UNIT 2

- **Understanding Requirements – Introduction (refer PPT)**
- **Requirement Types**

There are a number of different type of requirement that system engineers will have to develop on a acquisition program through it life-cycle. These requirements range from very high level concept focused to very specific for a part. The main types of requirements are:

- Functional Requirements
- Performance Requirements
- System Technical Requirements
- Specifications

Functional Requirements

A functional requirement is simply a task (sometimes called an action or activity) that must be accomplished to provide an operational capability (or satisfy an operational requirement). Some functional requirements that are associated with operations and support can be discerned from the needed operational capability (see Operational Requirements). Others often result only from diligent systems engineering. Experience in systems engineering has identified eight generic functions that most systems must complete over their life cycle: development, manufacturing, verification, deployment, training, operations, support, and disposal. These are known as the eight primary system functions. Each must usually be considered to identify all the functional requirements for a system.

Performance Requirements

A performance requirement is a statement of the extent to which a function must be executed, generally measured in terms such as quantity, accuracy, coverage, timeliness, or readiness

System Technical Requirements

Result in both allocated and derived requirements.

- **Allocated Requirements**: flow directly from the system requirements down to the elements of the system.

- **Derived Requirements:** dependent on the design solution (and so are sometimes called design requirements). They include internal interface constraints between the elements of the system.

Specifications

A specification is a detailed, exact statement of particulars, especially a statement prescribing materials, dimensions, and quality of work for something to be built, installed, or manufactured. The overall purpose of a specification is to provide a basis for obtaining a product or service that will satisfy a particular need at an economical cost and to invite maximum reasonable competition. By definition, a specification sets limits and thereby

eliminates, or potentially eliminates, items that are outside the boundaries drawn. A good specification should do four (4) things:

1. Identify minimum requirements
2. List reproducible test methods to be used in testing for compliance with specifications
3. Allow for a competitive bid
4. Provide for an equitable award at the lowest possible cost.

Introduction to requirement engineering

In requirements engineering, requirements elicitation is the practice of collecting the requirements of a system from users, customers and other stakeholders. The practice is also sometimes referred to as "requirement gathering". ... Commonly used elicitation processes are the stakeholder meetings or interviews.

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follow:

1. Inception/start

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2.Elicitation/find

Elicitation means to find the requirements from anybody.
The requirements are difficult because the following problems occur in elicitation.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.
- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

Eliciting Requirements

Eliciting requirement helps the user for collecting the requirement.

Eliciting requirement steps are asfollows:

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.

- QFD consist of three types of requirement:

    - Normal requirements

    - The objective and goal are stated for the system through the meetings with the customer.
    - For the customer satisfaction these requirements should be there.

<u>Expected requirement</u>

- These requirements are implicit.
- These are the basic requirement that not be clearly told by the customer, but also the customer expect that requirement.

<u>Exciting requirements</u>

- These features are beyond the expectation of the customer.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.
  For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more excited about that feature.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

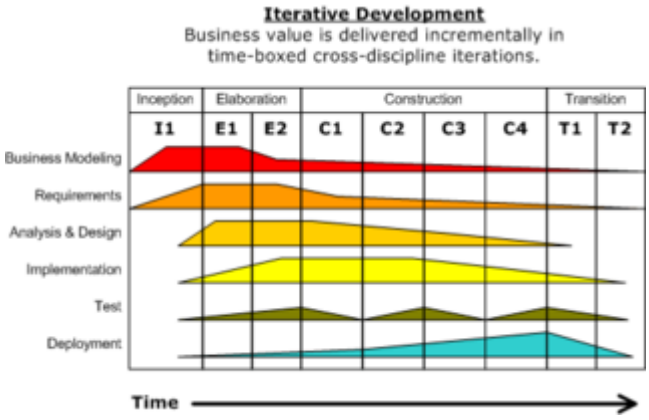
4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.

