Software Testing Assignment

Module 1

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.

There are a number of different development models.

- Requirement Gathering/collection
- Analysis
- Designing
- Implementation
- Testing
- Maintenance

2. What is Software Testing?

- Software Testing is a process used to identify the correctness, completeness and quality of developed computer software.
- In simple words testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements.

3. What is agile methodology?

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.

4. What is SRS?

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

It includes a set of use cases that describe all of the interactions that the users will have with the software.

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

Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

Recommended approaches for the specification of software requirements are described by IEEE 830-1998.

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

- Types of Requirement:-

  1 Customer Requirements
  2 Functional Requirements
  3 Non-Functional Requirements

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 is derived from abstract data type 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.

- Concept of OOP:-

  1 Object
  2 Class
  3 Encapsulation
  4 Inheritance
  5 Polymorphism
    - Overriding
    - Overloading
  6 Abstraction

6. Write Basic Concepts of oops?

1. **Object:-** Any Entity Which has its own state and behaviour that is called an object.
   Example: Any Living things

2. **Class:-** Collection of objects is called Class.
   Example: Human Body

3. **Encapsulation:-** Binding of data / wrapping up of data.
   Example: Capsule
4. **Inheritance:-** When one object acquires all the properties and behaviour of the parent class.
   Example: Father - Son

5. **Polymorphism:-** Many ways to perform anything.
   Example: Road ways.

6. **Abstraction:-** Hiding internal details and showing functionalities.

7. What is an Object?

An object represents an individual, identifiable item, unit, or entity, either real or 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 functions that operate on data are bundled as a unit called an object.

8. What is Class?

When you define a class, you define a blueprint for an object.

This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class represents an abstraction of the object and abstracts the properties and behaviour of that object.

Class can be considered as the blueprint or definition or a template for an object and describes the properties and behaviour of that object, but without any actual existence.

An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class.

In the case of a car or laptop, there will be a blueprint or design created first and then the actual car or laptop will be built based on that. We do not actually buy these blueprints but the actual objects.

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.

While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you a framework to place the data and the relevant functions together in the same object.

Encapsulation in Java is the process of wrapping up 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).

In Java, everything is enclosed within a class or interface, unlike languages such as C and C++, where we can have global variables outside classes.

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.

We can protect the internal state of an object by hiding its attributes from the outside world (by making it private), and then exposing them through setter and getter methods. Now modifications to the object internals are only controlled through these methods.

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 called 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. A class can get some of its characteristics from a parent class and then add unique features of its own.

In general, Java supports single-parent, multiple-children inheritance and multilevel inheritance (Grandparent-> Parent -> Child) for classes and interfaces. Java supports multiple inheritances (multiple parents, single child) only through interfaces.

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

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.

