DHANALAKSHMI COLLEGE OF ENGINEERING, CHENNAI

Department of Computer Science and Engineering

CS6502 - OBJECT ORIENTED ANALYSIS AND DESIGN

2 & 16 Mark Questions & Answers

***Year / Semester**: III / V*

***Regulation:** 2013*

***Academic year:** 2017 - 2018*

-

UNIT III

1. What is an elaboration? (M/J − 12) (2 Marks)

Elaboration is used to build the core architecture, resolve the high-risk elements, define most requirements, and estimate the overall schedule and resources.

2. List the tasks performed in elaboration. (2 Marks)

The tasks performed in elaboration are:

1. The core, risky software architecture is programmed and tested
2. The majority of requirements are discovered and stabilized
3. The major risks are mitigated or retired

3. What are the key ideas and best practices that will manifest in elaboration? (2 Marks)

The key ideas and best practices are:

1. Do a short time boxed risk-driven iterations
2. Start programming early
3. Adaptively design, implement, and test the core and risky parts of the architecture
4. Test early, often, realistically
5. Adapt based on feedback from tests, users, developers
6. Write most of the use cases and other requirements in detail, through a series of workshops, once per elaboration iteration

4. What is domain model? (A/M − 11) (2 Marks)

A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest. They have also been called conceptual models, domain object models, and analysis object models.

5. List the steps involved in creating a domain model. (2 Marks)

The steps involved in creating a domain model are,

1. Find the conceptual classes
2. Draw them as classes in a UML class diagram
3. Add associations and attributes

6. List the ways to illustrate a domain model. (2 Marks)

Applying UML notation, a domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined. It provides a conceptual perspective. It shows the following,

1. Domain objects or conceptual classes

2. Associations between conceptual classes
3. Attributes of conceptual classes

7. Why domain model is called as a "Visual Dictionary"? (2 Marks)

The information domain model illustrates could alternatively have been expressed in plain text. But it's easy to understand the terms and especially their relationships in a visual language, since our brains are good at understanding visual elements and line connections. Therefore, the domain model is a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

8. List the elements not suitable in a domain model. (2 Marks)

The elements not suitable in a domain model are:

1. Software artifacts, such as a window or a database, unless the domain being modeled are of software concepts, such as a model of graphical user interfaces.
2. Responsibilities or methods.

9. Define − Conceptual Classes (2 Marks)

The conceptual class is defined as an idea, thing, or object. More formally, a conceptual class may be considered in terms of its symbol, intension, and extension.

10. Define − Description Class (2 Marks)

A description class is defined as information that describes something else. For example, a Product Description that records the price, picture, and text description of an Item.

11. List the three strategies to find conceptual classes. (2 Marks)

The three strategies used to find conceptual classes are:

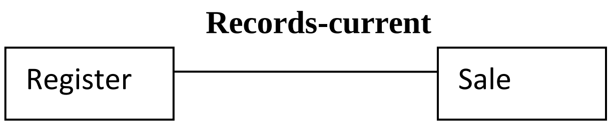
1. Reuse or modify existing models.
2. Use a category list.
3. Identify noun phrases

12. What is an association? (2 Marks)

An association is a relationship between classes (more precisely, instances of those classes) that indicates some meaningful and interesting connection.

13. Draw the UML notation for an association. (2 Marks)

Associations are defined as semantic relationship between two or more classifiers that involve connections among their instances.



14. State the reason to avoid adding many associations. (2 Marks)

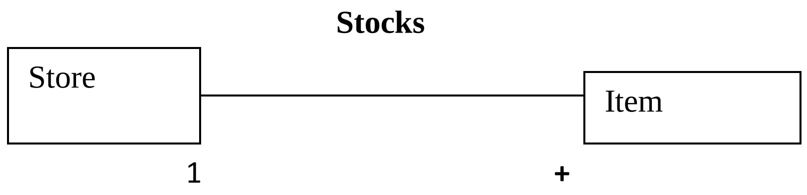
We need to avoid adding too many associations to a domain model. In discrete mathematics, in a graph with n nodes, there can be associations to other nodes a potentially very large number. A domain model with 20 classes could have 190 associations' lines! Many lines on the diagram will obscure it with "visual noise." Therefore, be parsimonious about adding association lines.

15. Write the format to name an association in UML. (2 Marks)

Name in an association is based on a Class Name-Verb Phrase- Class Name format where the verb phrase creates a sequence that is readable and meaningful.

16. What is multiplicity?

Multiplicity defines how many instances of class A can be associated with one instance of a class B.



17. What are association role names? (2 Marks)

Each end of an association is a role, which has various properties, such as:

1. Name
2. Multiplicity

A role name identifies an end of an association and ideally describes the role played by objects in the association.

18. What is qualified association? (2 Marks)

A qualifier may be used in an association; it distinguishes the set of objects at the far end of the association based on the qualifier value. An association with a qualifier is a qualified association. For example, Product Descriptions may be distinguished in a ProductCatalog by their itemID.

19. What is an attribute? (2 Marks)

An attribute is a logical data value of an object. It is useful to identify those attributes of conceptual classes that are needed to satisfy the information requirements of the current scenarios under development.

20. What is a derived attributes? (2 Marks)

The derived attribute is an attribute whose value is calculated from some other attribute. The total attribute in the Sale can be calculated or derived from the information in the SalesLineItems.

21. How does domain model is further refined after the first iteration? (2 Marks)

Generalization and specializations are fundamental concepts in domain modeling. Conceptual class hierarchies are often inspiration for software class hierarchies that exploits inheritance and reduce duplication of code.

Packages are a way to organize large domain models into smaller units.

Domain model is further refined with Generalization, Specialization, Association classes, Time intervals, Composition and packages, usage of subclasses.

22. State the way to develop a domain model incrementally. (2 Marks)

The domain model is incrementally developed by considering concepts in the requirements for this iteration.

23. What is generalization? (2 Marks)

Generalization is the activity of identifying commonality among concepts and defining super class (general concept) and subclass (specialized concept) relationships.

24. Differentiate single inheritance from multiple inheritances. (N/D –12) (2 Marks)

In single inheritance one class inherits its state (attributes) and behavior from exactly one super class. When one class inherits its state (attributes) and behavior from more than one super class, it is referred to as multiple inheritances.

