Assignments

Name: Roy Fenal Naranbhai

Software Testing

1. What Is SDLC ?

➢ SDLC is a structure imposed on the development of a software product that defines the process for planning, implementation, testing, documentation, deployment, and ongoing maintenance and support.

❖ SDLC Phase

Requirements Collection/Gathering	Establish Customer Needs
Analysis	Model And Specify the requirements- "What"
Design	Model And Specify a Solution – "Why"
Implementation	Construct a Solution In Software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to the new requirements

➢ Requirement Gathering In Three Type Problem :

- **Lack of clarity :** It is hard to write documents that are both precise and easy-to-read.

- **Requirements confusion :** Functional and Non-functional requirements tend to be intertwined.

- **Requirements Amalgamation :** Several different requirements may be expressed together.

❖ **Requirement Gathering :**

- Features
- Usage scenarios
- Plan for change
- Requirements will Change !

❖ **Types of Requirements :**

- **<u>Functional Requirements</u> :** Describe system services or functions.

- **<u>Non-Functional Requirements</u> :** Are constraints on the system or the development process.

➢ **<u>Analysis Phase :</u>**

- This phase defines the problem that the customer is trying to solve.
- The deliverable result at the end of this phase is a requirement document.

❖ **Design Phase :**

- Design Architecture Document
- Implementation Plan
- Performance Analysis
- Test Plan

➢ **<u>Implementation Phase :</u>**

- In the implementation phase, the team builds the components either Design

Architecture Document

- The end deliverable is the product itself.

➢ **Testing Phase :**

- Simply stated, quality is very important.
- Regression Testing
- Internal Testing
- Unit Testing
- Application Testing
- Stress Testing

➢ **Maintenance Phase** :

- Software maintenance is also one of the phases in the System Development Life Cycle
- Configuration and version management
- Updating all analysis, design and user documentation

❖ **Maintenance Phase (Content)**

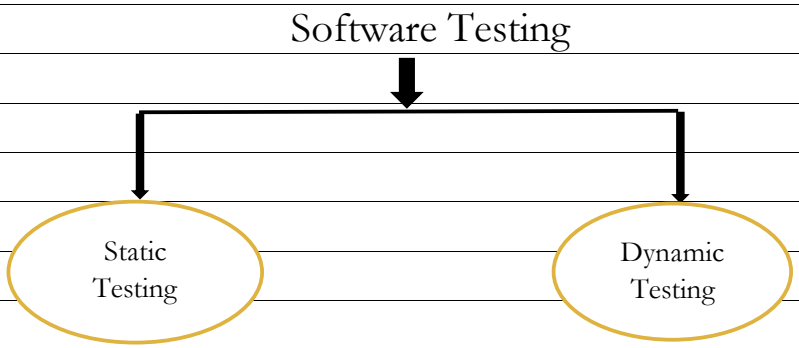
- **Corrective maintenance :** Identifying and repairing defects.

- **Adaptive maintenance :** Adapting the existing solution to the new platforms.

- **Perfective Maintenance :** Implementing the new requirements In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered "maintenance"!

2. What Is Software Testing ?

➢ Software Testing is a process used to identify the correctness, completeness, and quality of developed computer software.



❖ Static Testing :

➢ It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis.

❖ Dynamic Testing :

➢ In dynamic testing the software code is executed to demonstrate the result of running tests. It's done during validation process.

❖ Testing Activities :

- Planning and control
- Designing test cases
- Choosing test conditions
- Choosing test conditions
- Evaluating completion criteria
- Reporting on the testing process and system under test
- Finalizing or closure
- Testing also includes reviewing of documents

❖ Test Objectives :

- Finding defects.
- Gaining confidence in and providing information about the level of quality.
- Preventing defects.
- Both dynamic testing and static testing can be used as a means for achieving these Objectives.

❖ When to test ?

➢ For the betterment, reliability and performance of an Information System, it is always better to involve the Testing team right from the beginning of the Requirement Analysis phase.

❖ Why Testing is Necessary ?

- Testing is necessary because we all make mistakes.
- Some of those mistakes are unimportant, but some of them are expensive or dangerous.

Example :

1. Banking and Financial institutions
2. Transport
3. Medicine

❖ When to start Software testing ?

- All the high priority bugs are fixed.
- The rate at which bugs are found is too small.
- The testing budget is exhausted.
- The project duration is completed.
- The risk in the project is under acceptable limit.

