generate:

The book will be the output as the book demanded by the students will be given to them. Information on the demanded book should be displayed by the library information system that can be used by the student while selecting the book which makes it easier for the student.
To master DFDs and create effective models for your systems, the System Design Course provides detailed instructions on data flow diagramming and practical examples.

Level 0 DFD –



Level 1 DFD – At this level, the system has to show or exposed with more details of processing. The processes that are important to be carried out are:
Book delivery
Search by topic
List of authors, List of Titles, List of Topics, the bookshelves from which books can be located are some information that is required for these processes. Data store is used to represent this type of information



Level 1 DFD

Unit-III

6 a) Explain in detail about the various design concepts. CO3 L2 7M

Abstraction

Refinement
Modularity
Architecture
Patterns
Refactoring
Functional Independence
Information Hiding
OO Design Concepts
Abstraction
At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment.
At lower levels of abstraction, a more detailed description of the solution is provided.
As we move through different levels of abstraction, we work to create procedural and data abstractions.
A procedural abstraction refers to a sequence of instructions that have a specific and limited function
A data abstraction is a named collection of data that describes a data object.
Architecture
Software architecture alludes to the "overall structure of the software and the ways in which the structure provides conceptual integrity for a system."
In its simplest form, architecture is the structure of organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components.
The goal of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which detailed design activities are constructed.
The architectural design can be represented using one or more of a number of different models.
Structural models represent architecture as an organized collection of program components.
Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of applications.
Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.
Process models focus on the design of business or technical process that the system must accommodate.
Functional models can be used to represent the functional hierarchy of a system.

b) List out the golden rules for User interface design. CO3 L3 7M

Three principles of user interface design.
The first is to place the user in control (which means have the computer interface support the user's understanding of a task and do not force the user to follow the computer's way of doing things).
The second (reduce the user's memory load) means place all necessary information on the screen at the same time.
The third is consistency of form and behavior.
The three "golden rules" are:
Place the user in control
Reduce the user's memory load
Make the interface consistent
Place the User in Control
Theo Mandel defines a number of design principles that allow the user to maintain control:
Define interaction modes in a way that does not force a user into unnecessary or undesired actions. The user should always be able to enter and exit the mode with little or no effort.

Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided by using keyboard commands, mouse movements, digitizer pen or voice recognition commands.

Allow user interaction to be interruptible and undoable. A user should be able to interrupt a sequence of actions to do something else without losing the work that has been done. The user should always be able to "undo" any action.
Streamline interaction as skill levels advance and allow the interaction to be customized.
Allow to design a macro if the user is to perform the same sequence of actions repeatedly.

Hide technical internals from the casual user. The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.

Design for direct interaction with objects that appear on the screen. The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.
Reduce the User's Memory Load
Reduce demand on short-term memory. Provide visual cues that enable a user to recognize past actions, rather than having to recall them.

Establish meaningful defaults. A user should be able to specify individual preferences; however, a reset option should be available to enable the redefinition of original default values.

Define shortcuts that are intuitive. "Example: Alt-P to print. Using easy to remember mnemonics."
The visual layout of the interface should be based on a real-world metaphor: Anything you represent on a screen if it is a metaphor for a real-world entity then users would easily understand.

Disclose information in a progressive fashion. The interface should be organized hierarchically. The information should be presented at a high level of abstraction.
Make the Interface Consistent
The interface should present and acquire information in a consistent manner:
A set of design principles that help make the interface consistent:
Allow the user to put the current task into a meaningful context. The user should be able to determine where he has come from and what alternatives exist for a transition to a new task.
Maintain consistency across a family of applications. "MS Office Suite"
If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

(OR)

7	a)	What is object oriented design process? why it is so important for software development.	CO3	L3	7M

Software design is the last software engineering action within the modeling activity and sets the stage for construction (code generation and testing).
The analysis model, manifested by scenario-based, class-based, flow-oriented and behavioral elements, feed the design task.
The architectural design defines the relationship between more structural elements of the software, the architectural styles and design patterns that can be used to achieve the requirements defined for the system, and the constraints that affect the way in which the architectural design can be implemented.
The architectural design can be derived from the System Specs, the analysis model, and interaction of subsystems defined within the analysis model.
The interface design describes how the software communicates with systems that interpolate with it, and with humans who use it. An interface implies a flow of information (data, and or control) and a specific type of behavior.
The component-level design transforms structural elements of the software architecture into a procedural description of software components.
The importance of software design can be stated with a single word – quality. Design is the place where quality is fostered in software engineering.
Design is the only way that we can accurately translate a customer's requirements into a finished software product or system.

	b)	Explain about taxonomy of Architecture styles and patterns?	CO3	L4	7M

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,
(4) Semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.
An architectural style is a transformation that is imposed on the design of an entire system.
An architectural pattern, like an architectural style, imposes a transformation on the design of an architecture.
A pattern differs from a style in a number of fundamental ways:
The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.
A pattern imposes a rule on the architecture, describing how the S/W will handle some aspect of its functionality at the infrastructure level.
Architectural patterns tend to address specific behavioral issues within the context of the architectural.
Architectural Patterns
A S/W architecture may have a number of architectural patterns that address issues such as concurrency, persistence, and distribution.
Concurrency—applications must handle multiple tasks in a manner that simulates parallelism
operating system process management pattern
task scheduler pattern
Persistence—Data persists if it survives past the execution of the process that created it. Persistent data are stored in a database or file and may be read and modified by other processes at a later time.
Two patterns are common:
a database management system pattern that applies the storage and retrieval capability of a DBMS to the application architecture

an application level persistence pattern that builds persistence features into the application architecture

Distribution— the manner in which systems or components within systems communicates with one another in a distributed environment.

