E-notes Software Engineering

By Sagar N. Gharate

Syllabus

References:

1. Software Engineering : A Practitioner's Approach - Roger S. Pressman, McGraw hill(Eighth Edition) ISBN-13: 978-0-07-802212-8, ISBN-10: 0-07-802212-6
2. A Concise Introduction to Software Engineering - PankajJalote, Springer ISBN: 978-1-84800-301-9
3. The Unified Modeling Language Reference Manual - James Rambaugh, Ivar Jacobson, Grady Booch ISBN 0-201-30998-X

Chapter 4

Requirements Modeling

Requirements Modeling

Requirements modeling in software engineering are essentially the planning stage of a software application or system. Generally, the process will begin when a business or an entity (for example, an educational institution) approaches a software development team to create an application or system from scratch or update an existing one. Requirements modeling comprises several stages, or 'patterns': scenario-based modeling, data modeling, flow-oriented modeling, class-based modeling and behavioral modeling. Each of these stages/patterns examines the same problem from a different perspective.

Requirements modeling uses a combination of text and diagrammatic forms to depict requirements in a way that is relatively easy to understand To validate software requirements, you need to examine them from a number of different points of view. In this presentation we'll consider requirements modeling from three different perspectives: scenario- based models, data(information) models, and class-based models. Each represents requirements in a different "dimension," thereby increasing the probability that errors will be found, that inconsistency will surface, and that omissions will be uncovered.

The requirements model as a bridge between the system description and the design model. Overall Objectives and Philosophy requirements modeling, your primary focus is on what, not how. What user interaction occurs in a particular circumstance, what objects does the system manipulate, what functions must the system perform, what behaviors does the system exhibit, what interfaces are defined, and what constraints apply The requirements model must achieve three primary objectives: (1) to describe what the customer requires, (2) to establish a basis for the creation of a software design, and (3) to define a set of requirements that can be validated once the software is built.

Identifying Requirements

Requirements in this context are the conditions that a proposed solution or application must meet in order to solve the business problem. Identifying requirements is not an exclusively technical process, and initially involves all the stakeholders, like the representatives of the entity that has commissioned the software project, who may not necessarily be from a technical background, as well as the software developers, who are not necessarily the technical team. Together, they discuss and brainstorm about the problem, and decide what functions the proposed application or system must perform in order to solve it.

Functional vs. Non-Functional Requirements

A **functional requirement** specifies something that the application or system should do. Often, this is defined as a behavior of the system that takes input and provides output. For example, a traveller fills out a form in an airline's mobile application with his/her name and passport details (input), submits the form, and the application generates a boarding pass with the traveller's details (output).

Non-functional requirements, sometimes also called quality requirements, describe how the system should be, as opposed to what it should do. Non-functional requirements of a system include performance (e.g., response time), maintainability and scalability, among many others. In the airline application example, the requirement that the application must display the boarding pass after a maximum of five seconds from the time the traveller presses the 'submit' button would be a non-functional requirement.

Introduction to UML

The requirements model as a bridge between the system description and the design model. Overall Objectives and Philosophy requirements modeling, your primary focus is on what, not how. What user interaction occurs in a particular circumstance, what objects does the system manipulate, what functions must the system perform, what behaviors does the system exhibit, what interfaces are defined, and what constraints apply The requirements model must achieve three primary objectives: (1) to describe what the customer requires, (2) to establish a basis for the creation of a software design, and (3) to define a set of requirements that can be validated once the software is built.

The requirements model as a bridge between the system description and the design model. Overall Objectives and Philosophy requirements modeling, your primary focus is on what, not how. What user interaction occurs in a particular circumstance, what objects does the system manipulate, what functions must the system perform, what behaviors does the system exhibit, what interfaces are defined, and what constraints apply The requirements model must achieve three primary objectives: (1) to describe what the customer requires, (2) to establish a basis for the creation of a software design, and (3) to define a set of requirements that can be validated once the software is built.

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously making efforts to create a truly industry standard.

- UML stands for **Unified Modeling Language**.
- UML is different from the other common programming languages such as C++, Java, COBOL, etc.
- UML is a pictorial language used to make software blueprints.
- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.
- Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization, UML has become an OMG standard.

Do we really need UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non-programmers essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

Goals of UML

- *A picture is worth a thousand words*, this idiom absolutely fits describing UML. Object-oriented concepts were introduced much earlier than UML. At that point of time, there were no standard methodologies to organize and consolidate the object-oriented development. It was then that UML came into picture.