❖ General Testing Principles :

1. Testing shows presence of Defects
2. Exhaustive Testing is Impossible!
3. Early Testing
4. Defect Clustering
5. The Pesticide Paradox
6. Testing is Context Dependent
7. Absence of Errors Fallacy

Project & Product

Project	Product
The main goal of a project is to form a new product that has not already been made.	The main goal of the product is to complete the work successfully (solve a specific problem).
Project is undertaken to form a new software.	Product is the final production of the project.
A project is done only once to get a new software.	A product can be made again and again for the purpose of distribution among users.
It is handled by the project managers.	It is handled by the product managers

❖ Software Architecture :

- Presentation Layer
- Application Layer
- Data Layer

❖ **Presentation Layer :**

- It is also known as the Client layer.
- The top most layer of an application.
- The main function of this layer is to communicate with the Application layer.

❖ **Application Layer :**

- It is also known as Business Logic Layer which is also known as the logical layer
- As per the Gmail login page example, once the user clicks on the login button, the Application layer interacts with the Database layer and sends required information to the Presentation layer.

❖ **Data Layer :**

- The data is stored in this layer.
- The application layer communicates with the Database layer to retrieve the data.
- It contains methods that connect the database and performs required action.

❖ **<u>Types of Software Architecture :</u>**

1. One Tier Architecture
2. Two Tier Architecture
3. Three Tier Architecture
4. N-Tier Architecture

❖ **One Tier Architecture :**

➢ One-tier architecture has all the layers such as Presentation, Business, Data Access layers in a single software package.

➢ The data is stored in the local system or a shared drive.

❖ Two Tier Architecture :

- The Two-tier architecture is divided into two parts:
  - Client Application (Client Tier)
  - Database (Data Tier)
- The client system handles both Presentation and Application layers and the Server system handles the Database layer.

❖ Three Tier Architecture :

- The Three-tier architecture is divided into three parts:
  - Presentation layer (Client Tier)
  - Application layer (Business Tier)
  - Database layer (Data Tier)
- The client system handles the Presentation layer, the Application server handles the Application layer, and the Server system handles the Database layer.

❖ N-Tier Architecture :

- N-tier architecture refers to the structure of a software application divided into multiple tiers. A tier is a layer of the application that operates on its own infrastructure or server, while a layer is a division of the application that carries out a discrete set of functions.

❖ <u>System Environments :</u>

DEV (dev tests passed)

↓

Deploy to QA (QA integration tests passed)

↓

Deploy to UAT (UAT acceptance tests passed)

↓

Deploy to Prod

3. What is agile methodology ?

- In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

- Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

- Agile Methods break the product into small incremental builds.

- These builds are provided in iterations.

- At the end of the iteration a working product is displayed to the customer and important stakeholders.

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.

❖ Agile (Pros) :

- Is a very realistic approach to software development.
- Resource requirements are minimum.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

❖ Agile (Cons) :

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.
- There is very high individual dependency, since there is minimum documentation generated.

❖ **Agile Manifesto :**

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

4. What is SRS ?

➢ A software requirements specification (SRS) is a complete description of the behaviour of the system to be developed.

➢ Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements.

➢ This standard describes possible structures, desirable contents, and qualities of a software requirements specification.

❖ **Types of Requirements :**

- Customer Requirements
- Functional Requirements
- Non-Functional Requirements

❖ **Customer Requirements :**

➢ The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer.

➢ Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing :

- Operational distribution or deployment: Where will the system be used ?
- Mission profile or scenario : How will the system accomplish its mission Objective ?
- Utilization environments : How are the various system components to be Used ?

❖ Functional Requirements :

➢ Functional Requirements are very important system requirements in the system design process. These requirements are the technical specifications, system design parameters and guidelines, data manipulation, data processing, and calculation modules etc , of the proposed system.

Example:

- The following are the requirements of Google Email Service
- The system shall support the ability to receive emails
- The system shall support the ability to send emails

❖ Non-Functional Requirements :

➢ Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviours.

➢ Example non-functional requirements for a system include :

- system must run on Windows Server 2003
- system must be secured against Trojan attacks

➢ Non-functional requirements can be divided into following categories :

- Usability
- Reliability
- Performance
- Security

5. What is oops ?

- Identifying objects and assigning responsibilities to these objects.
- Objects communicate to other objects by sending messages.
- Messages are received by the methods of an object
- An object is like a black box.
- The internal details are hidden.
- Object-oriented programming has a web of interacting objects, each house-keeping its own state.
- Objects of a program interact by sending messages to each other.

❖ **Everything in the world is an object :**

- A flower, a tree, an animal
- A student, a professor
- A university, a city, a country
- The world, the universe

6. Write Basic Concepts of oops ?

- Object
- Class
- Encapsulation
- Inheritance
- Polymorphism
  - → Overriding
  - → Overloading
- Abstraction

7. What is object ?

- An object represents an individual, identifiable item, unit, or entity, either real, abstract, with a well-defined role in the problem domain.
- An "object" is anything to which a concept applies.
- This is the basic unit of object oriented programming(OOP).
- That is both data and function that operate on data are bundled as a unit called as object.

