1 UNIFIED PROCESS AND USE CASE DIAGRAMS

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

1.1 Introduction to OOAD With OO Basics

In software engineering, a **software development process** is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a **software development life cycle**.

1.1.1 Two approaches

- Traditional Approach
- Objected-Oriented Approach

Traditional approach

- Collection of programs or functions.
- A system that is designed for performing certain actions.
- Algorithms + Data Structures = Programs.
- Software Development Models (Waterfall, Spiral, Incremental, etc.)

Object oriented approach

- OO development offers a different model from the traditional software - **based on functions and procedures.**
- Software is a collection of discrete object that encapsulate their data as well as the functionality.
- Each object has attributes (properties) and method (procedures).
- Software by building self-contained modules or objects that can be easily **replaced, modified and reused.**
- Objects grouped in to classes and objects are responsible for itself.

1.1.2 Benefits of Object Orientation

- Faster development,
- Reusability
- Increased quality
- modeling the real world and provides us with the stronger equivalence of the real world_s entities (objects).
- Raising the level of abstraction to the point where application can be implemented in the same terms as they are described.
- ease of description

- extensibility
- readability
- computational efficiency and
- ability to maintain

1.1.3 **Why Object Orientation?**

- ✓ OO Methods enables to develop **set of objects that work together**→ software → similar to traditional techniques.
- ✓ It adapts to
  - ➢ Changing requirements
  - ➢ Easier to maintain
  - ➢ More robust
  - ➢ Promote greater design
  - ➢ Code reuse Other uses:
- ✓ Higher level of abstraction
- ✓ Seamless transition among different phases of software development.
- ✓ Encouragement of good programming technique. ✓ Promotion of reusability.

1.1.4 **Examples of object oriented systems**

1. In OO system, ―everything is object‖.
2. A spreadsheet cell, bar chart, title in bar chart, report,
3. Numbers, arrays, records, fields, files, forms, an invoice, etc.
4. A window object is responsible for things like opening, sizing, and closing itself.
5. A chart object is responsible for things like maintaining data and labels even for drawing itself.

1.1.5 **Objects and classes**

The concepts of objects and classes are intrinsically linked with each other and form the foundation of object– oriented paradigm.

Object

An object is a real world element in an object-oriented environment that may have a physical or a conceptual existence.

Each object has:

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

The term object was first formally utilized in the Simula language to simulate some aspect of reality. The term **object** means a combination of **data** and **logic** that represents some real world entity. Attributes or properties describe object_s state (data) and methods (properties or functions) define its behavior.

- An object is an entity.
  - It knows things (has attributes)

- It does things (provides services or has methods)

Object's Attributes

- ✓ Attributes represented by data type.
- ✓ They describe objects states.
- ✓ In the Car example the car's attributes are: color, manufacturer, cost, owner, model, etc.

Object's Methods

- ✓ Methods define objects behaviour and specify the way in which an Object's data are manipulated.
- ✓ In the Car example the car's methods are: drive it, lock it, tow it, carry passenger in it.

- Object is whatever an application wants to talk about
  - For example, Parts and assemblies might be objects of bill of material application.
  - Stocks and bonds might be objects of financial investment applications.

Objects are Grouped In Classes

- The role of a class is to define the attributes and methods (the state and behaviour) of its instances. Used to distinguish one type of object from the other.
- Set of objects, that shares common methods, structure, behaviour.
- Single object is simply an instance of class.
- The class car, for example, defines the property color. Each individual car (object) will have a value for this property, such as "maroon," "yellow" or "white."

Class

- A class represents a collection of objects having same characteristic properties that exhibit common behavior.
- Creation of an object as a member of a class is called instantiation.
- Thus, object is an instance of a class.
- The constituents of a class are:
- A set of attributes for the objects that are to be instantiated from the class. ▪ Attributes are often referred as class data
- A set of operations that portray the behavior of the objects of the class.

 Operations are also referred as functions or methods ○ **Example:** Circle – a class x, to the center a, to denote the radius of the circle ○ Some of its operations can be defined as follows: findArea(), method to calculate area findCircumference(), method to calculate circumference

1.1.6 **Encapsulation and Information Hiding**

Encapsulation or Information Hiding is a design goal of an object oriented system

Information Hiding

- **Information hiding** is the principle of concealing the internal data and procedures of an object and providing an interface to each object in such a way as to reveal as little as possible about its inner workings.

- A class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct access. This process of insulating an object‘s data is called data hiding or information hiding.

Encapsulation

- It is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside.
- It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Inheritance

- It is the property of object oriented systems that allows objetcs to be built from other objects.
- It is a relationship between classes where one class is the parent class of another (derived) class. The parent class also is known as the base class or superclass.
- Some object oriented systems permit a class to inherit its state (attributes) and behaviours from more than one superclass. This kind of inheritance is referred to as **Multiple Inheritance.**
- **Dynamic inheritance** allows object to change and evolve over time. Since base classes provide properties and attributes for objects, changing base classes changes the properties and attributes of a class.

1.1.7 **Polymorphism**

- Poly means _many‘ and morph means _form‘.
- Polymorphism means that the same operation may beahve differently on different classes. Booch defines polymorphism as the relationship of objects of many different classes by some common superclass thus any of the objects designates by this name is able to respond to some common set of operations in a different way.

1.1.8 **Object Relationships**

Association represents the relationship between objects and classes.

- Bidirectional traversed in both directions
- The direction implied by the name is the forward direction and the opposite direction is the inverse direction

Aggregation

Each object has an identity, one object can refer to other objects. This is known as aggregation where an attribute can be an object itself.

Composition, also known as **composite aggregation**, is a strong kind of whole –part aggregation and is useful to show in some models. It is an instance of a part belongs to only one composite instance.

Navigability is a property of the role that indicates that it is possible to navigate uni-directionally across the association from objects of the source to target class.

- Navigability implies visibility, usually attribute visibility
- A UML operation is a declaration, with a name, parameters, return type, exceptions list and possibly a set of constraints of pre and post conditions.

- A UML method is the implementation of an operation, if constraints are defined, the method must satisfy them.

1.1.9 **What is OOAD?**

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects.

Analysis- understanding, finding and describing concepts in the problem domain.
Analysis emphasizes an *investigation* of the problem and requirements, rather than a solution. For example, if a new computerized library information system is desired, how will it be used?

Design - understanding and defining software solution/objects that represent the analysis concepts and will eventually be implemented in code.

Design emphasizes a *conceptual solution* that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects. Ultimately, designs can be implemented.

OOAD - A software development approach that emphasizes a logical solution based on objects.

- ✓ Software development is dynamic and always undergoing major change.
- ✓ System development refers to all activities that go into producing information system solution.
- ✓ System development activities consist of :
  - system analysis,
  - modelling,
  - design,
  - implementation,
  - testing and maintenance.
- ✓ A software development methodology
  - series of processes
  - can lead to the development of an application.
- ✓ Practices, procedures, and rules used to develop software, totally based on system requirements

During **object-oriented analysis,** there is an emphasis on finding and describing the objects—or concepts—in the problem domain. For example, in the case of the library information system, some of the concepts include *Book*, *Library*, and *Patron*.
During **object-oriented design,** there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. For example, in the library system, a *Book* software object may have a *title* attribute and a *getChapter* method.

1.2 Unified 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.
- The detailed refinement of the unified process is Rational Unified Process or RUP ➢ The Most Important UP Idea: **Iterative Development**

1.2.1 **Unified process**

- Is an iterative process.
- Provides an example structure for how to do object oriented analysis and design.
- Is flexible.
- Can be applied to lightweight and agile approach.
- Combines iterative lifecycle and risk driven development into a cohesive and well documented process description.