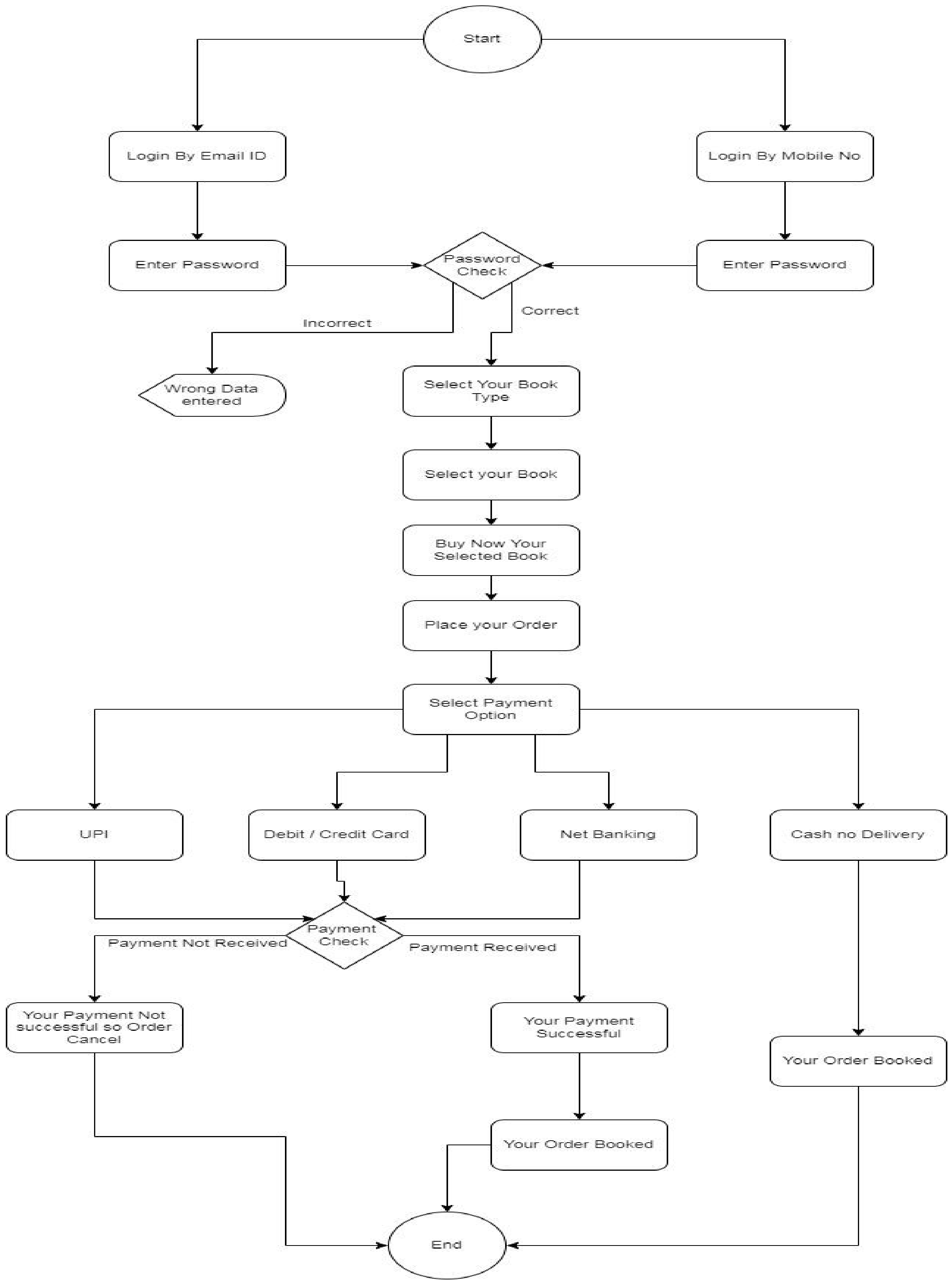
The most important aspect of an object is its behaviour (the things it can do). A behaviour is initiated by sending a message to the object (usually by calling a method).

The ability to use an operator or function in different ways, in other words giving different meaning or functions to the operators or functions is called polymorphism.