❖ The two parts of an object :

- Object = Data + Methods

❖ What is an object ?

- Tangible Things - as a car, printer
- Roles - as employee, boss
- Incidents - as flight, overflow
- Interactions - as contract, sale
- Specifications - as colour, shape

8. What is class ?

- When you define a class, you define a blueprint for an object.
- A class represents an abstraction of the object and abstracts the properties and behaviour of that object.
- An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class.
- We do not actually buy these blueprints but the actual objects.

❖ **The two steps of Object Oriented Programming :**

- Making Classes
- Making Objects interact

❖ **Making Classes :**

Creating, extending or reusing abstract data types.

❖ **Making Objects interact :**

Creating objects from abstract data types and defining their relationships.

9. What is encapsulation ?

- Encapsulation is the practice of including in an object everything it needs hidden from other objects. The internal state is usually not accessible by other objects.
- Encapsulation is placing the data and the functions that work on that data in the same place.
- Encapsulation in Java is the process of wrapping up of data (properties) and behaviour (methods) of an object into a single unit and the unit here is a Class (or interface).
- Encapsulate in plain English means to enclose or be enclosed in or as if in a capsule. In Java, a class is the capsule (or unit).

❖ **Encapsulation (Cont…)**

- Encapsulation enables data hiding, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user.
- We can expose our operations hiding the details of what is needed to perform that operation.

10. What is inheritance ?

- Inheritance means that one class inherits the characteristics of another class. This is also called a "is a" relationship

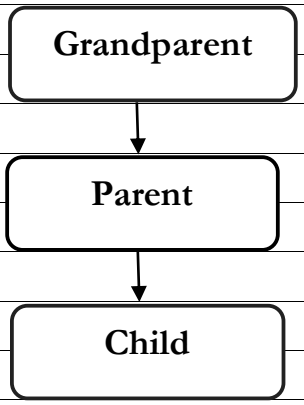
- One of the most useful aspects of object-oriented programming is code reusability.

- As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed as derived class.

- This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

- Inheritance describes the relationship between two classes.

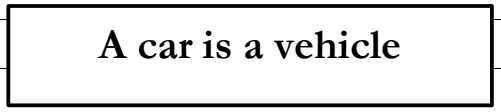
- In a class context, inheritance is referred to as implementation inheritance, and in an interface context, it is also referred to as interface inheritance.

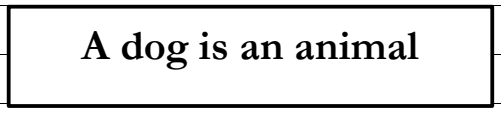


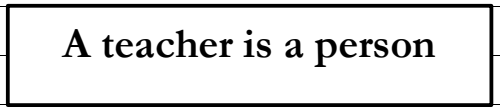
❖ Inheritance(Cont…)

- For example consider a Vehicle parent class and its child class Car.

→ Here, Vehicle is known as base class, parent class, or super class.
→ Car is known as derived class, Child class or subclass.







11. What is polymorphism ?

- Polymorphism means "having many forms".

- It allows different objects to respond to the same message in different ways, the response specific to the type of the object.

- The most important aspect of an object is its behaviour (the things it can do) A behaviour is initiated by sending a message to the object (usually by calling a method).

- Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

❖ The ability to change form is known as polymorphism.

