14CS501

Hall Ticket Number:

III/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

October, 2018 Computer Science And Engineering

Fifth Semester SOFTWARE ENGINEERING

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1 Answer all questions (1X12=12 Marks)

a) What is Software Engineering in IEEE terms?
Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software and the study of approaches.

b) List out process framework activities.
Communication
Planning
Modeling
Construction
Deployment

c) What is agility? Define in your own terms.
Agility is dynamic, context specific, change embracing and growth oriented.

d) What is Class based model?
Class-based modeling represents the objects that the system will manipulate, the operations (also called methods or services) that will be applied to the objects to effect the manipulation, relationships (some hierarchical) between the objects, and the collaborations that occur between the classes that are defined.

e) Name different types of Requirements.
Business Requirements (BR)
Market Requirements (MR)
Functional Requirements (FR) – Use Cases
Non-Functional Requirements (NFR)
UI Requirements (UIR)

f) Define flow oriented modelling.
Data flow modeling is a core modeling activity in *structured analysis.*

g) Outline architectural styles and patterns.
An Architectural Style is the application design at the highest level of abstraction.
An Architectural Pattern is a way to implement an Architectural Style.

h) What are class based components?
When an object-oriented software engineering approach is chosen, component-level design focuses on the elaboration of problem domain specific classes and the definition and refinement of infrastructure classes contained in the requirements model.

i) State the Golden rules of User Interface Design.
Place the user in control.
Reduce the user's memory load.
Make the interface consistent.

j) Write different Project domains.
Computer Theory
Algorithms
Cryptography
Distributed Computing

k) Define Software Reliability.

Software reliability is defined as the probability of failure-free operation of a computer program in a specified environment for a specified time.

l) What are different test strategies?

Test Strategies for Conventional Software

→Unit Testing

→ Integration Testing

Test Strategies for Object-Oriented Software

→ Unit Testing in the OO Context

→ Integration Testing in the OO Context

Test Strategies for WebApps

UNIT I

2 a) What is CMMI? Describe in detail about CMMI. 6M

CMMI meansCapability Maturity Model Integration.

CMMI is a Capability Maturity Model developed by Software Engineering Institute(SEI), part of Carnegie mellon university, Pittsbuegh, USA.

CMMI can be used to guide process improvement across a project, a division, or an entire organization.

The CMMI represents a process meta-model in two different ways:

(1) continuous model

(2) Staged model.

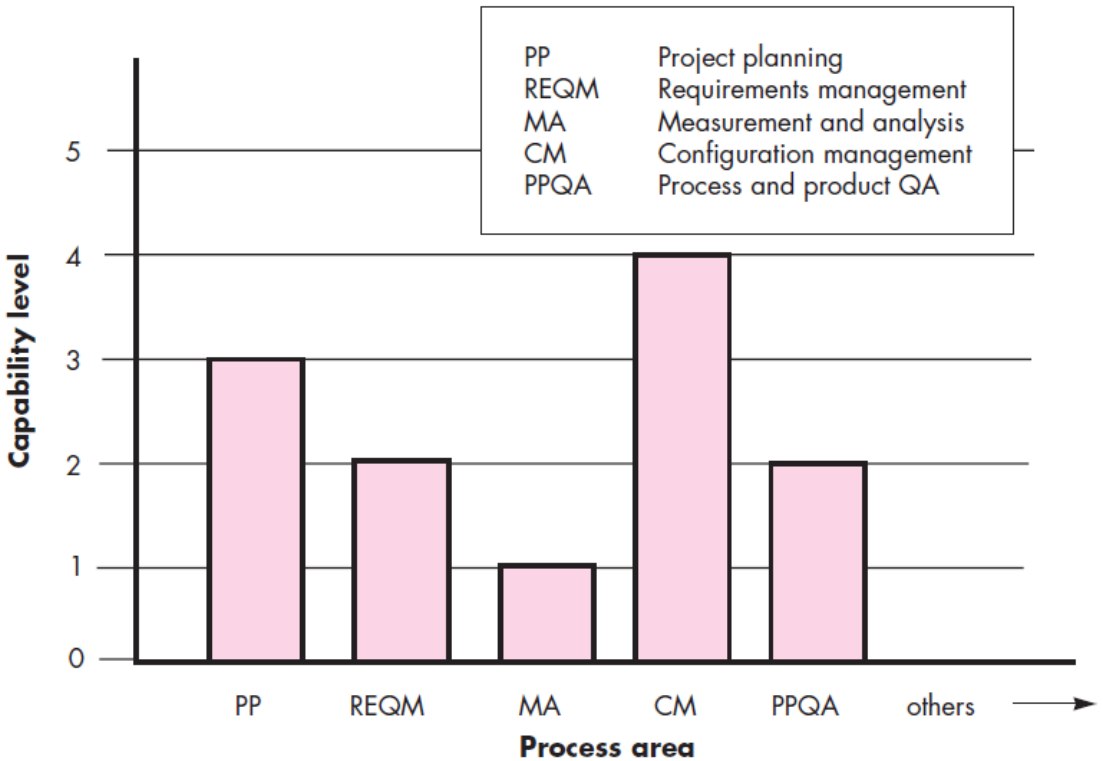
(1) Continuous model:-

The continuous CMMI meta-model describes a process in two dimensions as process area (e.g., project planning or requirements management) is formally assessed against specific goals and practices and is rated according to the following capability levels:

Level 0: *Incomplete*—the process area is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability for the process area.