Benefits of iterative development include:

- ✓ Less project failure, better productivity and lower defect rates
- ✓ early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)
- ✓ early visible progress
- ✓ early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
- ✓ managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
- ✓ the learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

Key concepts in the UP include:

- ✓ tackle high-risk and high-value issues in early iterations
- ✓ continuously engage users for evaluation, feedback, and requirements
- ✓ build a cohesive, core architecture in early iterations
- ✓ continuously verify quality; test early, often, and realistically
- ✓ apply use cases
- ✓ model software visually (with the UML)
- ✓ carefully manage requirements
- ✓ practice change request and configuration management

1.2.2 **The UP Phases**

A UP project organizes the work and iterations across four major phases:

1. Inception
2. Elaboration
3. Construction
4. Transition

Inception Phase: *Approximate vision, business case, scope, vague estimates* ➢ Inception is the smallest phase in the project, and ideally it should be quite short.

- If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

Goals for the Inception phase:

- Establish a justification or business case for the project.
- Establish the project scope and boundary conditions.
- Outline the use cases and key requirements that will drive the design tradeoffs outline one or more candidate architectures.
- Identify risks.
- Prepare a preliminary project schedule and cost estimate.
- The Lifecycle Objective Milestone marks the end of the Inception phase.

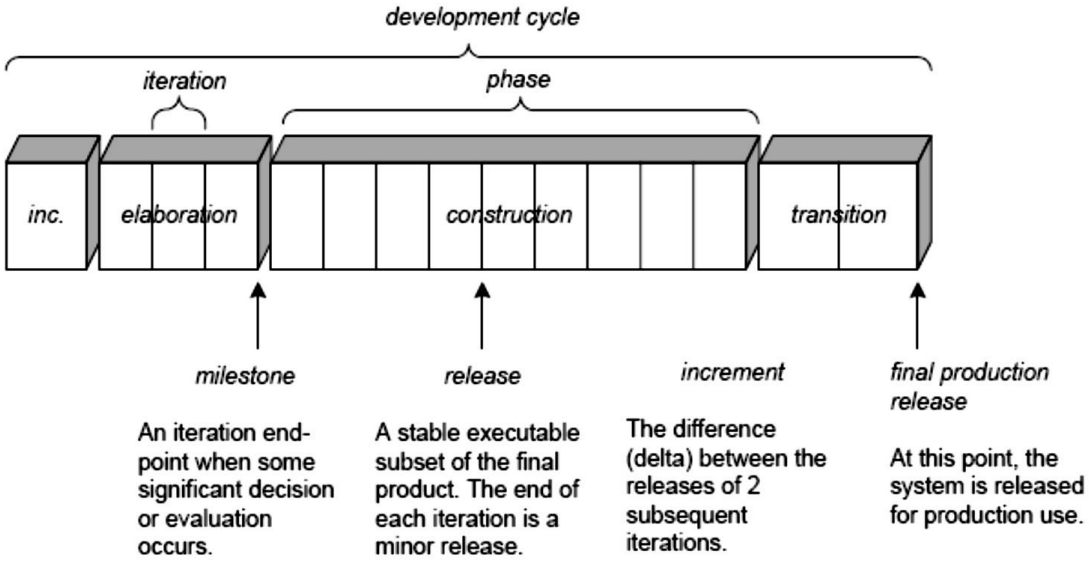


Figure: UP Phases

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

- Elaboration is the initial series of iterations during which:
  - the majority of requirements are discovered and stabilized
  - the major risks are mitigated or retired
  - the core architectural elements are implemented and proven
- During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements.
- The primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture.
- Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).
- The architecture is validated primarily through the implementation of an Executable Architecture Baseline.
- It is built in a series of small, time boxed iterations.

- By the end of the Elaboration phase, the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost.
- The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase.
- The plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.
- The Lifecycle Architecture Milestone marks the end of the Elaboration phase.

Construction Phase: *Iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.*

- Construction is the largest phase in the project.
- In this phase the remaining of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time boxed iterations.
- Each iteration results in an executable release of the software.
- It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration.
- The Initial Operational Capability Milestone marks the end of the Construction phase.
- Common UML (Unified Modeling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.

Transition Phase: *Beta tests, deployment*

- The final project phase is Transition. In this phase the system is deployed to the target users after beta test.
- Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations.
- The Transition phase also includes system conversions and user training.
- The Product Release Milestone marks the end of the Transition phase.

1.2.3 The UP Disciplines

- The UP describes work activities, such as writing a use case, within **disciplines** (originally called **workflows).**
- A discipline is a set of activities (and related artifacts) in one subject area, such as the activities within requirements analysis.
- In the UP, an **artifact** is the general term for any work product: code, Web graphics, database schema, text documents, diagrams, models, and so on. ➢ Some important artifacts:

1. **Business Modeling**
    a. When developing a single application, this includes domain object modeling.
    b. When engaged in large-scale business analysis or business process reengineering, this includes dynamic modeling of the business processes across the entire enterprise.

2. **Requirements** - Requirements analyses for an application, such as writing use cases and identifying non-functional requirements.
3. **Design** - All aspects of design, including the overall architecture, objects, databases, networking, and the like high-risk issues.

- In the UP, **Implementation** means programming and building the system, not deployment.
- The **Environment** discipline refers to establishing the tools and customizing the process for the project—that is, setting up the tool and process environment.

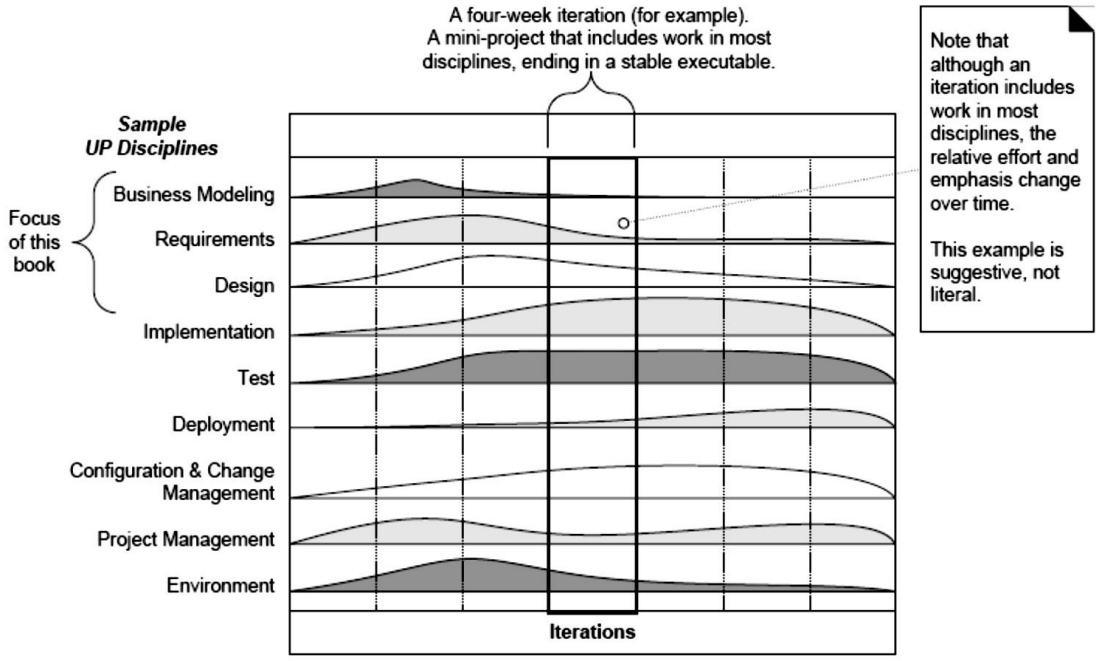


Figure: UP Disciplines

1.3 UML Diagrams

UML

The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of software systems.

1.3.1 Three ways to apply UML *1.*

UML as sketch:

- Informal and incomplete diagrams.
- Created to explore difficult parts of the problem.

2. UML as blueprint:

- Detailed design diagram.
- Used for better understanding of code.

3. UML as programming language:

- Complete executable specification of a software system in UML.

1.3.2 Three perspectives to apply UML

1. ***Conceptual perspective*:** Diagrams describe the things of real world.
2. ***Specification perspective:*** Diagrams describe software abstractions or components with specifications and interfaces.
3. ***Implementation perspective:*** Diagrams describe software implementation in a particular technology.