- **There is two types of polymorphism in Java :**

    1. Compile time polymorphism(Overloading)
    2. Runtime polymorphism(Overriding)

❖ **Overloading :**

- The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

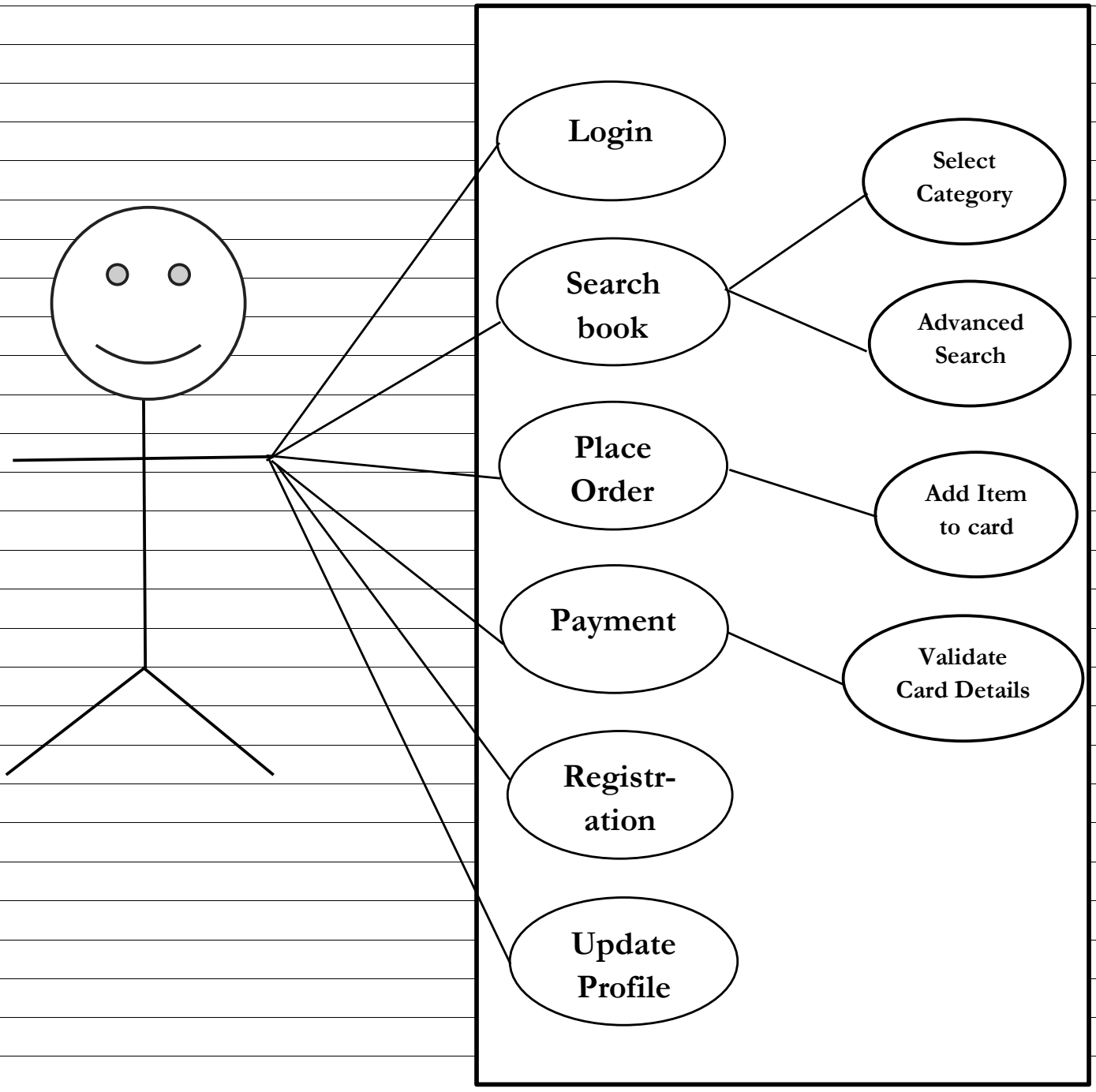
- The same method name (method overloading) or operator symbol (operator overloading) can be used in different contents.

- In method overloading, multiple methods having same name can appear in a class, but with different signature.

