

Internal Assessment Test 3 – October 2024

Sub:	Software Engineering						Sub Code:	22MCA23	
Date:	15/10/2023	Duration:	90 min's	Max Marks:	50	Sem:	II	Branch	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
PART I				
1	What is System Modeling? Explain the different System Perspectives. OR	[10]	CO1	L1
2	a. What is design pattern? Explain four elements of design pattern. b. Explain Interaction and Behavioral Models	[10]	CO1	L1
PART II				
3	Draw and explain the state machine model of simple microwave oven. OR	[10]	CO5	L3
4	Draw and explain the Activity Diagram for Insulin Pump Operation.	[10]	CO5	L2
PART III				
5	Draw a sequence diagram describing the sequence of operations for an ATM system. OR	[10]	CO5	L3
6	What is Model Driven Engineering? State the three types of abstract system models produced.	[10]	CO3	L2
PART IV				
7	State and explain Development Testing and its three levels - unit testing, component testing and system testing OR	[10]	CO4	L2
8	Write short notes on: a). Verification & Validation (b) Unit testing (c) Regression Testing	[10]	CO4	L2
PART V				
9	With the help of neat diagram, explain various stages of acceptance testing process. OR	[10]	CO3	L2
10	What are the different types of maintenance? Explain the challenges associated with each.	[10]	CO3	L2

...

Solution

1. What is System Modeling? Explain the different System Perspectives.

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language (UML)**. Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

Models can explain the system from **different perspectives**:

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

- **Activity** diagrams, which show the activities involved in a process or in data processing.
- **Use case** diagrams, which show the interactions between a system and its environment.
- **Sequence** diagrams, which show interactions between actors and the system and between system components.
- **Class** diagrams, which show the object classes in the system and the associations between these classes.
- **State** diagrams, which show how the system reacts to internal and external events.

2. a. What is design pattern? Explain four elements of design pattern

The design pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings.

→ The pattern is not a detailed specification.

→ Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience.

→ Design patterns are usually associated with object-oriented design.

→ The general principle of encapsulating experience in a pattern is one that is equally applicable to any kind of software design

→ The four essential elements of design patterns were defined by the 'Gang of Four' in their patterns book:

- A name that is a meaningful reference to the pattern.
- A description of the problem area that explains when the pattern may be applied.
- A solution description of the parts of the design solution, their relationships, and their responsibilities. This is not a concrete design description. It is a template for a design solution that can be instantiated in different ways. This is often expressed graphically and shows the relationships between the objects and object classes in the solution.

- A statement of the consequences—the results and trade-offs—of applying the pattern. This can help designers understand whether or not a pattern can be used in a particular situation.

2. b. Explain Interaction and Behavioral Models

From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system.

This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose of Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

Behavioral Model is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

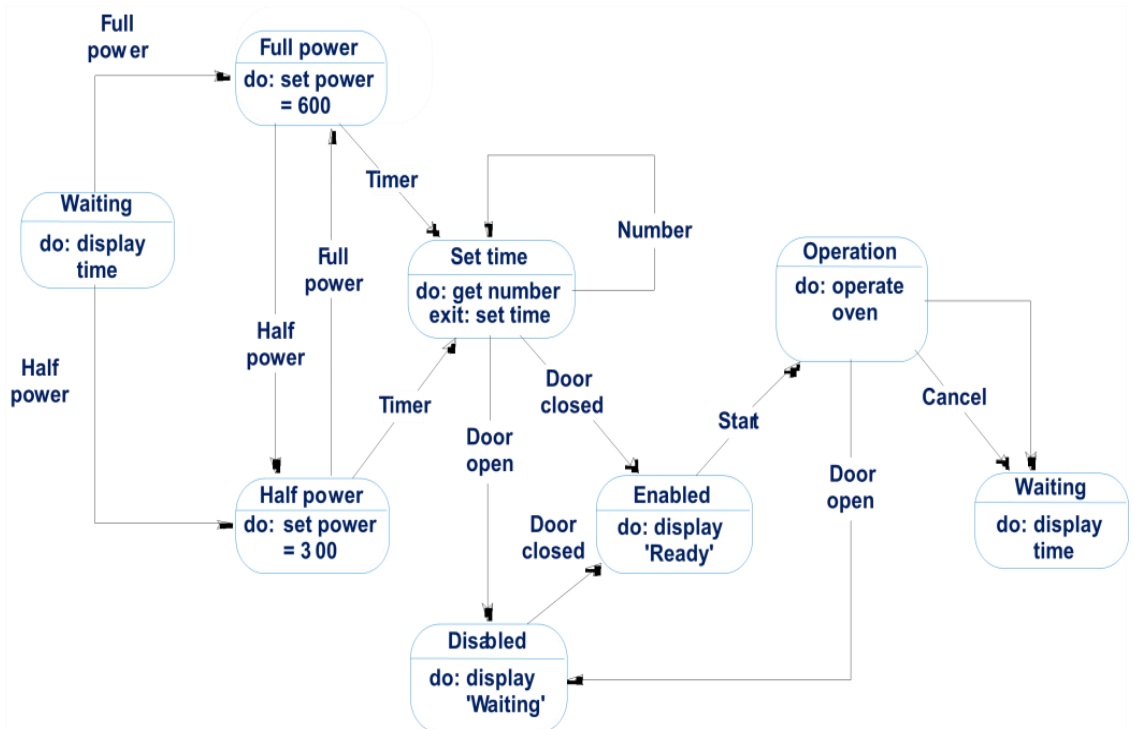
So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.

