MODULE: 1

Q-1 What is software? What is software engineering?

Answer: Software engineering has two parts: software and engineering.

Software is a collection of codes, documents, and triggers that does a specific job and fills a specific requirement.

Engineering is the development of products using best practices, principles, and methods.

It is a branch of engineering that deals with the development of software products. It operates within a set of principles, best practices, and methods that have been carefully honed throughout the years, changing as software and technology change.

Software engineering leads to a product that is reliable, efficient, and effective at what it does. While software engineering can lead to products that do not do this, the product will almost always go back into the production stage.

Types of Software Engineering

Software engineering studies the design, development, and maintenance of software as an umbrella definition. Still, there are different types of software engineering that a company or product may need. Problems tend to emerge when software is low-quality or isn't properly vetted before deployment.

There has been a lot of demand for software engineers because of the rate of change in user requirements, statutes, and the platforms we use.

Software engineering works on a few different levels:

Operational Software Engineering: Software engineering on the operational level focuses on how the software interacts with the system, whether or not it is on a budget, the usability, the functionality, the dependability, and the security.

Transitional Software Engineering: This type focuses on how software will react when it is changed from one environment to another. It typically takes some scalability or flexibility in the development.

Software Engineering Maintenance: Recurrent software engineering focuses on how the software functions within the existing system, as all parts of it change.

Software engineering functions at all parts of the software development lifecycle, including analysis, design, development, testing, integration, implementation, maintenance, and even retirement.

It is important to understand that software engineering isn’t a new practice, but it is constantly changing and can feel new on a regular basis. As our scientific and technical knowledge grows, so does the skill set required for software engineering. Software engineering is different from other branches of engineering in that professionals are working, at least at the start, in theory instead of with something real. Software is used in everything around us, so it is important to ensure that all software is working properly. If it does not, it can result in loss of money, loss of reputation, and even in some cases, loss of life.

Q- 2. Explain types of software.

Answer: Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part.

The two main categories of software are application software and system software. An application is software that full fills a specific need or performs tasks. System software is designed to run a computer's hardware and provides a platform for applications to run on top of.

Other types of software include programming software, which provides the programming tools software developers need; middleware, which sits between system software and applications; and driver software, which operates computer devices and peripherals.

Early software was written for specific computers and sold with the hardware it ran on. In the 1980s, software began to be sold on floppy disks, and later on CDs and DVDs. Today, most software is purchased and directly downloaded over the internet. Software can be found on vendor websites or application service provider websites.

Examples and types of software

Among the various categories of software, the most common types include the following:

- **Application software.** The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application. An application can be self-contained, or it can be a group of programs that run the application for the user. Examples of modern applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.
- **System software.** These software programs are designed to run a computer's application programs and hardware. System software coordinates the activities and functions of the hardware and software. In addition, it controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in. The OS is the best example of system software; it manages all the other computer programs. Other examples of system software include the firmware, computer language translators and system utilities.
- **Driver software.** Also known as device drivers, this software is often considered a type of system software. Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks. Every device that is connected to a computer needs at least one device driver to function. Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.
- **Middleware.** The term *middleware* describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word. It is also used to send a remote work request from an application in a computer that has one kind of OS, to an

application in a computer with a different OS. It also enables newer applications to work with legacy ones.

- **Programming software.** Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs. Examples of programming software include assemblers, compilers, debuggers and interpreters.

3. What is SDLC? Explain each phase of SDLC

Answer: A system development life cycle or SDLC is essentially a project management model. It defines different stages that are necessary to bring a project from its initial idea or conception all the way to deployment and later maintenance.

7 Phases of the System Development Life Cycle

There are seven primary stages of the modern system development life cycle. Here's a brief breakdown:

- Stage 1: Planning Stage
- Stage 2: Feasibility or Requirements of Analysis Stage
- Stage 3: Design and Prototyping Stage
- Stage 4: Software Development Stage
- Stage 5: Software Testing Stage
- Stage 6: Implementation and Integration
- Stage 7: Operations and Maintenance Stage

Now let's take a closer look at each stage individually.

Stage 1: Planning Stage

Before we even begin with the planning stage, the best tip we can give you is to take time and acquire a proper understanding of the app development life cycle.

The planning stage (also called the feasibility stage) is exactly what it sounds like the phase in which developers will plan for the upcoming project.

It helps to define the problem and scope of any existing systems, as well as determine the objectives for their new systems.