1.3.3 **UML Diagrams**

UML includes the following diagrams:

1. Use Case Diagrams
2. Class Diagrams
3. Object Diagrams
4. Package Diagrams
5. Interaction Diagrams Sequence
   Collaboration
6. Activity Diagrams
7. State Transition Diagrams
8. Component Diagrams
9. Deployment Diagrams

1. Use Case Diagram

- Use case diagrams are a set of use cases, actors and their relationships. They represent the use case view of a system. A use case represents a particular functionality of a system.

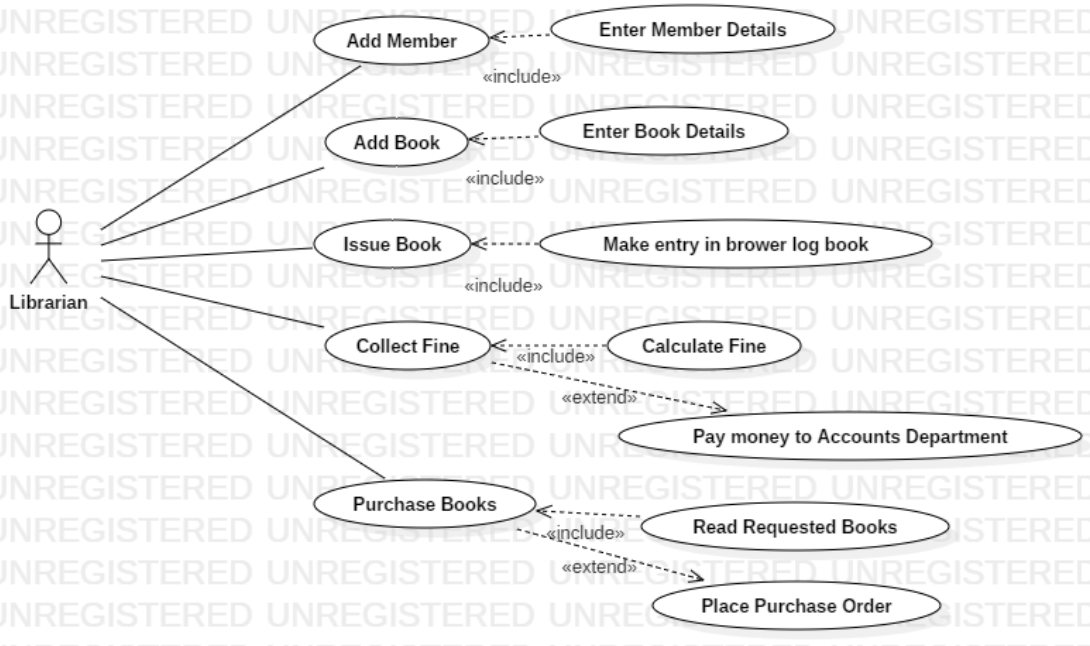


Figure: Use Case Diagram

2. **Class Diagram**

- Class diagrams are widely used to describe the types of objects in a system and their relationships.

- Class diagrams model class structure and contents using design elements such as classes, packages and objects.

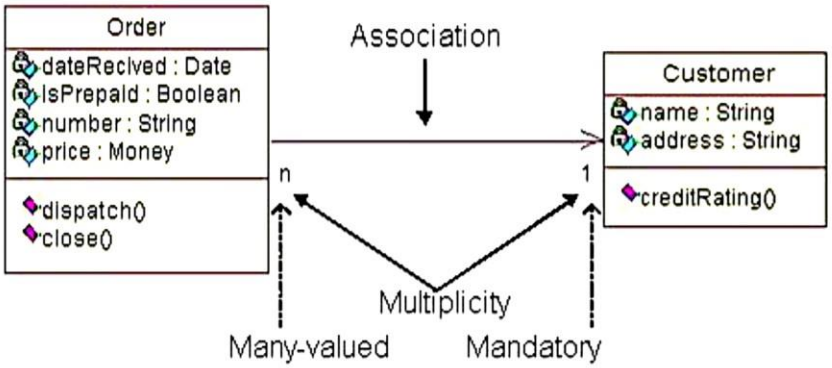


Figure: Class diagram

3. Object Diagram

- Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.
- Object diagrams represent an instance of a class diagram.
- Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

- The purpose of the object diagram can be summarized as − o Forward and reverse engineering. o Object relationships of a system o Static view of an interaction. o Understand object behaviour and their relationship from practical perspective

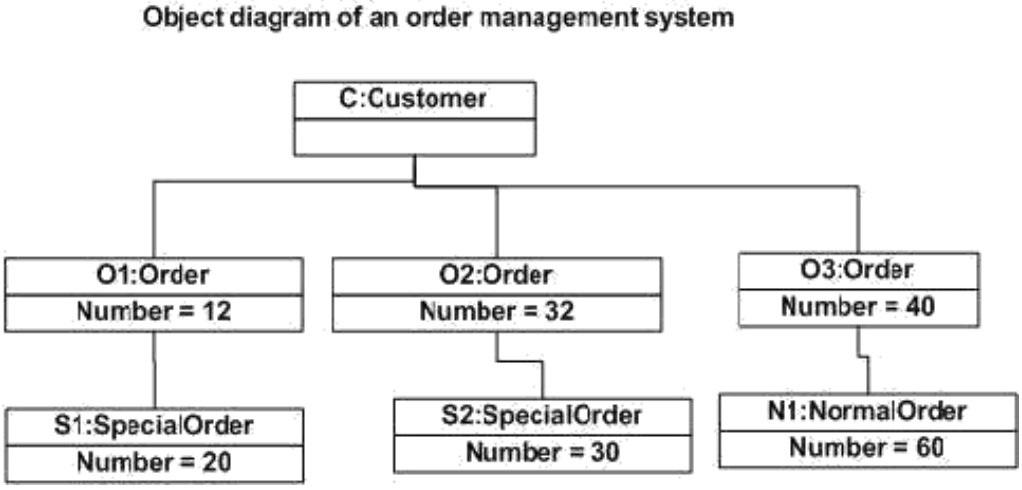


Figure: Object Diagram

4. Package Diagram

- A **package diagram** is a UML **diagram** composed only of **packages** and the dependencies between them.

- A **package** is a UML construct that enables you to organize model elements, such as use cases or classes, into groups.
- **Packages** are depicted as file folders and can be applied on any UML **diagram**.

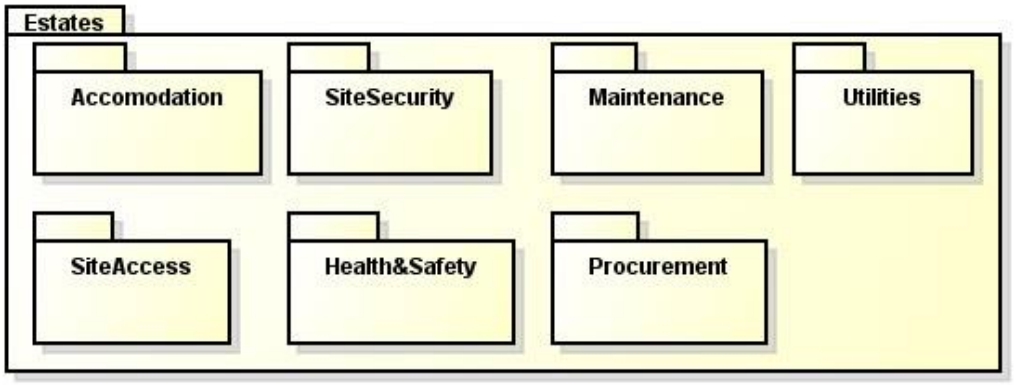


Figure: Package Diagram

5. Interaction Diagram

a. Sequence Diagram

- A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- Sequence diagrams describe how and in what order the objects in a system function.
- These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