- There are a number of goals for developing UML but the most important is to define some general purpose modeling language, which all modelers can use and it also needs to be made simple to understand and use.
- UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system. The system can be a software or non-software system. Thus it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system.
- In conclusion, the goal of UML can be defined as a simple modeling mechanism to model all possible practical systems in today's complex environment.

A Conceptual Model of UML

To understand the conceptual model of UML, first we need to clarify what is a conceptual model? and why a conceptual model is required?

- A conceptual model can be defined as a model which is made of concepts and their relationships.
- A conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other.

As UML describes the real-time systems, it is very important to make a conceptual model and then proceed gradually. The conceptual model of UML can be mastered by learning the following three major elements which we studied in previous chapter.

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

Use Case Diagram/Model

- 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. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers.
- These internal and external agents are known as actors. Use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.
- Hence to model the entire system, a number of use case diagrams are used.

Purpose of Use Case Diagrams

- The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.
- Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analysed to gather its functionalities, use cases are prepared and actors are identified.
- When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows -

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Shows the interactions among the requirements are actors.

How to Draw a Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analysed, the functionalities are captured in use cases.

We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

- Functionalities to be represented as use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram

- The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.

- Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.
- Use notes whenever required to clarify some important points.

Following is a sample use case diagram representing the order management system. Hence, if we look into the diagram then we will find three use cases (Order, SpecialOrder, and NormalOrder) and one actor which is the customer.

The SpecialOrder and NormalOrder use cases are extended from *Order* use case. Hence, they have extended relationship. Another important point is to identify the system boundary, which is shown in the picture. The actor Customer lies outside the system as it is an external user of the system.

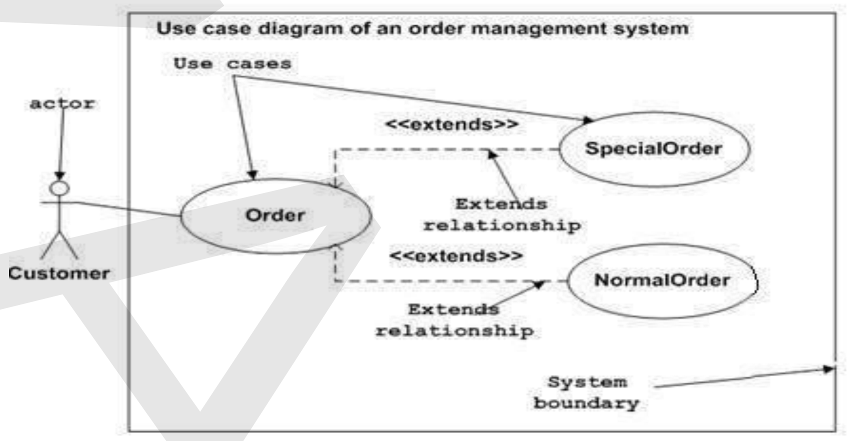


Figure: Sample Use Case diagram

Where to Use a Use Case Diagram?

- As we have already discussed there are five diagrams in UML to model the dynamic view of a system. Now each and every model has some specific purpose to use. Actually these specific purposes are different angles of a running system.
- To understand the dynamics of a system, we need to use different types of diagrams. Use case diagram is one of them and its specific purpose is to gather system requirements and actors.
- Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.
- These diagrams are used at a very high level of design. This high level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, and exceptions. These extra elements are used to make test cases when performing the testing.
- Although use case is not a good candidate for forward and reverse engineering, still they are used in a slightly different way to make forward and reverse engineering. The same is true for reverse engineering. Use case diagram is used differently to make it suitable for reverse engineering.
- In forward engineering, use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application.

Use case diagrams can be used for -

- Requirement analysis and high level design.
- Model the context of a system.
- Reverse engineering.
- Forward engineering

Example: ATM use cases:

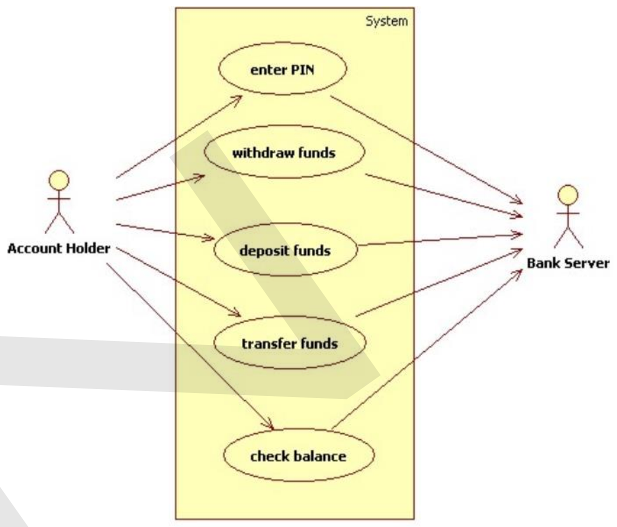


Fig No. 1

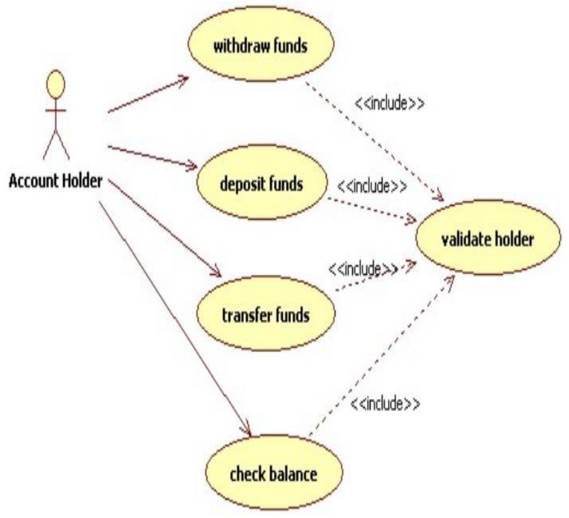


Fig No. 2

Fig No.1- Primary actors usually initiate a use case. This is shown by an arrow running from the actor to the use case. Secondary actors respond to use cases.

Fig No.2- Sometimes an action sequence appears as a subsequence in many scenarios. In this case we can treat the subsequence as a separate use case and include it in other use cases. For example, most of the ATM use cases begin by validating the holder: We can create a "Validate User" use case and allow other use cases to explicit invoke it. This is modelled by the "include" arrow.

Class Diagram/Model

- Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.
- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.
- Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

- The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.
- UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as -

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

How to Draw a Class Diagram?

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram -

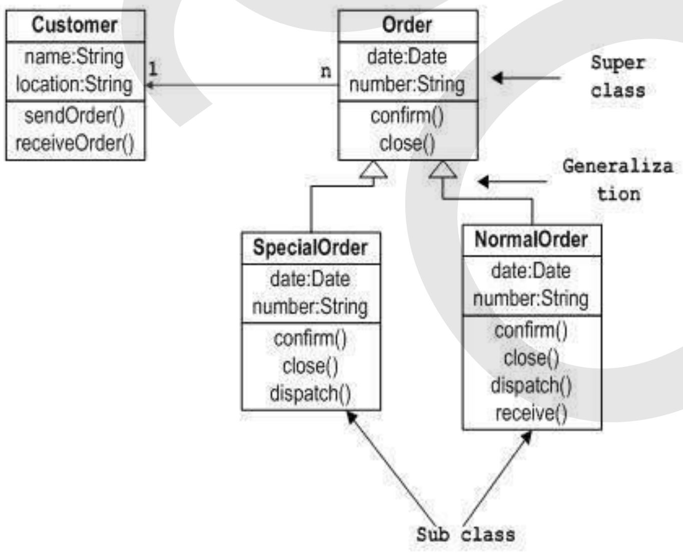
- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

The following class diagram has been drawn considering all the points mentioned above.

Sample Class Diagram



Where to Use Class Diagrams?

- Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.
- Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.
- Generally, UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception.
- Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose.

In a nutshell it can be said, class diagrams are used for -

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

Example: ATM Class Diagram:

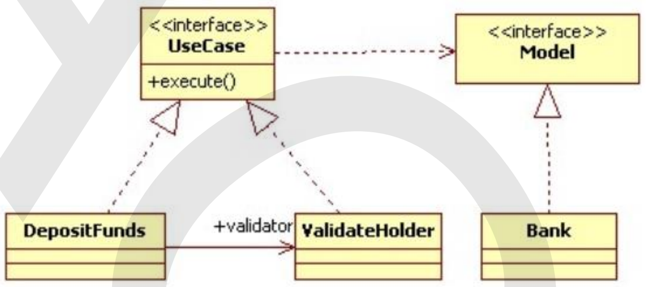


Fig No. 3

Fig No. 3- An included use case is often implemented as an explicit call to another use case. For example, assume all use cases are implemented as instances of classes implementing a use case controller interface

Behavioral Modeling/Diagram

Interaction Modeling/Diagrams

Interaction diagrams, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled. For example, the sequence diagram shows how objects communicate with each other regarding a sequence of messages.