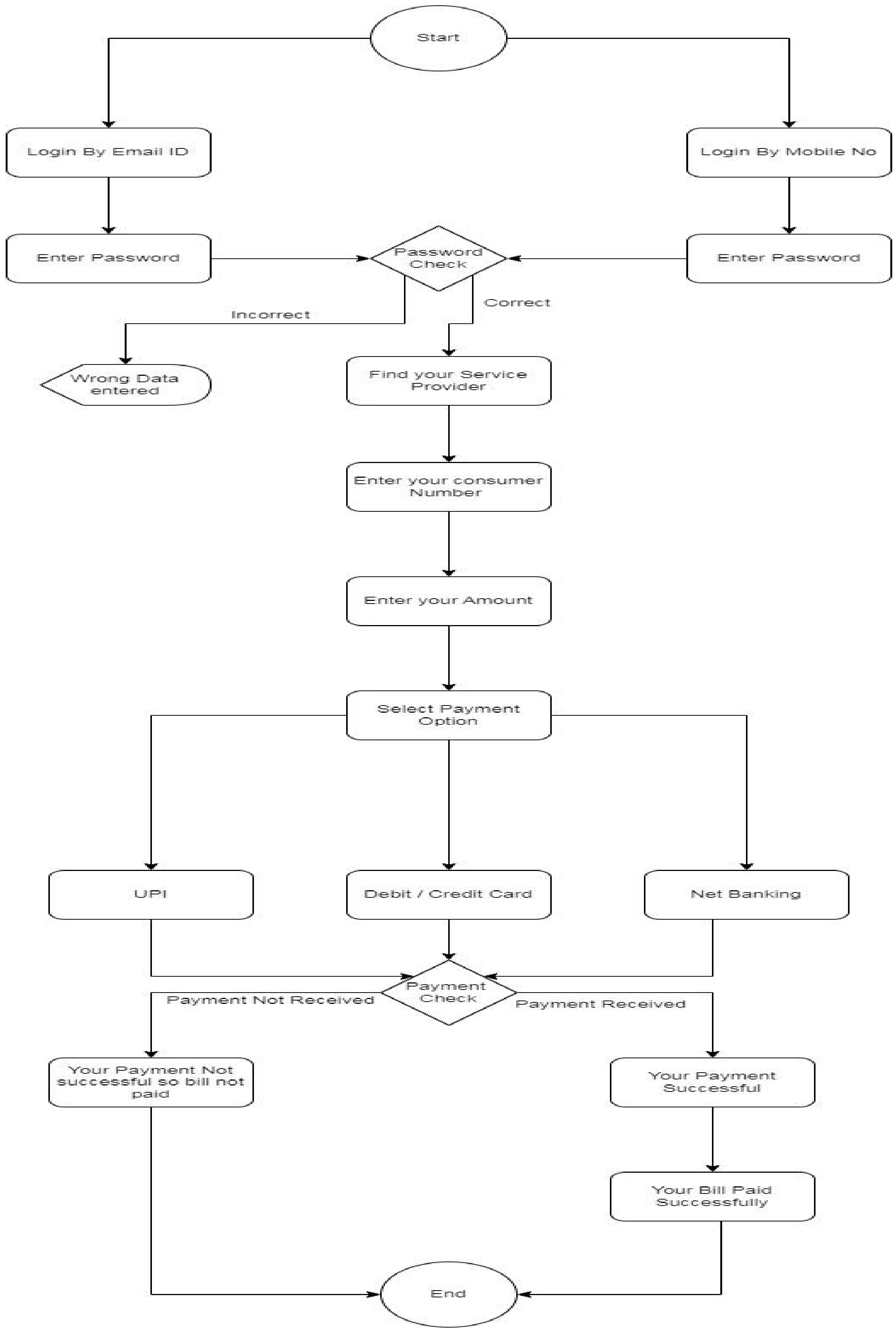
There is two types of polymorphism in Java

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

12. Draw Usecase on Online book shopping



13. Draw Usecase on online bill payment system (paytm)



14. Write SDLC phases with basic introduction?

There are 6 types of phases

1. **Requirement Gathering:-** The first phase of SDLC is requirement gathering from customers. Here we gathered requirements of functional and nonfunctional requirements by customer.

2. **Analysis:-** The second phase of SDLC is analysis of gathered requirements from customers. And analyse how these requirements will be accomplished.

3. **Design:-** The third phase of SDLC is the design phase, on the basis of analysis of gathered requirements. In this phase the software design documents are prepared. There are two kinds of design documents: HLD & LLD.

4. **Implementation:-** The fourth phase of SDLC is implementation In this phase developers start the develop build by writing the code.

5. **Testing:-** The fifth phase of SDLC is testing, testing for giving bug free and quality products to the customer. In testing phase tester will check the application by the help of comparing expected result and actual result.

6. **Maintenance:-** 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”!

15. Explain Phases of the waterfall model?

1. Requirement Gathering
2. Analysis
3. Design
4. Implementation
5. Testing
6. Maintenance

The waterfall is unrealistic for many reasons, especially:

1. Requirements must be “frozen” to early in the life cycle
2. Requirements are validated too late

The Classical software lifecycle models the software development as a step by step waterfall between the various development phases.