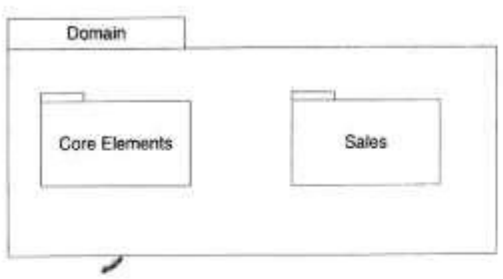
25. What is the need for packages? (2 Marks)

A domain model can easily grow large enough that it is desirable to factor it into packages of strongly related concepts, as an aid to comprehension and parallel analysis work in which different people do domain analysis within different sub-domains.

Following Sections illustrate a package structure for the UP Domain Model:

1. UML Package Notation
2. Ownership and References
3. Package Dependencies

26. Draw the UML Package with an example. (2 Marks)



A UML package is shown as tabbed folder as shown above. Subordinate packages may be shown within it. The package name is within the tab if the package depicts its elements; otherwise, it is centered within the folder itself.

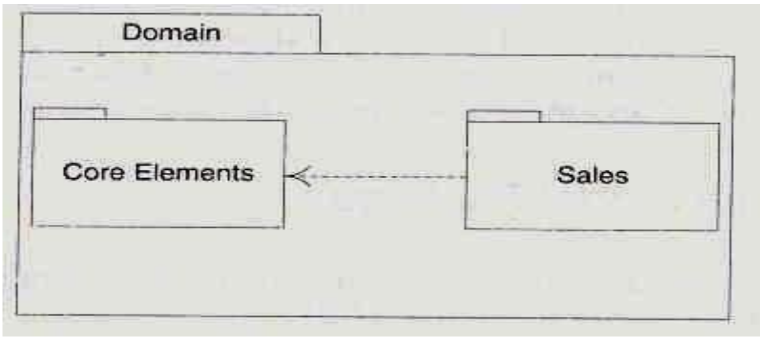
27. What is package dependency? (2 Marks)

If a model element is in some way dependent on another,

1. The dependency may be shown with a dependency relationship
2. Depicted with Arrowed Line
3. A package dependency indicates elements of the dependent package know about or are coupled to elements in the target package

Example:

The sales package has a dependency on the Core Elements package.



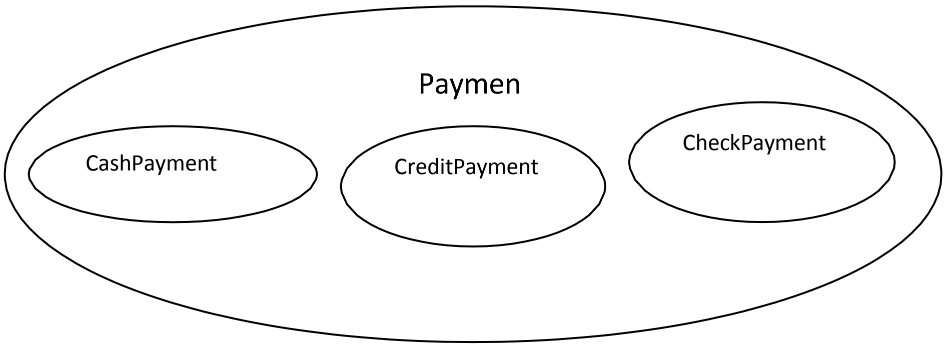
Finding conceptual class hierarchies

28. What are the uses of defining conceptual super classes and subclasses? (2 Marks)

Defining is valuable to identify conceptual super and subclasses, it is useful to clearly and precisely understand generalization, super classes, and subclasses in terms of class definition and class sets.

29. State the role of conceptual subclass and super classes in set membership. (2 Marks)

Conceptual subclasses and super classes are related in terms of set membership. By definition, all members of a conceptual subclass set are members of their super class set. For example, in terms of set membership, all instances of the set CreditPayment are also members of the set Payment. This is shown in the Venn diagram shown below.



30. What is 100% rule? (2 Marks)

100% of the conceptual Super class's definition should be applicable to the subclass. The subclass must conform to 100% of the Super class's attributes and associations.

31. Write the guidelines followed in defining a super class. (2 Marks)

The guidelines are:

Create a super class in a generalization relationship to subclasses when :

1. The potential conceptual subclasses represent variations of a similar concept
2. The subclasses will confirm to the 100% and Is-A rules
3. All subclasses have the same attribute that can be factored out and expressed in the super class
4. All subclasses have the same association that can be factored out and related to the super class

32. What are the strong motivations to partition a conceptual class with subclasses? (2 Marks)

The following are the strong motivations to partition a class into subclasses :

Create a conceptual subclass of a super class when :

1. The subclass has additional attributes of interest
2. The subclass has additional associations of interest
3. The subclass concept is operated on, handled, reacted-to, or manipulated differently than the super class or other subclasses

33. What is aggregation? (A/M – 11, M/J – 12) (2 Marks)

Aggregation is a vague kind of association in the UML that loosely suggests whole-part relationships (as do many ordinary associations). It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.

34. What is composition? (A/M – 11, M/J – 12) (2 Marks)

Composition, also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models. A composition relationship implies that 1) an instance of the part belongs to only one composite instance at a time, 2) the part must always belong to a composite and 3) the composite is responsible for the creation and deletion of its parts either by itself creating/deleting the parts, or by collaborating with other objects.

35. What are the benefits of showing composition? (2 Marks)

Showing composition clarifies the domain constraints regarding the eligible existence of the part independent of the whole. In composite aggregation, the part may not exist outside of the lifetime of the whole.

1. During design work, this has an impact on the create-delete dependencies between the whole and part software classes and database elements(in terms of referential integrity and cascading delete paths)

2. It consists in the identification of a creator (the composite) using the GRASP Creator Pattern.
3. Operations – such as copy and delete – applied to the whole often propagate to the parts.

36. Define – Inception (2 Marks)

The inception is defined as an envision of product scope, vision, and business case.

The purpose of inception stage is not to define all the requirements. The Up is not the waterfall and the first phase inception is not the time to do all requirements or creates believable estimates or plans. That happens during elaboration.

37. State the intent of inception phase. (2 Marks)

The intent of inception is to establish some initial common vision for the objectives of the project, determine if it is feasible, and decide if it is worth some serious investigation in elaboration. It can be brief.