Level 1: *Performed*—all of the specific goals of the process area have been satisfied.

Level 2: *Managed*—all capability level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy.



Level 3: *Defined*—all capability level 2 criteria have been achieved. In addition, the process is tailored from the organization's set of standard processes according to the organization's

tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets.

Level 4: *Quantitatively managed*—all capability level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment.

Level 5: *Optimized*—all capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

(2) staged model:-

Process Area required to achieve a matured level.

Level	**Focus**	**Process Area**
Optimizing	Continuous process improvement	Organizational innovation and deployment Causal analysis and resolution
Quantitatively managed	Quantitative management	Organizational process performance Quantitative project management
Defined	Process standardization	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Integrated supplier management Risk management Decision analysis and resolution Organizational environment for integration Integrated teaming
Managed	Basic project management	Requirements management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
Performed		

b) Explain about incremental process model 6M

The incremental model combines elements of the waterfalls model applied in an iterative fashion.

Each linear sequence produces deliverable "increments" of the software.

The first incremental is often called as "core-product". In the core product basic requirements are addressed but many supplementary features, remain undelivered

The core product is used by customer and any modifications to core product and additional features and functions are added to the operational product.

Some situations when to use Incremental model:

1. Requirements of the system are clearly understood.
2. When demand for early release of product arises.
3. When the staff are less.

Advantages:-

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its

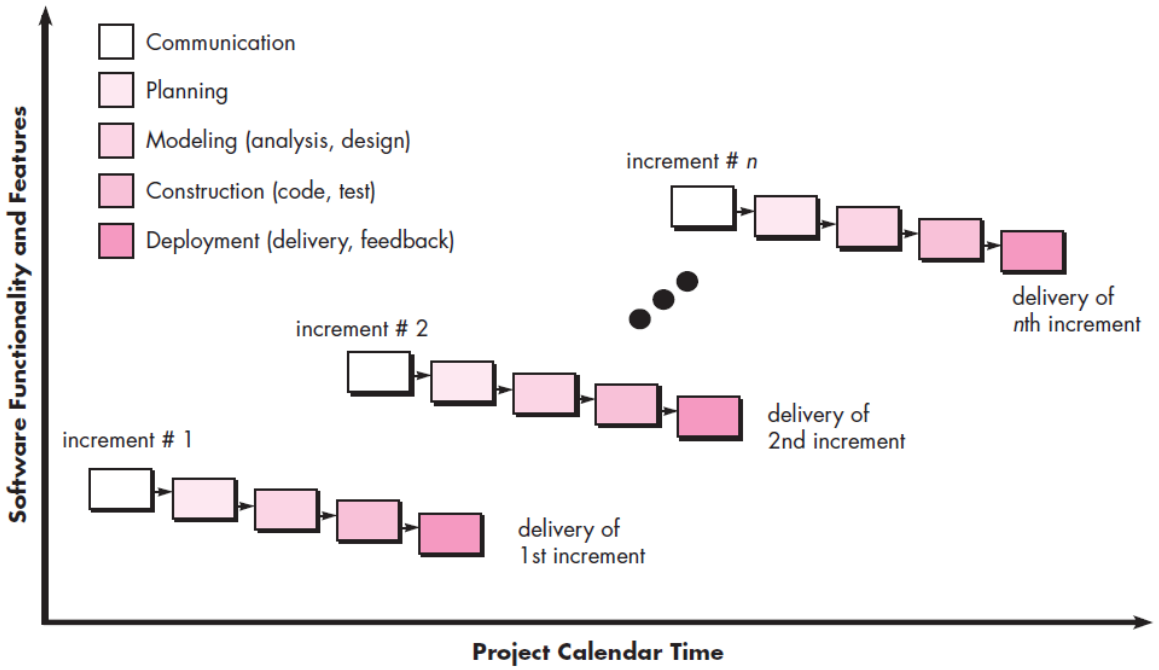
iteration.

- Each iteration is an easily managed milestone.

Disadvantages:-

Each phase of an iteration is rigid and do not overlap each other.

- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.



