* 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. A Software Development Life Cycle is essentially a series of steps, or phases, that provide a

• model for the development and lifecycle management of an application or piece of software. The methodology within the SDLC process can vary across industries and organizations, but

• standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition, and configuration of software systems.

**elow is the diagrammatic representation of the SDLC cycle:**



Requirements will Change!

User and business needs change during the project

Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.

Build constant feedback into the project plan Plan for change

• Early prototyping [e.g., UI] can help clarify the requirements

• Functional and Non-Functional

• Requirements definitions usually consist of natural language, supplemented by (e.g.,• UML) diagrams and tables.

Types of Requirements:

• Functional Requirements: describe system services or functions.

• Compute sales tax on a purchase

• Update the database on the server

• Non-Functional Requirements: are constraints on the system or the development

• process. Non-functional requirements may be more critical than functional requirements.

• If these are not met, the system is useless!

Three types of problems can arise:

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

• Requirements confusion: Functional and Non-functional requirements tend to be

intertwined.

Requirements Amalgamation: Several different requirements may be expressedtogether.

### 2) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

### #3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

### #4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer’s expectation. Testers refer SRS document to make sure that the software is as per the customer’s standard.

### #5) Deployment

Once the product is tested, it is deployed in the production environment or first [UAT (User Acceptance testing)](https://www.softwaretestinghelp.com/what-is-user-acceptance-testing-uat/) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

### #6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What is software testing?**

* Software Testing is a process used to identify the correctness, completeness, and quality of developed computer software
* Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not.
* 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

**What is agile methodology?**

**Agile test plan** includes types of testing done in that iteration like test data requirements, infrastructure, test environments, and test results. Unlike the waterfall model, in an agile model, a test plan is written and updated for every release. Typical test plans in agile includes



## Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

**1. Requirements gathering:** In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility

**2. Design the requirements:** When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

**3. Construction/ iteration:** When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

**4. Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

**5. Deployment:** In this phase, the team issues a product for the user's work environment.

**6. Feedback:** After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feed back

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

What is SRS

SRS means Software Requirements Specification

A software requirements specification (SRS) is a complete description of the behavior 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 Requirements

Requirements are categorized in several ways. The following are common categorizations of

• requirements that relate to technical management: 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?

• Performance and related parameters: What are the critical system parameters to

• accomplish the mission? Utilization environments: How are the various system components to be used?

• Effectiveness requirements: How effective or efficient must the system be in

• performing its mission?

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. For 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

• The system shall support the ability to create new folders

• The system shall support the ability to filter emails in different folders

• The system shall support the ability to attach different kind of attachment

• The system shall support the ability to create and maintain address book

• The system shall support the ability to create unlimited user accounts with different

• email addresses

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 behaviors. Non-functional requirements are qualities or standards that the system under development must have or comply with, but which are not tasks that will be automated by the system. Example non-functional requirements for a system include:

• system must be built for a total installed cost of $1,050,000.00

• system must run on Windows Server 2003

• system must be secured against Trojan attacks

• A software development methodology helps to identify, document, and realize the

• requirements. Nonfunctional requirements can be divided into following categories: Usability• Reliability

• Performance

• Security

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What is oops**

Object Oriented Programming

Programming is like writing.

• If you can write a demonstration, you

• can make a program. So, programming is also easy

.• But, actually, programming is not so

• easy, because a real good program is not easily programmed. It needs the programmers’ lots of wisdom, lots of knowledge about programming and lots of experience. It is like writing, to be a good writer

• needs lots of experience and lots of knowledge about the world. Learning and practice is necessary

Object-Oriented Languages An object-based programming language is one which easily supports object-orientation

.• Smalltalk : 1972-1980

• Founder of Alan Kay

• C++ : 1986, Bjarne Stroustrup• Java(Oak) : 1992 (Smalltalk + C++)

• Founder of James Gosling• Developed by Sun Microsystem overtake by Oracle

.• C# :• Developed at Microsoft by Anders Hejlsberg et al, 2000

• Event driven, object oriented, visual programming language (C++ and Java)

What is OOP?

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.

Everything in the world is an object A flower, a tree, an animal

• A student, a professor

• A desk, a chair, a classroom, a building

• A university, a city, a country

• The world, the universe

• A subject such as CS, IS, Math, History, …

Concepts of OO Object

• Class

• Encapsulation

• Inheritance

• Polymorphism

• Overriding

• Overloading

• Abstractio

The two parts of an object Object = Data + Methods Or To say the same differently An object has the responsibility to know and the responsibility to do.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

What is 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 function that operate on data are

• bundled as a unit called as object.