38. Define – Requirements (2 Marks)

Requirements are capabilities and conditions, to which the system and more broadly, the project must conform,

1) The UP promotes a set of best practices, one of which is to manage requirements.
2) In the context of changing and unclear stakeholder's wishes – Managing requirements means a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system.
   A prime challenge of requirements analysis is to find, communicate, and remember (To write down) what is really needed, in a form that clearly speaks to the client and development team members.

39. List the types and categories of requirements. (2 Marks)

In the UP, requirements are categorized as the following,

2) Functional - features, capabilities, security.
3) Usability - human factors, help, documentation.
4) Reliability - frequency of failure, recoverability, predictability.
5) Performance - response times, throughput, accuracy, availability, resource usage.
6) Supportability - adaptability, maintainability, internationalization, configurability.

40. List the key requirements of artifacts. (2 Marks)

The Key requirements artifacts are:

1) Use Case Model - A set of typical scenarios of using a system. There are primarily for functional (behavioural) requirements.
2) Supplementary Specification - Basically, everything not in the use cases. This artifact is primarily for all non-functional requirements, such as performance or licensing. It is also the

place to record functional features not expressed (or expressible) as use cases; for example, a report generation.

3) Glossary - In its simplest form, the Glossary defines noteworthy terms. It also encompasses the concept of the data dictionary, which records requirements related to data, such as validation rules, acceptable values, and so forth. The Glossary can detail any element: an attribute of an object, a parameter of an operation call, a report layout, and so forth.
4) Vision - Summarizes high-level requirements that are elaborated in the Use-Case Model and Supplementary Specification, and summarizes the business case for the project.
5) Business Rules - Business rules (also called Domain Rules) typically describe requirements or policies that transcend one software project they are required in the domain or business, and many applications may need to conform to them. An excellent example is government tax laws. Domain rule details may be recorded in the Supplementary Specification, but because they are usually more enduring and applicable than for one software project, placing them in a central Business Rules artifact (shared by all analysts of the company) makes for better reuse of the analysis effort.

41. Define -- Actors, Scenarios and Use Cases (A/M-08) (2 Marks)

An actor is something with behaviour, such as a person (identified by role), computer system, or organization; for example, a cashier.

A scenario is a specific sequence of actions and interactions between actors and the system; it is also called a use case instance. It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit payment denial.

A use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

42. Define – Use Case Modelling (2 Marks)

Use-Case Model is the set of all written use cases; it is a model of the system's functionality and environment. Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams.

The Use-Case Model is not the only requirement artifact in the UP. There are also the Supplementary Specification, Glossary, Vision, and Business Rules. These are all useful for requirements analysis, but secondary at this point.

The Use-Case Model may optionally include a UML use case diagram to show the names of use cases and actors, and their relationships. This gives a nice context diagram of a system and its environment. It also provides a quick way to list the use cases by name.

43. List the three kinds of actors. (2 Marks)

Actors are roles played not only by people, but by organizations, software, and machines. The three kinds of external actors in relation to the SuD are,

1) Primary actor has user goals fulfilled through using services of the SuD. For example, the cashier.
2) Supporting actor provides a service (for example, information) to the SuD. The automated

payment authorization service is an example. Often a computer system, but could be an organization or person.

3) Offstage actor has an interest in the behaviour of the use case, but is not primary or supporting; for example, a government tax agency.

44. What are the preconditions and post conditions of scenarios? (2 Marks) Preconditions state what must always be true before a scenario is begun in the use case.

Preconditions are not tested within the use case; rather, they are conditions that are assumed to be true. Typically, a precondition implies a scenario of another use case, such as logging in, that has successfully completed.

Success guarantees (or post conditions) state what must be true on successful completion of the use case either the main success scenario or some alternate path. The guarantee should meet the needs of all stakeholders.

EXAMPLE:

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Post conditions): Sale is saved. Tax is correctly calculated.

Accounting and Inventory are updated. Commissions recorded. Receipt is generated.

45. List the steps to find use cases. (N/D-12)

(2 Marks) Use cases are defined to satisfy the goals of the primary actors. Hence, the basic procedures are,

1) Choose the system boundary
2) Identify the primary actors those that have goals fulfilled through using services of the system.
3) Identify the goals for each primary actor.
Define use cases that satisfy user goals; name

PART B

1. Explain in detail, the Case study on the Next Gen POS system.

USECASE MODELING:

The Use Case Model describes the proposed functionality of the new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. A Use Case is a single unit of meaningful work; for example login to system, register with system and create order are all Use Cases. Each Use Case has a description which describes the functionality that will be built in the proposed system. A Use Case may 'include' another Use Case's functionality or 'extend' another Use Case with its own behavior. Use Cases are typically related to 'actors'. An actor is a human or machine entity that interacts with the system to perform meaningful work. Actors

An Actor is a user of the system. This includes both human users and other computer systems. An Actor uses a Use Case to perform some piece of work which is of value to the business.
The set of Use Cases an actor has access to define their overall role in the system and the scope of their action.

Constraints, Requirements and Scenarios

The formal specification of a Use Case includes:

1. Requirements:

These are the formal functional requirements that a Use Case must provide to the end user. They correspond to the functional specifications found in structured methodologies. A requirement is a contract that the Use Case will perform some action or provide some value to the system.

2. Constraints:

These are the formal rules and limitations that a Use Case operates under, and includes pre- post- and invariant conditions. A pre-condition specifies what must have already occurred or be in place before the Use Case may start. A post-condition documents what will be true once the Use Case is complete. An invariant specifies what will be true throughout the time the Use Case operates.

3. Scenarios:

Scenarios are formal descriptions of the flow of events that occurs during a Use Case instance. These are usually described in text and correspond to a textual representation of the Sequence Diagram.

USE CASE RELATIONSHIPS.

Use case relationships is divided into three types

1. Include relationship
2. Extend relationship
3. Generalization

1. Include relationship:

It is common to have some practical behavior that is common across several use cases.
It is simply to underline it or highlight it in some fashion

Example:

Paying by credit: Include Handle Credit Payment

2. Extend relationship:

Extending the use case or adding new use case to the process Extending use case is triggered by some conditions called extension point.

3. Generalization:

Complicated work and unproductive time is spending in this use case relationship. Use case experts are successfully doing use case work without this relationship.