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

This ***interactionbehavior*** is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration/Communication diagram**. The basic purpose of both the diagram/model is similar.

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

Purpose of Interaction Diagrams

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

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

The purpose of interaction diagram is -

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

How to Draw an Interaction Diagram?

- As we have already discussed, the purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized. Dynamic aspect can be defined as the snapshot of the running system at a particular moment
- We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

Following things are to be identified clearly before drawing the interaction diagram -

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Following are two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

Interaction Notation

Interaction is basically a message exchange between two UML components. The following diagram represents different notations used in an interaction.Interaction is used to represent the communication among the components of a system.

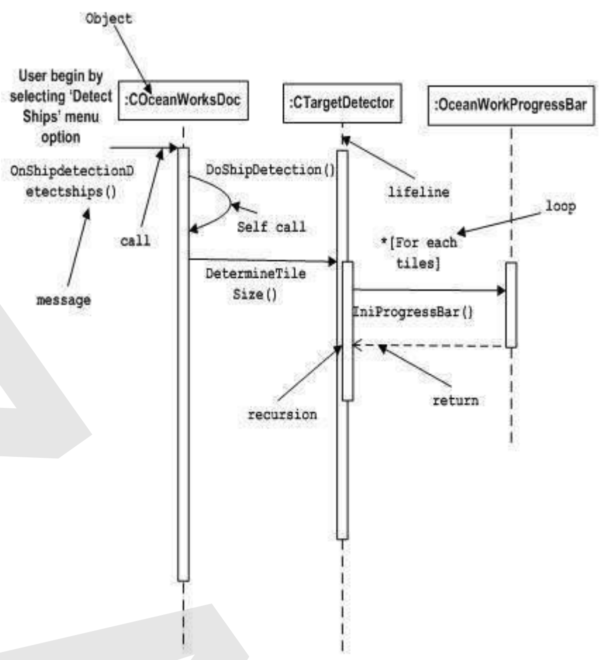


Fig.Interaction Notation

The Sequence Diagram/Model

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).
- The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object. It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.
- The first call is *sendOrder ()* which is a method of *Order object*. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

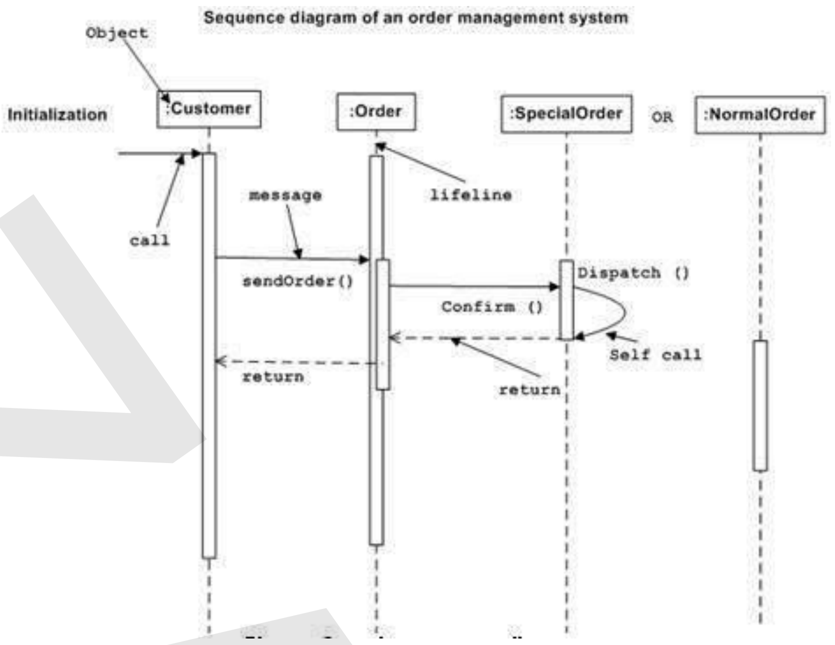


Fig. sequence diagram

The Collaboration/Communication Diagram/Model

Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.