The two parts of an object Object = Data + Methods Or To say the same differently An object has the responsibility to know and the responsibility to do.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

• behavior of that object. Class can be considered as the blueprint or definition or a template for an object and

• describes the properties and behavior 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.

The two steps of Object Oriented Programming Making Classes:

Creating, extending or reusing abstract data types.

• Making Objects interact: Creating objects from abstract data types and defining their relationships

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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 framework to place the data and the relevant functions together in the same object. Encapsulation in Java is the process of wrapping up of data (properties) and behavior

• (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 thatoperation. 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.

Example: The Animal class

public class Animal

{

private String animalName; private String animalType; public Animal()

{

System.out.println(“in default constructor ”);

}

public Animal(String animalName, String animalType)

{

this.animalName = animalName; this.animalType = animalType;

}

public String getAnimalName()

{

return animalName;

}

public void setAnimalName(String animalName)

{

this.animalName = animalName;

}

public String getAnimalType()

{

return animalType;

}

public void setAnimalType(String animalType)

{

this.animalType = animalType;

}

}

public static void main(String args[])

{

Animal a = new Animal(“Cow” , “Mammal”);

System.out.println(“name : ” + a. animalName);

System.out.println(“type : ” + a. animalType);

Animal a1 = new Animal(); a1.setAnimalName(“Dog”);

a1.setAnimalType(“Mammal”);

System.out.println(“ name : ” + a1.getAnimalName);

System.out.println(“type : ” + a1.getAnimalType);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

What is polymorphism

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 behavior 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.

Poly refers too many.

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

E.g. the message displayDetails() of the Person class should give different results

when send to a Student object (e.g. the enrolment number). The ability to change form is known as polymorphism.

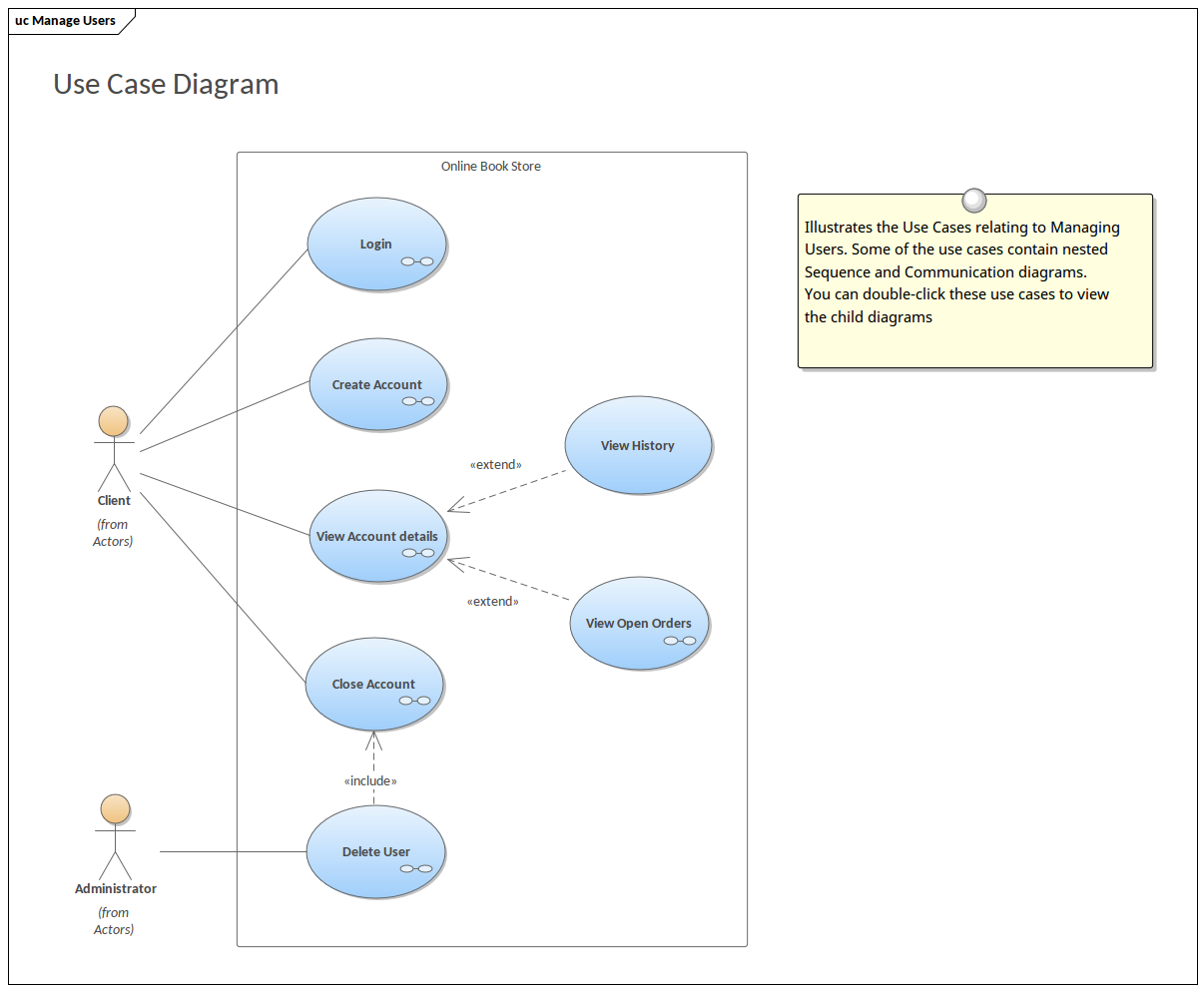
• There is two types of polymorphism in Java