16. Write phases of spiral model?

1. Planning
2. Risk Analysis
3. Engineering
4. Customer Evaluation

Spiral Model is very widely used in the software industry as it is in sync 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 of the Spiral model.

When costs 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.

Customers are not sure of their requirements which are usually the case.

Requirements are complex and need evaluation to get clarity.

New product line which should be released in phases to get enough customer feedback.

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

- Pros (Why It works)
    1. Users see the system early.
    2. Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.
    3. Changing requirements can be accommodated.
    4. Allows for extensive use of prototypes
    5. Requirements can be captured more accurately.

- Cons (Why It doesn't work)
    1. Management is more complex.
    2. End of the project may not be known early.
    3. Not suitable for small or low risk projects and could be expensive for small projects.
    4. Process is complex
    5. Spiral may go indefinitely.
    6. Large number of intermediate stages requires excessive documentation.

17. Explain the working methodology of the agile model and also write pros and cons.

The 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.

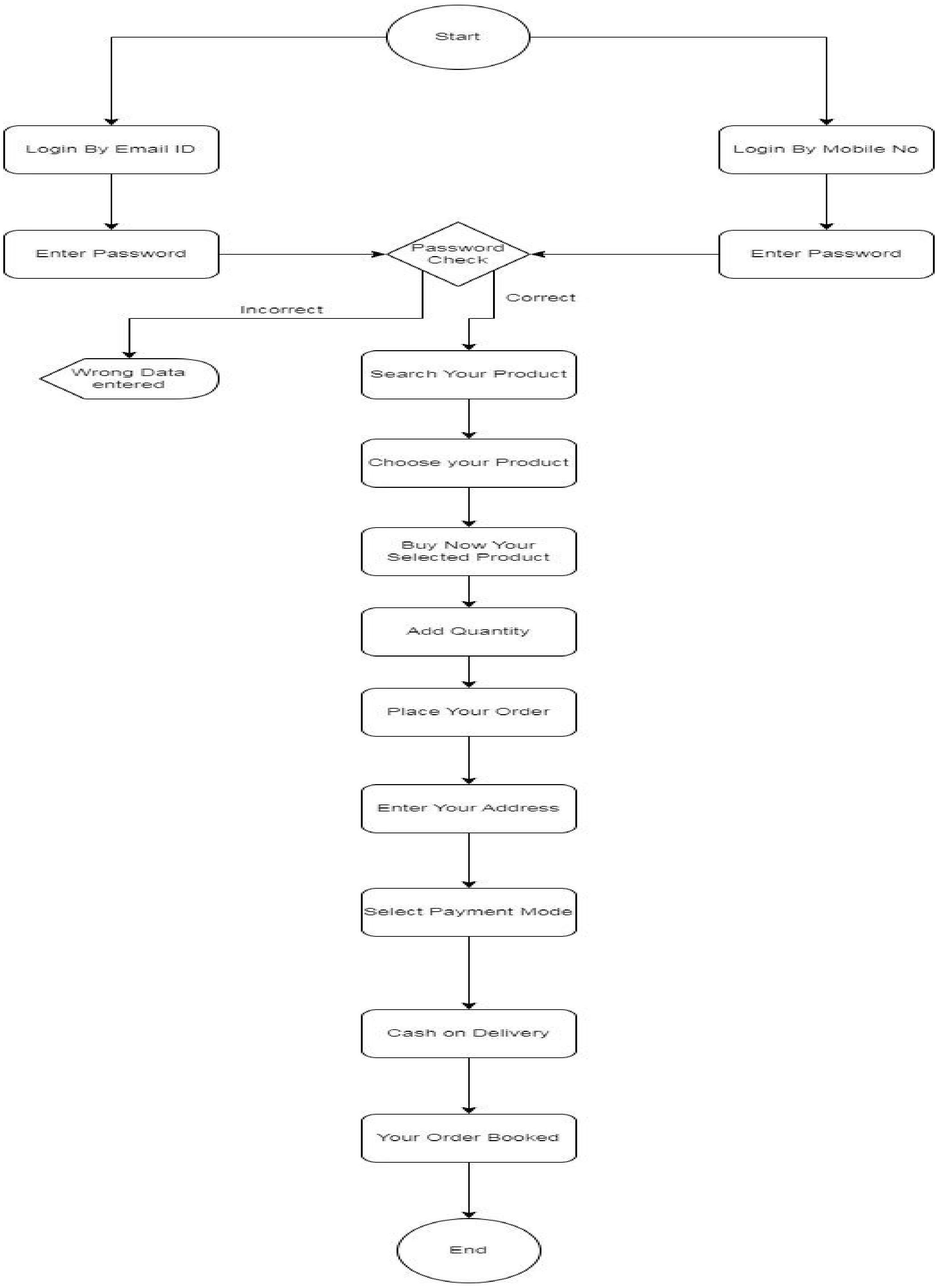
Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

