CS8592 OBJECT ORIENTED ANALYSISANDDESIGN L T PC
3 0 0 3

UNIT I UNIFIED PROCESS AND USECASEDIAGRAMS 9

Introduction to OOAD with OO Basics - Unified Process – UML diagrams – Use Case –Case study – the Next Gen POS system, Inception -Use case Modelling – Relating Use cases – include, extend and generalization – When to use Use-cases

UNIT II STATICUML DIAGRAMS 9

Class Diagram— Elaboration – Domain Model – Finding conceptual classes and description classes – Associations – Attributes – Domain model refinement – Finding conceptual class Hierarchies – Aggregation and Composition - Relationship between sequence diagrams and use cases – When to use ClassDiagrams

UNIT III DYNAMIC AND IMPLEMENTATIONUML DIAGRAMS 9

Dynamic Diagrams – UML interaction diagrams - System sequence diagram – Collaboration diagram – When to use Communication Diagrams - State machine diagram and Modelling –When to use State Diagrams - Activity diagram – When to use activity diagrams

Implementation Diagrams - UML package diagram - When to use package diagrams - Component and Deployment Diagrams – When to use Component and Deployment diagrams

UNIT IVDESIGNPATTERNS 9

GRASP: Designing objects with responsibilities – Creator – Information expert – Low Coupling – High Cohesion – Controller

Design Patterns – creational – factory method – **structural** – Bridge – Adapter – **behavioral** – Strategy – observer –Applying GoF design patterns – Mapping design to code

UNITVTESTING 9

Object Oriented Methodologies – Software Quality Assurance – Impact of object orientation on Testing – Develop Test Cases and Test Plans

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Craig Larman, —Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development‖, Third Edition, Pearson Education,2005.

2. Ali Bahrami - Object Oriented Systems Development - McGraw Hill International Edition -1999

REFERENCES:

1. Erich Gamma, a n d Richard Helm, Ralph Johnson, John Vlissides, —Design patterns: Elements of Reusable Object-Oriented Software‖, Addison-Wesley,1995.

2. Martin Fowler, —UML Distilled: A Brief Guide to the Standard Object Modeling Language‖, Thirdedition, Addison Wesley,2003.

COURSE OUTCOMES

On completion of this course, the students will:

CS8592.1	Identify various scenarios based on software requirements
CS8592.2	Express software design with static UML diagrams
CS8592.3	Express software design with dynamic and implementation UML diagrams
CS8592.4	Transform UML based software design into pattern based design using design patterns
CS8592.5	Understand the various testing methodologies for OO software

CS8592 OBJECT ORIENTED ANALYSISANDDESIGN L T PC
3 0 0 3

UNIT I UNIFIED PROCESS AND USECASEDIAGRAMS 9

Introduction to OOAD with OO Basics - Unified Process – UML diagrams – Use Case –Case study – the Next Gen POS system, Inception -Use case Modelling – Relating Use cases – include, extend and generalization – When to use Use-cases

PART-A

1. **Define an object. Identify the probable attributes that will be modeled in a Library database for the objectBOOK.**

   Anything in the real world in an object. It has,

   - properties/attributes - describe the state of anobject
   - Methods/procedures - define itsbehavior

   Attributes for BOOK are AuthorName, Title, Price, Publication

2. **What is Analysis?**

   It emphasizes an investigation of the problem and requirements, rather than a solution. Example: If a new computerized library information system is desired, how will it be used? What are its functions?

3. **What is Design?**

   It emphasizes a conceptual solution that fulfills the requirements, rather than its implementation. Ultimately, designs can be implemented.

   Example: A description of a database schema and software objects.

4. **Define Object-Oriented Analysis and Design.**

   Object-oriented analysis is an emphasis on finding and describing the objects—or concepts—in the problem domain.

   Example: In the case of the library information system, some of the concepts include Book, Library, and Patron.

   Object-oriented design is an emphasis on defining software objects and how they collaborate to fulfill the requirements. Finally, during implementation or object-oriented programming, design objects are implemented, such as a Book class in Java.

   Example: In the library system, a Book software object may have a title attribute and a getChaptermethod.

5. **What is an Object Modeling Language?**

   An object-modeling language is a standardized set of symbols used to model a software system using anobject-orientedframework A modeling language is usually associated witha methodology for object-oriented development. Themodeling language defines the elements of the model.

6. **Write the difference between Traditional and Object Oriented Approach.**

Traditional Approach	**Object Oriented System Development**
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step ofprocess	Focuses on object, classes, modules that easily replaced, modified andreused
Moving from one phase to another phase is complex	Moving from one phase to another phaseis
Increases duration of project	Decreases duration of project
Increases complexity	Reduces complexity and redundancy

7. **What is the difference between a Class and an Object?**

Object	Class
Object is an instance of a class.	Class is a blueprint or template from which objects are created.
Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a logical entity.
Object is createdmany times as per requirement.	Class is declared once
Object allocates memory when it is created.	Class doesn't allocate memory when it is created.

8. **Why object orientation?**

To create sets of objects that work together concurrently to produce software that better, model their problem domain that similarly system produced by traditional techniques.