3. Draw and explain the state machine model of simple microwave oven

Microwave oven state table description

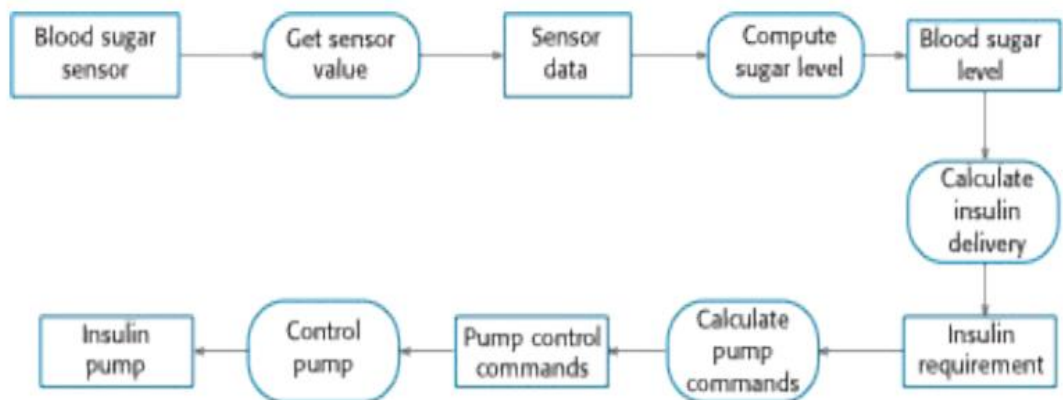
State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Microwave oven model (State diagram)