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

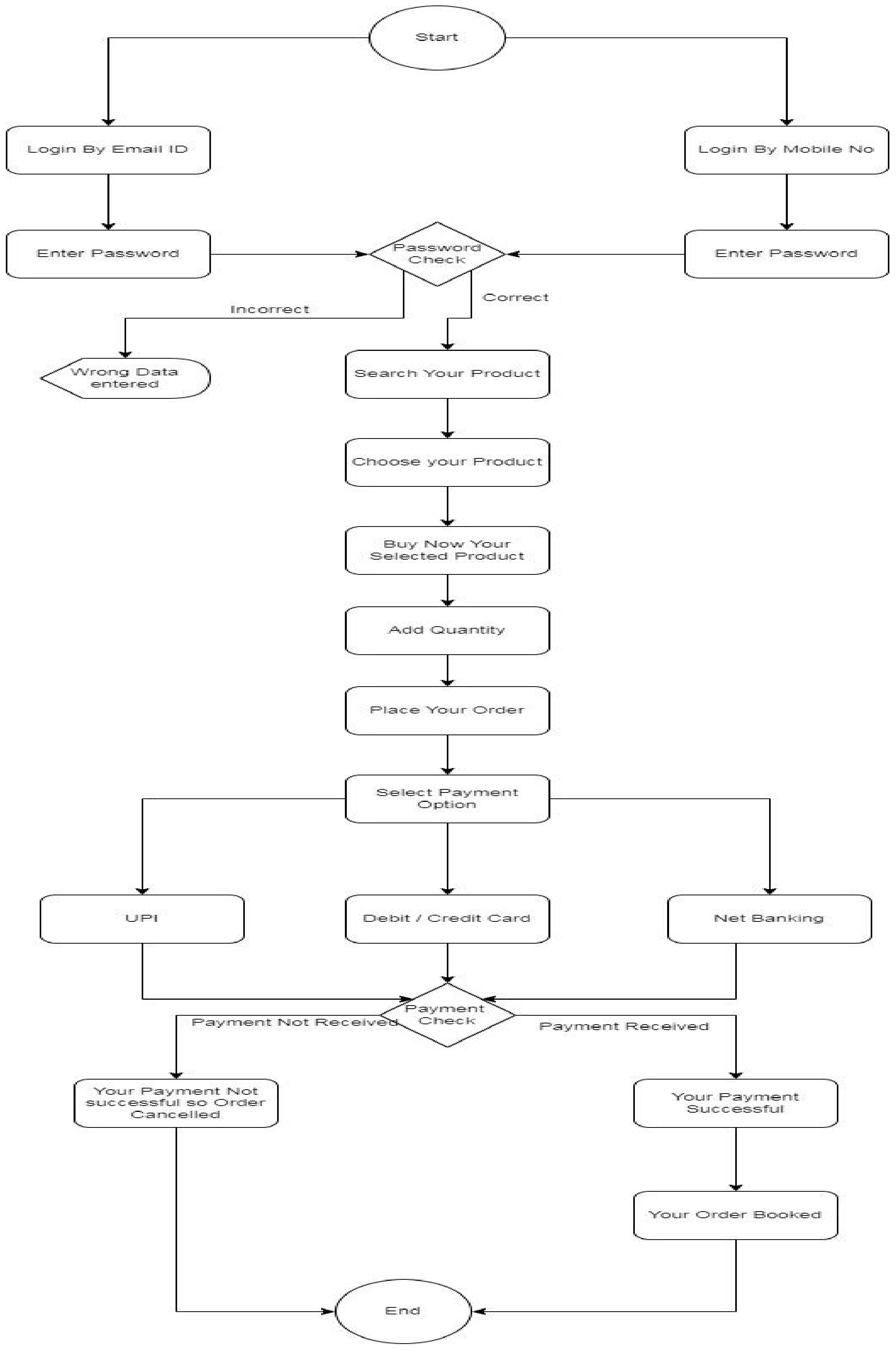
- Pros (Why It works)
    1. Good model for environments that change steadily.
    2. Minimal rules, documentation easily employed.
    3. Enables concurrent development and delivery within an overall planned context.
    4. Little or no planning required
    5. Easy to manage
    6. Gives flexibility to developers