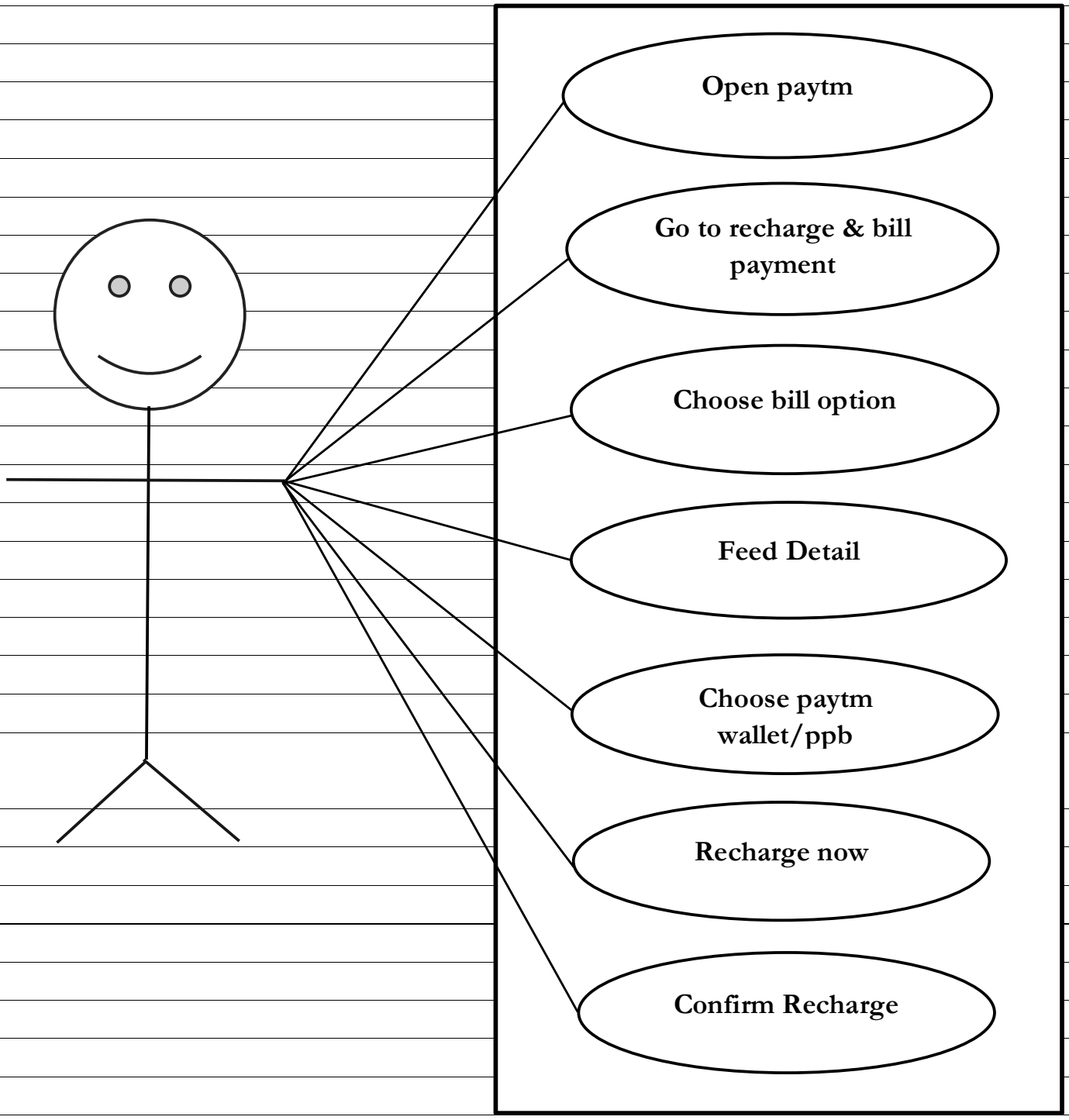
- And based on the number and type of arguments we provide while calling the method, the correct method will be called.

- Java doesn't allow operator overloading yet + is overloaded for class String. The '+' operator can be used for addition as well as string concatenation.

12. Draw Use case on Online book shopping



13. Draw Use case on online bill payment system (paytm) :



14. Write SDLC phases with basic introduction.

Requirements Collection/Gathering	Establish Customer Needs
Analysis	Model and Specify the requirements- "What"
Design	Model And Specify a Solution – "Why"
Implementation	Construct a Solution In Software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to the new requirements

❖ Requirement Gathering :

- Features
- Usage scenarios
- Plan for change
- Functional and Non-Functional
- Requirements will Change!

❖ Requirement Gathering(Cont…)

- Requirements definitions usually consist of natural language, supplemented by diagrams and tables.

❖ Three types of problems can arise:

- **Lack of clarity :** It is hard to write documents that are both precis and easy-to read.

- **Requirements confusion :** Functional and Non-functional requirements tend to be intertwined.

- **Requirements Amalgamation :** Several different requirements may be expressed together.

❖ **Requirement Gathering(Cont…)**

- Types of Requirements :

1. **Types of Requirements :** describe system services or functions.

2. **Non-Functional Requirements :** are constraints on the system or the development process.

❖ **<u>Analysis Phase :</u>**

- This phase defines the problem that the customer is trying to solve.

- The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.

- The deliverable result at the end of this phase is a requirement document.

- This analysis represents the "what" phase.

- The architecture defines the components, their interfaces and behaviours.

- The deliverable design document is the architecture.

❖ **<u>Design Phase :</u>**

- Design Architecture Document
- Implementation Plan
- Critical Priority Analysis
- Performance Analysis
- Test Plan

- The Design team can now expand upon the information established in the requirement document.
- The requirement document must guide this decision process.

❖ **Implementation Phase :**

- In the implementation phase, the team builds the components either from scratch or by composition.

- The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.

- The end deliverable is the product itself. There are already many established techniques associated with implementation.

❖ **Testing Phase :**

- Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.

- It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.

- Quality is a distinguishing attribute of a system indicating the degree of excellence.