• Compile time polymorphism(Overloading)

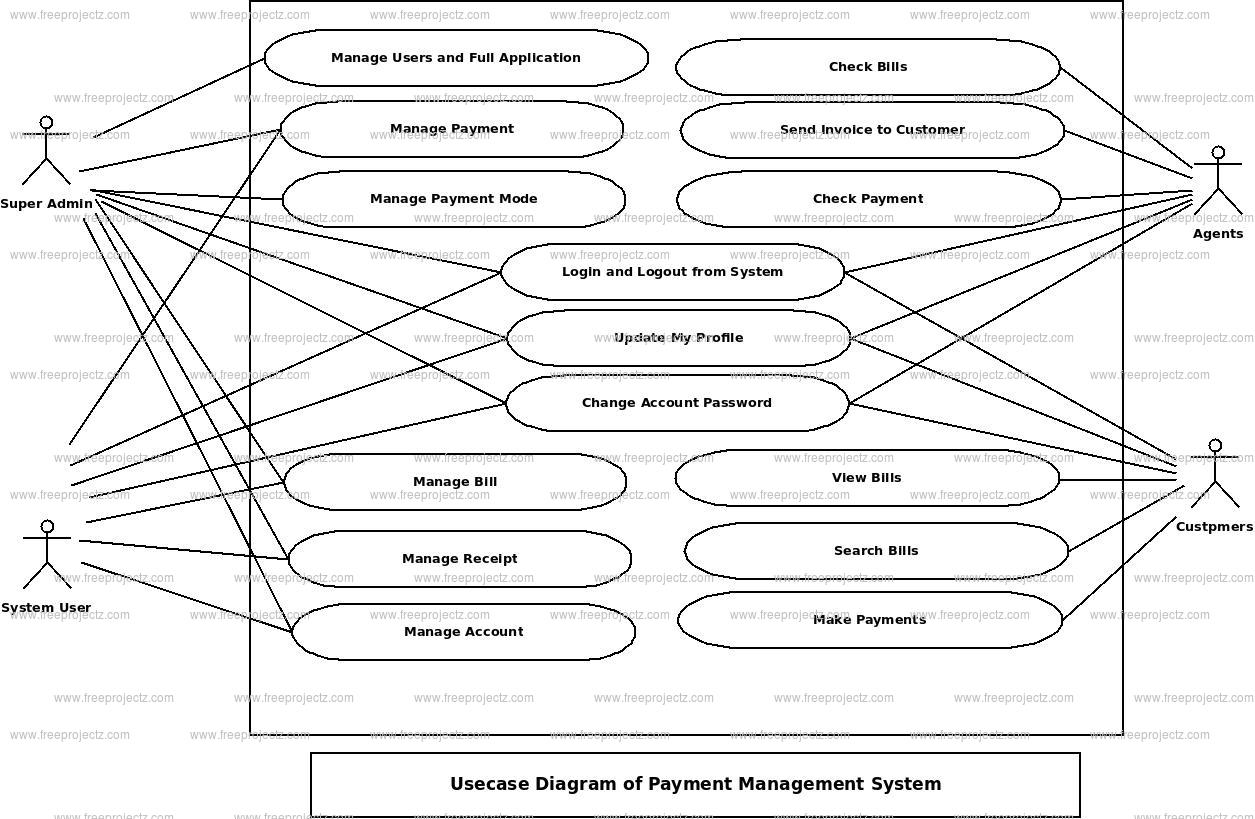
• Runtime polymorphism(Overriding)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Draw Usecase on Online book shopping