Includes and Extends relationships between Use Cases

One Use Case may include the functionality of another as part of its normal processing. Generally, it is assumed that the included Use Case will be called every time the basic path is run. An example may be to list a set of customer orders to choose from before modifying a selected order in this case the <list orders> Use Case may be ncluded every time the <modify order> Use Case is run. A Use Case may be included by one or more Use Cases, so it helps to reduce duplication of functionality by factoring out common behavior into Use Cases that are re-used many times. One Use Case may extend the behavior of another - typically when exceptional circumstances are encountered.

Relationships between Use Cases

Use cases could be organized using following relationships:

Generalization
Association
Extend
Include
Generalization between Use Cases

Generalization between use cases is similar to generalization between classes; child use case inherits properties and behavior of the parent use case and may override the behavior of the parent.

NOTATION:

Generalization is rendered as a solid directed line with a large open arrowhead (same as generalization between classes).

Generalization between use cases
Association between Use Cases

Use cases can only be involved in binary Associations. Two use cases specifying the same subject cannot be associated since each of them individually describes a complete usage of the system.

Extend Relationship

Extend is a directed relationship from an extending use case to an extended use case that specifies how and when the behavior defined in usually supplementary (optional) extending use case can be inserted into the behavior defined in the use case to be extended.

The extension takes place at one or more extension points defined in the extended use case. The extend relationship is owned by the extending use case. The same extending use case can extend more than one use case, and extending use case may itself be extended.

Extend relationship between use cases is shown by a dashed arrow with an open arrowhead from the extending use case to the extended (base) use case. The arrow is labeled with the keyword Registration use case is meaningful on its own, and it could be extended with optional Get Help On Registration use case. The condition of the extend relationship as well as the references to the extension points are optionally shown in a Note attached to the corresponding extend relationship.

Registration use case is conditionally extended by Get Help On Registration use case in extension point Registration Help

Include Relationship

An include relationship is a directed relationship between two use cases, implying that the behavior of the required (not optional) included use case is inserted into the behavior of the including (base) use case. Including use case depends on the addition of the included use case. The include relationship is intended to be used when there are common parts of the behavior of two or more use cases. This common part is extracted into a separate use case to be included by all the base use cases having this part in common. As the primary use of the include relationship is to reuse common parts, including use cases are usually not complete by themselves but dependent on the included use cases.

Include relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case. The arrow is labeled with the keyword «include».

2. Explain in detail, the Elaboration in Domain Models, Hierarchy.

A domain model captures the most important types of objects in the context of the business. The domain model represents the 'things' that exist or events that transpire in the business environment."

Gives a conceptual framework of the things in the problem space

Helps you think – focus on semantics

Provides a glossary of terms – noun based

It is a static view - meaning it allows us convey time invariant business rules

Foundation for use case/workflow modelling

Based on the defined structure, we can describe the state of the problem domain at any time.

Features of a domain model

The following features enable us to express time invariant static business rules for a domain:-

Domain classes – each domain class denotes a type of object.

Attributes – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.

Associations – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.

Additional rules – complex rules that cannot be shown with symbology can be shown with attached notes

FINDING CONCEPTUAL CLASSES AND DESCRIPTION CLASSES:

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

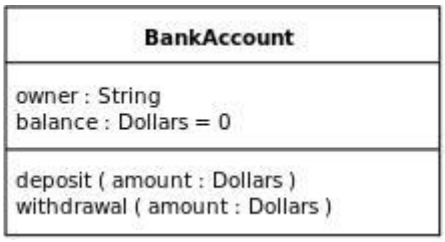
The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling.

The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.
A class with three sections.

In the diagram, classes are represented with boxes which contain three parts:

The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
The middle part contains the attributes of the class. They are left-aligned and the first letter is lowercase.
The bottom part contains the methods the class can execute. They are also left-aligned and the first letter is lowercase.

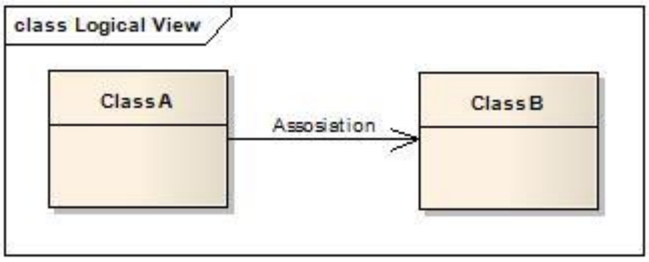
In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.



3.Describe in detail, the Associations,Attributes,Aggregation and Composition give suitable example.

The UML Class diagram is used to visually describe the problem domain in terms of types of object (classes) related to each other in different ways. There are three primary inter-object relationships: association, aggregation, and composition. Using the right relationship line is important for placing implicit restrictions on the visibility and propagation of changes to the related classes, matter which play major role in reducing system complexity. Association

The most abstract way to describe static relationship between classes is using the 'Association' link, which simply states that there is some kind of a link or a dependency between two classes or more.

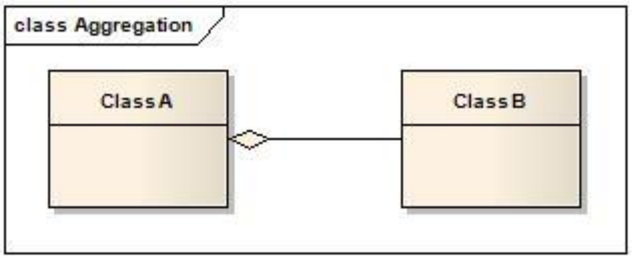


Weak Association

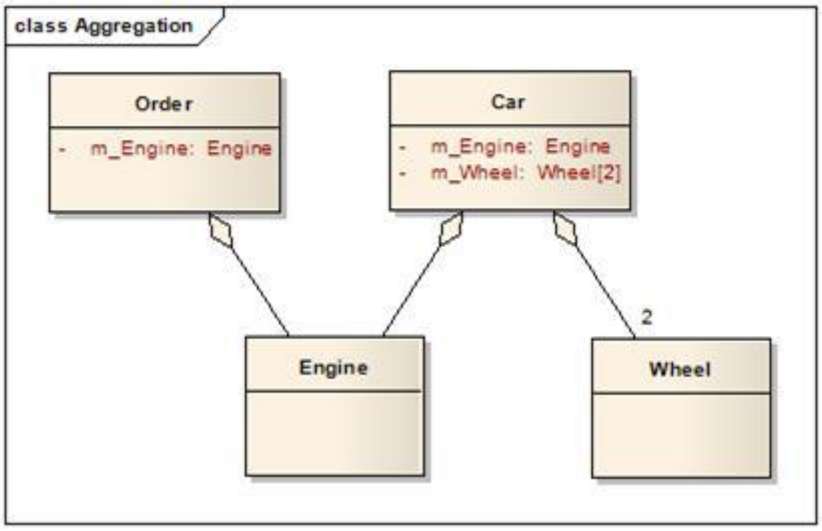
ClassA may be linked to ClassB in order to show that one of its methods includes parameter of ClassB instance, or returns instance of ClassB.