- Regression Testing

- Internal Testing

- Unit Testing

- Application Testing

- Stress Testing

❖ Testing Phase(Cont…)

- The testing phase is a separate phase which is performed by a different team after the implementation is completed.

- There is merit in this approach; it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times.

- If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases.

- an attitude change must take place to guarantee quality.

- Regardless if testing is done after the-fact or continuously, testing is usually based on a regression technique split into several major focuses, namely internal, unit, application, and stress.

❖ Maintenance Phase :

- Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.

- The developing organization or team will have some mechanism to document and track defects and deficiencies.

- configuration and version management

- reengineering (redesigning and refactoring)

- updating all analysis, design and user documentation

- Repeatable, automated tests enable evolution and refactoring

❖ **Maintenance Phase(Cont…)**

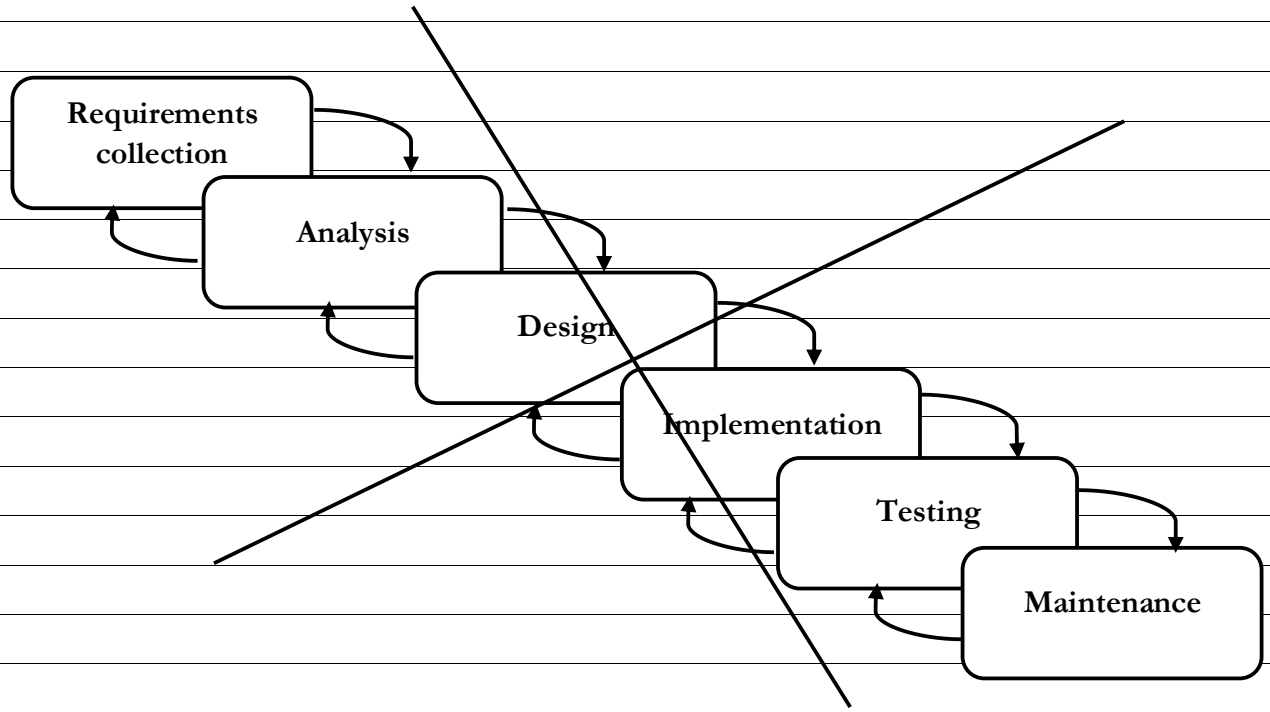
Maintenance is the process of changing a system after it has been deployed.

- **Corrective maintenance :** identifying and repairing defects
- **Adaptive maintenance :** adapting the existing solution to the new platforms.
- **Perfective Maintenance :** implementing the new requirements In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered "maintenance"!

→ Software just like most other products is typically released with a known set of defects and deficiencies.

15. Explain Phases of the waterfall model.

➢ The Classical Software lifecycle models the Software Development as a step-By-step " Waterfall" between the various development phases.



- **The waterfall is unrealistic for many reasons, especially :**

  - Requirements must be "**frozen**" to early in the life cycle.