The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.
- Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

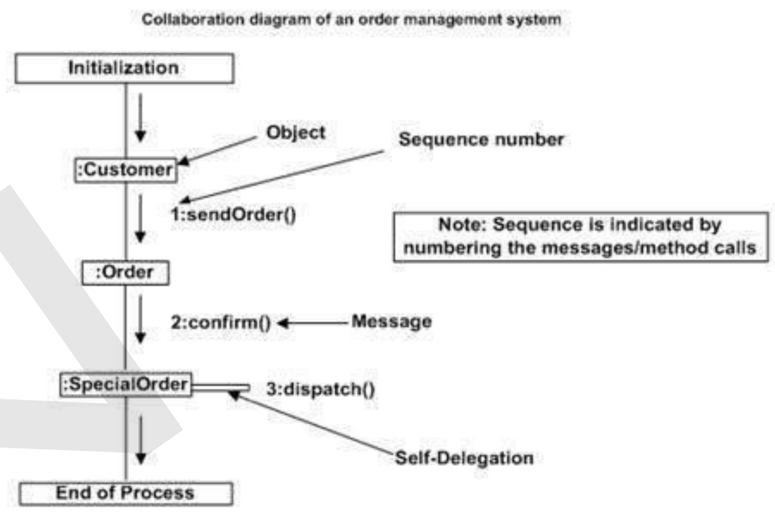


Fig. Collaboration/CommunicationDiagram

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.

Where to Use Interaction Diagrams?

- We have already discussed that interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.
- The main purposes of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.
- Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.
- Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

Interaction diagrams can be used -

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

Activity Diagram/Model

- Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.
- Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.
- Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.
- Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed
- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

Purpose of Activity Diagrams

- The basic purposes of activity diagrams are similar to other above diagram/models. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.
- Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.
- It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as -

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

How to Draw an Activity Diagram?

- Activity diagrams are mainly used as a flowchart that consists of activities performed by the system. Activity diagrams are not exactly flowcharts as they have some additional capabilities. These additional capabilities include branching, parallel flow, swimlane, etc.
- Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

Before drawing an activity diagram, we should identify the following elements -

- Activities
- Association
- Conditions
- Constraints

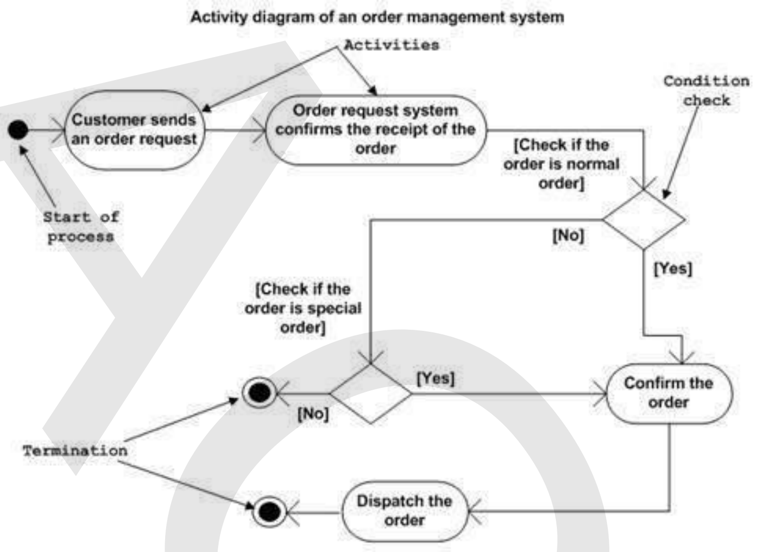
Once the above mentioned parameters are identified, we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

Following is an example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and is mainly used by the business users.

Following diagram is drawn with the four main activities -

- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order

After receiving the order request, condition checks are performed to check if it is normal or special order. After the type of order is identified, dispatch activity is performed and that is marked as the termination of the process.


Activity diagram of an order management system
Activities
Condition check
Customer sends an order request
Order request system confirms the receipt of the order
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

Where to Use Activity Diagrams?

- The basic usage of activity diagram is similar to other four UML diagrams. The specific usage is to model the control flow from one activity to another. This control flow does not include messages.
- Activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes the flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues, or any other system.
- We will now look into the practical applications of the activity diagram. From the above discussion, it is clear that an activity diagram is drawn from a very high level. So it gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person.
- This diagram is used to model the activities which are nothing but business requirements. The diagram has more impact on business understanding rather than on implementation details.

Activity diagram can be used for -

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

Architectural Modeling

Component Diagrams/Model

- Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.
- Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Purpose of Component Diagrams

- Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.
- Thus from that point of view, 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.
- A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as -

- Visualize the components of a system.
- Construct executable by using forward and reverse engineering.
- Describe the organization and relationships of the components.

How to Draw a Component Diagram?

- Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc
- The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.
- Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.
- This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.

Before drawing a component diagram, the following artifacts are to be identified clearly -

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

After identifying the artifacts, the following points need to be kept in mind.