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Draw Usecase on online bill payment system (paytm)•



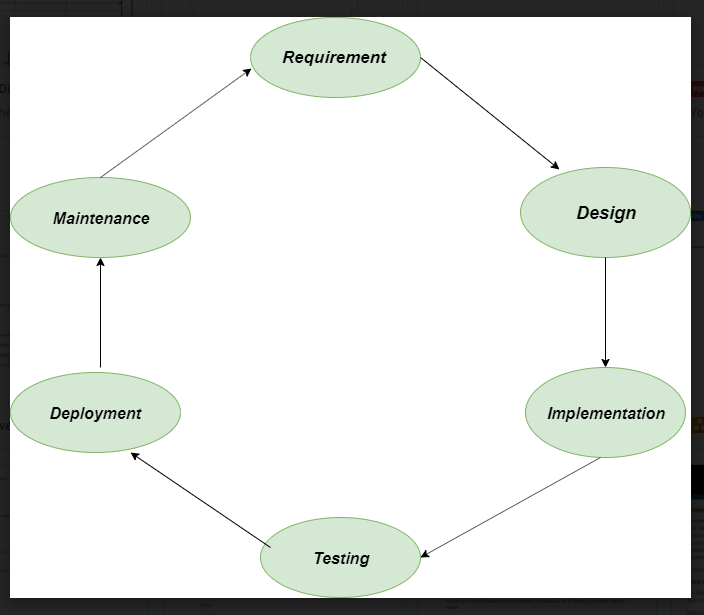
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write SDLC phases with basic introduction

Software Development Life Cycle is the application of standard business practices to building software applications. It’s typically divided into six to eight steps: Planning, Requirements, Design, Build, Document, Test, Deploy, Maintain.

There are a number of different development models.

A Software Development Life Cycle is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software



**Requirement:** In this phase, all the requirements are collected from the customer/client. They are provided in a document called Businessmen requirement specification (BRS) and System requirement specification (SRS). All the details are discussed with the customer/client in detail.

**2. Design:** It has two steps:

* **High-level design (HLD):** It gives the architecture of software products.
* **Low-level design (LLD):** It describes how each and every feature in the product should work and every component.

**3. Implementation:**

* This is the longest phase.
* This phase consists of Front end + Middleware + Back-end.
* **In front-end:**Development of coding is done even SEO settings are done.
* **In Middleware:** They connect both the front end and back end.
* **In the back-end:** A database is created.

**4. Testing:** Testing is carried out to verify the entire system. The aim of the tester is to find out the gaps and defects within the system and also to check whether the system is running according to the requirement of the customer/client.

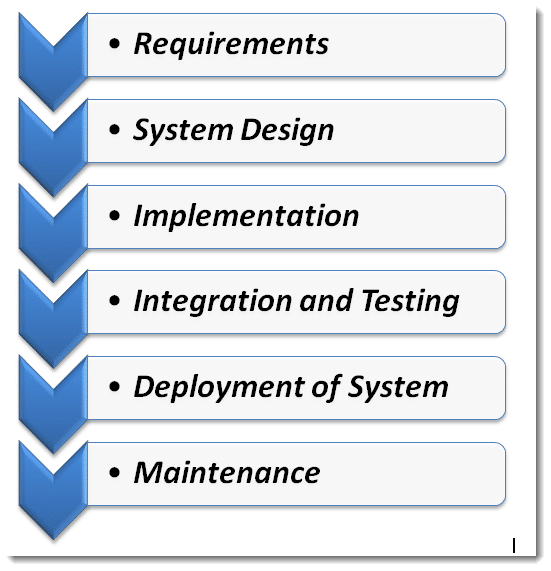
**5. Deployment:** After successful testing, the product is delivered/deployed to the client, and even clients are trained on how to use the product.