(OR)

3 a) Discuss Unified process. 6M

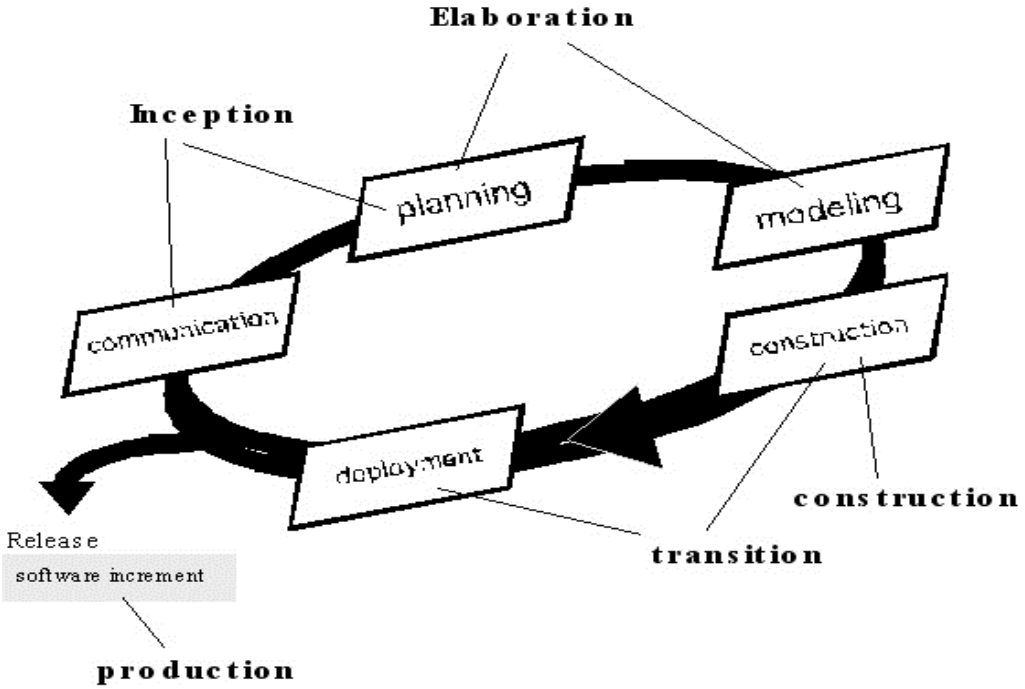
Unified process model is a use-case driven, architecture-centric, iterative and incremental software process closely aligned with the Unified Modeling Language (UML).

Phases of unified process:

The figure below depicts the phases of the UP and relates them to the generic activities.

The **Inception phase** of the UP encompasses both customer communication and planning activities.

By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the ensuing project is developed.



A use-case describes a sequence of actions that are performed by an actor (person, machine, another system) as the actor interacts with the Software.

The **elaboration phase** encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software - the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The **construction phase** of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users.

The **transition phase** of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity.

Software is given to end-users for beta testing, and user feedback reports both defects and necessary changes.

At the conclusion of the **transition phase**, the software increment becomes a usable software release user manuals, trouble-shooting guides, and installation procedures.

The production phase of the UP coincides with the development activity of the generic process.

b) Differentiate Team Process model & Personal process model. 6M

Personal Software Process (PSP)? • The Personal Software Process (PSP) shows engineers how to - manage the quality of their projects - make commitments they can meet - improve estimating and planning - reduce defects in their products PSP emphasizes the need to record and analyze the types of errors you make, so you can develop strategies to eliminate them. Personal Software Process • Because personnel costs constitute 70 percent of the cost of software development, the skills and work habits of engineers largely determine the results of the software development process. • Based on practices found in the CMMI, the PSP can be used by engineers as a guide to a disciplined and structured approach to developing software. The PSP is a prerequisite for an organization planning to introduce the TSP.

Team Software Process (TSP)? • The Team Software Process (TSP), along with the Personal Software Process, helps the highperformance engineer to - ensure quality software products - create secure software products - improve process management in an organization

PSP Framework Activities	PSP Framework Activities
Planning	Launch high level design
High level Design	Implementation
High level Design Review	Integration
Development	Test
Postmortem	postmortem