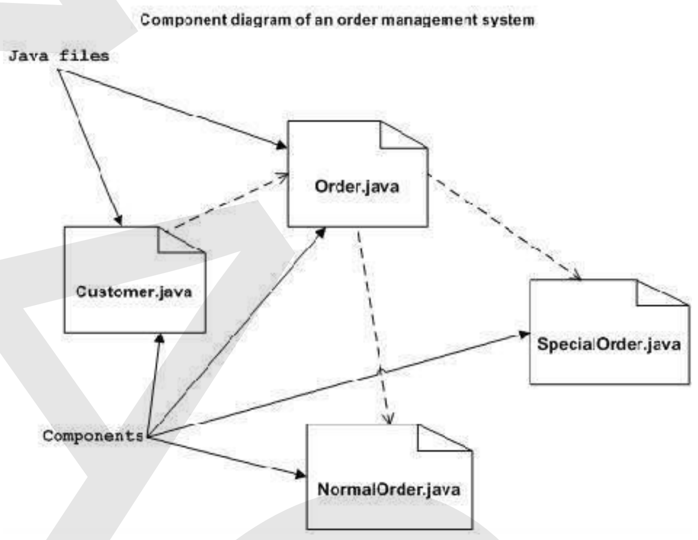
- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing the using tools.

- Use notes for clarifying important points.

Following is a component diagram for order management system. Here, the artifacts are files. The diagram shows the files in the application and their relationships. In actual, the component diagram also contains dlls, libraries, folders, etc.

In the following diagram, four files are identified and their relationships are produced. Component diagram cannot be matched directly with other UML diagrams discussed so far as it is drawn for completely different purpose.

The following component diagram has been drawn considering all the points mentioned above.



Where to Use Component Diagrams?

- We have already described that component diagrams are used to visualize the static implementation view of a system. Component diagrams are special type of UML diagrams used for different purposes.
- These diagrams show the physical components of a system. To clarify it, we can say that component diagrams describe the organization of the components in a system.
- Organization can be further described as the location of the components in a system. These components are organized in a special way to meet the system requirements.
- As we have already discussed, those components are libraries, files, executables, etc. Before implementing the application, these components are to be organized. This component organization is also designed separately as a part of project execution.
- Component diagrams are very important from implementation perspective. Thus, the implementation team of an application should have a proper knowledge of the component details

Component diagrams can be used to −

- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

Deployment Diagrams/Model

- Deployment diagramsare 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 of Deployment Diagrams

- The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.
- Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.
- UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.
- Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as -

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

How to Draw a Deployment Diagram?

Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters -

- Performance
- Scalability
- Maintainability
- Portability

Before drawing a deployment diagram, the following artifacts should be identified -

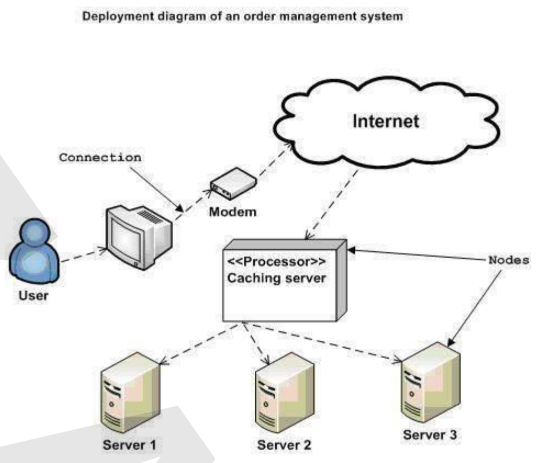
- Nodes
- Relationships among nodes

Following is a sample deployment diagram to provide an idea of the deployment view of order management system. Here, we have shown nodes as -

- Monitor
- Modem
- Caching server
- Server

The application is assumed to be a web-based application, which is deployed in a clustered environment using server 1, server 2, and server 3. The user connects to the application using the Internet. The control flows from the caching server to the clustered environment.

The following deployment diagram has been drawn considering all the points mentioned above.


Deployment diagram of an order management system
Internet
Connection
Modem
User
<<Processor>> Caching server
Nodes
Server 1
Server 2
Server 3

Where to Use Deployment Diagrams?

- Deployment diagrams are mainly used by system engineers. These diagrams are used to describe the physical components (hardware), their distribution, and association.
- Deployment diagrams can be visualized as the hardware components/nodes on which the software components reside.
- Software applications are developed to model complex business processes. Efficient software applications are not sufficient to meet the business requirements. Business requirements can be described as the need to support the increasing number of users, quick response time, etc.
- To meet these types of requirements, hardware components should be designed efficiently and in a cost-effective way.
- Now-a-days software applications are very complex in nature. Software applications can be standalone, web-based, distributed, mainframe-based and many more. Hence, it is very important to design the hardware components efficiently.