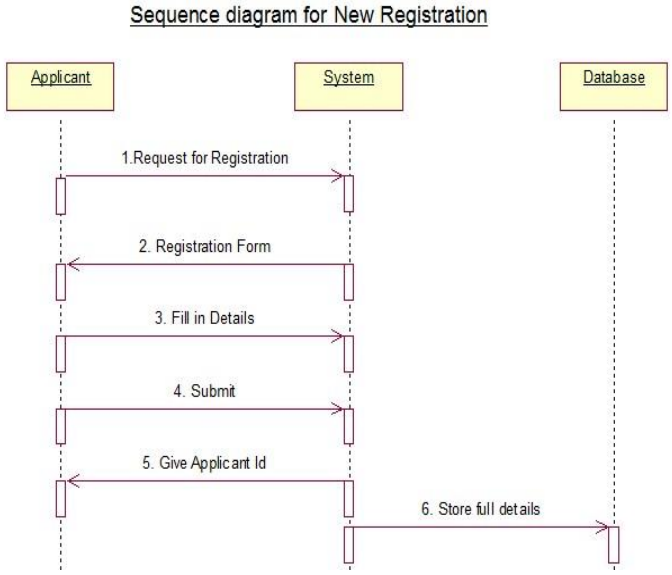


Figure: Sequence Diagram

b. Collaboration Diagram

- **Collaboration Diagram** represents the interaction of the objects to perform the behavior of a particular use case or a part of use case.
- The designers use the Sequence diagram and Collaboration Diagrams to define and clarify the roles of the objects that perform a particular flow of events of a use case.

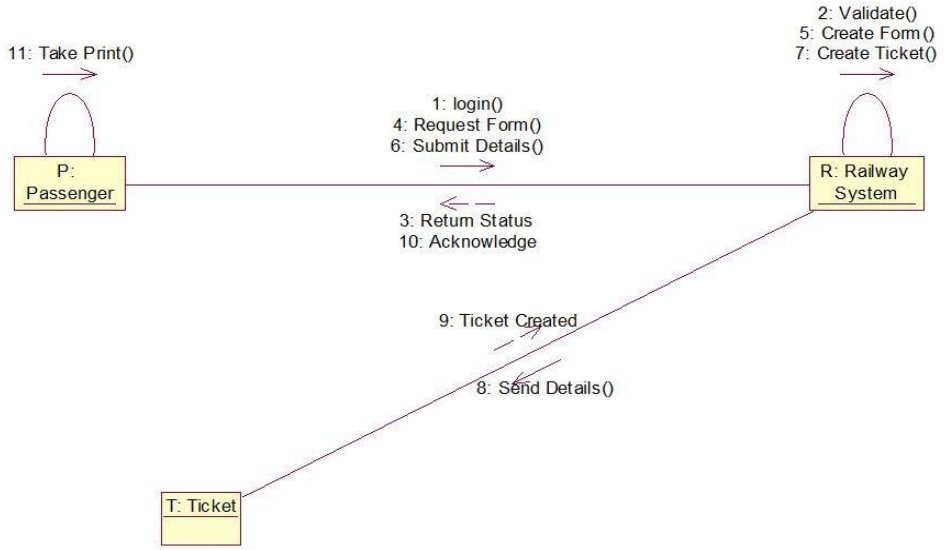


Figure: Collaboration Diagram

6. Activity Diagram

- Activity diagram describes the flow of control in a system. So it consists of activities and links. The flow can be sequential, concurrent or branched.
- Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.
- An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- We can depict both sequential processing and concurrent processing of activities using an activity diagram.
- They are used in business and process modeling where their primary use is to depict the dynamic aspects of a system.
- An activity diagram is very **similar to a flowchart**.

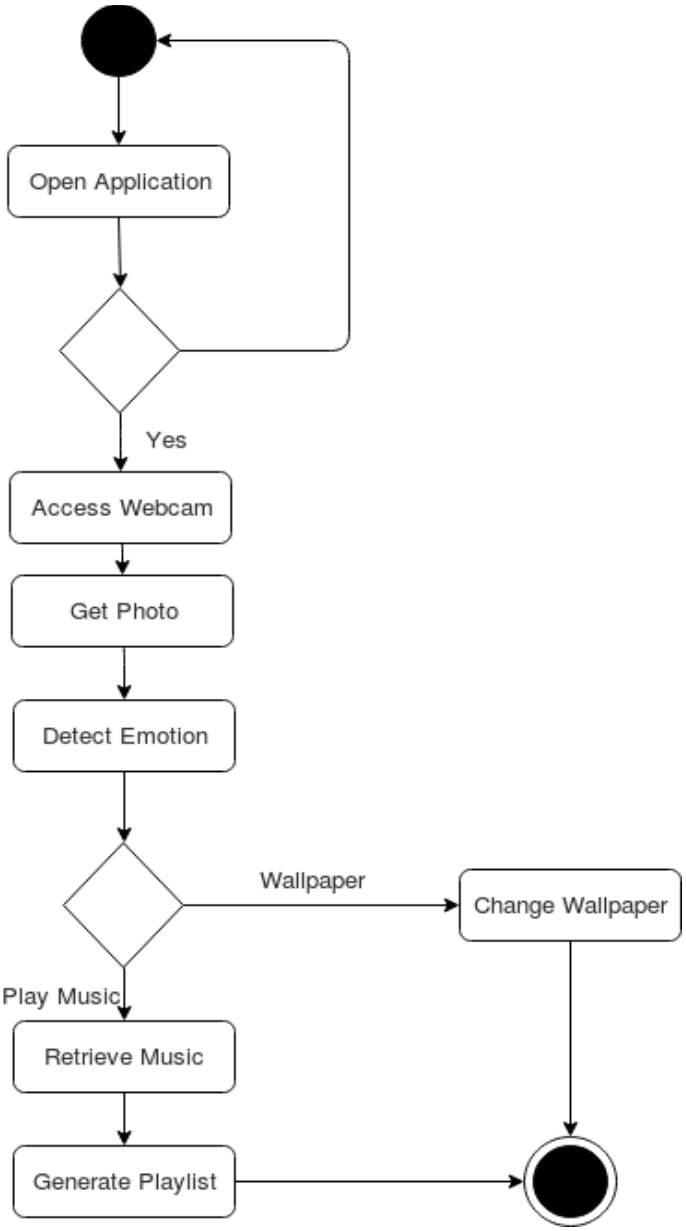


Figure: Activity Diagram

7. StateChart Diagram

- A UML state chart diagram illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event.
- A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time.
- It's a **behavioral** diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**.
- State chart diagram is used to visualize the reaction of a system by internal/external factors.

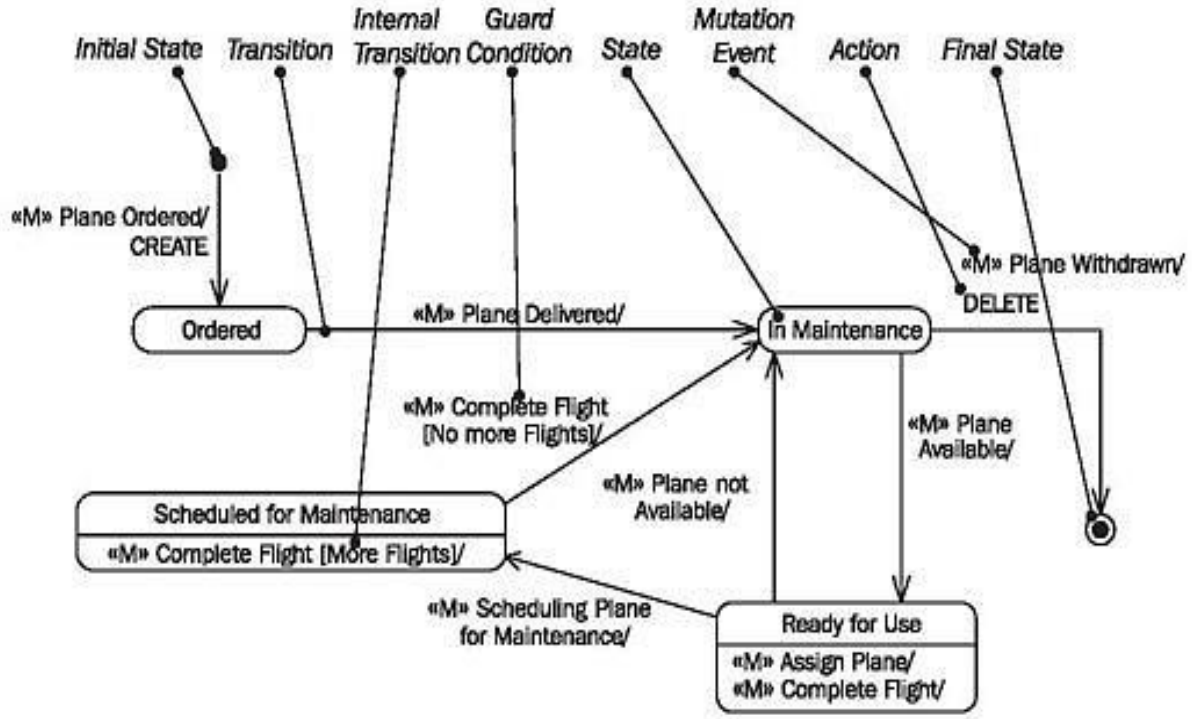


Figure: State chart diagram

8. Component Diagrams

- Component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.
- Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.
- The purpose of the component diagram can be summarized as − Visualize the components of a system.