- Cons (Why It doesn't work)
    1. Not suitable for handling complex dependencies.
    2. More risk of sustainability, maintainability and extensibility.
    3. An overall plan, an agile leader and agile PM practice is a must without which it will not work.
    4. Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.

18. Draw use case on Online shopping product using COD.



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



20. What are 7 key principles? Explain in detail?

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

1. Testing shows presence of Defects:-

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness. We test to find Faults As we find more defects, the probability of undiscovered defects remaining in a system reduces. However Testing cannot prove that there are no defects present.

2. Exhaustive Testing is Impossible:-

Testing everything including all combinations of inputs and preconditions are not possible.
So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts.
For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30 517 578 125 (515) tests.This is very unlikely that the project timescales would allow for this number of tests. So, accessing and managing risk is one of the most important activities and reasons for testing in any project. We have learned that we cannot test everything (i.e. all combinations of inputs and pre-conditions).That is we must Prioritise our testing effort using a Risk Based Approach.

3. Early Testing:-

Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives. Testing activities should start as early as possible in the development life cycle These activities should be focused on defined objectives – outlined in the Test Strategy Remember from our Definition of Testing, that Testing doesn’t start once the code has been written.

4. Defect Clustering:-

A small number of modules contain most of the defects discovered during pre-release testing, or are responsible for the most operational failures. Defects are not evenly spread in a system. They are ‘clustered’ In other words, most defects found during testing are usually confined to a small number of modules. Similarly, most operational failures of a system are usually confined to a small number of modules. An important consideration in test prioritisation.

5. The Pesticide Paradox:-

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects. Testing identifies bugs, and programmers respond to fix them As bugs are eliminated by the programmers, the software improves As software improves the effectiveness of previous tests erodes 28. Therefore we must learn, create and use new tests based on new techniques to catch new bugs N.B It's called the "pesticide paradox" after the agricultural phenomenon, where bugs such as the boll weevil build up tolerance to pesticides, leaving you with the choice of ever-more powerful pesticides followed by ever-more powerful bugs or an altogether different approach.’ – Beizer 1995

6. Testing is Context Dependent:-

Testing is basically context dependent. Testing is done differently in different contexts. Different kinds of sites are tested differently. For example Safety – critical software is tested differently from an e-commerce site. Whilst, Testing can be 50% of development costs, in NASA's Apollo program it was 80% testing 3 to 10 failures per thousand lines of code (KLOC) typical for commercial software 1 to 3 failures per KLOC typical for industrial software 0.01 failures per KLOC for NASA Shuttle code! Also different industries impose different testing standards.

7. Absence of Errors Fallacy:-

If the system built is unusable and does not fulfil the user’s needs and expectations then finding and fixing defects does not help. If we build a system and, in doing so, find and fix defects .... It doesn’t make it a good system Even after defects have been resolved it may still be unusable and/or does not fulfil the users’ needs and expectations