AGGREGATION (SHARED ASSOCIATION)

In cases where there's a part-of relationship between ClassA (whole) and ClassB (part), we can be more specific and use the aggregation link instead of the association link, taking special notice that ClassB can also be aggregated by other classes in the application (therefore aggregation is also known as shared association).



So basically, the aggregation link doesn't state in any way that ClassA owns ClassB nor that there is a parent-child relationship (when parent deleted all its child's are being deleted as a result) between the two. Actually, quite the opposite! The aggregation link usually used to stress the point that ClassA is not the exclusive container of ClassB, as in fact ClassB has another container.

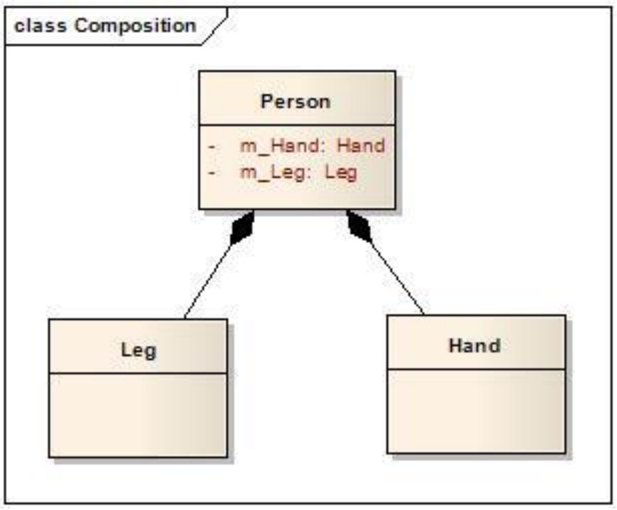


COMPOSITION (NOT-SHARED ASSOCIATION)

In cases where in addition to the part-of relationship between ClassA and ClassB - there's a strong life cycle dependency between the two, meaning that when ClassA is deleted then ClassB is also deleted as a result, we should be more specific and use the composition link instead of the aggregation link or the association link.

The composition link shows that a class (container, whole) has exclusive ownership over other class/s (parts), meaning that the container object and its parts constitute a parent-child/s relationship.

Unlike association and aggregation, in the composition relationship, the composed class cannot appear as a return type or parameter type of the composite class, thus changes in the composed



class cannot be propagated to the rest of the system. Consequently, usage of composition limits complexity growth as the system grows.

SIGNIFICANCE:

-Provides standard for software development.
- Reducing of costs to develop diagrams of UML using supporting tools.
- Development time is reduced.
- The past faced issues by the developers are no longer exists.
- Has large visual elements to construct and easy to follow.

APPLICATION AREA:

Web applications of UML can be used to model user interfaces of web applications and make the purpose of the website clear. Web applications are software-intensive systems and UML is among the efficient choice of languages for modeling them. Web software complexity of an application can be minimized.

UNIT IV

APPLYING DESIGN PATTERNS

1. Explain in detail, the System Sequence Diagrams.

If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances.

Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages.

If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn’t have to wait for a response.

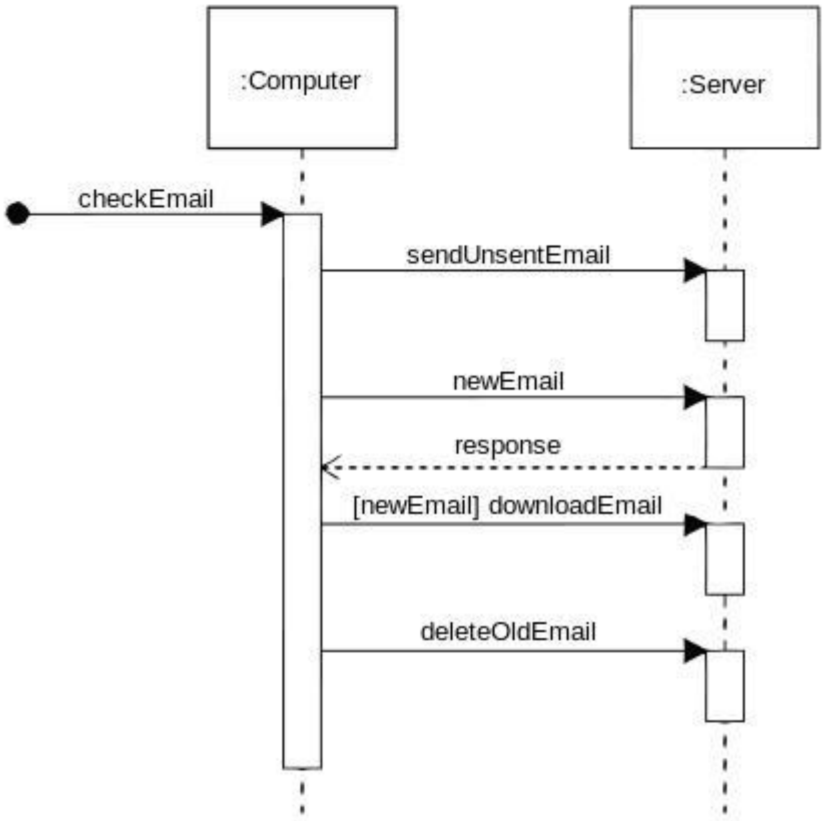
Asynchronous calls are present in multithreaded applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML).

Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing.