  Construct executables by using forward and reverse engineering. Describe the organization and relationships of the components.

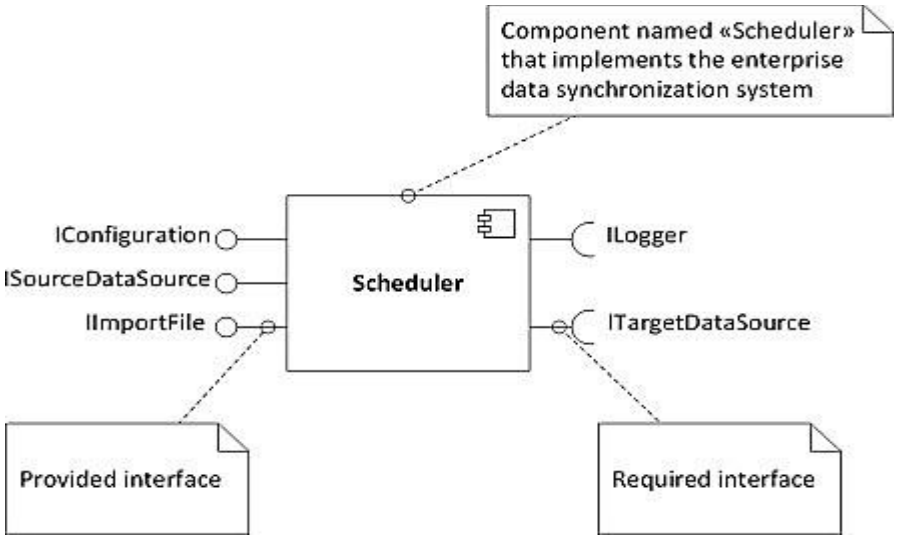


Figure: Component diagram

9. Deployment Diagrams

- Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.
- Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose:

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

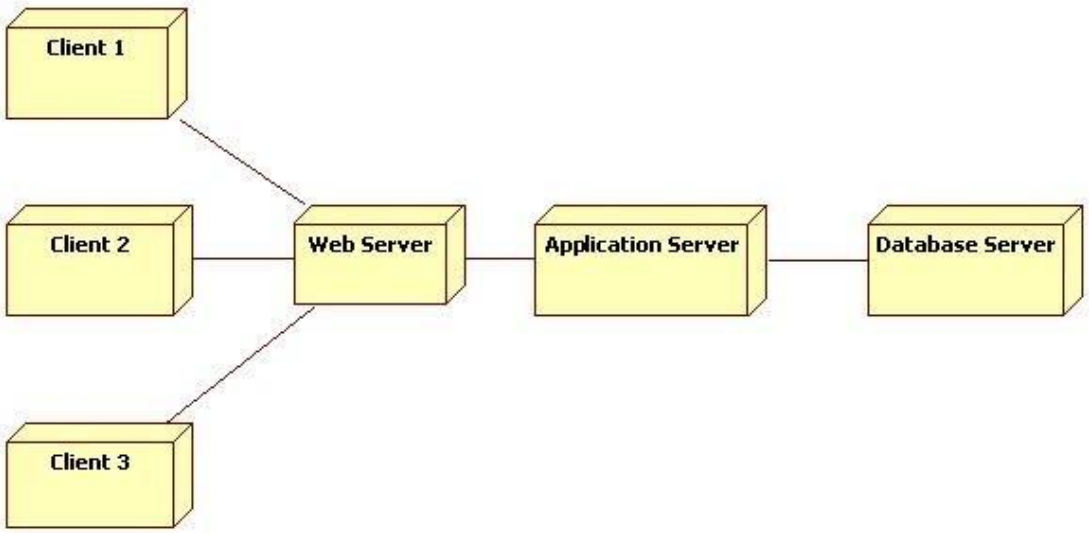


Figure: Deployment diagram

There are two types of diagrams in UML:

1. **Structure Diagrams –** Used to model the static structure of a system, for example- class diagram, package diagram, object diagram, deployment diagram etc.

2. **Behavior diagram –** Used to model the dynamic change in the system over time. They are used to model and construct the functionality of a system. So, a behavior diagram simply guides us through the functionality of the system using Use case diagrams, Interaction diagrams, Activity diagrams and State diagrams.

1.4 Use Case

- A **use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal.
- Use cases are requirements; primarily they are functional requirements that indicate what the system will do.
- Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing.
- However, the UML defines a use case diagram to illustrate the names of use cases and actors, and their relationships.
- A set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor.

1.5 Case Study - The Nextgen POS System

- A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store.
- It includes hardware components such as a computer and bar code scanner, and software to run the system.
- It interfaces to various service applications, such as a third-party tax calculator and inventory control.
- These systems must be relatively fault-tolerant; that is, even if remote services are temporarily unavailable (such as the inventory system), they must still be capable
- A POS system increasingly must support multiple and varied client-side terminals and interfaces.
- These include a thin-client Web browser terminal, a regular personal computer with something like a Java Swing graphical user interface, touch screen input, wireless PDAs, and so forth.
- Furthermore, we are creating a commercial POS system that we will sell to different clients with disparate needs in terms of business rule processing. Each client will desire a unique set of logic to execute at certain predictable points in scenarios of using the system, such as when a new sale is initiated or when a new line item is added. Therefore, we will need a mechanism to provide this flexibility and customization.
- Using an iterative development strategy, we are going to proceed through requirements, objectoriented analysis, design, and implementation.

1.6 Inception

- Inception phase is akin to a feasibility study to decide if it is even worth investing in a business process

- 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
- Artifacts of Inception phase:
  - ✓ **Vision and Business Case** - Describes the high-level goals and constraints, the business case, and provides an executive summary.
  - ✓ **Use-Case Model** - Describes the functional requirements, and related non-functional requirements.
  - ✓ **Supplementary Specification** - Describes other requirements.
  - ✓ **Glossary** - Key domain terminology.
  - ✓ **Risk List & Risk Management Plan** - Describes the business, technical, resource, schedule risks, and ideas for their mitigation or response.
  - ✓ **Prototypes and proof-of-concepts** - To clarify the vision, and validate technical ideas.
  - ✓ Iteration **Plan** - Describes what to do in the first elaboration iteration.
  - ✓ **Phase Plan & Software Development Plan** - Low-precision guess for elaboration phase duration and effort.
  - ✓ **Tools, people, education, and other resources** - Development Case A description of the customized UP steps and artifacts for this project. In the UP, one always customizes it for the project.
- **Requirements** are capabilities and conditions to which the system or the project must conform.

  Types of Requirements
  1. **Functional**—features, capabilities, security.
  2. **Usability**—human factors, help, documentation.
  3. **Reliability**—frequency of failure, recoverability, predictability
  4. **Performance**—response times, throughput, accuracy, availability, resource usage.
  5. **Supportability**—adaptability, maintainability, internationalization, configurability.
- The "+" in FURPS+ indicates ancillary and sub-factors, such as:
  a) **Implementation**—resource limitations, languages and tools, hardware, ...
  b) **Interface**—constraints imposed by interfacing with external systems.
  c) **Operations**—system management in its operational setting.
  d) **Packaging**
  e) **Legal**—licensing and so forth.