By developing an effective outline for the upcoming development cycle, they'll theoretically catch problems before they affect development.

And help to secure the funding and resources they need to make their plan happen.

Perhaps most importantly, the planning stage sets the project schedule, which can be of key importance if development is for a commercial product that must be sent to market by a certain time.

Stage 2: Analysis Stage

The analysis stage includes gathering all the specific details required for a new system as well as determining the first ideas for prototypes.

Developers may:

- Define any prototype system requirements

- Evaluate alternatives to existing prototypes
- Perform research and analysis to determine the needs of end-users

Furthermore, developers will often create a software requirement specification or SRS document.

This includes all the specifications for software, hardware, and network requirements for the system they plan to build. This will prevent them from overdrawing funding or resources when working at the same place as other development teams.

Stage 3: Design Stage

The design stage is a necessary precursor to the main developer stage.

Developers will first outline the details for the overall application, alongside specific aspects, such as its:

- User interfaces
- System interfaces
- Network and network requirements
- Databases

They'll typically turn the SRS document they created into a more logical structure that can later be implemented in a programming language. Operation, training, and maintenance plans will all be drawn up so that developers know what they need to do throughout every stage of the cycle moving forward.

Once complete, development managers will prepare a design document to be referenced throughout the next phases of the SDLC.

Stage 4: Development Stage

The development stage is the part where developers actually write code and build the application according to the earlier design documents and outlined specifications.

This is where Static Application Security Testing or SAST tools come into play. Product program code is built per the design document specifications. In theory, all of the prior planning and outlining should make the actual development phase relatively straightforward.

Developers will follow any coding guidelines as defined by the organization and utilize different tools such as compilers, debuggers, and interpreters.

Programming languages can include staples such as C++, PHP, and more. Developers will choose the right programming code to use based on the project specifications and requirements.

Stage 5: Testing Stage

Building software is not the end. Now it must be tested to make sure that there aren't any bugs and that the end-user experience will not negatively be affected at any point.

During the testing stage, developers will go over their software with a fine-tooth comb, noting any bugs or defects that need to be tracked, fixed, and later retested.

It’s important that the software overall ends up meeting the quality standards that were previously defined in the SRS document.

Depending on the skill of the developers, the complexity of the software, and the requirements for the end-user, testing can either be an extremely short phase or take a very long time. Take a look at our top 10 best practices for software testing projects for more information.

Stage 6: Implementation and Integration Stage

After testing, the overall design for the software will come together. Different modules or designs will be integrated into the primary source code through developer efforts, usually by leveraging training environments to detect further errors or defects.

The information system will be integrated into its environment and eventually installed. After passing this stage, the software is theoretically ready for market and may be provided to any end-users.

Stage 7: Maintenance Stage

The SDLC doesn’t end when software reaches the market. Developers must now move into maintenance mode and begin practicing any activities required to handle issues reported by end-users.

Furthermore, developers are responsible for implementing any changes that the software might need after deployment.

This can include handling residual bugs that were not able to be patched before launch or resolving new issues that crop up due to user reports. Larger systems may require longer maintenance stages compared to smaller systems.

Role of System Analyst

An SDLC’s system analyst is, in some ways, an overseer for the entire system. They should be totally aware of the system and all its moving parts and can help guide the project by giving appropriate directions.

The system analyst should be:

- An expert in any technical skills required for the project
- A good communicator to help command his or her team to success
- A good planner so that development tasks can be carried out on time at each phase of the development cycle

Thus, systems analysts should have an even mix of interpersonal, technical, management, and analytical skills altogether. They’re versatile professionals that can make or break an SDLC.

Their responsibilities are quite diverse and important for the eventual success of a given project. Systems analysts will often be expected to:

- Gather facts and information
- Make command decisions about which bugs to prioritize or what features to cut

- Suggest alternative solutions
- Draw specifications that can be easily understood by both users and programmers
- Implement logical systems while keeping modularity for later integration
- Be able to evaluate and modify the resulting system as is required by project goals
- Help to plan out the requirements and goals of the project by defining and understanding

cycle, so implementing the changes is often less expensive.

bugs from spiraling out of control.

Benefits of SDLC (System Development Life Cycle)

SDLC provides a number of advantages to development teams that implement it correctly.

Used:

System development life cycles are typically used when developing IT projects. Software development managers will utilize SDLCs to outline various development stages, make sure everyone completes stages on time and in the correct order, and that the project is delivered as promptly and as bug-free as possible.