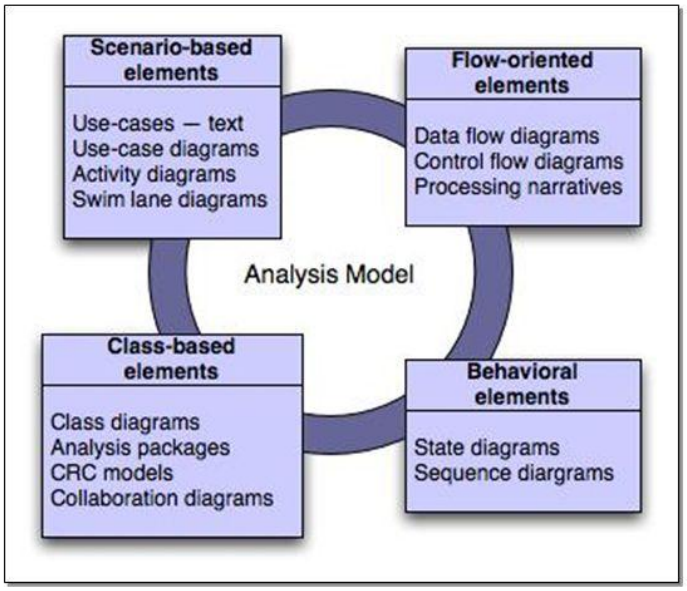
UNIT II

4 a) Write the procedure of building Analysis model. 6M

Analysis model operates as a link between the 'system description' and the 'design model'.

In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modelling'.

Elements of Analysis Model



9

Elements of the analysis model

1. Scenario based element(1.5M)

This type of element represents the system user point of view.

Scenario based elements are use case diagram, user stories.

2. Class based elements(1.5M)

The object of this type of element manipulated by the system.

It defines the object,attributes and relationship.

The collaboration is occurring between the classes.

Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements(1.5M)

Behavioral elements represent state of the system and how it is changed by the external events.

The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements(1.5M)

An information flows through a computer-based system it gets transformed.

It shows how the data objects are transformed while they flow between the various system functions.

The flow elements are data flow diagram, control flow diagram.

b) Determine Class based modelling with example. 6M

Class-based modeling is a stage of requirements modeling. In the context of software engineering, requirements modeling examines the requirements a proposed software application or system must meet in order to be successful. Typically, requirements modeling begin with scenario-based modeling, which develops a use case that will help with the next stages, like data and class-based modeling. Class-based modeling takes the use case and extracts from it the classes, attributes, and operations the application will use. Like all

modeling stages, the end result of class-based modeling is most often a diagram or series of diagrams, most frequently created using UML, Unified Modeling Language.

How do you Identify Classes?(2M)

With this done, class-based modeling then uses several general classifications to help identify elements that can be considered potential classes:

1. **External entities** that produce or consume information to be used by the application. (In the example use case, Sunny Beach Hotel.)
2. **Things** that are part of the information domain of the problem. (The advertisement that Sunny Beach Hotel purchases.)
3. **Occurrences** or events that occur within the context of system operation. (The transaction of Sunny Beach Hotel purchasing the advertisement from Acme Publishing)
4. **Roles** played by people who interact with the system. (The marketing representative at Sunny Beach Hotel that places the order for the advertisement with Acme Publishing and the salesperson at Acme publishing that receive the order.)
5. **Organizational units** those are relevant to an application. (Divisions, groups or teams, like the Sales and Editorial departments at Acme Publishing.)
6. **Places** that establish the context of the problem and the overall function of the system. (Sunny Beach Hotel's location, Acme Publishing's office location.)
7. **Structures** that define a class of objects or related classes of objects. (The 'Best Hotels' travel book is one of several travel books that Acme Publishing publishes, so within this context 'travel book' is a class of objects.)

Selection Characteristics

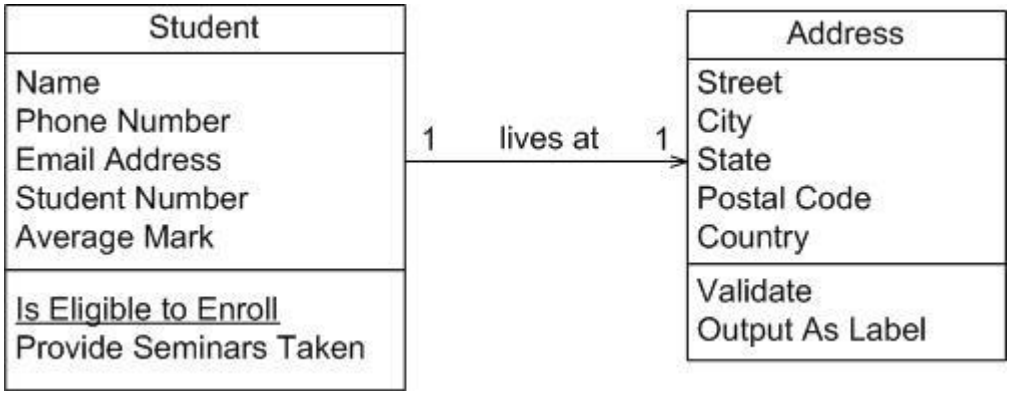
Once the potential classes have been identified, six selection characteristics are used to decide which classes to include in the model:

1. **Retained information** - Information must be remembered about the system over time. (The fact that the advertisement has been sold to the Sunny Beach Hotel.)
2. **Needed services** - A set of operations that can change the attributes of a class. (The number of advertisements sold will change the number of pages of the 'Best Hotels' travel book and also the amount of income it has generated for Acme Publishing.)
3. **Multiple attributes** - A class must have multiple attributes. If something identified as a potential class has only a single attribute, it is best included as a variable. (The 'Best Hotels' travel book has multiple attributes, like number of pages, page size, and retail price.)
4. **Common attributes** - All instances of a class share the same attributes. (All travel books published by Acme Publishing will have the same attributes: number of pages, page size, and retail price. The **values** of these attributes may differ from one travel book to another, for example, one travel book may have 170 pages while another will have 200 pages, but they both still have a number of pages, which is one of their common attributes.)
5. **Common operations** - A set of operations apply to all instances of a class. (All travel books published by Acme Publishing have pages that need to be filled with advertisements or other content, they all need to be printed and shipped to point of sale..)
6. **Essential requirements** - Entities that produce or consume information. (The 'Best Hotels' travel book consumes information by storing the advertisement's content in its pages and produces information with the amount of income earned for the sale of that advertisement.)

Specifying Attributes: Attributes are the set of data objects that fully define the class within the context of the problem.

Defining Methods (a.k.a. Operations, Services) An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class.A method is invoked via message passing.

(SPECIFYING ATTRIBUTES AND METHODS 2M)



(ANY EXAMPLE GIVE 2MARKS)

(OR)

5 a) Describe in detail about construction practice and deployment. 6M

Construction Practice **(1.5M)**

Coding Principle and Concepts

Preparation principles: Before you write one line of code, be sure you:
1. Understand the problem you're trying to solve.
2. Understand basic design principles and concepts.
3. Pick a programming language that meets the needs of the software to the hilt and the environment in which it will operate.
4. Select a programming environment that provides tools that will make your work easier.
5. Create a set of unit tests that will be applied once the component you code is completed.

Coding principles: As you begin writing code, be sure you:
1. Constraint your algorithm by following structured programming practice.
2. Select data structure that will meet the needs of the design.
3. Understand the software architecture and create interfaces that are consistent.
4. Keep conditional statement as simple as possible.
5. Create nested loops in a way that makes them easily testable.
6. Select meaningful variable names and follow other local coding standards.
7. Write code that is self-documenting.
8. Create a visual layout that aids understanding.

Validation principles: After you've completed your first coding pass, be sure you:
1.Conduct a code walkthrough when appropriate.
2. Perform Unit tests and correct errors you have uncovered.
3. Refactor the code.

Testing Principles: **(1.5M)**

Principle #1: All tests should be traceable to customer requirements.
Principle #2: Tests should be planned long before testing begins.
Principle #3: The pare to principle applies to software testing
Principle #4: Testing should begin "in the small" and progress toward testing "in the large
Principle #5: Exhaustive testing is not possible.

Deployment Practices **(3M)**

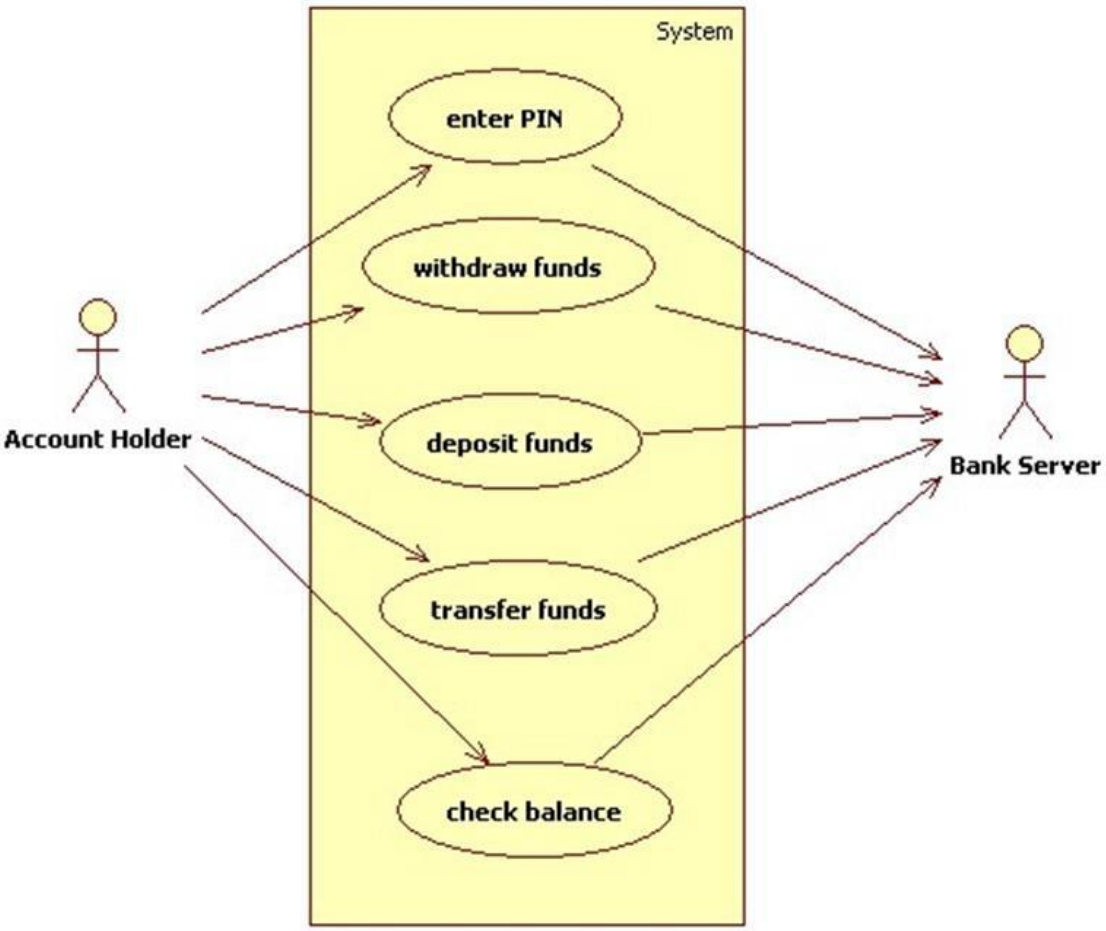
Principle#1: Customer expectations for the software must be managed
Principle#2: A complete delivery package should be assembled and tested. .
Principle #3: A support regime must be established before the software is delivered
Principle#4: Appropriate instructional materials must be provided to end-users.
Principle #5: Buggy software should be fixed first, delivered later.