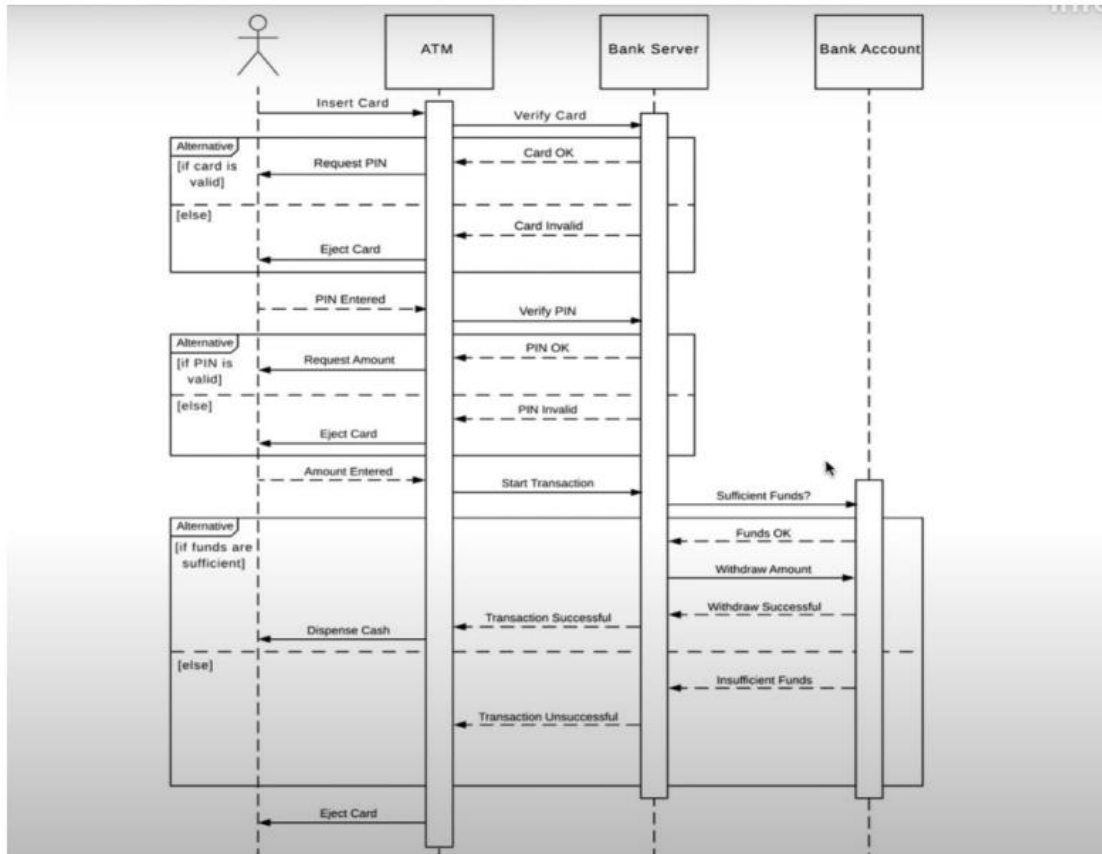


4. Draw and explain the Activity Diagram for Insulin Pump Operation

Activity Diagram



5. Draw a sequence diagram describing the sequence of operations for an ATM system



6. What is Model Driven Engineering? State the three types of abstract system models produced.

- Model-driven engineering (**MDE**) is an approach of software development where **models** rather than programs are **the principal outputs** of the development process.
- The programs that are executed on a hardware/software platform are then **generated automatically** from the **models**.

MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.

Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

Types of Models:

- **A computation independent model (CIM)**

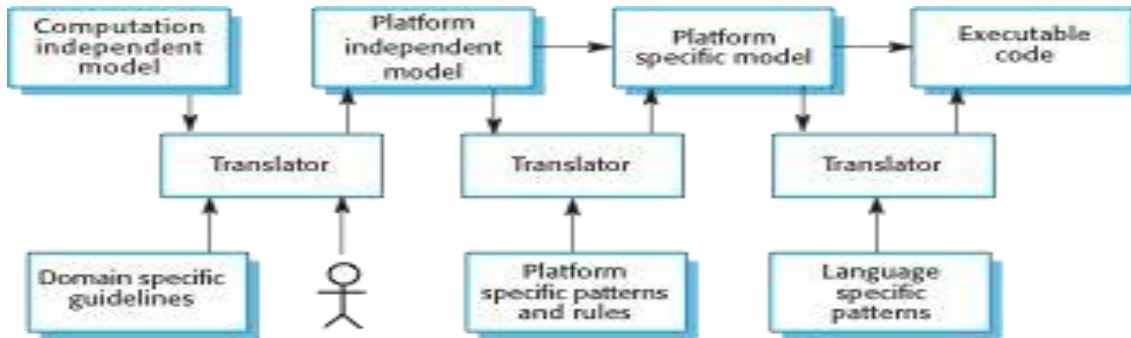
These model the important domain abstractions used in a system. CIMs are sometimes called domain models.

- **A platform independent model (PIM)**

These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.

- **Platform specific models (PSM)**

These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.



- **State and explain Development Testing and its three levels - unit testing, component testing and system testing**

Development testing includes all testing activities that are carried out by the team developing the system.

The tester of the software is usually the programmer who developed that software, although this is not always the case.

1. Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
2. Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
3. System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

8. Differentiate between

a). Verification & Validation

(b) Unit testing

(c) Regression testing

Verification	Validation
The verifying process includes checking documents, design, code, and program	It is a dynamic mechanism of testing and validating the actual product
It does not involve executing the code	It always involves executing the code
Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc.	It uses methods like Black Box Testing, White Box Testing , and non-functional testing
Whether the software conforms to specification is checked	It checks whether the software meets the requirements and expectations of a customer
It finds bugs early in the development cycle	It can find bugs that the verification process can not catch
Target is application and software architecture, specification, complete design, high level, and database design etc.	Target is an actual product
QA team does verification and make sure that the software is as per the requirement in the SRS document.	With the involvement of testing team validation is executed on software code.
It comes before validation	It comes after verification

Unit Testing:

Unit testing involves testing individual components or units of code, typically functions or methods, to ensure they work as expected. It helps developers verify that each piece of code behaves correctly in isolation.