SDLCs can also be more specifically used by systems analysts as they develop and later implement a new information system.

It largely depends on what your team's goals and resource requirements are. The majority of IT development teams utilize the agile methodology for their SDLC. However, others may prefer the iterative or spiral methodologies.

All three of these methods are popular since they allow for extensive iteration and bug testing before a product is integrated with greater source code or delivered to the market.

Q-4. What is DFD? Create a DFD diagram on Flipkart

Answer: DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. It is a graphical tool, useful for communicating with users, managers and other personnel. it is useful for analyzing existing as well as proposed system.

It should be pointed out that a DFD is not a flowchart. In drawing the DFD, the designer has to specify the major transforms in the path of the data flowing from the input to the output. DFDs can be hierarchically organized, which helps in progressively partitioning and analyzing large systems.

It provides an overview of

- What data is system processes.
- What transformation are performed.
- What data are stored.
- What results are produced, etc.

Data Flow Diagram can be represented in several ways. The DFD belongs to structured-analysis modelling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.

Characteristics of DFD

- DFDs are commonly used during problem analysis.
- DFDs are quite general and are not limited to problem analysis for software requirements specification.
- DFDs are very useful in understanding a system and can be effectively used during analysis.
- It views a system as a function that transforms the inputs into desired outputs.
- The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced.
- The processes are shown by named circles and data flows are represented by named arrows entering or leaving the bubbles.
- A rectangle represents a source or sink and it is a net originator or consumer of data. A source sink is typically outside the main system of study.

Components of DFD

The Data Flow Diagram has 4 components:

- **Process** Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle. The process is named a short sentence, in one word or a phrase to express its essence
- **Data Flow** Data flow describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved. Data flow also represents material along with information that is being moved. Material shifts are modeled in systems that are not merely informative. A given flow should only transfer a single type of

information. The direction of flow is represented by the arrow which can also be bi-directional.

- **Warehouse** The data is stored in the warehouse for later use. Two horizontal lines represent the symbol of the store. The warehouse is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet. The data warehouse can be viewed independent of its implementation. When the data flow from the warehouse it is considered as data reading and when data flows to the warehouse it is called data entry or data updating.
- **Terminator** The Terminator is an external entity that stands outside of the system and communicates with the system. It can be, for example, organizations like banks, groups of people like customers or different departments of the same organization, which is not a part of the model system and is an external entity. Modeled systems also communicate with terminator.

Rules for creating DFD

- The name of the entity should be easy and understandable without any extra assistance (like comments).
- The processes should be numbered or put in ordered list to be referred easily.
- The DFD should maintain consistency across all the DFD levels.
- A single DFD can have a maximum of nine processes and a minimum of three processes.

Symbols Used in DFD

- **Square Box:** A square box defines source or destination of the system. It is also called entity. It is represented by rectangle.
- **Arrow or Line:** An arrow identifies the data flow i.e. it gives information to the data that is in motion.
- **Circle or bubble chart:** It represents as a process that gives us information. It is also called processing box.
- **Open Rectangle:** An open rectangle is a data store. In this data is store either temporary or permanently.

Levels of DFD

DFD uses hierarchy to maintain transparency thus multilevel DFD's can be created. Levels of DFD are as follows:

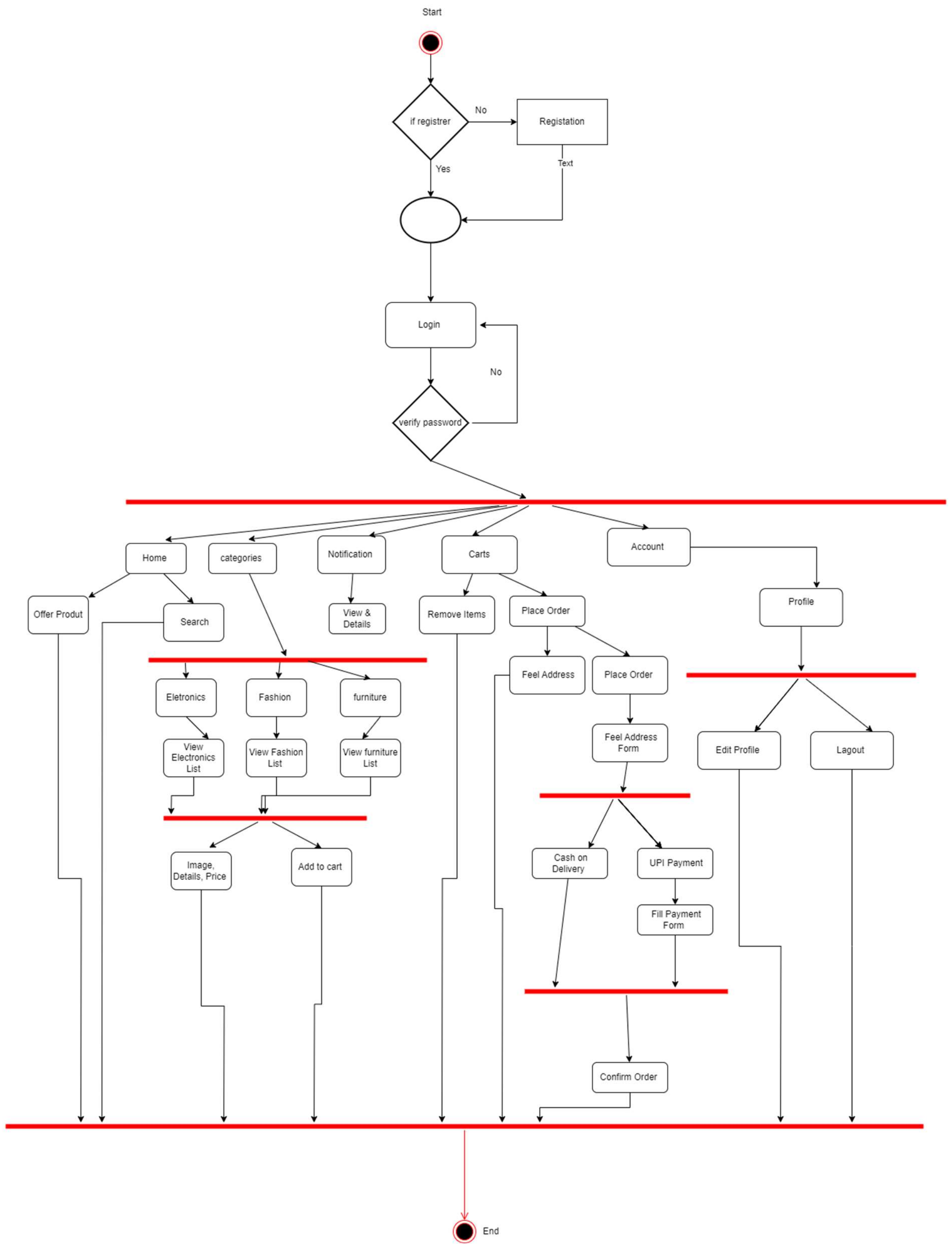
- 0-level DFD: It represents the entire system as a single bubble and provides an overall picture of the system.
- 1-level DFD: It represents the main functions of the system and how they interact with each other.
- 2-level DFD: It represents the processes within each function of the system and how they interact with each other.
- 3-level DFD: It represents the data flow within each process and how the data is transformed and stored.

Advantages of DFD

- It helps us to understand the functioning and the limits of a system.
- It is a graphical representation which is very easy to understand as it helps visualize contents.
- Data Flow Diagram represent detailed and well explained diagram of system components.
- It is used as the part of system documentation file.
- Data Flow Diagrams can be understood by both technical or nontechnical person because they are very easy to understand.

Disadvantages of DFD

- At times DFD can confuse the programmers regarding the system.
- Data Flow Diagram takes long time to be generated, and many times due to this reasons analyst are denied permission to work on it.


Start
if registrer
No
Registation
Yes
Text
Login
No
verify password
Home
categories
Notification
Carts
Account
Offer Produt
Search
View & Details
Remove Items
Place Order
Profile
Eletronics
Fashion
furniture
Feel Address
Place Order
View Electronics List
View Fashion List
View furniture List
Feel Address Form
Edit Profile
Lagout
Image, Details, Price
Add to cart
Cash on Delivery
UPI Payment
Fill Payment Form
Confirm Order
End

Q-5 What is Flow chart? Create a flowchart to make addition of two numbers
Answer: Flowcharts are nothing but the graphical representation of the data or the algorithm for a better understanding of the code visually. It displays step-by-step solutions to a problem, algorithm, or process. It is a pictorial way of representing steps that are preferred by most beginner-level programmers to understand algorithms of computer science, thus it contributes to troubleshooting the issues in the algorithm. A flowchart is a picture of boxes that indicates the process flow sequentially. Since a flowchart is a pictorial representation of a process or algorithm, it's easy to interpret and understand the process. To draw a flowchart, certain rules need to be followed which are followed by all professionals to draw a flowchart and are widely accepted all over the countries.