b) How can you develop Use cases with an example? **6M**

(ANY RELEVANT MATTER GIVES 4M)

Use cases are a type of textual requirements specification that captures how a user will interact with a solution to achieve a specific goal. They describe the step by step process a

user goes through to complete that goal using a software system.
Use cases capture all the possible ways the user and system can interact that result in the user achieving the goal. They also capture all the things that can go wrong along the way that prevent the user from achieving the goal.
Step 1: Identify who is going to be using the system directly. These are the Actors.
Step 2: Pick one of those Actors.
Step 3: Define what that Actor wants to do with the system. Each of these things that the actor wants to do with the system becomes a Use Case.
Step 4: For each of those Use Cases decide on the most usual course when that Actor is using the system. What normally happens.
Step 5: Describe that basic course in the description for the use case.
Step 6: Now consider the alternatives and add those as extending use cases.

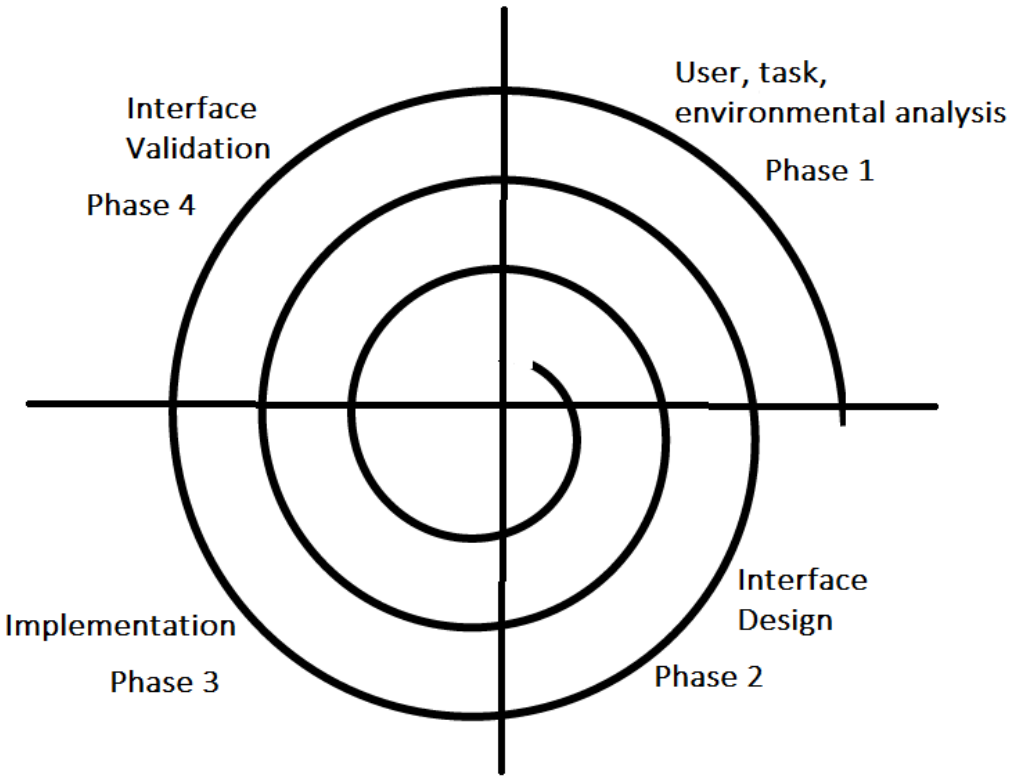


(ANY EXAMPLE GIVES 2M)

UNIT III

6 a) Explain about user interface design and analysis. **6M**

The overall process for analyzing and designing a user interface begins with the creation of different models of system function.