1.7 Use Case Modeling – Writing Requirements In Context

- This is the set of all use cases
- It is a model of the system's functionality and environment.
- They are stories of using a system to meet goals **Definitions:**

- **Use cases** are text documents. Doing use case work means to write text.
- An **actor** is something with behavior, 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 under discussion; 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 card transaction denial.

Actors

- An actor is anything with behavior, including the system under discussion (SuD) itself when it calls upon the services of other systems.
- Primary and supporting actors will appear in the action steps of the use case text.
- Actors are not only roles played by people, but organizations, software, and machines. ➢ There are three kinds of external actors in relation to the SuD:

Primary actor has user goals fulfilled through using services of the SuD.
For example: the cashier
Why identify? To find user goals, which drive the use cases.
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.
Why identify? To clarify external interfaces and protocols.
Offstage actor has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.
Why identify? To ensure that *all* necessary interests are identified and satisfied. Offstage actor interests are sometimes subtle or easy to miss unless these actors are explicitly named.

Use Cases Types and Formats

- **Black-box use cases** are the most common and recommended kind; they do not describe the internal workings of the system, its components, or design.
- By defining system responsibilities with black-box use cases, it is possible to specify *what* the system must do (the functional requirements) without deciding *how* it will do it (the design).

Black-box style	**Not**
The system records the sale.	The system writes the sale to a database. ...or (even worse): The system generates a SQL INSERT statement for the sale...

Formality Types

Use cases are written in different formats, depending on need. In addition to the black-box versus white-box *visibility* type, use cases are written in varying degrees *of formality:*

- **brief**—terse one-paragraph summary, usually of the main success scenario. The prior *Process Sale* example was brief.
- **casual**—informal paragraph format. Multiple paragraphs that cover various scenarios. The prior *Handle Returns* example was casual.
- **fully dressed**—the most elaborate. All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

Fully Dressed Example: Process Sale

Fully dressed use cases show more detail and are structured; they are useful in order to obtain a deep understanding of the goals, tasks, and requirements.

Use Case UC1: Process Sale

Primary Actor: Cashier Stakeholders
and Interests:

- Cashier:

Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.

- Salesperson:

Wants sales commissions updated.

- Customer:

Wants purchase and fast service with minimal effort. Wants proof of purchase to support returns.

-Company:

Wants to accurately record transactions and satisfy customer interests.

Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable.

Wants automatic and fast update of accounting and inventory.

- Government Tax Agencies:

Want to collect tax from every sale.

May be multiple agencies, such as national, state, and county.

- Payment Authorization Service:

Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated.
Accounting and Inventory are updated. Commissions recorded. Receipt is generated.
Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.

Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.

10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.

2a. System detects anomalies preventing recovery:

1. System signals error to the Cashier, records the error, and enters a clean state.
2. Cashier starts a new sale.

3a. Invalid identifier:

1. System signals error and rejects entry. 3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.

2.System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:

1. System records sale so that it is available for retrieval on any POS terminal.

4a.The system generated item price is not wanted (e.g., Customer complained about something and is offered a lower price):

1. Cashier enters override price.
2. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:

1. System restarts the service on the POS node, and continues. 1a. System detects that the service does not restart.

1. System signals error.
2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):

1. Cashier signals discount request.
2. Cashier enters Customer identification.
3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:

1. Cashier signals credit request.
2. Cashier enters Customer identification.
3. Systems applies credit up to price=0, and reduces remaining credit. 6a. Customer says they intended to pay by cash but don‘t have enough cash: 1a. Customer uses an alternate payment method.

1b. Customer tells Cashier to cancel sale. Cashier cancels sale on System.7a. Paying by cash:

1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.

3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:
1. Customer enters their credit account information.
2. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.

2a. System detects failure to collaborate with external system:
1. System signals error to Cashier.
2. Cashier asks Customer for alternate payment.
3. System receives payment approval and signals approval to Cashier.

3a. System receives payment denial:
1. System signals denial to Cashier.
2. Cashier asks Customer for alternate payment.
4. System records the credit payment, which includes the payment approval.
5. System presents credit payment signature input mechanism.
6. Cashier asks Customer for a credit payment signature. Customer enters signature. 7c. Paying by check...

7d. Paying by debit...

7e. Customer presents coupons:
1. Before handling payment, Cashier records each coupon and System reduces price as appropriate. System records the used coupons for accounting reasons.

1a. Coupon entered is not for any purchased item:
1. System signals error to Cashier.

9a.There are product rebates:1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible):
1. Cashier requests gift receipt and System presents it.

Special Requirements:
- Touch screen Ul on a large flat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90% of the time.
- Somehow, we want robust recovery when access to remote services such the inventory system is failing.
- Language internationalization on the text displayed.
- Pluggable business rules to be insertable at steps 3 and 7.

Technology and Data Variations List:

3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.

3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.

7a. Credit account information entered by card reader or keyboard.

7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

Frequency of Occurrence: Could be nearly continuous.

Open Issues:
- What are the tax law variations?

- Explore the remote service recovery issue.
- What customization is needed for different businesses?
- Must a cashier take their cash drawer when they log out?
- Can the customer directly use the card reader, or does the cashier have to do it?

The Two-Column Variation

Some prefer the two-column or conversational format, which emphasizes the fact that there is an interaction going on between the actors and the system.

Use Case UC1: Process Sale

Primary Actor: ... ... as before ... **Main Success Scenario:** Actor Action (or Intention) 1. Customer arrives at a POS checkout with goods and/or services to purchase. 2. Cashier starts a new sale. 3. Cashier enters item identifier. Cashier repeats steps 3-4 until indicates done.	System Responsibility 4. Records each sale line item and presents item description and running total. 5. System presents total with taxes calculated. 1. Handles payment. 9. Logs the completed sale and sends information to the external accounting (for all accounting and commissions) and inventory systems (to update inventory). System
6. Cashier tells Customer the total, and asks for payment. 7. Customer pays.	presents receipt.

Finding Primary Actors, Goals, and Use Cases

Use cases are defined to satisfy the user goals of the primary actors.

Basic procedure is:

1. **Choose the system boundary:** Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
2. **Identify the primary actors:** those that have user goals fulfilled through using services of the system.
3. **For each, identify their user goals**: Raise them to the highest user goal level that satisfies the EBP guideline.
4. **Define use cases that satisfy user goals:** name them according to their goal.

1.8 Relating Use Cases - Include, Extend and Generalization

The include Relationship

- This is the most common and important relationship.
- It is common to have some partial behavior that is common across several use cases.
- This is the most commonly used relationship which denotes that a given use case may include another. It is denoted by

<<include>>

- The use case at the arrow head position is the use case which is included by use case on the other side of the arrow.
- When there are multiple steps to carry out the single task then this task is divided into sub functions.
- For example, the description of paying by credit occurs in several use cases, including *Process Sale, Process Rental, Contribute to Lay-away Plan,* and so forth. Rather than duplicate this text, it is desirable to separate it into its own sub-function use case, and indicate its inclusion. This is simply refactoring and linking text to avoid duplication.

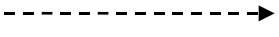
For example: **UC1:** Process Sale **Main**

Success Scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
7. Customer pays and System handles payment. **Extensions:**
7b. Paying by credit: Include *Handle Credit Payment*. *7c.*
Paying by check:Include Handle Check Pavment.

The extend Relationship

- Suppose a use case's text should not be modified (at least not significantly) for some reason. Perhaps continually modifying the use case with myriad new extensions and conditional steps is a maintenance headache, or the use case has been baselined as a stable artifact, and can't be touched. How to append to the use case without modifying its original text?
- The **extend** relationship provides an answer. The idea is to create an extending or addition use case, and within it, describe where and under what condition it extends the behavior of some base use case.
- Extension occur in two cases
  1. When some part of the use case is optional and we need to show separate behavior of the system.
  2. When we want to show a sub flow of the system when some specific conditions occur.