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

Uses of Flowcharts in Computer Programming/Algorithms

The following are the uses of a flowchart:

- It is a pictorial representation of an algorithm that increases the readability of the program.
- Complex programs can be drawn in a simple way using a flowchart.
- It helps team members get an insight into the process and use this knowledge to collect data, detect problems, develop software, etc.
- A flowchart is a basic step for designing a new process or adding extra features.
- Communication with other people becomes easy by drawing flowcharts and sharing them.

Flowcharts are mainly used in the below scenarios:

- It is most importantly used when programmers make projects. As a flowchart is a basic step to make the design of projects pictorially, it is preferred by many.
- When the flowcharts of a process are drawn, the programmer understands the non-useful parts of the process. So flowcharts are used to separate sound logic from the unwanted parts.
- Since the rules and procedures of drawing a flowchart are universal, a flowchart serves as a communication channel to the people who are working on the same project for better understanding.
- Optimizing a process becomes easier with flowcharts. The efficiency of the code is improved with the flowchart drawing.

Types of Flowcharts

Three types of flowcharts are listed below:

Process flowchart: This type of flowchart shows all the activities that are involved in making a product. It provides a pathway to analyze the product to be built. A process flowchart is most commonly used in process engineering to illustrate the relation between the major as well as minor components present in the product. It is used in business product modeling to help understand employees about the project requirements and gain some insight into the project.

Data flowchart: As the name suggests, the data flowchart is used to analyze the data, specifically it helps in analyzing the structural details related to the project. Using this flowchart, one can easily understand the data inflow and outflow from the system. It is most commonly used to manage data or to analyze information to and fro from the system.

Business Process Modeling Diagram: Using this flowchart or diagram, one can analytically represent the business process and help simplify the concepts needed to understand business activities and the flow of information. This flowchart illustrates the business process and models graphically which paves the way for process improvement**.**

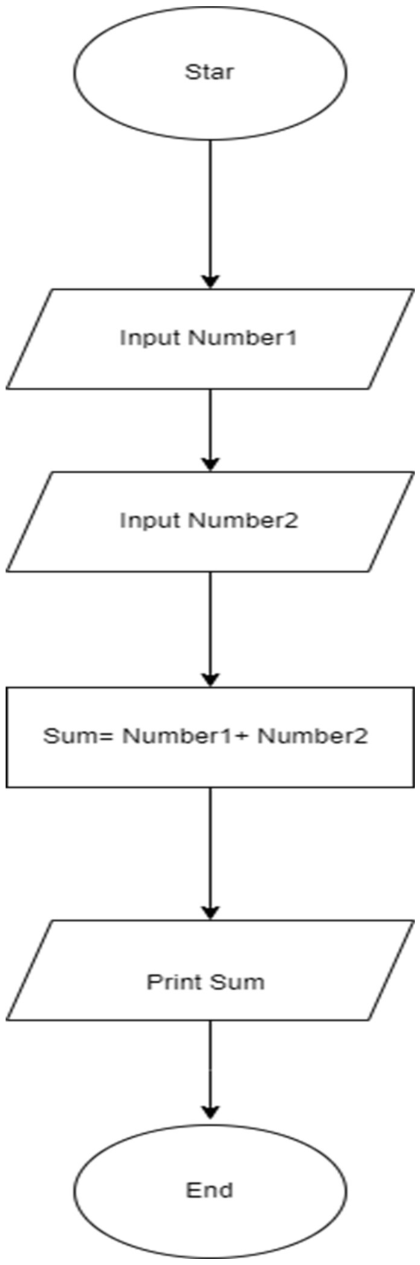


Fig 2 Flowchart to addition of two numbers

Q-6 What is Use case Diagram? Create a use-case on bill payment on Paytm.

Answer: A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.
A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.

2. Use Case Diagram Notations

UML notations provide a visual language that enables software developers, designers, and other stakeholders to communicate and document system designs, architectures, and behaviors in a consistent and understandable manner.