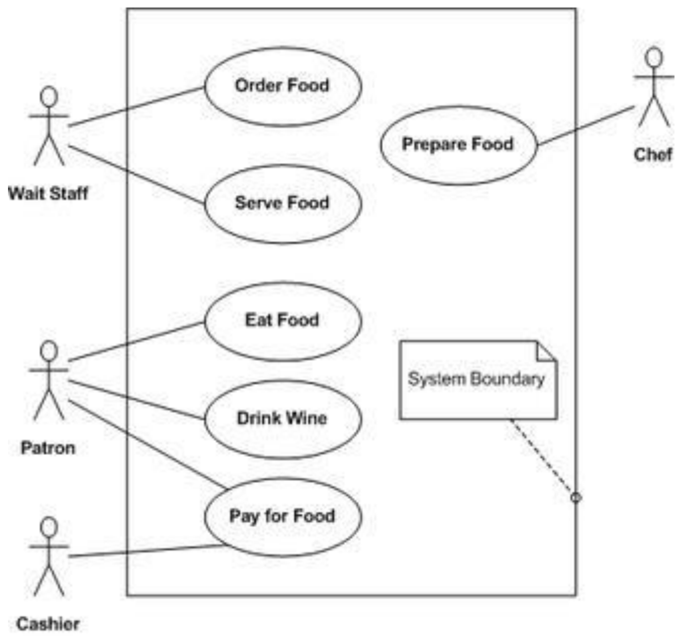
When an object is destroyed (removed from memory), an X is drawn on top of the lifeline, and the dashed line ceases to be drawn below it (this is not the case in the first example though). It should be the result of a message, either from the object itself, or another.

A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of the sequence diagram (gate in UML).

UML has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments[2] which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments,[3] which are then used to model interactions that include parallelism, conditional branches, optional interactions.


:Computer
:Server
checkEmail
sendUnsentEmail
newEmail
response
[newEmail] downloadEmail
deleteOldEmail

2. Describe differentiate between Relationship between Sequence Diagrams and Use Cases.



use case diagram:

Realizing use cases by means of sequence diagrams is an important part of our analysis. It ensures that we have an accurate and complete class diagram.

The sequence diagrams increase the completeness and understandability of our analysis model. Often, analysts use the sequence diagram to assign responsibilities to classes. The behavior is associated with the class the first time it is required, and then the behavior is reused for every other use case that requires the behavior.

When assigning behaviors or responsibilities to a class while mapping a use case to the analysis model, you must take special care to assign the respon- sibility to the correct class. The responsibility or behavior belongs to the class if it is something you would do to the thing the class represents.

For example, if our class represented a table and our application must keep track of the physical location of the table, we would expect to find the method MOVE in the class. We also expect the object to maintain all the information associated with an object of a given type.

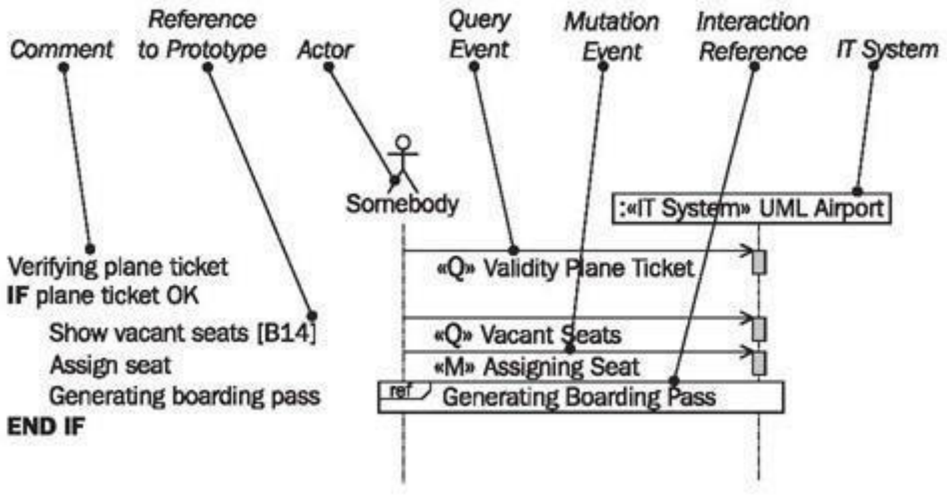
Therefore, the TABLE class should include the method WHERE LOCATED.

This simple rule of thumb states that a class will include any methods needed to provide information about an object of the given class, will provide necessary methods to manipulate the objects of the given class, and will be responsible for creating and deleting instances of the class.

Another guideline helpful in building an analysis model is to examine the model from the whole-part perspective.

The start of the sequence diagram is normally the most difficult aspect. It is important to have access to the object on which you will call a method to start the sequence. It is often the case that you will have to return to the whole to navigate through the whole-part relationship to arrive at the class you will be working with.

A simple example illustrates the whole-part relationship navigation. Suppose you were asked to read the first paragraph of three chapters of a book. First, you would need to know where to go to get the book. We might state that all books we are referring to are available at the Fourth St. library.



4.Explain in detail, the Logical Architecture and Package Diagram.

Logical Architecture and UML Package Diagrams...

• High level, large scale Architecture Model.

• At this level, the design of a typical OO system is based on several architectural layers, such as a UI layer, an application logic (or "domain") layer, and so forth.

• Goal is to design a logical architecture with layers and partitions using UML package diagrams

Logical Architecture And Layers

• Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.

• Logical - because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (deployment architecture)

Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.

It's called the logical architecture because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (these latter decisions are part of the deployment architecture)

LOGICAL ARCHITECTURE REFINEMENT :

UML CLASS DIAGRAM:

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling.

The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
The middle part contains the attributes of the class. They are left-aligned and the first letter is lowercase.
The bottom part contains the methods the class can execute. They are also left-aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

Links

A Link is the basic relationship among objects.

Association

An association represents a family of links. A binary association (with two ends) is normally represented as a line. An association can link any number of classes. An association with three links is called a ternary association. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.

There are four different types of association: bi-directional, uni-directional, Aggregation (includes Composition aggregation) and Reflexive. Bi-directional and uni-directional associations are the most common ones. For instance, a flight class is associated with a plane class bi-directionally. Association represents the static relationship shared among the objects of two classes.