<<extend>>

- **Example:** ***online purchase system***, when user browses for the product, it may browse either by the product name or by product ID.

For example:

UC1: Process Sale (the base use case)

Extension Points: *VIP Customer,* step 1. *Payment,* step 7. Main Success **Scenario:**

1. Customer arrives at a POS checkout with goods and/or services to purchase.

…

7. Customer pays and System handles payment.

…

The generalize Relationship

- In this type of relationship there are two types of use case

1. Parent use case.
2. Child use case.

- A parent use case may be specialized into one or more child use cases that represent more specific forms of the parent.
- The child use cases inherit all the structure and behavior and relationships of parents.
- Generalization is used when we find two or more use cases having common structure and behavior and purpose.

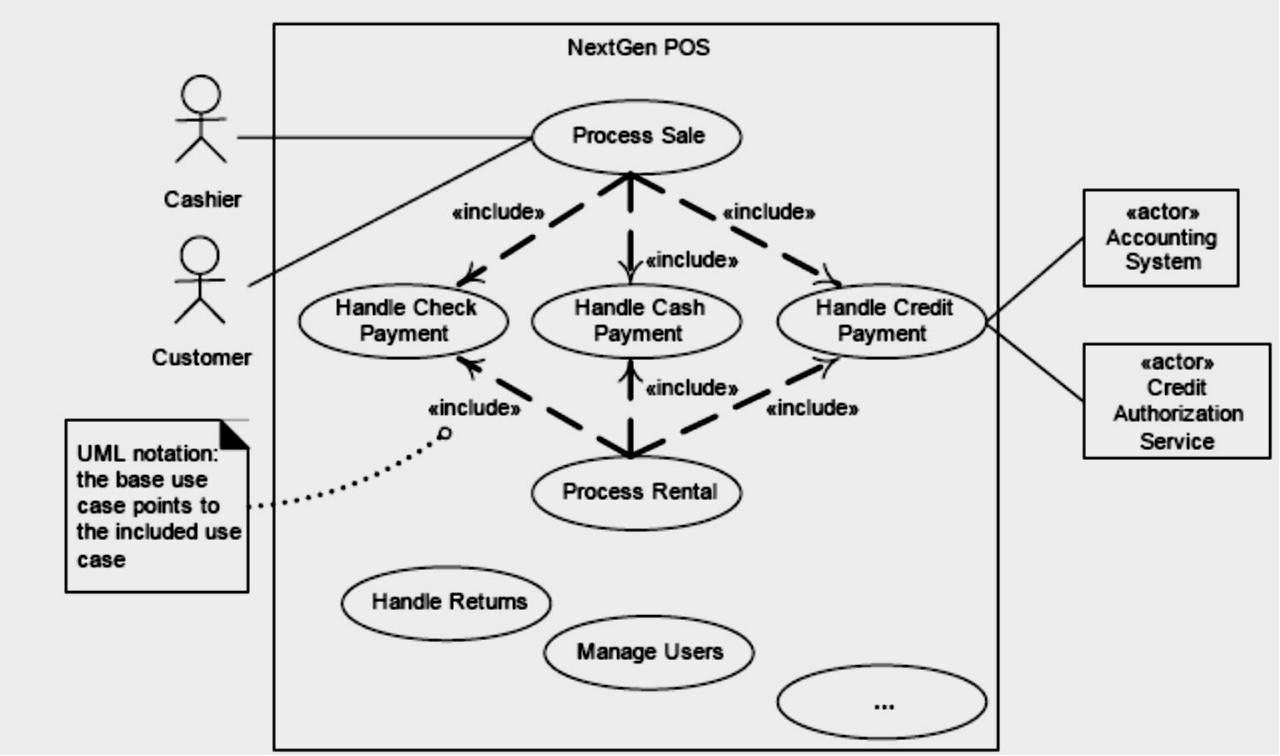


Figure: Use case include relationship in the Use-Case Model.

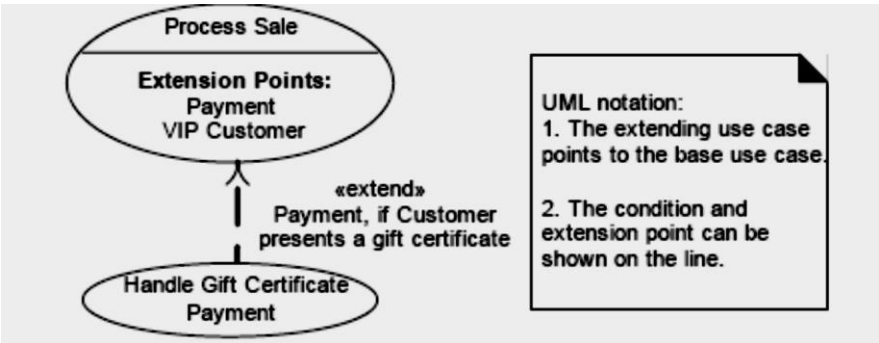


Figure: The extend Relationship

1.9 When To Use Use- Cases

- Use case diagrams are typically develop in early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases Developed by analysts together with domain experts ➢ Use cases are used to model the behavior of the system.

Part A

1. **Define an object. Identify the probable attributes that will be modeled in a Library database for the object Book. [Dec 2017] [Nov 2012]**
   - ➢ An object is a combination of data and logic; the representation of some real world entity **Example:** Saab automobile where **Saab** is an object.
   - ➢ The data part of this object will be car_s name, color, price etc.
   - ➢ The logic part can be collection of programs (show mileage, change mileage, stop.
   - ➢ Object is a tangible entity that exhibits some well-defined behaviour.

   Attributes for Object BOOK in a Library database:

   Book-ID
   Book-Name
   Author
   Publisher-Name
   Edition
   Issue –Date
   Return-Date

2. **What is OO system development methodology? [Nov 2012]**

Object oriented system development methodology is a way to develop software by **building selfcontained modules** or objects that can be easily replaced, modified and reused.

3. **Why do we need object oriented systems development? [Dec 2012]**
   1. Higher level of abstraction
   2. Seamless transition between different phases of software development
   3. Encouragement of good programming techniques
   4. Promotion of reusability

4. **What is analysis and design? [May 2013] (or) What is design? Give an example. [Dec 2017]**
   Analysis emphasizes an **investigation of the problem and requirements**, rather than a solution. For example, if a new computerized library information system is desired, how will it be used? **Design** emphasizes a **conceptual solution** that fulfils the requirements, rather than its implementation. For example, a description of a database schema and software objects. Ultimately, designs can be implemented.

5. What is OO analysis and design? Define OOAD.[May 2017][Nov 2013][May2014][Nov 2014] [May 2015][May 2011]

During **object-oriented analysis,** there is an emphasis on **finding and describing the objects** or concepts in the problem domain.

Example: In library information system, *Book*, *Library*, and *Patron*.

During **object-oriented design**, there is an emphasis on **defining software objects** and how they collaborate to fulfil the requirements.

Example: In library system, a *Book* software object may have a *title* attribute and a *getChapter* method.

6. What is encapsulation?

It is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside.

7. Define information hiding.

Information hiding is the principle of concealing the internal data and procedures of an object and providing an interface to each object in such a way as to reveal as little as possible about its inner workings.

8. What is inheritance?

- It is the property of object oriented systems that allows objetcs to be built from other objects.
- It is a relationship between classes where one class is the parent class of another (derived) class. The parent class also is known as the base class or superclass.

9. What are the types of inheritance?

- Single Inheritance
- Multiple Inheritance
- Dynamic Inheritance

10. Define Polymorphism.

- Polymorphism means that the same operation may beahve differently on different classes.
- Booch defines polymorphism as the relationship of objects of many different classes by some common superclass thus any of the objects designates by this name is able to respond to some common set of operations in a different way.