Interface analysis means understanding:

(1) The people (end-users) who will interact with the system through the interface;
(2) The tasks that end-users must perform to do their work,
(3) The content that is presented as part of the interface,
(4) The environment in which these tasks will be conducted**.**

Once interface analysis has been completed, all tasks required by the end-user have been identified in detail.

1. Using information developed during interface analysis (Section 12.3), define interface objects and actions (operations).
2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end-user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

b) Write about assessing alternative architectural designs. 6M

Two different approaches for the assessment of alternative architectural designs.

- (1) The first method uses an iterative method to assess design trade-offs.
- (2)The second approach applies a pseudo-quantitative technique for assessing design quality.

<u>An Architecture Trade-Off Analysis Method</u> **(3M)**

The Software Engineering Institute (SEI) has developed an architecture trade-off analysis method (ATAM) that establishes an iterative evaluation process for software architectures. The design analysis activities that follow are performed iteratively.

- 1. Collect scenarios: A set of use cases is developed to represent the system from the user's point of view.
- 2. Elicit (Bring out) requirements, constraints, and environment description. This information is determined as part of requirements engineering and is used to be certain that all stakeholder concerns have been addressed.
- 3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements.
- 4. Evaluate quality attributes: Quality attributes for architectural design assessment include reliability, performance, security, maintainability, flexibility, testability, portability, reusability, and interoperability.

- 5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style. This can be accomplished by making small changes in the architecture and determining how sensitive a quality attribute, say performance, is to the change. Any attributes that are significantly affected by variationin the architecture are termed sensitivity points..
- 6. Critique (Assess) candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

Architectural description language (*ADL*) **(3M)**

It provides a semantics and syntax for describing software architecture.

- An ADL should provide the designer with the ability to decompose architectural components,
- Compose individual components into larger architectural blocks,
- Represent interfaces (connection mechanisms) between components.

- Once descriptive, language based techniques for architectural designs have been established, it is more likely that effective assessment methods for architectures will be established as the design evolves.

(OR)

7 a) Describe about Component level design and illustrate with an example. 6M

Component level design is the definition and design of components and modules after the architectural design phase. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component for the system development.

Component level design includes the following actions: **(3M)**

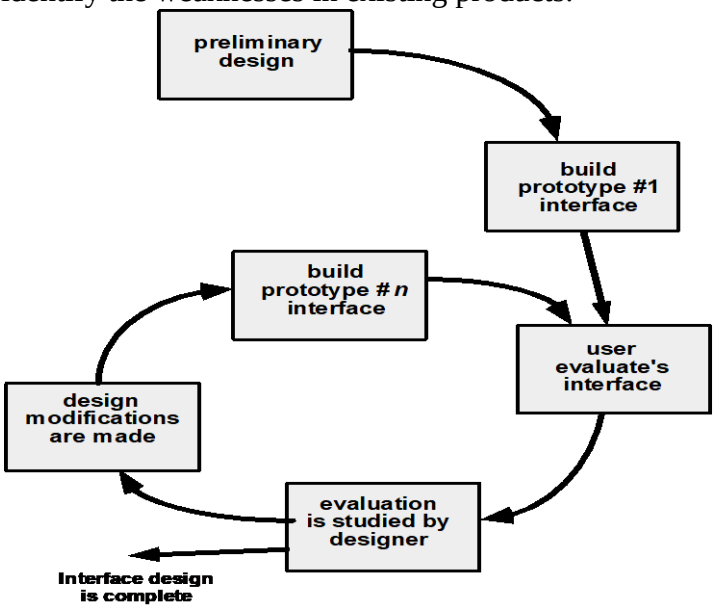
1. Identify Design Classes in Problem Domain
2. Identify Infrastructure Design Classes
3. Elaborate Design Classes
4. Describe Persistent Data Sources
5. Elaborate Behavioral Representations
6. Elaborate Deployment Diagrams
7. Refactor Design and Consider Alternatives

(ANY RELEVANT EXAMPLE-3M)

b) Explain about Design Evaluation. 6M

Design Evaluation

The process of learning how to design good user interfaces often begins with learning to identify the weaknesses in existing products.