- **RUP**

The Rational Unified Process (RUP) is an iterativesoftware development process framework created by the Rational Software Corporation, a division of IBM since 2003.[1] RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the Unified Process.

Four project life-cycle phases



RUP phases and disciplines

The RUP has determined a project life-cycle consisting of four phases. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although in essence the key to the process lies in the iterations of development that lie within all of the phases. Also, each phase has one key objective and milestone at the end that denotes the objective being accomplished. The visualization of RUP phases and disciplines over time is referred to as the RUP hump chart.

1. ***Inception phase***

The primary objective is to scope the system adequately as a basis for validating initial costing and budgets. In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc.), and financial forecast is established. To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated. After these are completed, the project is checked against the following criteria:

- Stakeholder concurrence on scope definition and cost/schedule estimates.

- Requirements understanding as evidenced by the fidelity of the primary use cases.
- Credibility of the cost/schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Establishing a baseline by which to compare actual expenditures versus planned expenditures.

If the project does not pass this milestone, called the life cycle objective milestone, it either can be cancelled or repeated after being redesigned to better meet the criteria.

2. ***Elaboration phase***

The primary objective is to mitigate the key risk items identified by analysis up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

The outcome of the elaboration phase is:

- A use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.
- A description of the software architecture in a software system development process.
- An executable architecture that realizes architecturally significant use cases.
- Business case and risk list which are revised.
- A development plan for the overall project.
- Prototypes that demonstrably mitigate each identified technical risk.
- A preliminary user manual (optional)

This phase must pass the lifecycle architecture milestone criteria answering the following questions:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration indicate that major risk elements are addressed and resolved?
- Is the construction phase plan sufficiently detailed and accurate?
- Do all stakeholders agree that the current vision can be achieved using current plan in the context of the current architecture?

- Is the actual vs. planned resource expenditure acceptable?

If the project cannot pass this milestone, there is still time for it to be canceled or redesigned. However, after leaving this phase, the project transitions into a high-risk operation where changes are much more difficult and detrimental when made.

The key domain analysis for the elaboration is the system architecture.

3. Construction phase

The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.

This phase produces the first external release of the software. Its conclusion is marked by the initial operational capability milestone.

4. Transition phase

The primary objective is to 'transit' the system from development into production, making it available to and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The system also goes through an evaluation phase, any developer which is not producing the required work is replaced or removed. The product is also checked against the quality level set in the Inception phase.

If all objectives are met, the product release milestone is reached and the development cycle is finished

Note : Introduction to UML, UML diagrams, purpose of UML ,use caseetc. refer PPT

- **Use case diagrams**

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists 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 analyzed 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.
- Show the interaction among the requirements and 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 analyzed, 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.