Unit-IV

8	a)	Discuss different types of integration testing.	CO4	L3	7M

Integration testing often forms the heart of the test specification document. Don't be dogmatic about a "pure" top down or bottom up strategy. Rather, emphasize the need for an approach that is tied to a series of tests that (hopefully) uncover module interfacing problems.

Options:

The “big bang” approach: all components are combined in advance; the entire program is tested as a whole.

An incremental Integration: is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct.

Integration Testing is a systematic technique for constructing the S/W architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

Top-down Integration

Top-down Integration testing is an incremental approach to construction of the S/W arch.

Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).

Modules subordinate to the main control module are incorporated into the structure in either depth-first or breadth-first manner.

Depth-first integration integrates all components on a major control path of the program structure. Selection of a major path is somewhat arbitrary and depends on application-specific characteristics.

Breadth-first integration incorporates all components directly subordinate at each level, moving across the structure horizontally.

The integration process is performed in a series of 5 steps:

The main control module is used as a test driver, and stubs are substituted for all components directly subordinate to the main control module.

Depending on the integration approach selected subordinate stubs are replaced one at a time with actual components.

Tests are conducted as each component is integrated.

On completion of each set of tests, another stub is replaced with the real component.

Regression testing may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

Top-down strategy sounds relatively uncomplicated, but, in practice, logistical problems can arise.

	b)	What are the various testing strategies (White box and Black box) for software testing? Discuss them briefly.	CO4	L2	7M

What is Black Box testing?

In Black-box testing, a tester doesn’t have any information about the internal working of the software system.

Black box testing is a high level of testing that focuses on the behavior of the software.

It involves testing from an external or end-user perspective.

Black box testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance.

What is White Box testing?

White-box testing is a testing technique which checks the internal functioning of the system.

In this method, testing is based on coverage of code statements, branches, paths or conditions.

White-Box testing is considered as low-level testing.

It is also called glass box, transparent box, clear box or code base testing.

The white-box Testing method assumes that the path of the logic in a unit or program is known.

Black Box Testing	White Box Testing
1. Black box testing techniques are also called functional testing techniques.	1. White box testing techniques are also called structural testing techniques.
2Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	2. White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
3. It is mainly applicable to higher levels of testing such as Acceptance Testing and System Testing	3. Mainly applicable to lower levels of testing such as Unit Testing and Integration Testing
4. Black box testing is generally done by Software Testers	4. White box testing is generally done by Software Developers
5. Programming knowledge is not required	5. Programming knowledge is required
6. Implementation knowledge is not required.	6. Implementation knowledge is required

(OR)

9 a) What is cyclomatic complexity? Explain with an example of how to construct a flow graph for a program (Fibonacci series) and compute cyclomatic complexity. CO4 L4 7M

Cyclomatic complexity is grounded in the structure of a program's control flow graph, which represents all paths traversing through the program during its execution. In this graph: Nodes represent parts of the source code that have no control flow statements, meaning they are blocks of instructions executed linearly

Cyclomatic complexity is grounded in the structure of a program's control flow graph, which represents all paths traversing through the program during its execution. In this graph:

Nodes represent parts of the source code that have no control flow statements, meaning they are blocks of instructions executed linearly.

Edges symbolize the control flow between these blocks, dictated by programming constructs like loops (for, while), conditional statements (if, switch), etc.

A program with a straightforward, linear execution path without control flow statements (like loops or conditions) has a cyclomatic complexity of 1, indicating minimal complexity. Conversely, each additional control flow element introduces new paths, increasing the complexity.

b) Explain the Test Strategies for Object-Oriented Software. CO4 L3 7M

Unit Testing

Both black-box and white-box testing techniques have roles in testing individual software modules.

Unit Testing focuses verification effort on the smallest unit of S/W design.

Unit Test Considerations:

Module interface is tested to ensure that information properly flows into and out of the program unit under test.

Local data structures are examined to ensure that data stored temporarily maintains its integrity.

All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.

All error handling paths are tested.

If data do not enter and exit properly, all other tests are moot.

Comparison and control flow are closely coupled. Test cases should uncover errors such as

comparison of different data types

incorrect logical operators or precedence

expectation of equality when precision error makes equality unlikely

incorrect comparison of variables

improper loop termination

failure to exit when divergent iterations is encountered

improperly modified loop variables

Error handling:

when error handling is evaluated, potential errors should be tested:

error description is unintelligible

error noted does not correspond to error encountered

error condition causes O/S intervention prior to error handling

exception-condition processing is incorrect

error description does not provide enough information to assist the location of the cause of the error.

Unit Test Procedures

Because a component is not a stand-alone program, driver and/or stub S/W must be

developed for each unit test.
In most applications, a driver is nothing more than a “main program” that accepts test case data, passes such data to the component, and prints relevant results.
Stubs serve to replace modules that are subordinate to the component to be tested. A stub “dummy program” uses the subordinate module’s interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing.

SIGNATURE OF STAFF1

SIGNATURE OF STAFF2

SIGNATURE OF HEAD OF THE DEPARTMENT