(DIAGRAM -3M)

Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assess in evaluating the interface. Questions can be a simple Y/N

response, numeric response, scaled response, Likert scale (strongly agree, etc.), percentage response, and open-ended ones.
(RELATED THEORY-3M)

UNIT IV

8 a) How metrics are integrating with process? 6M

Integrating Metrics with Software Process

→ Many software developers do not collect measures.

→ Without measurement it is impossible to determine whether a process is improving or not.

→ Baseline metrics data should be collected from a large, representative sampling of past software projects.

→ Getting this historic project data is very difficult, if the previous developers did not collect data in an on-going manner.

Arguments for Software Metrics

→ If you don't measure you have no way of determining any improvement.

→ By requesting and evaluating productivity and quality measures software teams can establish meaningful goals for process improvement.

→ Software project managers are concerned with developing project estimates, producing high quality systems, and delivering product on time.

→Using measurement to establish a project baseline helps to make project managers tasks possible.

Baselines

→ Establishing a metrics baseline can benefit portions of the process, project, and product levels.

→ Baseline data must often be collected by historical investigation of past project (better to collect while projects are on-going).

→ To be effective the baseline data needs to have the following attributes:

I Data must be reasonably accurate, not guesstimates.

II Data should be collected for as many projects as possible.

III Measures must be consistent.

IV Applications should be similar to work that is to be estimated.

b) Describe in detail about ISO 9000 standards. 6M

ISO 9000 Quality Standards

→ Quality assurance systems are defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

→ ISO 9000 describes the quality elements that must be present for a quality assurance system to be compliant with the standard, but it does not describe how an organization should implement these elements.

→ ISO 9001:2000 is the quality standard that contains 20 requirements that must be present in an effective software quality assurance system.

(OR)

9 a) Explain about unit and integrating testing. 6M

Unit Testing

Focuses testing on the function or software module

Concentrates on the internal processing logic and data structures

Is simplified when a module is designed with high cohesion

--Reduces the number of test cases

--Allows errors to be more easily predicted and uncovered

Concentrates on critical modules and those with high cyclomaticcomplexity when testing resources are limited

Targets for Unit Test Cases

Module interface-Ensure that information flows properly into and out of the module

Local data structures-Ensure that data stored temporarily maintains its integrity during all steps in analgorithm execution
Boundary conditions-Ensure that the module operates properly at boundary values established to limit or restrict processing.
Independent paths (basis paths)-Paths are exercised to ensure that all statements in a module have beenexecuted at least once
Error handling paths-Ensure that the algorithms respond correctly to specific error conditions

Drivers and Stubs for Unit Testing

Driver-A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results
Stubs-Serve to replace modules that are subordinate to (called by) the component to be tested. It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing
Drivers and stubs both represent overhead-Both must be written but don't constitute part of the installed software Product

Integration Testing

Defined as a systematic technique for constructing the software architecture
-At the same time integration is occurring, conduct tests to uncover errors associated with interfaces
Objective is to take unit tested modules and build a program structure based on the prescribed design
Two Approaches
-Non-incremental Integration Testing
-Incremental Integration Testing

Non-incremental Integration Testing

Commonly called the "Big Bang" approach
All components are combined in advance
The entire program is tested as a whole
Chaos results
Many seemingly-unrelated errors are encountered
Correction is difficult because isolation of causes is complicated
Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

Incremental Integration Testing

Three kinds
-Top-down integration
-Bottom-up integration
-Sandwich integration
The program is constructed and tested in small increments
Errors are easier to isolate and correct
Interfaces are more likely to be tested completely
A systematic test approach is applied

Top-down Integration

Modules are integrated by moving downward through the controlhierarchy, beginning with the main module
Subordinate modules are incorporated in either a depth-first or breadth-first fashion
-DF: All modules on a major control path are integrated
-BF: All modules directly subordinate at each level are integrated

Advantages

This approach verifies major control or decision points early in the test process

Disadvantages

-Stubs need to be created to substitute for modules that have not been built ortested yet; this code is later discarded
-Because stubs are used to replace lower level modules, no significant data flowcan occur

until much later in the integration/testing process

Bottom-up Integration

Integration and testing starts with the most atomic modules in the control hierarchy

Advantages

-This approach verifies low-level data processing early in the testing process

-Need for stubs is eliminated

Disadvantages

-Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version

-Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available

b) Write about formal technical reviews. 6M

Formal Technical Reviews

→ Involves 3 to 5 people (including reviewers)

→ Advance preparation (no more than 2 hours per person) required

→ Duration of review meeting should be less than 2 hours

→ Focus of review is on a discrete work product

→ Review leader organizes the review meeting at the producer's request

→ Reviewers ask questions that enable the producer to discover his or her own error (the product is under review not the producer)

→ Producer of the work product walks the reviewers through the product

(alternative: in inspections a "reader" who is not the producer presents the work product)

→ Recorder writes down any significant issues raised during the review

→ Reviewers decide to accept or reject the work product and whether to require additional reviews of product or not.