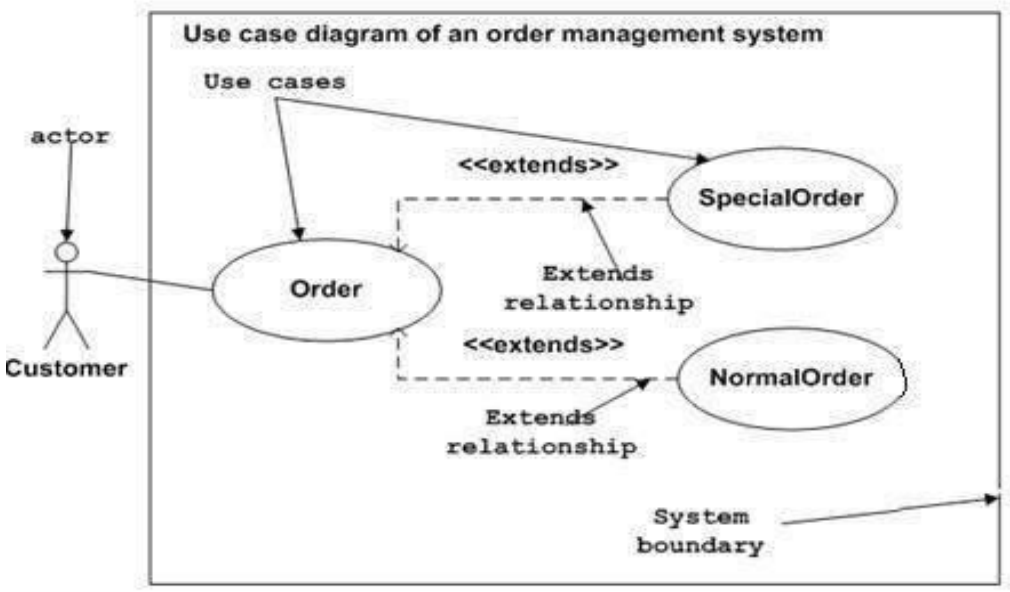


Figure: Sample Use Case diagram

Types of actors in use case diagrams:

Actors can be primary or secondary actors. Primary actors initiate a use case, while secondary actors support a use case or receive something of value from the use case. While this answer might score you some points in the interview, there is another way to classify actors that is important to know and can show that you understand some of the finer points of use case diagramming.

Actors can be:

1. Human
2. Systems/Software
3. Hardware
4. Timer/Clock

Many analysts miss key actors during the use case diagramming process because they only identify human actors. Categorizing use case actors in this ways helps the analyst ensure they haven't overlooked any critical actors within the use case diagram.

- **use-case realization**

A use-case realization represents how a use case will be implemented in terms of collaborating objects. The realizations reside within the design. By walking through a design

exercise of showing how the design elements will perform the use case, the team gets confirmation that the design is robust enough to perform the required behavior.

The realization can take various forms. It may include, for example, a textual description (a document), class diagrams of participating classes and subsystems, and interaction diagrams (communication and sequence diagrams) that illustrate the flow of interactions between class and subsystem instances.

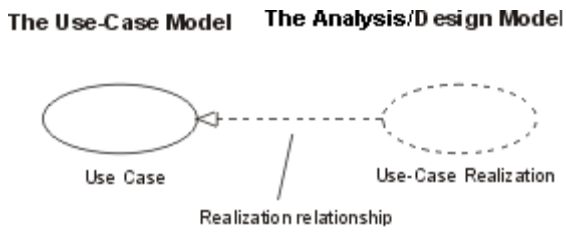
The reason for separating the use-case realization from its use case is that doing so allows the requirements, in the form of use cases, to be managed separately from the design, in the form of realizations. This decoupling will prove invaluable if the architecture is changed enough that the realization needs to be reworked while the requirements remain unaffected. Even without such a circumstance, the clear separation of concerns between requirements and design is valuable.

In a model, a use-case realization is represented as a UML (Unified Modeling Language) collaboration that groups the diagrams and other information (such as textual descriptions) that form the use-case realization.

Like other aspects of the design, the UML diagrams that support use-case realizations can be produced at various levels of abstraction. A first pass in creating the realization might produce a diagram at an analysis level of abstraction where the participants will be high-level elements that are expected to be revisited and detailed down to the design level in a second pass. If the architecture and design idioms are well-understood, the realization could immediately be created at a low level of abstraction that specifies more detail on the elements and how they will collaborate to realize the behavior of the use case. In the latter case, it is valuable to model patterns and architectural mechanisms to reduce the amount of low-level detail in each realization.

For each use case in the requirements, there can be a use-case realization in the design with a realization relationship to the use case, as the following figure shows. In UML, this is shown as a dashed arrow with an arrowhead, like a generalization relationship, indicating that a realization is a kind of inheritance, as well as a dependency (see the figure that follows).

The UML notation for use-case realization



- **Activity Diagrams**

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.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

Purpose of Activity Diagrams

It captures the dynamic behaviour of the system. 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.