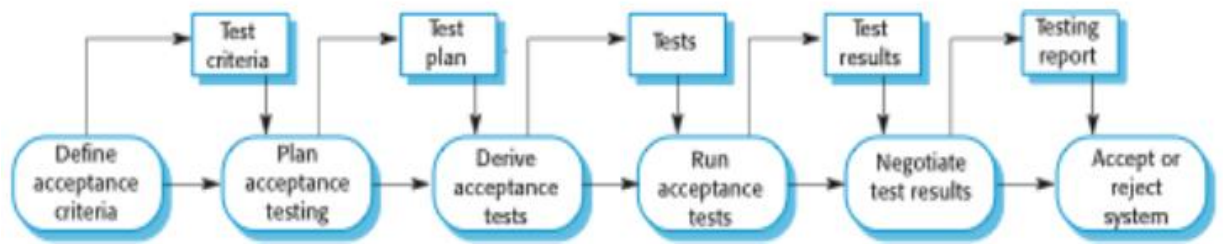
- **Example:** Testing a function that calculates the sum of two numbers to confirm it returns the correct result for different inputs.

Regression Testing:

Regression testing is the process of re-running previously completed tests on modified code to ensure that recent changes haven't introduced new bugs or broken existing functionality. It ensures that updates, bug fixes, or new features don't negatively impact the existing codebase.

- **Example:** After adding a new feature to a software application, running a suite of tests on both new and old functionalities to ensure everything still works as intended.

9. With the help of neat diagram explain various stages of acceptance testing process



Stages in the acceptance testing process

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

10. What are the different types of maintenance? Explain the challenges associated with each.

Software maintenance refers to the process of updating and improving software after its initial release to correct issues, enhance performance, adapt to new environments, or meet evolving user needs. Maintenance is a significant part of the software lifecycle and typically consumes a large portion of time and resources. Here are the four main types of software maintenance and the associated challenges with each:

1. Corrective Maintenance

Involves fixing defects or bugs found in the software after deployment. It includes addressing issues reported by users, such as errors in code or unexpected behaviors.

- Challenges :
- Identifying Root Causes : Diagnosing the exact cause of a defect, especially in large systems, can be difficult and time-consuming.
- Impact Analysis : Fixing one bug can inadvertently create new issues elsewhere in the software, so understanding dependencies and the impact of changes is crucial.
- Resource Allocation : Corrective maintenance can be unpredictable, requiring immediate attention and resources, disrupting planned work schedules.

2. Adaptive Maintenance

Involves modifying the software to work in new or changing environments. This could include updates for new hardware, operating systems, or compatibility with other software.

- Challenges :

- Continuous Compatibility : Keeping up with frequent updates to operating systems, databases, and third-party tools can be challenging and may require regular testing and adjustments.

- Resource Demands : Adaptive maintenance often requires extensive knowledge of new technologies, necessitating time and training for the development team.

- Technical Debt : If the software was not originally designed with adaptability in mind, making these updates can lead to technical debt, where future maintenance becomes harder.

3. Perfective Maintenance

Involves enhancing the software by adding new features or improving functionality based on user feedback or new requirements. It aims to increase usability, efficiency, or performance.

- Challenges :

- Scope Creep : Adding new features can lead to expanding requirements and scope, which can be challenging to manage within time and budget constraints.

- Complexity Management : As new features are added, the software can become more complex, making it harder to maintain in the long run.

- User Expectations : Balancing user requests with technical feasibility and ensuring enhancements do not affect the current functionality or user experience can be challenging.

4. Preventive Maintenance

- Definition : Involves making proactive improvements to the software to prevent future issues, optimize performance, and enhance maintainability. This may include code refactoring, updating documentation, and addressing technical debt.

- Challenges :

- Justifying Costs : Preventive maintenance often doesn't provide immediate visible benefits, making it harder to justify the investment to stakeholders focused on short-term gains.

- Balancing with Other Priorities : Since preventive maintenance isn't urgent, it often gets deprioritized in favor of more immediate needs like bug fixes or new features.

- Avoiding Over-Engineering : There's a risk of spending too much time making improvements that may not yield significant benefits or may become obsolete as requirements change.