  - Requirements are validated too late

❖ **Applications (When to use?)**

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

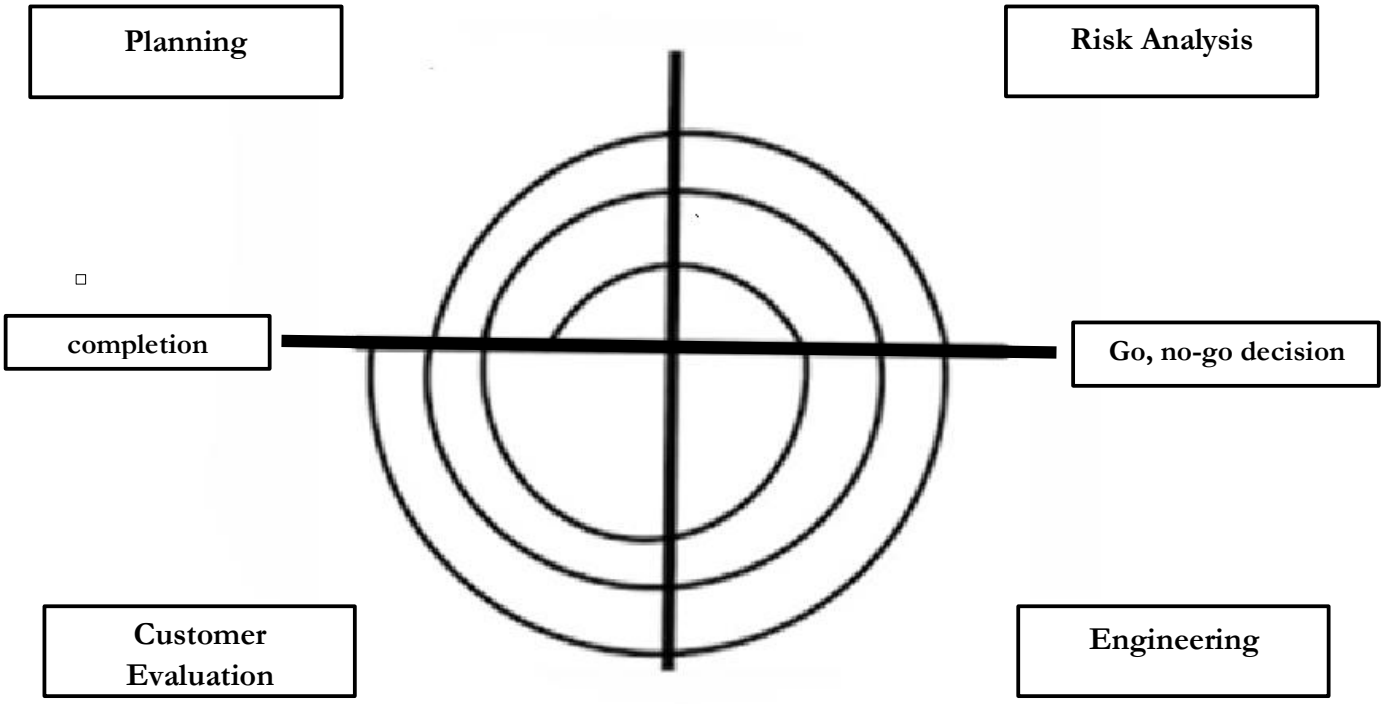
❖ **Pros (Why Waterfall Model)**

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

❖ **Cons (Why not Waterfall Model)**

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- It is difficult to measure progress within stages.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

16. Write phases of spiral model.



❖ Application :

➢ Spiral Model is very widely used in the software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves Minimum risk for the customer as well as the development firms.
Following are the typical uses Spiral model :

- When costs there are a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which are usually the case.
- Requirements are complex and need evaluation to get clarity.

- Significant changes are expected in the product during the development cycle. cycle.

❖ **Why It works ?**

- Changing requirements can be accommodated.
- Allows for extensive use of prototypes
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

❖ **Why It doesn't work ?**

- Why It doesn't work
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

17. Write agile manifesto principles

- The four core values of Agile software development as stated in the Agile Manifesto are as follows:

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a project plan.

- **Individuals and interactions over processes and tools :**

- Suppose the team finds any issue in software then they search for another process or tool to resolve the issue. But, in Agile, it is preferable to interact with client, manager or team regarding issue and make sure that the issue gets resolved.

- **Working software over comprehensive documentation :**

- Documentation is needed, but working software is much needed. Agile is not saying that documentation is not needed, but working software is much needed. For example, you have 20-page documents, but you do not have a single prototype of the software. In such a case, the client will not be happy because, in the end, the client needs a document.

- **Customer collaboration Over contract negotiation :**

- Contract negotiation is important as they make the budget of software, but customer collaboration is more important than over contract negotiation. For example, if you stuck with the requirements or process, then do not go for a contract which we have negotiated. You need to interact with the customer, gather their requirements.