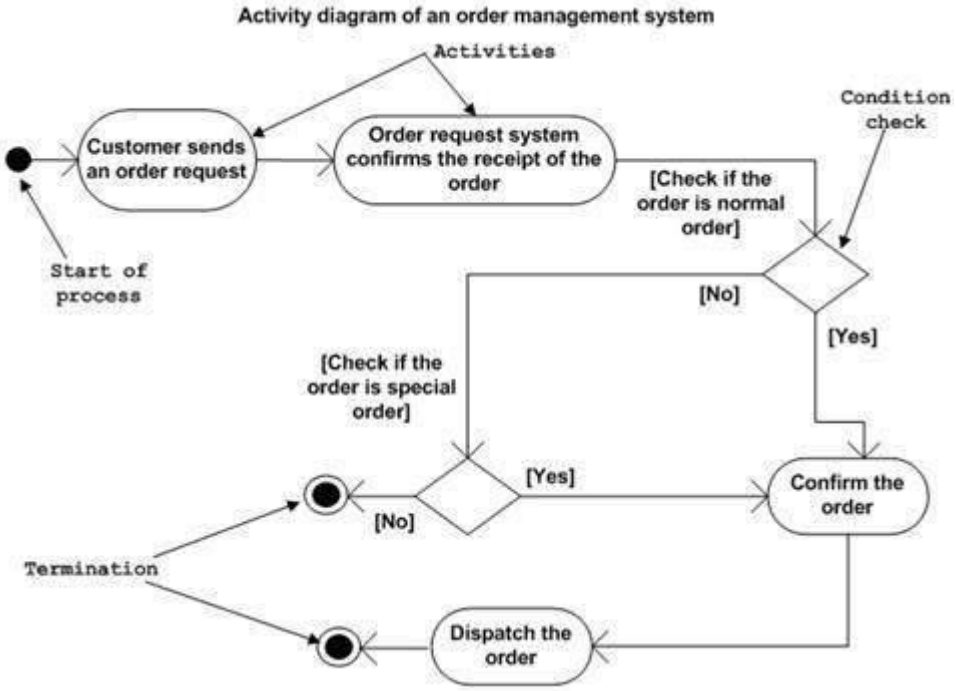
Following is an example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with

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.



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.

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.

Activity Diagram Notations –

1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.

   Figure – notation for initial state or start state
   A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

   For example – Here the initial state is the state of the system before the application is opened.

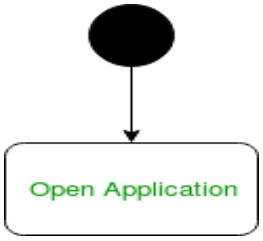
   

   Figure – initial state symbol being used
2. **Action or Activity State –** An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

   Figure – notation for an activity state
   For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

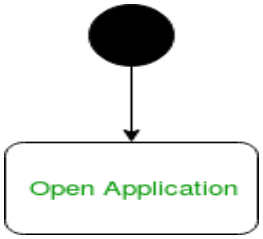
   

   Figure – activity state symbol being used
3. **Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

   Figure – notation for control Flow
   An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.

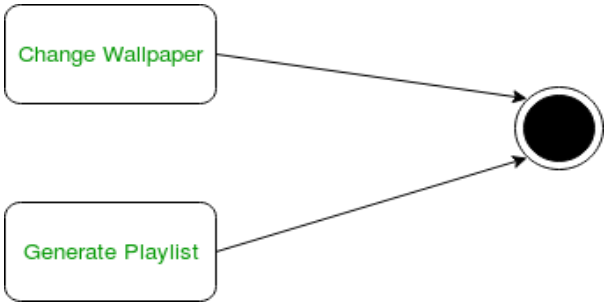


Figure – using action flows for transitions

4. **Decision node and Branching –** When we need to make a decision before deciding the flow of control, we use the decision node.