It adapts to

- Changingrequirements
- Easier tomaintain
- Morerobust
- Promote greater design
- Code reuse

9. **Define Software development process.**

A software development process describes an approach to building, deploying, and possibly maintaining software. The Unified Process is a popular software development process for building object-oriented systems.

10. **What is a Unified Process?**

The Unified Process is a popular software development process for building object-oriented systems. It combines commonly accepted best practices, such as an iterative lifecycle and risk-driven development, into a cohesive and well-documented description. UP practices provide an example structure to talk about how to do—and how tolearn—OOA/D.

11. **What are the key concepts/best practices in the UP?**

- tackle high-risk and high-value issues in earlyiterations
- continuously engage users for evaluation, feedback, andrequirements
- build a cohesive, core architecture in earlyiterations
- continuously verify quality; test early, often, andrealistically
- apply usecases
- model software visually (with theUML)
- carefully managerequirements
- practice change request and configurationmanagement

12. **What is an Iterative UP?**

The UP is an iterative process. In this approach, development is organized into a series of short, fixed-length (for example, four week) mini-projects called **iterations;** the outcome of each is a tested, integrated, and executable system. Each iteration includes its own requirements analysis, design, implementation, and testing activities.

13. **Mention the four major phases of UP?**

1. Inception— approximate vision, business case, scope, vagueestimates.

2. Elaboration—refinedvision,iterativeimplementationofthecorearchitecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.

3. **Construction**—iterative implementation of the remaining lower risk and easier elements, and preparation fordeployment.
4. **Transition**—beta tests,deployment.

4. **Define Rational Unified Process (RUP).**

RUP is a complete software-development process framework, developed by Rational Corporation. It's an iterative development methodology based upon six industry-proven best practices. Processes derived from RUP vary from lightweight—addressing the needs of small projects —to more comprehensive processes addressing the needs of large, possibly distributed project teams.

5. **Define UML.**

The Unified Modeling Language (UML) is a visual language for specifying, constructing, and documenting the artifacts of systems. The UML is the de factostandard diagramming notation for drawing or presenting pictures.

6. **What are the primary goals in the design of UML? / What is the significance of UML?**

The primary design goals of the UML are as follows:

- Provide users with a ready-to-use, expressive visual modeling language to develop and exchange Meaningful models.
- Furnish extensibility and specialization mechanisms to extend the coreconcepts.
- Support specifications that are independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modelinglanguage.
- Encourage the growth of the object toolsmarket.
- Support higher-level development concepts such as components, collaborations, frameworks andpatterns.

7. **What are the three ways to apply UML?**

1. **UML as sketch** – Informal and incomplete diagrams (often hand sketched on whiteboards)
2. **UML as blueprint** – Relatively detailed design diagrams used
   either for a)reverse engineeringor
   b)code generation
3. **UMLasprogramminglanguage**–Completeexecutablespecificationofasoftware system in UML. Executable code will be automatically generated, but is not normally seen or modified by developers.

8. **What are the three perspectives to apply UML?**

1. **Conceptual Perspective** - diagrams are interpreted as describing things in a real world situation or domain ofinterest.
2. **Specification or software perspective** – diagrams describes the software abstractions of components with specifications andinterfaces.
3. **Implementation of software perspective** – diagrams describe software implementation in a particular technology such asJava.

9. **Define Use Case.**

Use cases are text stories, widely used to discover and record requirements. They are text stories of some actor using a system to meet goals. They are not diagrams.

Informally, a **use case** is a collection of related success and failure scenarios that describe actors using a system to support agoal.

10. **Outline the purpose of using use cases, to describe requirements.**

Use cases are not an object oriented artifact-they are simply written stories. However, they are a popular tool in requirements analysis and are an important part of the unified process. ausecaseisalistofsteps,typicallydefininginteractionsbetweenarole(knowninUnified ModelingLanguage (UML)asan"actor")andasystem,toachieveagoal.Theactorcanbea human,an external system, or time. Describing requirements are based on

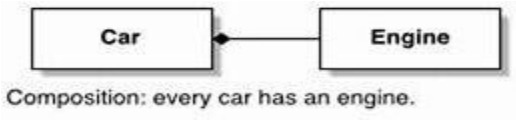
ly what is required of thesoftware

- communicate this understanding precisely to all developmentparties

13. **Define Composition.**

A special type of aggregation where parts are destroyed when the whole is destroyed. The relationship is displayed as a **solid line with a filled diamond at the association end, which is connected to the class that represents the whole orcomposite**.

Example: Objects of Engine live and die with Car. Engine cannot stand by itself.



Composition: every car has an engine.

14. **Define Elaboration.**

Elaboration is the initial series of iterations during which:

- the majority of requirements are discovered andstabilized
- the major risks are mitigated orretired
- the core architectural elements are implemented andproven

15. **What is a domain model?**

A domain model is a visual representation of conceptual classes or real-world objects in adomainofinterest.Theyhavealsobeencalledconceptualmodels,domainobjectmodels,and analysis object models.

16. **How to Create/Make a Domain Model?**

1. **Find the candidate conceptual classes** using the Conceptual Class Category List and noun phrase identification techniques related to the current requirements under consideration.
2. **Draw them as classes** in a domainmodel.
3. **Add the associations** necessary to record relationships for which there is a need to preserve somememory.
4. **Add the attributes** necessary to fulfill the informationrequirements.