Deployment diagrams can be used -

- To model the hardware topology of a system.
- To model the embedded system.
- To model the hardware details for a client/server system.
- To model the hardware details of a distributed application.
- For Forward and Reverse engineering.

Artifact Model

Artefacts play a vital role in software and systems development processes. Other terms like documents, deliverables, or work products are widely used in software development communities instead of the term artefact.

An artifact model shows how tangible elements (physical or electronic) are used and structured in the business process flow of doing the work. For software processes, artefact orientation has become an indispensable tool to support flexibility in the project lifecycle, precision in the results, and towards standardisation of terminology within and across projects.

- Work artifacts are one of the most important entities that get passed from one work role to another within the flow model. Examples include paper memos, email messages, correspondence templates, product change orders, and other things people create and use while working.
- Sometimes artifacts are work products, information objects used in daily operation of the enterprise, for example, an order form being filled out, that reveal traces of people's work practices.
- The contextual inquiry team must pay close attention to how these artifacts are created, communicated, and used. What are those notes scribbled on those forms? Why are some fields in this form left blank? Why is there a sticky note on this form? Perhaps a signature is required for approval on other kinds of documents.
- This model is one reason why observers and interviewers must collect as many artifacts as possible during their contextual inquiry field visits to users.

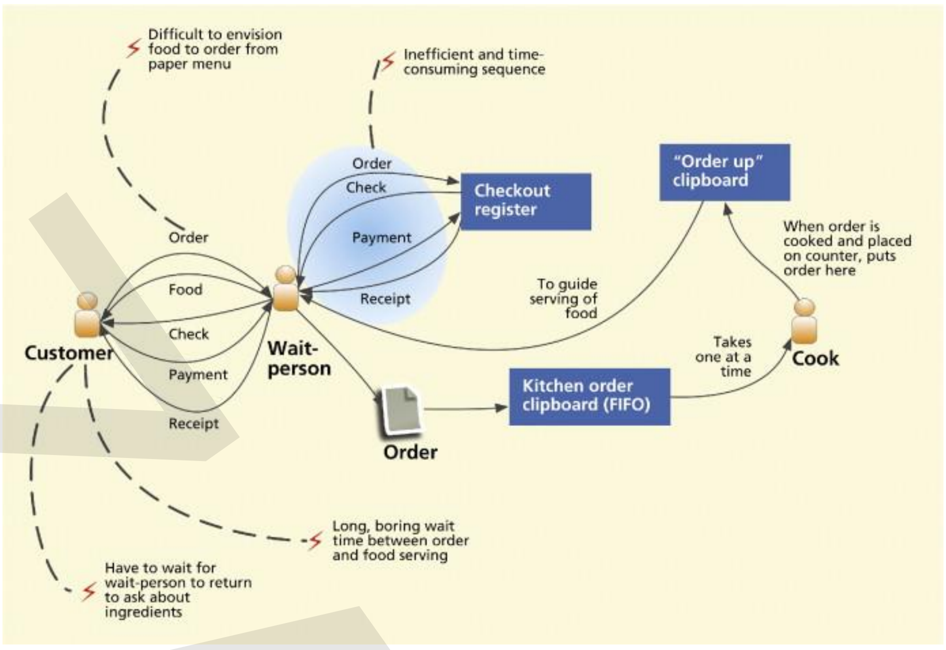
What's in an Artifact Model?

At this point, your artifact model is probably just a collection of labelled artifacts plus some notes about them. Examples of artifacts include:

- Work practice forms.
- Sketches.
- Props.
- Memos.
- Significant email messages.
- Correspondence templates.
- Product change orders.
- An order form.
- A receipt.
- Paper or electronic forms.
- Templates.
- Physical or electronic entities that users create, retrieve, use, or reference within a task and/or pass on to another person in the work domain.
- Photos (with permission) of the work place and work being performed.
- Other objects that play a role in work performed.

Example: Artifact Model from a Restaurant

- It is easy to think of artifacts associated with a restaurant. In below figure the first artifact encountered by a person in the customer work role, delivered by the person in the wait-staff work role, is a menu, used by the customer work role to decide on something to order.
- Other usual restaurant work artifacts include the order form on which the wait-staff person writes the original order and the guest check, which can be the same artifact or a printed check if the order is entered into a system. Finally, there might be a regular receipt and, if a credit card is used, a credit card signature form and a credit card receipt. Artifacts in restaurants, as they do in most enterprises, are the basis for at least part of the flow model. In below Figure you can see how restaurant artifacts help show the workflow from order to serving to concluding payment.