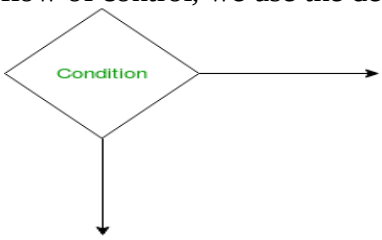


Figure – notation for decision node

The outgoing arrows from the decision node can be labelled with conditions or guard expressions.It always includes two or more output arrows.

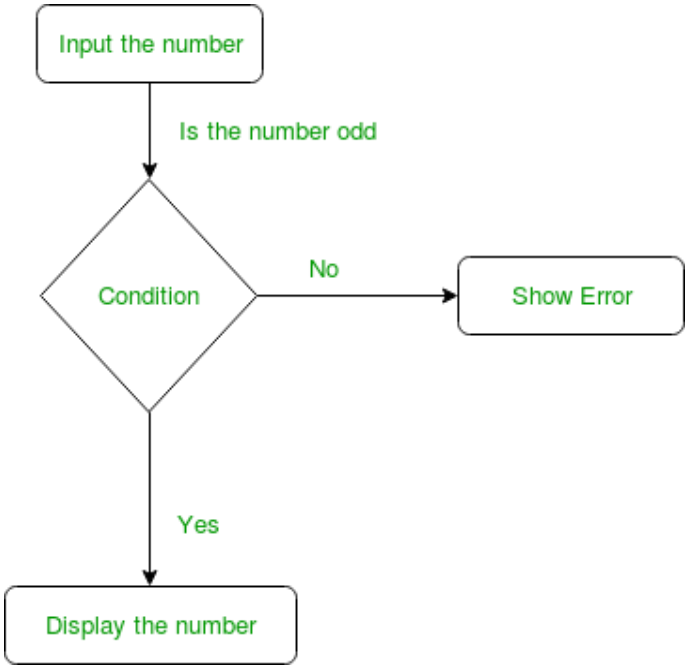


Figure – an activity diagram using decision node

5. **Guards –** A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

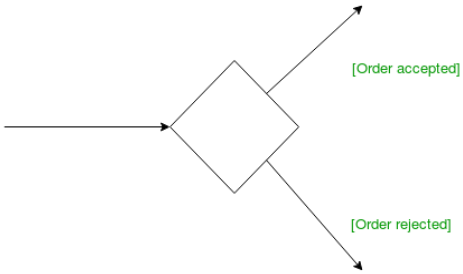


Figure – guards being used next to a decision node
The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

6. **Fork –** Fork nodes are used to support concurrent activities.

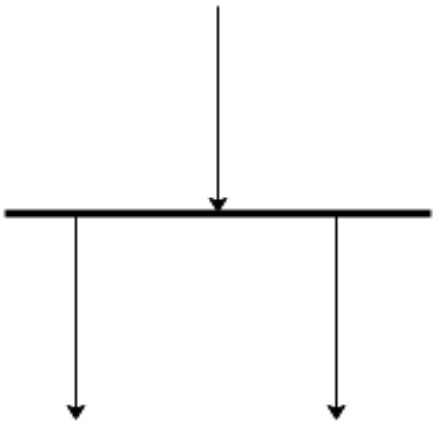


Figure – fork notation
When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.
We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.
For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.