17. **Define Conceptual Classes.**

The domain model illustrates conceptual classes or vocabulary in the domain. Informally, a conceptual class is an idea, thing, or object. More formally, a conceptual class may be considered in terms of its symbol, intension, and extension.

- **Symbol**—words or images representing a conceptualclass.
- **Intension**—the definition of a conceptualclass.
- **Extension**—the set of examples to which the conceptual classapplies.

18. **What are the three strategies to find conceptual classes?**

There are three strategies.

- Reuse or modify existingmodels.
- Use a categorylist
- Identify nounphrases.

19. **What are the tasks performed in elaboration?**

Refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.

20. **Why use description classes?/When are Description classes useful?**

Add a specification or description conceptual class (for example, ProductSpecification) when:

- There needs to be a description about an item or service, independent of the current existence of any examples of those items orservices.
  Deleting instances of things they describe (for example, Item) results in a loss of informationthatneedstobemaintained,duetotheincorrectassociationofinformationwith the deleted thing.
- It reduces redundant or duplicatedinformation.

21. **How to Find Associations?**

Two main ways:

- By reading the current, relevant, requirements and asking ourselves what information is needed to fulfil theserequirements
- Using a list of associationcategories.

22. **How to Partition the Domain Model?**

To partition the domain model into packages, place elements together that:

- are in the same subject area — closely related by concept orpurpose
- are in a class hierarchytogether
- participate in the same usecases
- are stronglyassociated

PART-B

1. Explain the various relationship between classes with notations.
2. Draw and explain the class diagram for a banking application.
3. Explain the Domain Model and define all the classes involved in the model?
4. Explain about Association and different types about associations.
5. How to find conceptual classes? Explain the concept of description classes.
6. What is Domain Model Refinement? Explain with suitable examples.
7. Discuss the concept of aggregation and composition?
8. Write about elaboration and discuss the differences between elaboration and inception with suitable diagram for university domain.
9. Construct design for Library Information system which comprises and following notations. (i)Aggregations (ii)Composition (iii)Associations
10. Explain with an example generalization and specialization and write a note on abstract class and abstract operation.
11. Illustrate with an example, the relationship between sequence and use case diagrams.

UNIT III DYNAMIC AND IMPLEMENTATIONUMLDIAGRAMS 9

Dynamic Diagrams – UML interaction diagrams - System sequence diagram – Collaboration diagram – When to use Communication Diagrams - State machine diagram and Modelling –When to use State Diagrams - Activity diagram – When to use activity diagrams

Implementation Diagrams - UML package diagram - When to use package diagrams - Component and Deployment Diagrams – When to use Component and Deployment diagrams

PART-A

1. **Define Interaction Diagrams.**

Object oriented design is concerned with defining software objects and their collaborations. A common notation to illustrate this collaboration is the interaction diagram. It shows the flow of messages between software objects and thus the invocation of methods.

2. **Name the two types of UML interaction diagrams.**

Two types of UML interaction diagrams are

- Sequence Diagram
- CollaborationDiagram

3. **List out the components involved in interaction diagrams.**

The components that are involved in communication interaction diagrams are:

- Objects
- Links
- Messages
- Conditional messages
- Looping or iterationmessages
- Sequence numbers

4. **What is the use of interaction diagrams?**

The UML interaction diagrams are used to illustrate how objects interact via messages.

They are used for dynamic object modeling.

5. **Define SSD. Mention its use.**

A system sequence diagram (SSD) is a picture that shows, for a particular scenario of use case, the events that external actors generate their order and inter system events. All systems are treated as a black box. The emphasis of the diagram is the events that cross the system boundary from actors to systems.

6. **Define System events and the system boundary.**

The system sequence diagrams shows system Events or I/O messages relative to the system. Input system events imply the system has standalone system operations to handle the events, just as an Object Oriented message is handled by an Object oriented method.

To identify system events, it is necessary to be clear on the choice of system boundary. For the purposes of software development, the system boundary is usually chosen to be the software (and possibly hardware) system itself; in this context, a system event is an external event that directly stimulates the software.

7. **7. Define System Behaviour.**

System Behaviour is a description of what a system does, without explaining how it does it.

One part of the description is the system sequence diagram. Other parts include the use case and system operation contracts.

8. **List out the frame operators in sequence diagram.**

The common frame operators:

- Alt : alternate fragment for mutualexclusion
- Loop : loop fragment while guard istrue
- Opt : Optional fragment that executes if guard istrue
- Par: parallel fragments that execute inparallel.
- Region: critical region within which only one thread canrun.

9. **How to name system events and operations?**

System events (and their associated system operations) should be expressed at the level of intent (abstract level) rather than in terms of the physical input medium or interface widget level. It also improves clarity to start the name of a system event with a verb, since it emphasizes the command orientation of theseevents.