Difficult to envision food to order from paper menu
Inefficient and time-consuming sequence
Order
Check
Payment
Receipt
Checkout register
"Order up" clipboard
When order is cooked and placed on counter, puts order here
Order
Food
Check
Payment
Receipt
Customer
Wait-person
To guide serving of food
Takes one at a time
Cook
Kitchen order clipboard (FIFO)
Order
Long, boring wait time between order and food serving
Have to wait for wait-person to return to ask about ingredients

- The artifacts, especially when arranged as part of a flow model, can help make a connection from contextual data to thinking ahead about design. For example, the waiting and boredom shown in Figure pose a "barrier" to the customer.
- This leads us to ask questions such as: How can we make that experience for the customer placing the order more fun, engaging, and informed? This kind of question posed now will later provide a great starting point for design brainstorming later: Would not it be cool if each dining table contained an embedded interactive touch tablet. Users could pass time by playing games, doing email, or surfing the Web.
- Another barrier shown in Figure is the difficulty of ordering food from a textual description in a paper menu. Interviewing restaurant customers about their experiences, you find that many people, when they order a dish and then see something else someone has ordered, wish to get that dish instead.
- Paper menus do not leverage this rich human sensual connection to the food! However, this discussion of restaurant artifacts does help us ask questions that will later inspire design: If the table contained an interactive display, then why not let the customer use it to interact with the kitchen, ask questions about ingredients, and see images of the dish being served? In fact, why not let the customers place their orders themselves?

Constructing the Artifact model

- How do you make the model? Well, the artifact model is mainly a collection of artifacts, but you can organize it for analysis and use. In contextual inquiry you will have collected the artifacts, usually visual, by making a drawing, a copy, or a photograph or by having collected a real example of the artifact. An example of a tangible work artifact is a guest check from a restaurant. If an artifact is more aural than visual, a recording of the sound could be an artifact.
- Next, the team should make "artifact posters." Attach samples of each artifact to a separate flip chart page. Add annotation to your "exhibits" to provide more information about your observations of them in the work practice. Add explanations of how each artifact is used in the work practice and workflow.
- Annotate artifacts with stick-on notes associating them with tasks, user goals, and task barriers. Each poster drives discussion to explain the artifact's use while trying to tease out associated issues and

user needs. As usual, the process can generate additional user work activity notes from what is learned about the artifacts and how they are used.

- Artefact models provide a basis for a process agnostic flexible backbone for project operation in which the team agrees on the content and the structure of the artefacts to be created until a specific point in time rather than a strict process and their synchronisation.
- At the same time, artefact models leave open the way of creating the results yet having a clear notion of responsibilities assigned to each artefact to be created.

.

Chapter 4 Questions:

A. MCQ's:

1. UML is a standard language for specifying, visualizing, constructing, and _______the artifactsof software systems.

a. planning	b. designing
c. documenting	d. organizing

2. Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the ________of the objects

a.structural organization	b. Functionalorganization
c. divisionalorganization	d. matrix organization

3. Class diagram describes the attributes and operations of a class and also the _______imposed on the system.

a. assurance	b. constraints
c. collaboration	d. assumption

Ans:

1.c 2.a 3.b

B. State True or False

1. Use case diagrams are used to gather the requirements of only internal system influences.
2. Deployment diagramsare used to visualize the topology of the physical components of a system, where the software components are deployed.
3. UML is a programming language that can be used to generate code in various languages using UML diagrams.
4. Activity diagrams are used to visualize the flow of controls in a system.

Ans:

1. False2.True3.False 4. True

C. 1 mark questions

1. Which UML diagram/model can be mapped directly with object-oriented languages?
2. List out the Interaction Diagrams/Model
3. Where to use a Use Case Diagram?
4. Give the purpose of activity diagrams
5. Where to use Component Diagrams?

D. Brief Questions

1. Difference between Sequence diagram and Collaboration diagram
2. What is The Collaboration/Communication Diagram/Model?
3. Where we can use a Use Case Diagram/Model?
4. Difference between Sequence diagram and Activity diagram
5. What is Deployment Diagrams/Model
6. How to draw a deployment diagram?
7. Write a short note on Artifact Model/diagram
8. What is Component model?