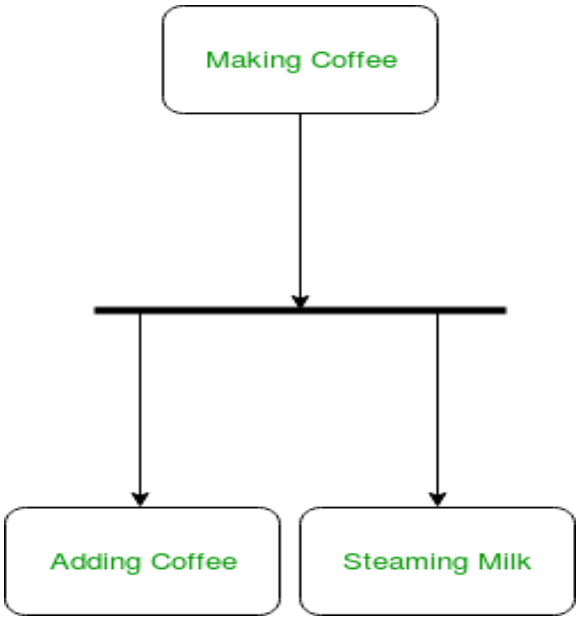


Figure – a diagram using fork

7. **Join –** Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

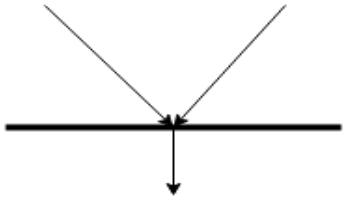


Figure – join notation

For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.

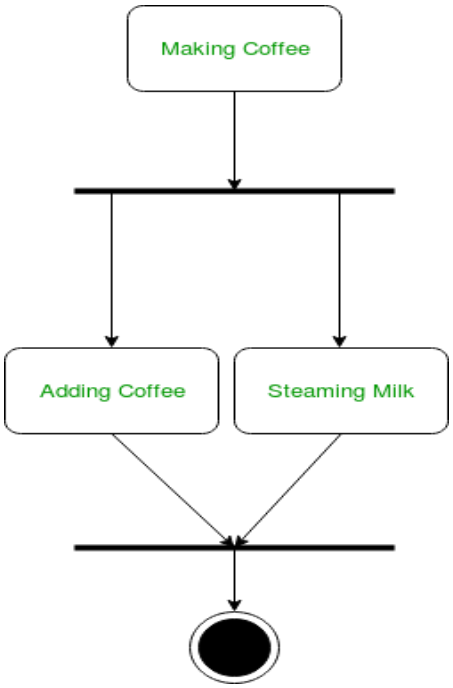


Figure – a diagram using join notation

8. **Merge or Merge Event –** Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

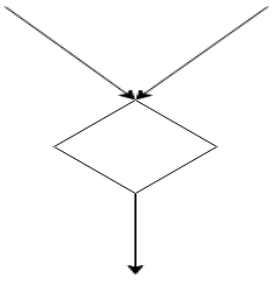


Figure – merge notation

For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time.