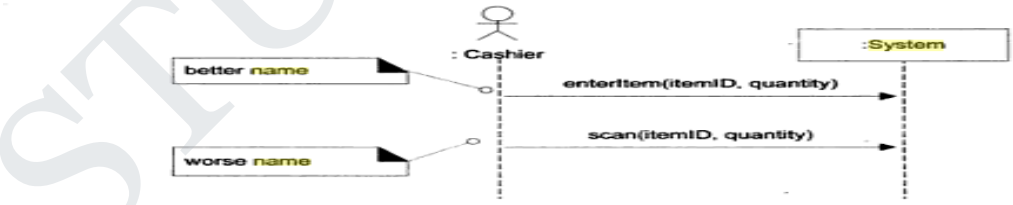


Figure 10.4 Choose event and operation names at an abstract level.

10. **Give the strength and weakness of sequence and collaboration diagram.**

Type	Strengths	Weaknesses
Sequence	Clearly shows sequence or time ordering of Messages.	Forced to extend to the right when adding new objects.
	Large set of detailed notation option	Consumes horizontal space
Collaboration	Space economical	Difficult to see sequence of messages
	Flexibility to add new objects in two dimensions	Fewer notation option

11. **Define system operation.**

Operation that the system as a black box component offers in its public interface is called system operation. System operations can be identified while sketching System sequence diagrams. The system sequence diagram show system events or I/O messages relative to the

system.

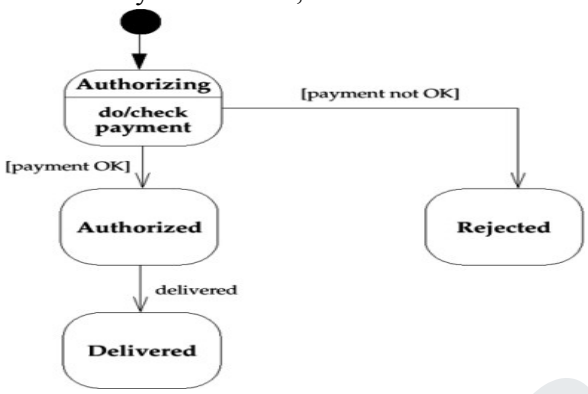
12. **Define State Diagram.**

State diagrams are used to describe the behaviour of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system. The basic elements are rounded boxes representing the state of the object and arrows indicting the transition to the nextstate.

The activity section of the state symbol depicts what activities the object will be doing while it is in that state.

13. **Define Concurrent State Diagrams**

Concurrent State Diagrams In addition to states of an order that are based on the availability of the items, there are also states that are based on payment authorization.

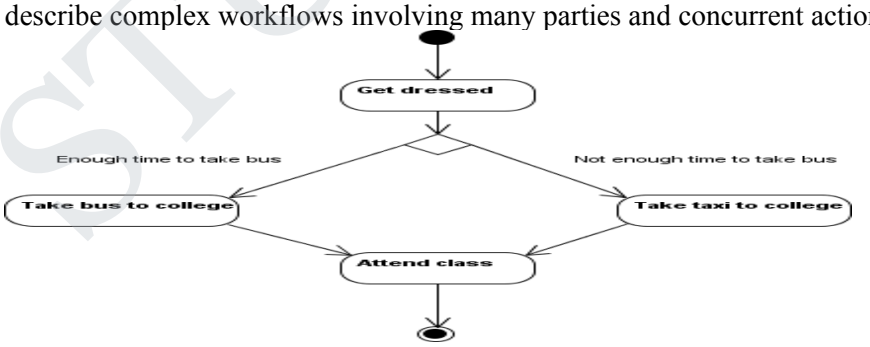


14. **When to use State Diagrams?**

- State diagrams are good at describing the behavior of an object across several usecases.
- State diagrams are not very good at describing behavior that involves a number of objectscollaborating.
- Use state diagrams only for those classes that exhibit interesting behavior, where building the state diagram helps you understand what is goingon.

15. **Define activity diagram.**

A diagram which is useful to visualize workflows and business processes. These can be a useful alternative or adjunct to writing the use case text, especially for business use cases that describe complex workflows involving many parties and concurrent actions.



16. **How conditional behavior is delineated in activity diagram?**

Conditional behavior is delineated by branches and merges.
A branch has a single incoming transition and several guarded outgoing transitions. Only one ofthe outgoing transitions can be taken, so the guards should be mutually exclusive. Using [else] as a guard indicates that the "else" transition should be used if all the other guards on the branch are false.A merge has multiple input transitions and a single output. A merge marks the end of conditional behavior started by a branch.

17. **How Parallel behavior of activity diagram is indicated?**

Parallel behavior is indicated by forks and joins.

A fork has one incoming transition and several outgoing transitions. When the incoming

transition is triggered, all of the outgoing transitions are taken in parallel.

The join before the Close Order activity. With a join, the outgoing transition is taken only when all the states on the incoming transitions have completed their activities.

18. **Define package and draw the UML notation for package.**

A UML package diagram provides a way to group elements. It can group anything: classes, other packages, use cases. UML package diagrams are often used to illustrate the logical architecture of a system -the layers, the subsystems, packages.

The package name is placed on the tab if the package shows the inner members or on the main folder if not. Dependency or coupling is shown by the UML – dependency line – a dashed line with arrow pointing towards depended on package. Fully qualified names are represented in UML for example as java :: util::date

UML notation for package

19. **Give the benefits of using layers.**

- Relaxed complexity is encapsulated anddecomposable.
- Lower layers contain reusablefunctions
- Some layers (primarily the domain and technical services) can bedistributed.
- Development by team is aided because of the logicalsegmentation
- Some layers can be replaced with newimplementations.