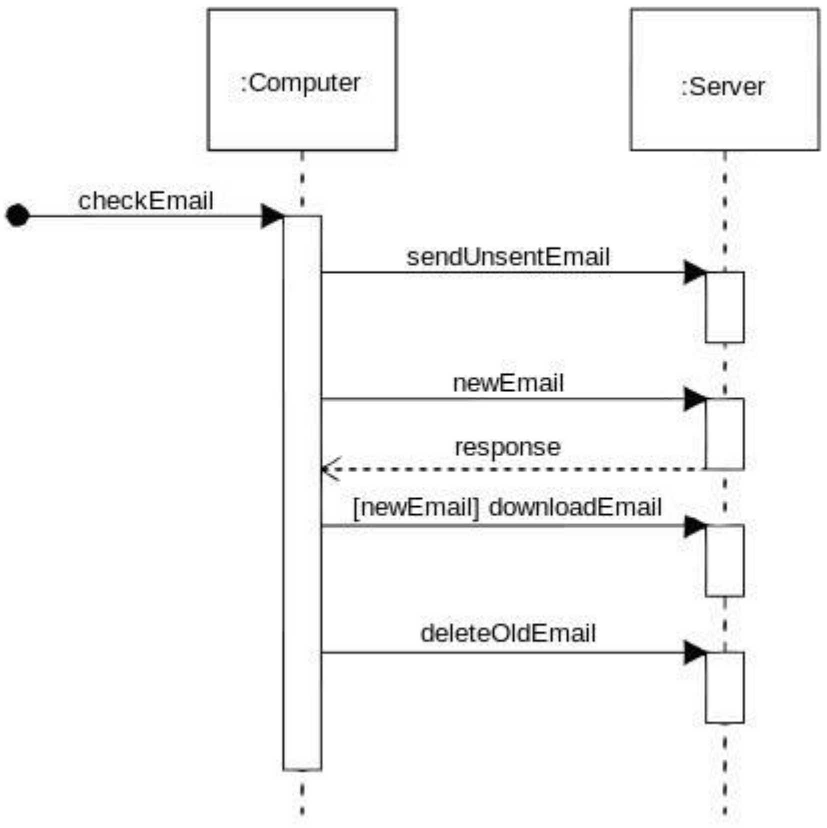
Aggregation

Aggregation is a variant of the "has a" association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship. As shown in the image, a Professor 'has a' class to teach. As a type of association, an aggregation can be named and have the same adornments that an association can. However, an aggregation may not involve more than two classes; it must be a binary association.

Aggregation can occur when a class is a collection or container of other classes, but the contained classes do not have a strong lifecycle dependency on the container. The contents of the container are not automatically destroyed when the container is.

In UML, it is graphically represented as a hollow diamond shape on the containing class with a single line that connects it to the contained class. The aggregate is semantically an extended object that is treated as a unit in many operations, although physically it is made of several lesser objects.

UML INTERACTION DIAGRAMS:



Interaction Overview Diagram is one of the thirteen types of diagrams of the Unified Modeling Language (UML), which can picture a control flow with nodes that can contain interaction diagrams.

The interaction overview diagram is similar to the activity diagram, in that both visualize a sequence of activities.

The difference is that, for an interaction overview, each individual activity is pictured as a frame which can contain a nested interaction diagrams. This makes the interaction overview diagram useful to "deconstruct a complex scenario that would otherwise require multiple if-then-else paths to be illustrated as a single sequence diagram".

The other notation elements for interaction overview diagrams are the same as for activity diagrams.

These include initial, final, decision, merge, fork and join nodes. The two new elements in the interaction overview diagrams are the "interaction occurrences" and "interaction elements."

UNIT V

CODING AND TESTING

1. Explain in detail, the Object Oriented Testing Strategies.

While there are efforts underway to develop more automated testing processes from test models of the object model characteristics (for example states, data flows, or associations), testing is still based on the creation of test cases and test data by team members using a structural (White Box Testing) and/or a functional (SeeBlack Box Testing) strategy.

Overview of Object Orientated Unit Testing

Implications of Object Oriented Testing

Once a class is testing thoroughly it can be reused without being unit tested again

UML class state charts can help with selection of test cases for classes

Classes easily mirror units in traditional software testing

Objective of OO is to facilitate easy code reuse in the form of classes

To allow this each class has to be rigiriously unit tested

Due to classes potentially used in unforeseeable ways when composed in new systems

Example: A XML parser for a web browser

Classes must be created in a way promoting loose coupling and strong cohesion

CLASS TESTING:

Class testing is testing that ensures a class and its instances (objects) perform as defined. (Source: Object Testing Patterns).

When designing a society of classes, an excellent goal is to structure classes so they can be tested with a minimum of fuss. After all, a class that requires compiling the application, starting the application, logging into the application, navigating to a particular point in the application ...will likely not get rigorously tested.

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure.

In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method.

Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects,[5] fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended**.**

OO INTEGRATION TESTING:

Research confirms that testing methods proposed for procedural approach are not adequate for OO approach
Ex. Statement coverage

OO software testing poses additional problems due to the distinguishing characteristics of OO Ex. Inheritance

Testing time for OO software found to be increased compared to testing procedural software

Typical OO software characteristics that impact testing ...

State dependent behavior
Encapsulation
Inheritance
Polymorphism and dynamic
Binding
Abstract and generic classes
Exception handling
Procedural software

unit = single program, function, or procedure Object oriented
software unit = class
unit testing =intra-class testing
integration testing = inter-class testing

cluster of classes dealing with single methods separately is usually too expensive (complex scaffolding), so methods are usually tested in the context of the class they belong to.

2. Describe in detail, the GUI Testing and OO System Testing.

Overview

‰ This chapter discusses the testing of object-oriented systems. The process of testing object-oriented systems begins with a review of the object-oriented analysis and design models. Once the code is written object-oriented testing (OOT) begins by testing "in the small" with class testing (class operations and collaborations). As classes are integrated to become subsystems class collaboration problems are investigated. Finally, use-cases from the OOA model are used to uncover software validation errors.

‰ OOT similar to testing conventional software in that test cases are developed to exercise the classes, their collaborations, and behavior.

‰ OOT differs from conventional software testing in that more emphasis is placed assessing the completeness and consistency of the OOA and OOD models as they are built.

‰ OOT tends to focus more on integration problems than on unit testing.