11. Differentiate OO approach and traditional approach.

Traditional approach	OO approach
Focus in functions of the system	Focus on the object which covers both functions and data
Develop software by functions and procedures	Develop software by creating self contained modules or objects that can be easily modified and reused

12. What do you mean by unifies process? [Dec 2018]

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. The detailed refinement of the unified process is Rational Unified Process or RUP. The Most Important UP Idea: **Iterative Development**

13. List the four phases in UP.

1. Inception
2. Elaboration
3. Construction
4. Transition

14. What are the benefits of iterative development?

- Less project failure, better productivity and lower defect rates.
- Early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth).
- Early visible progress.
- Early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders.

15. What is UML? [May 2012, 2013, Dec 2014, Dec 2017] (or) What is Unified Modeling Language? [May 2019]

The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems. UML and the modeling process can help improve quality, completeness and scalability and reduce production time in many software projects.

16. What are the primary goals in design of UML?[Dec 2016]

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.

17. What are the three ways and perspectives to apply UML? [Dec 2015] (or) State and explain the three perspectives of using UML.[Dec 2016, May 2017, Dec 2017]

Three ways to apply UML:

UML as sketch:

- Informal and incomplete diagrams.
- Created to explore difficult parts of the problem.

UML as blueprint:

- Detailed design diagram.
- Used for better understanding of code.

UML as programming language:

- Complete executable specification of a software system in UML.

Three perspectives to apply UML:

- *Conceptual perspective*: Diagrams describe the things of real world.
- *Specification perspective:* Diagrams describe software abstractions or components with specifications and interfaces.
- *Implementation perspective:* Diagrams describe software implementation in a particular technology.

18. What is the purpose of extends and include relationship in use case diagram? [May 2017] The <<Include>> relationship is used to include common behavior from an included use case into a base use case in order to support re-use of common behavior. The extend relationships are important because they show optional functionality or system behavior.

19. List the principles of modeling in UML. [May 2019] Four Principles of Modeling:

1. The models we choose have a profound influence on the solution we provide
2. Every model may be expressed at different levels of abstraction
3. The best models are connected to reality
4. No single model is sufficient, a set of models is needed to solve any nontrivial system

20. What is the need for modeling? [May 2014]

The main reason for modeling is the reduction of complexity. Also it:

- ✓ Provide structure for problem solving
- ✓ Experiment to explore multiple solutions
- ✓ Furnish abstractions to manage complexity
- ✓ Reduce time
- ✓ Decrease development cost ✓ Manage the risk of mistakes

21. Define POS system.

- ✓ A POS system is a computerized application used (in part) to record sales and handle payments
- ✓ It is typically used in a retail store.
- ✓ It includes hardware components such as a computer and bar code scanner, and software to run the system.

22. List out the components of a POS system. [May 2018]

a. Monitor
b. Barcode Scanner
c. Magnetic stripe readers
d. Printer

23. What is Inception? [May 2011] [Dec 2017]

Inception is the first phase in the UP. It is the **feasibility phase** where enough investigation is done to support a decision to continue or stop. It includes:

- approximate vision

- business case
- scope
- Vague estimates.

24. What are the UP disciplines?

There are several disciplines in UP. They are:

- ✓ Business Modeling
- ✓ Requirements
- ✓ Design
- ✓ Implementation
- ✓ Test
- ✓ Deployment
- ✓ Configuration & Change Management
- ✓ Project Management
- ✓ Environment

25. What are the advantages of inception?

- ✓ Estimation or plans are expected to be reliable.
- ✓ After inception, design architecture can be made easily because all the use cases are written in detail.

26. Define use case and actor. What do you mean by actor?[May 2018]

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

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

27. What are the uses of use case?

✓ Use cases are very good way to keep everything simple. ✓

It gives the user‘s goal and perspectives.

28. Define scenario.

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.

29. What is use case model?

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

The Use-Case Model may optionally include a UML use case diagram to show the names of use cases and actors, and their relationships.

30. What are the types of actors?

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

1. **Primary actor** has user goals fulfilled through using services of the SuD. For example, the cashier.
   Why identify? To find user goals, which drive the use cases.
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.
   Why identify? To clarify external interfaces and protocols.
3. **Offstage actor** has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.
   Why identify? To ensure that all necessary interests are identified and satisfied.

31. List the types of user interface free style.

- ✓ Essential style writing
- ✓ Concrete style writing

32. Define terse use case.

Terse use case is use case where noisy words or fuzzy words are avoided or deleted

33. How to Find Use Cases?

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.
4. Define use cases that satisfy user goals; name them according to their goal. Usually, usergoal level use cases will be one-to-one with user goals, but there is at least one exception, as will be examined.

34. What tests can help find useful use cases? [May 2016] There are three types of test used in use case.

- ✓ Boss test.
- ✓ EBP test. ✓ Size test.

35. Define EBP test.

Elementary business process is related to business engineering. The task is performed by one person in one place at one time in response to business events which gives measurable business value

36. List out the use case diagram components.

A use case diagram contains four components:

1. Boundary
2. Actors
3. Use case
4. Relationship

37. List the relationships used in use cases. [May 2014][May 2012] The relationships used in use cases are:

1. Include relationship.

2. External relationship
3. Generalization.

38. List out the steps for finding use cases. [Dec 2012]

The following are the steps to identify use cases:

Step 1: Identify candidate system actors.
Step 2: Identify the goals of the actors.
Step 3: Identify the candidate use cases.
Step 4: Identify the start point for each use case.
Step 5: Identify the end point for each use case. **Step 6**: Refine and scope units of interaction.

39. What is the significance of UML? [Dec 2018]

UML is for visualizing, specifying, constructing and documenting the components of software and non-software systems. UML notations are the most important elements in modeling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless, unless its purpose is depicted properly.

40. Outline the purpose of using use cases, to describe requirements. [Dec 2017] Purpose of using use cases:

- Used to gather requirements of a system
- Used to get an outside view of a system
- Identify external and internal factors influencing the system
- Show the interaction among the requirements through actors

41. What are the types of UML Diagrams?

10. Use Case Diagrams
11. Class Diagrams
12. Object Diagrams
13. Package Diagrams
14. Interaction Diagrams Sequence
    Collaboration
15. Activity Diagrams
16. State Transition Diagrams
17. Component Diagrams
18. Deployment Diagrams

42. Name the two types of UML interaction diagrams. [Nov 2017] [May 2019]

Interaction Diagrams are classified into two types. They are

- Sequence Diagram
- Collaboration Diagram

43. Differentiate method and message in object. [Dec 2015]

Method	Message
Methods are similar to functions, procedures or subroutines in more traditional programming languages	Message essentially are non-specific function calls
Method is an implementation	Message is an instruction
In an object oriented system, a method is invoked by sending an object a message	An object understands a message when it can match the message to a method that has the same name as the message.

PART B

1. **Present an outline of object oriented analysis and object oriented design.[7m] [Dec 2017]**
2. **Why the Unified process has emerged as a popular and effective software development process? [6M-Dec-2017]**

3. **Explain different phases of Unified process with suitable examples. [Nov2013, May2014, Nov 2014, May 2012, Nov 2011, Dec 2015, May 2011, May 2019, Dec 2017, May 2017]**
4. **What do you mean by unified process in OOAD? Explain the phases with suitable diagrams. [Dec 2011, Dec 2012, , Dec 2016, May 2017, May 2019]**
5. **Explain the conceptual model of the UML in detail? Explain various common mechanisms used in UML. [May 2019]**
6. **Explain about unified process phases [Nov 2015]**
7. **Explain the purpose of UML diagram and list out the types of UML diagrams with example. [May 2014] [May 2017]**
8. **Explain use case modeling with example.[May 2019-7M] (or) Explain with an example, how use case modeling is used to describe functional requirements. Identify the actors, scenario and use case for the example. [May 2014] [Nov 2015][Nov 2016]**
9. **Design a use case model for a case study of your choice. [Nov 2015]**
10. **Explain in detail about use case diagrams. [May 2018-6M]**
11. **Draw the use case diagram for the process sale and specify actor, use case and scenario.**
12. **For POS application, create a use case diagram that shows include, extend and generalization relationships. [Nov 2017]**
13. **Mention when to use use-cases.**
14. **Explain in detail about various relationships used in use case diagram.**