1.1. Actors
Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modeling system behavior.

1.2. Use Cases
Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be "Place Order," "Track Delivery," or "Update Product Information". Use cases are represented by ovals.
1.2. Use Cases
Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be "Place Order," "Track Delivery," or "Update Product Information". Use cases are represented by ovals.

Association Relationship
The Association Relationship represents a communication or interaction between an actor and a use case. It is depicted by a line connecting the actor to the use case. This relationship signifies that the actor is involved in the functionality described by the use case.
Example: Online Banking System

- **Actor:** Customer
- **Use Case:** Transfer Funds
- **Association:** A line connecting the "Customer" actor to the "Transfer Funds" use case, indicating the customer's involvement in the funds transfer process.

Include Relationship
The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.
Example: Social Media Posting

- **Use Cases:** Compose Post, Add Image

- **Include Relationship:** The “Compose Post” use case includes the functionality of “Add Image.” Therefore, composing a post includes the action of adding an image.

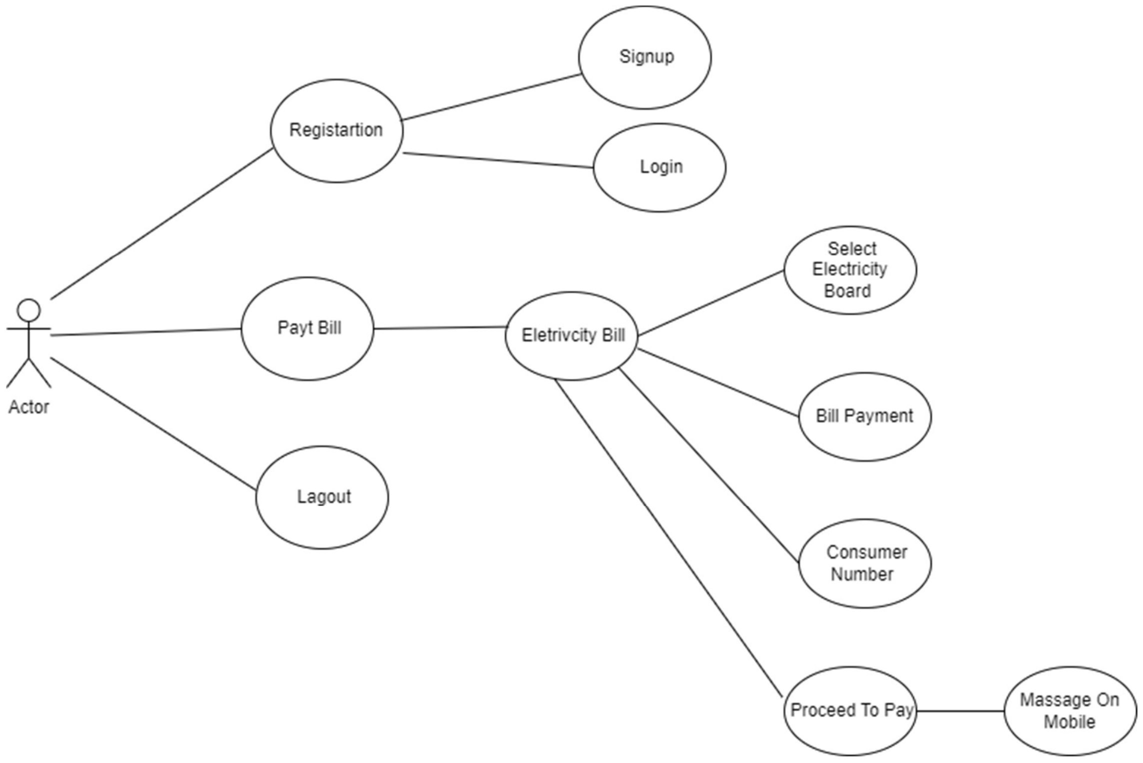
Extend Relationship

The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions. It is represented by a dashed arrow with the keyword “extend.” This relationship is useful for handling optional or exceptional behavior.

Example: Flight Booking System

- **Use Cases:** Book Flight, Select Seat
- **Extend Relationship:** The “Select Seat” use case may extend the “Book Flight” use case when the user wants to choose a specific seat, but it is an optional step.

O**nline Shopping System:**



1. Actors:
   - Customer
   - Admin
2. Use Cases:
   1. Browse Products
   2. Add to Cart
   3. Checkout
   4. Manage Inventory (Admin)
3. Relations:
   - The Customer can browse products, add to the cart, and complete the checkout.
   - The Admin can manage the inventory.