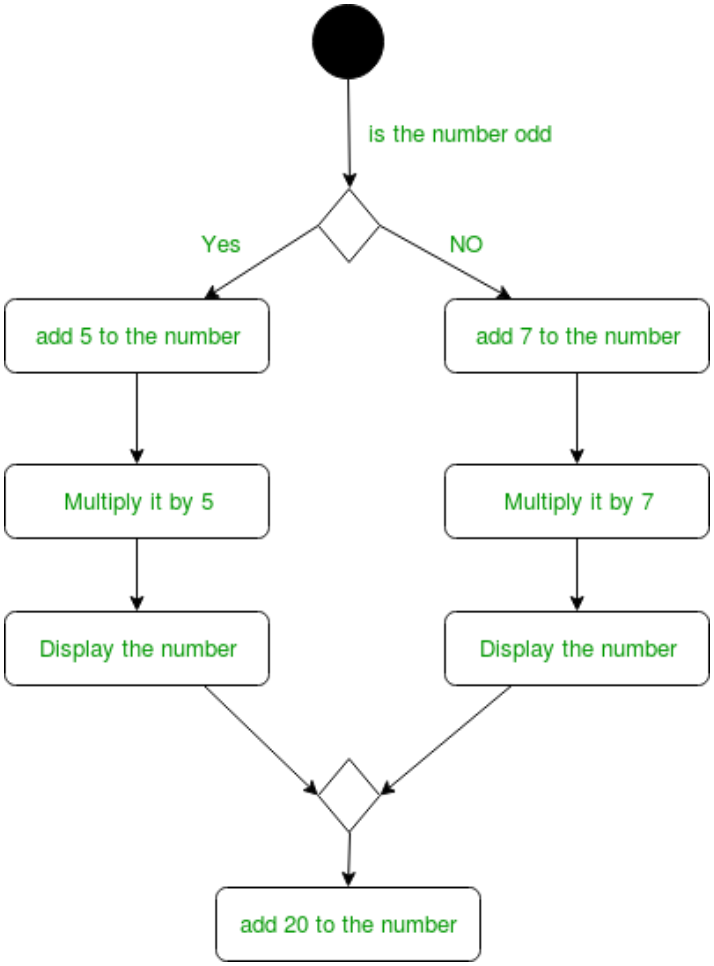


Figure – an activity diagram using merge notation

9. **Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

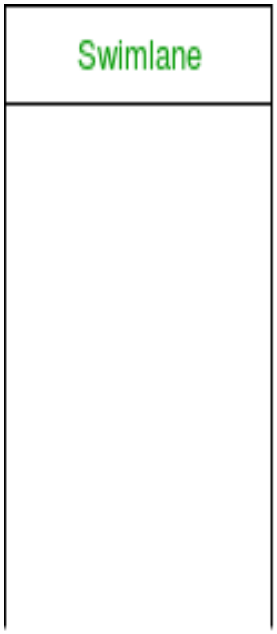


Figure – swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above.

For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.

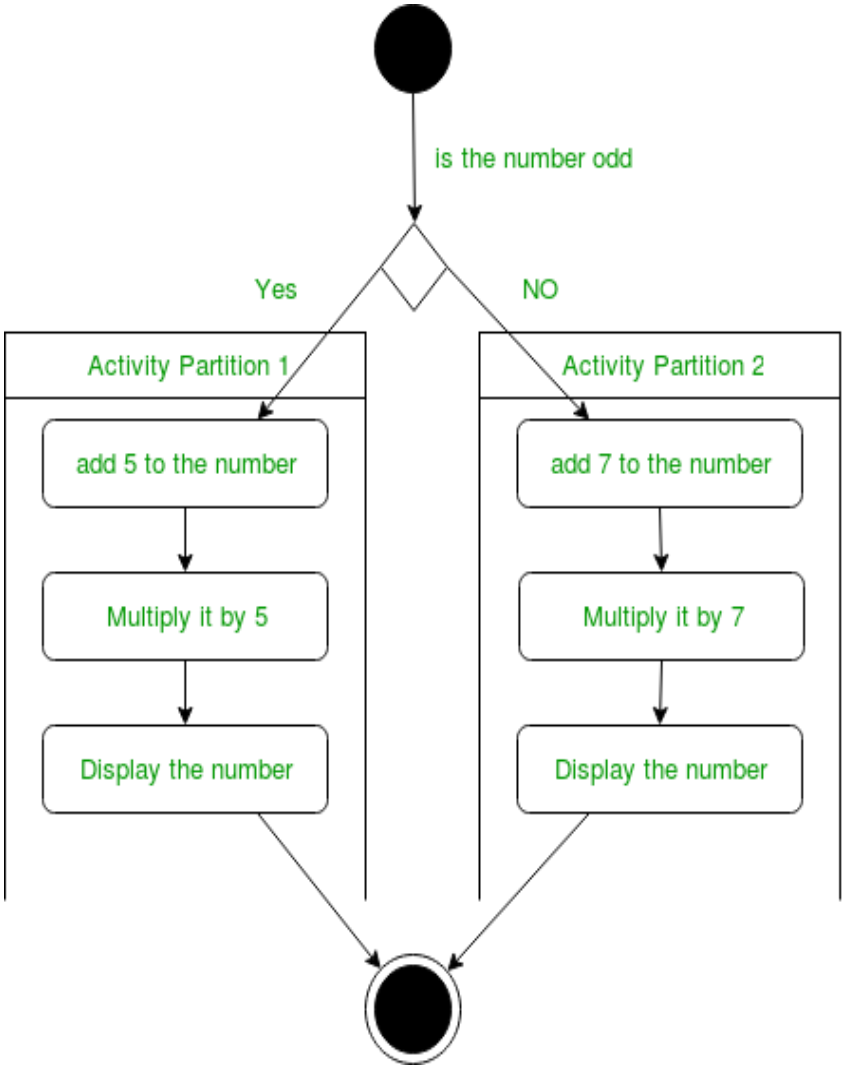


Figure – an activity diagram making use of swimlanes

10. **Time Event –**

Time Event

Figure – time event notation

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.