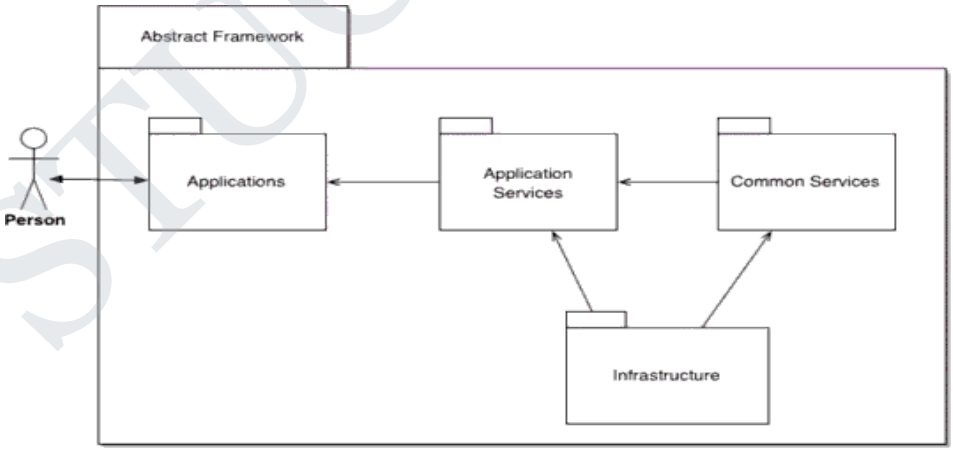
20. **Define dependencies of package diagram.**

A package diagram in the Unified Modeling Language depicts the dependencies between the packages that make up a model. There are two special types of dependencies defined betweenpackages:

- packageimport
- packagemerge

Dependencies exist for various reasons:

One class sends a message to another; one class has another as part of its data; one class mentions another as a parameter to an operation. If a class changes its interface, any message it sends may no longer be valid



21. **When to Use Package Diagrams and Collaborations?**

- Use packages whenever a class diagram that encompasses the whole system is no longer legible on a single letter-size (or A4) sheet ofpaper.
- Packages are particularly useful fortesting.
- Collaborations useful whenever you want to refer to a particularinteraction.
- Parameterized collaborations are useful when you have several similar collaborations in yoursystem.

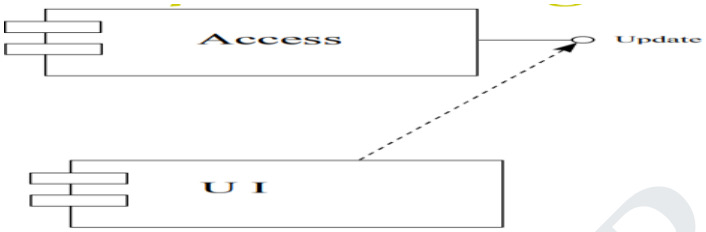
22. **Outline the key reason for modeling a package diagram.**

Packages are the key aspect of UML. A package contains a group or related use cases or model. They are most useful to organize use cases and other models when they get too large or

complex to represent in a simple diagram. A package diagram is one that shows "packages" of artifacts (e.g., use cases, class diagram, etc.) and their respective dependencies. A dependency between any two entities exists when events, actions and definitions in one entity influence events, actions and definitions in the other entity.

23. **Define Component Diagram.**

A Component Diagram models the physical components (source code, exe, UI) in a design.



Components connected by dependency relationships a component is represented by the boxed figure shown in above figure. Dependency is shown as a dashed arrow.

24. **Define Deployment diagram**

A deployment diagram the Unified Modeling Language models the physical deployment of artifacts on nodes. The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

PART-B

1. What are System Sequence Diagrams? What is the relationship between SSDs and use cases? Explain with an example. (or) Explain in detail about UML sequence diagrams.
2. Explain interaction diagrams with suitable examples.
3. Explain about Interaction Diagram Notation for inventory management system
4. Draw UML activity diagram for the Enroll in University use case.
5. Explain State diagrams with suitable examples.
6. Explain Activity diagrams with suitable examples.
7. With an example, explain the need for Activity Diagram.
8. Explain in detail about Package diagram.
9. How will you refine Logical Architecture? Give examples.
10. How do you see the application of UML diagrams for Iterative Software Development? Explain.
11. Discuss about UML deployment and component diagrams. Draw the diagrams for the banking application.
12. Explain in detail about when to use
    (i) Statediagram
    (ii) Activitydiagram
13. Explain in detail about when to use
    (i) Packagediagram
    (ii) Component and Deploymentdiagram

UNITIV DESIGN PATTERNS 9

GRASP: Designing objects with responsibilities – Creator – Information expert – Low Coupling – High Cohesion – Controller Design Patterns – creational – factory method – structural – Bridge – Adapter – behavioral – Strategy – observer –Applying GoF design patterns – Mapping design to code

PART-A

1. **Explain GRASP methodical approach to learning basic object design.**
General Responsibility Assignment Software Patterns (or Principles), abbreviated GRASP, consists of guidelines for assigning responsibility to classes and objects in object-oriented design. GRASP patterns are a learning aid to help one understand essential objectdesign, and apply design reasoning in a methodical, rational, explainable way.
2. **Define GRASP patterns.**

GRASP patterns are really more accurately described as best practices. GRASP patterns outline best practices which can be employed in any object-oriented design. These best practices, if used properly, will lead to maintainable, reusable, understandable, and easy to develop

software.