- **Responding to change over following a project plan :**

- In the waterfall model, everything is planned, i.e., at what time, each phase will be completed. Sometimes you need to implement the new requirements in the middle of the software, so you need to be versatile to make changes in the software.

18. **Explain working methodology of agile model and also write pros and cons.**

➢ Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

➢ Agile Methods break the product into small incremental builds.

➢ These builds are provided in iterations.

➢ Each iteration typically lasts from about one to three weeks.

➢ Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

➢ At the end of the iteration a working product is displayed to the customer and important stakeholders.

❖ **What is Agile ?**

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

- Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

❖ **Pros**

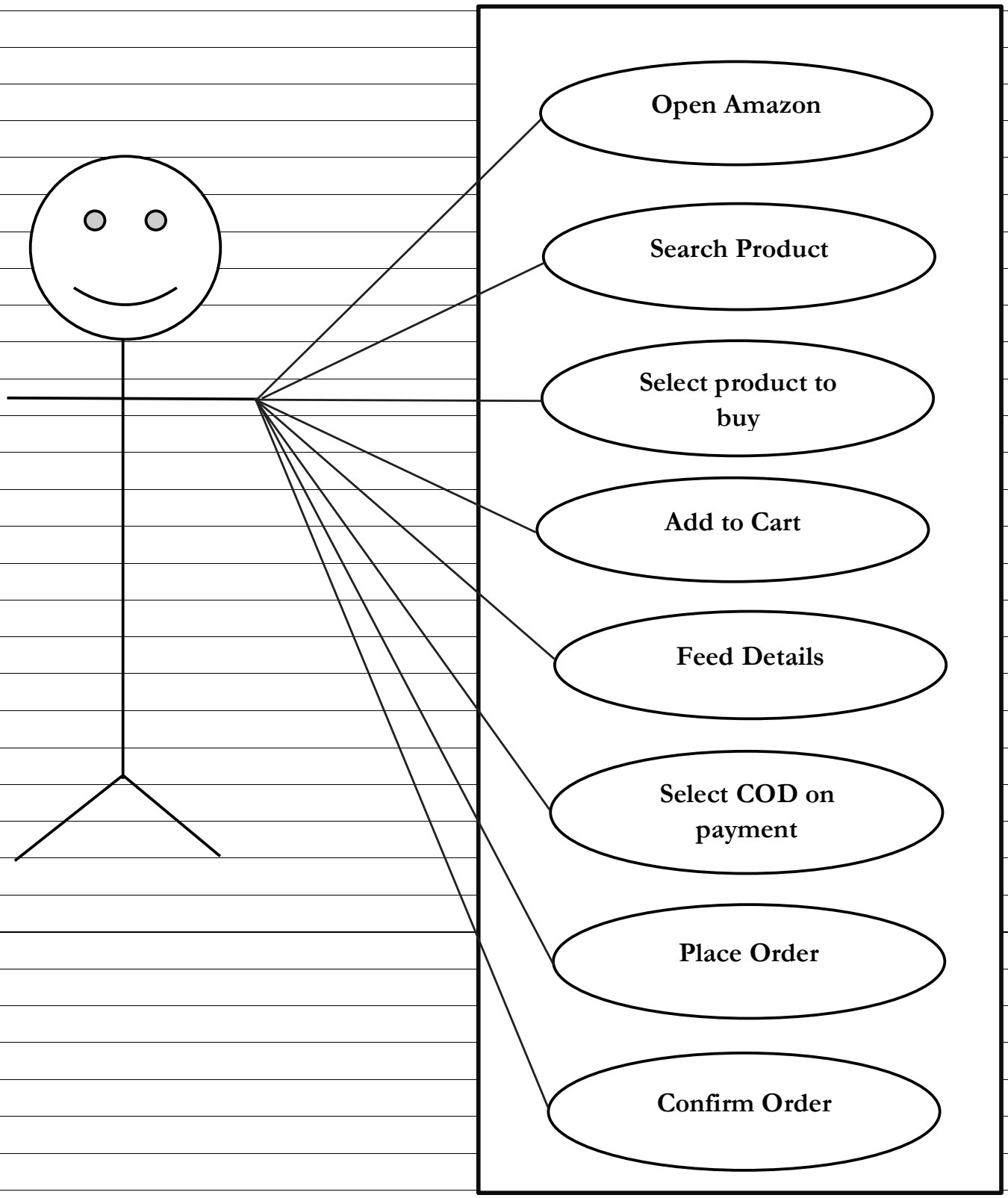
- Is a very realistic approach to software development

- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required
- Easy to manage
- Gives flexibility to developers

❖ **Cons**

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.

19. **Draw use case on Online shopping product using COD.**



20. Draw use case on Online shopping product using payment gateway.