For example – Let us assume that the processing of an image takes takes a lot of time. Then it can be represented as shown below.

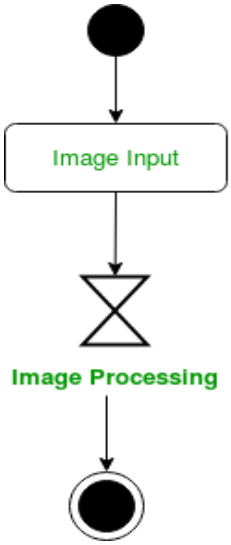


Figure – an activity diagram using time event

11. **Final State or End State –** The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

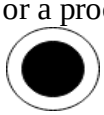


Figure – notation for final state

How to Draw an activity diagram –

1. Identify the initial state and the final states.
2. Identify the intermediate activities needed to reach the final state from he initial state.
3. Identify the conditions or constraints which cause the system to change control flow.
4. Draw the diagram with appropriate notations.

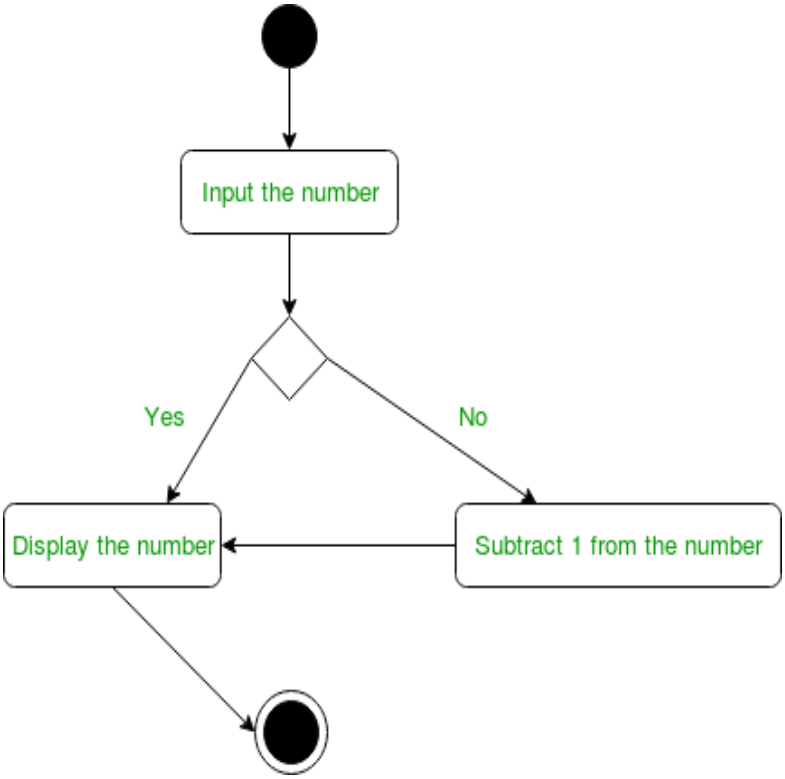


Figure – an activity diagram

The above diagram prints the number if it is odd otherwise it subtracts one from the number and displays it.

Uses of an Activity Diagram –

- Dynamic modelling of the system or a process.
- Illustrate the various steps involved in a UML use case.
- Model software elements like methods,operations and functions.
- We can use Activity diagrams to depict concurrent activities easily.
- Show the constraints, conditions and logic behind algorithms.