GRASP patterns: Creator, Information Expert, Low Coupling, Controller, High Cohesion, Indirection, Polymorphism, Protected Variations, Pure Fabrication.

3. **Define GRASP responsibilities.**

Responsibilities are "a contract or obligation of a classifier" (UML definition). Responsibilities can include behaviour, data storage, object creation and more. They often fall into two categories:

Doing responsibilities of an object include:

- Doing something itself, such as creating an object or doing acalculation
- Initiating action in otherobjects
- Controlling and coordinating activities in otherobjects

Knowing responsibilities of object include:

- Knowing about private encapsulateddata
- Knowing about relatedobjects
- Knowing about things it can derive orcalculate

4. **Define cohesion.**

Cohesion – the degree to which the information and responsibilities of a class are related to each other. Cohesion refers to the degree to which the elements of a module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is. Cohesion refers to what the class (or module) will do.

5. **Define coupling.**

Coupling of classes is a measure of how strongly a class is connected to another class. Coupling is the degree to which one class knows about another class. Consider two classes class A and class B. If class A knows class B through its interface only i.e it interacts with class B through its API then class A and class B are said to be loosely coupled.

6. **What is creator?**

The Creator GRASP ensures that coupling due to object instantiation only occurs on closely related classes. An aggregate or container of a class is already coupled with that class. Creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes. In general, a class B should be responsible for creating instances of class A if one, or preferably more, of the following apply:

- Instances of B contain or compositely aggregate instances ofA
- Instances of B record instances ofA
- Instances of B closely use instances ofA

Instances of B have the initializing information for instances of A and pass it on creation.

7. **Define low coupling.**

Assign a responsibility so that coupling remains low. This is pretty vague, but it means that low number of classes to which a class is coupled. Creator is a more specific case of Low Coupling, related to instantiation. Low Coupling is an evaluative pattern, which dictates how to assign responsibilities to support: lower dependency between the classes, change in one class having lower impact on other classes, higher reuse potential.

8. **Define high cohesion.**

High Cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of Low Coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, hard to reuse, hard to maintain and averse to change.

9. **What is creator and its responsibilities?**

Creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes. Assign the responsibility for receiving and handling a system event message to a class that is either:

- Representative of the entire subsystem (e.g. a FaçadeController).

- Representative of the entire use casescenario.

10. **What is facade controller?**

The GRASP mentioned that a Controller that represents an entire subsystem might be called a Façade Controller. A facade is an object that provides a simplified interface to a larger body of code, such as a class library.

11. **What is polymorphism?**

In object-oriented programming, polymorphism (from the Greek meaning "having multiple forms") is the characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

12. **What about pure fabrication?**

To support high cohesion and low coupling, where no appropriate class is present, invent one even if the class does not represent a problem domain concept .This is a compromise that often has to be made to preserve cohesion and low coupling. This kind of class is called "Service" in Domain-driven design.

13. **Define indirection.**

To avoid direct coupling between objects, assign an intermediate object as a mediator. Recall that coupling between two classes of different subsystems can introduce maintenance problems. Another possibility is that two classes would be otherwise reusable (in other contexts) except that one has to know of the other.

14. **What are protected variations?**

Assign responsibility to create a stable interface around an unstable or predictably variable subsystem or component. If a component changes frequently, the users of the component will also have to be modified. This is especially time consuming if the component has many users.

15. **What are the five GRASP patterns?**

Creator, Information Expert, Low Coupling, Controller and High Cohesion

16. **What is Responsibility-Driven Design?**

Responsibility-driven design is a design technique in object-oriented programming. A popular way of thinking about the design of software objects and also larger scale components are in terms of responsibilities, roles and collaborations. Responsibility-driven design is inspired by the client/server model. It focuses on the contract byasking:

- What actions is this object responsiblefor?
- What information does this objectshare?

This is part of a larger approach called responsibility driven design or RDD.

17. **What are the advantages of Factory objects?**

- Separate the responsibility of complex creation into cohesive helperobjects.
- Hide potentially complex creationlogic.
- Allow introduction of performance-enhancing memory management strategies, such as object caching orrecycling.

18. **Give example for Information Expert pattern or principle.**

Name: Information Expert
Problem: How to assign responsibilities to objects?
Solution: Assign responsibility to the class that has the information needed to fulfill it

- ATM application – to know the balance amt weneed
- The acc no of a customer whose balance amt must beknown.
- The transactions till date for the same customer. This information must be available in a classbank.

Therefore by information expert the bank class will return the acc no and balance details

19. **State the use of Design patterns?**

- Understandability: Classes are easier to understand inisolation.
- Maintainability: Classes aren't affected by changes in othercomponents.
- Reusability: easier to grab hold ofclasses.