Object-Oriented Testing Activities

‰ Review OOA and OOD models
‰ Class testing after code is written
‰ Integration testing within subsystems
‰ Integration testing as subsystems are added to the system
‰ Validation testing based on OOA use-cases

Testing OOA and OOD Models

‰ OOA and OOD cannot be tested but can review the correctness and consistency.
‰ Correctness of OOA and OOD models

Syntactic	Semantic
Judged by ensuring that proper modeling conventions and symbolism have been used	Based on the model's conformance to the real world problem domain

‰ **Consistency of OOA and OOD Models**

- Assess the class-responsibility-collaborator (CRC) model and object-relationship diagram
- Review system design (examine the object-behavior model to check mapping of system behavior to subsystems, review concurrency and task allocation, use use-case scenarios to exercise user interface design)
- Test object model against the object relationship network to ensure that all design object contain necessary attributes and operations needed to implement the collaborations defined for each CRC card
- Review detailed specifications of algorithms used to implement operations using conventional inspection techniques

Object-Oriented Testing Strategies

Unit testing in the OO context

- Smallest testable unit is the encapsulated class or object
- Similar to system testing of conventional software
- Do not test operations in isolation from one another
- Driven by class operations and state behavior, not algorithmic detail and data flow across module interface
- Complete test coverage of a class involves
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states
- Design of test for a class uses a verity of methods:
  - fault-based testing
  - random testing
  - partition testing
- each of these methods exercises the operations encapsulated by the class
- test sequences are designed to ensure that relevant operations are exercised
- state of the class (the values of its attributes) is examined to determine if errors exist

Integration testing in the OO context

focuses on groups of classes that collaborate or communicate in some manner
integration of operations one at a time into classes is often meaningless
thread-based testing (testing all classes required to respond to one system input or event)
use-based testing (begins by testing independent classes first and the dependent classes that make use of them)
cluster testing (groups of collaborating classes are tested for interaction errors)
regression testing is important as each thread, cluster, or subsystem is added to the system Levels of integration are less distinct in object-oriented systems

Validation testing in the OO context

- focuses on visible user actions and user recognizable outputs from the system

- validation tests are based on the use-case scenarios, the object-behavior model, and the event flow diagram created in the OOA model
- conventional black-box testing methods can be used to drive the validation tests

3.Describe in detail the Test Case Design for OO Software.

- Each test case should be uniquely identified and be explicitly associated with a class to be tested
- State the purpose of each test
- List the testing steps for each test including:
  - list of states to test for each object involved in the test
  - list of messages and operations to exercised as a consequence of the test
  - list of exceptions that may occur as the object is tested
  - list of external conditions needed to be changed for the test
  - supplementary information required to understand or implement the test

- Testing Surface Structure and Deep Structure
  - Testing surface structure (exercising the structure observable by end-user, this often involves observing and interviewing users as they manipulate system objects)
  - Testing deep structure (exercising internal program structure - the dependencies, behaviors, and communications mechanisms established as part of the system and object design)

Testing Methods Applicable at The Class Level

Random testing - requires large numbers data permutations and combinations, and can be inefficient

- Identify operations applicable to a class
- Define constraints on their use
- Identify a minimum test sequence
- Generate a variety of random test sequences.

Partition testing - reduces the number of test cases required to test a class

- state-based partitioning - tests designed in way so that operations that cause state changes are tested separately from those that do not.
- attribute-based partitioning - for each class attribute, operations are classified according to those that use the attribute, those that modify the attribute, and those that do not use or modify the attribute
- category-based partitioning - operations are categorized according to the function they perform: initialization, computation, query, termination

‰ Fault-based testing

- best reserved for operations and the class level uses the inheritance structure
- tester examines the OOA model and hypothesizes a set of plausible defects that may be encountered in operation calls and message connections and builds appropriate test cases
- misses incorrect specification and errors in subsystem interactions

‰ **Inter-Class Test Case Design**

- Test case design becomes more complicated as integration of the OO system begins – testing of collaboration between classes
- Multiple class testing
  - for each client class use the list of class operators to generate random test sequences that send messages to other server classes
  - for each message generated determine the collaborator class and the corresponding server object operator
  - for each server class operator (invoked by a client object message) determine the message it transmits
  - for each message, determine the next level of operators that are invoked and incorporate them into the test sequence
- Tests derived from behavior models
  - Use the state transition diagram (STD) as a model that represent the dynamic behavior of a class.
  - test cases must cover all states in the STD
  - breadth first traversal of the state model can be used (test one transition at a time and only make use of previously tested transitions when testing a new transition)
  - test cases can also be derived to ensure that all behaviors for the class have been adequately exercised

‰ **Testing Methods Applicable at Inter-Class Level**

Cluster Testing

Is concerned with integrating and testing clusters of cooperating objects

Identify clusters using knowledge of the operation of objects and the system features that are implemented by these clusters

- Approaches to Cluster Testing
  - Use-case or scenario testing
    - Testing is based on a user interactions with the system
    - Has the advantage that it tests system features as experienced by users
  - Thread testing – tests the systems response to events as processing threads through the system
  - Object interaction testing – tests sequences of object interactions that stop when an object operation does not call on services from another object

Use Case

Scenario-based Testing

- Based on
  - use cases
  - corresponding sequence diagrams
- Identify scenarios from use-cases and supplement these with interaction diagrams that show the objects involved in the scenario
- Concentrates on (functional) requirements
  - Every use case
  - Every fully expanded extension (<<extend>>) combination
  - Every fully expanded uses (<<uses>>) combination
  - Tests normal as well as exceptional behavior
- A scenario is a path through sequence diagram
- Many different scenarios may be associated with a sequence diagram
- using the user tasks described in the use-cases and building the test cases from the tasks and their variants
- uncovers errors that occur when any actor interacts with the OO software
- concentrates on what the use does, not what the product does
- you can get a higher return on your effort by spending more time on reviewing the use-cases as they are created, than spending more time on use-case testing

OO Test Design Issues

- White-box testing methods can be applied to testing the code used to implement class operations, but not much else
- Black-box testing methods are appropriate for testing OO systems
- Object-oriented programming brings additional testing concerns
  - classes may contain operations that are inherited from super classes
  - subclasses may contain operations that were redefined rather than inherited
  - all classes derived from an previously tested base class need to be thoroughly tested