20. **Define Patterns and design patterns.**

Principles and idioms codified in a structured format describing the problem, solution, and given a name are called patterns. Design pattern is a general repeatable solution to a

commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

21. **Distinguish between coupling and cohesion.**

Cohesion	Coupling
Cohesion is the indication of the relationship withinmodule.	Coupling is the indication of the relationships between modules.
Cohesion shows the module's relative functional strength.	Coupling shows the relative independenceamong the modules.
Cohesion is a degree (quality) to whicha component / module focuses on the singlething.	Coupling is a degree to which a component/ module is connected to the other modules.
While designing you should strive for high cohesion i.e. a module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should beless.
Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.	Making private fields, private methods and non public classes provides loose coupling.
Cohesion is Intra – Module Concept.	Coupling is Inter -Module Concept.
Various types-coincidental, logical,temporal, procedural, communication	Various types-Data, control, common, content

22. **Write the differences between design patterns and frameworks.**
    - Design patterns are more abstract thanframeworks.
    - Design patterns are smaller architectural elements thanframeworks.
    - Design patterns are less specialized thanframeworks.

23. **What are the various GoF design patterns?**
    - Adapter
    - Factory
    - Singleton
    - Strategy
    - Composite
    - Façade
    - Observer

24. **What is the use of GoF design pattern?**

The GoF design patterns help to resolve and simplify the key software development processes such as,

- Flexibility
- Extensibility
- Dependability
- Predictability
- Scalability
- Efficiency

25. **What are the steps for mapping designs to code?**

Write source code for:

- Class and interfacedefinitions
- Methoddefinitions

Work from OOA/D artifacts:

- Create class definitions for Domain Class Diagrams(DCDs)
- Create methods from Interactiondiagrams

PART-B

1. Explain about GRASP designing objects with responsibilities.
2. Write short notes on adapter, singleton, factory and observer patterns.
3. Explain Creator and Information Expert with an Example?
4. Explain Low Coupling and Controller with an Example?
5. Describe the features of Low Coupling and High Coupling with suitable examples.
6. Explain factory and observer patterns.
7. Explain the concept of High Cohesion.
8. Explain the concept of strategy pattern.
9. Explain the concept of bridge pattern.
10. Explain in detail about all types of Design Patterns with an example scenario.
11. Explain polymorphism, pure fabrication and protected variations?
12. (i) Compare Cohesion and Coupling with suitable example. (8)
    State the role of patterns while developing the system design. (8)
13. (i) Differentiate Bridge and Adapter.(8)
    (ii) How will you design the Behavioral pattern?(8)
    Explain about use case realization with GoF design pattern?
    Explain with a diagram gang of four (GoF) pattern summary and relationships.
    Explain Mapping design to code process with examples.
    How will you generate source code from design using UML? Illustrate.

UNITVTESTING **9**

Object Oriented Methodologies – Software Quality Assurance – Impact of object orientation on Testing – Develop Test Cases and Test Plans

PART-A

1. **Define testing.**

Testing is the process of using suitable test cases to evaluate and ensure the quality of a product by removing or sorting out the errors and discrepancies. It is also used to ensure that the product has not regressed (such as, breaking a feature that previously worked). Testing involves various types and levels based on the type of object/product under test.

Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.

2 **What is meant by Object Oriented Testing?**

Object – Oriented Testing is a collection of testing techniques to verify and validate object oriented software. In object-oriented systems, testing encompasses three levels, namely,

unit testing, subsystem testing, and system testing.

3. **Define Debugging and testing.**
   - Elimination of the syntactical bug is the process ofdebugging.
   - Detection and elimination of the logical bug is the process oftesting.
4. **List out the issues of Software Quality.**
   - Validation or user satisfaction - Validation ensures that the product actually meets the client'sneeds.
   - Verification or quality assurance. - Verification is the process of checking that a software achieves its goal without anybugs.
5. **List out the types of Error.**

   Language Error (syntax errors) are result of incorrectly constructed code, such as an incorrectly typed keyword or punctuations. They are easiest error to be detected on simple runningsystem

   Run time Error are detected on running, when a statement attempts an operation that is impossible to carry out. Eg. if program tries to access a non-exist file or object, it occurs

   Logic Error occur when expected output is not formed. They can be detected only by testing the code and analyzing the results performed by intended code.
6. **Define: Class Testing.**
   - In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces areerror-free.
   - Unit testing is the responsibility of the application engineer who implements the structure.
7. **How would you identify the bugs anddebug?**

   The steps to follow to identify thebugs:
   - Selecting appropriate testingstrategies.
   - Developing test cases and sound testplan.
   - Debuggingtools.
8. **Define Error based and Scenario Testing.**

   Error–based testing techniques search a given class's method for particular clues of interests, then describe how these clues should be tested. E.g: Boundary condition testing

   Scenario–based testing also called usage–based testing, concentrates on capturing use –cases. Then it traces user's task, performing them with their variants as tests. It can identify interaction bugs. These are more complex tests tend to exercise multiple subsystems in a single test coveringhigher visibility system interaction bugs.
9. **List out some testing strategies.**

   The objective of s/w testing is to uncover errors. The various testing strategies constitutes
   - Unit Testing – Black Box testing, White blacktesting
   - Integration Testing – Top–down testing, Bottom–up testing, Regressiontesting
   - Validation Testing – Alpha test, Beta testand
   - System Testing – Recovery testing, Security testing, Stress testing, Performancetesting
10. **List out the types of path testing.**
    - Statement testing coverage - Every statement in the program must be executed at least once
    - Branch testing cover ages - Every branch alternative must be executed at least once under sometest.

11. **Differentiate between Black box testing and white box testing.**

Black box testing	White box testing

Black 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	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.
Mainly applicable to higher levels of testing: • AcceptanceTesting • SystemTesting	Mainly applicable to lower levels of testing: • UnitTesting • IntegrationTesting
Generally, independent Software Testers	Generally, Software Developers

12. **Differentiate between Top-Down testing and Bottom-Up testing.**

Top-Down testing	Bottom-Up testing
Uses stubs as the momentary replacements for the invoked modules and simulates the behavior of the separated lower-level modules.	Use test drivers to initiate and pass the required data to the lower-level of modules.
If the significant defect occurs toward the top of theprogram.	If the crucial flawsencounters towards the bottom of the program.
Main function is written at first then the subroutines are called fromit.	Modules are created first then are integrated with the mainfunction.

13. **What are the impact of an object orientation on testing?**
    - Some types of errors could become less plausible (not worth fortesting)
    - Some types of errors could become more plausible (worth testing fornow)
    - Some new types of errors mightappear
14. **Define a test case. Give example.**

The usual approach to detecting defects in a piece of software is for the tester to select a set of input data and then execute the software with the input data under a particular set of conditions. The tester bundles this information into an item called test case.

Test cases for login form

- Test caseid
- Test case(unit totest)
- Preconditions
- Input testdata
- Priority
- Steps to beexecuted
- Expectedresult
- Actual result
- Pass/Fail Comments

15. **Interpret the impact of Inheritance in Testing.**
    - Inheritance does make testing a system moredifficult.
    - If you do not follow the OOD guidelines, you will end up with objects that are extremely hard to debug andmaintain.
16. **Give some Guidelines for developing Quality Assurance Test Cases.**

Freedman & Thomas have developed guidelines adapted for the Unified Approach

- Describe which feature or service (external of internal), test attempts tocover
- If test case is based on use case it must refer to use–case name and write test plan for that piece
- Specify what to test on which method along with test feature and expectedaction
- Test normal use of the object'smethods
- Test abnormal but reasonable use of the object'smethods
- Test abnormal and unreasonable use of object'smethods
- Test boundary conditions of number of parameters or input set ofobjects
- Test object's interactions & message sent among them with assist of sequencediagram

- On doing revision, document the cases so they become the starting basis for follow–up test

Attempt to reach agreement on answers of what–if questions and repeat process until stabilize

17. **What is a test plan? What steps are followed in developing a test plan?**

A test is developed to detect and identify potential problems before delivering the s/w to its users. The test plan need not be very large; in fact, devoting too much time to the plan can be counterproductive. The following steps are needed to create a plan

- Objectives of test: Create objectives of test and describe how to achievethem
- Development of test case: Develop test data, I/O based on domain of data & expected behavior
- Test analysis: It involves examination of test O/p and documentation of testresults

18. **Give some guidelines for developing Test Plan.**

Thomas stated following guidelines for writing test plans

- Specific appearance or format of test plan must include more details abouttest
- It should contain a schedule and a list of required resources including number of peoples &time
- Document every type of test planned with level of detail driven by severalfactors
- A configuration control system provides a way of tracking changes to code shouldexist
- Try to develop a habit of routinely bring test plan sync with product or product specification
- At end of each month or as reach each milestone, take time to complete routineupdates

19. **List out the steps followed in Continuous testing.**

- Understand and communicate the business case for improvedtesting.
- Develop an internal infrastructure to support continuoustesting.
- Look for leaders who will commit to and own theprocess.
- Measure and document your findings in a defect recordingsystem.
- Publicize improvements as they are made and let people know what they are doingbetter

20. **What are the Myer's Bug Locating principles?**

- Think
- If you reach an impasse, sleep on it.
- If the impasse remains, describe the problem to someoneelse
- Use debuggingtools
- Experimentation should be done as alast

21. **How would have Myers describes about testing?**

Testing is the process of executing a program with the intent of finding errors.

•A good test case is the one that has a high probability of detecting an as yet undiscoverederror.

•A successful test case is the one that detects an as-yet undiscovered error.

PART-B

1. Explain in detail about Bugs and Debugging.
2. Explain the various object oriented methodologies.
3. What is a testing? Discuss various testing strategies available?
4. Explain the impact of an object orientation on testing.
5. How to develop test cases?
6. How to develop test plans?
7. What is testing and what is the need for testing? Explain the different types of Testing.
8. What is GUI based Testing? How does it help improving software design? Explain.
9. Write in detail with an example about Object oriented integration testing. (or) What is integration testing? Explain the same with respect to object oriented systems.
10. What are the different issues involved in object oriented testing? Explain with examples.
11. What are guidelines for developing test plans?