**6. Maintenance:** Once the product has been delivered to the client a task of maintenance starts as when the client will come up with an error the issue should be fixed from time to time.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explain Phases of the waterfall model

As the ***Waterfall Model*** illustrates the software development process in a linear sequential flow; hence it is also referred to as a ***Linear-Sequential Life Cycle Model.***



* ***Requirements****: The first phase involves understanding what needs to design and what is its function, purpose, etc. Here, the specifications of the input and output or the final product are studied and marked.*
* ***System Design****: The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The software code to be written in the next stage is created now*.
* ***Implementation****: With inputs from system design, the system is first developed in small programs called units, which are integrated into the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.*
* ***Integration and Testing****: All the units developed in the implementation phase are integrated into a system after testing of each unit. The software designed, needs to go through constant*[***software testing***](https://www.toolsqa.com/software-testing/software-testing/)*to find out if there are any*[***flaws or errors***](https://www.toolsqa.com/software-testing/istqb/error-defect-failure/)*. Testing is done so that the client does not face any problem during the installation of the software*.
* ***Deployment of System****: Once the*[***functional and non-functional testing***](https://www.toolsqa.com/software-testing/functional-and-non-functional-testing/)*is done, the product is deployed in the customer environment or released into the market.*
* ***Maintenance****: This step occurs after installation, and involves making modifications to the system or an individual component to alter attributes or improve performance. These modifications arise either due to change requests initiated by the customer, or defects uncovered during live use of the system. The client is provided with regular maintenance and support for the developed software*.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (*like a waterfall*) through the phases. The next phase is started only after the defined set of goals is achieved for the previous phase and it is signed off, so the name "***Waterfall Model***".

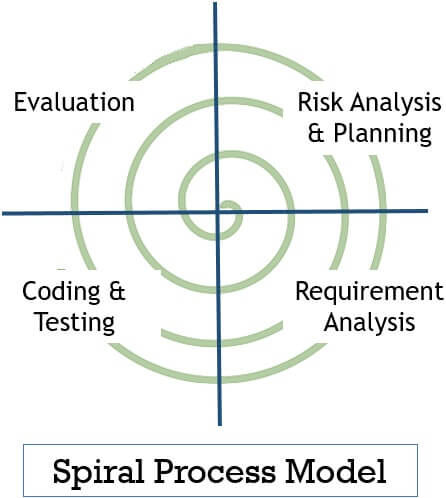
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write phases of spiral model

The spiral model is similar to the incremental development for a system, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Design, Construct and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model).

The spiral model is an SDLC model that combines elements of an iterative software development model with a waterfall model. It is advisable to use this model for expensive, large and complex projects.

In its diagrammatic representation, we have a coil having many cycles or loops. The number of cycles varies for each project and is usually specified by the project manager. Each spiral cycle is a stage in the software development process.



**. Risk Analysis & Planning**

Developers identify risks involved in the current iteration and also corrective measures to minimize risk. Then they evaluate corrective measures against objectives and constraints. Thus the deadline is set for the next stage.

**2. Requirement Analysis**

This phase analyzes the client’s requirements. Also, the requirements for developing the product are also established.

**3. Coding & Testing**

Developers develop multiple programs and integrated them to form a software or prototype. Like, in the early cycles the product of this stage would be a prototype. In subsequent cycles, the product of this stage becomes **developed software**.

The product of this stage is tested to find any error in coding.

**4. Evaluation**

This stage evaluates whether the project is going as per planning or not. It evaluates whether the objective set at the first stage has been achieved or not. The evaluation phase also helps developers decide the number of cycles required to complete the project.

The spiral model allows using other process models in one or more of its cycles. This is either to reduce the risk at some stage or to get the requirements of the user clear or for resolving technical risks.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write agile manifesto principles

1

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2

Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

3

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4

Business people and developers must work together daily throughout the project.

5

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7

Working software is the primary measure of progress.

8

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9

Continuous attention to technical excellence and good design enhances agility.

10

Simplicity–the art of maximizing the amount of work not done–is essential.

11

The best architectures, requirements, and designs emerge from self-organizing teams.

12

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Individuals and interactions - in agile development, self-organization and motivation are

• important, as are interactions like co-location and pair programming. Working software - Demo working software is considered the best means of

• communication with the customer to understand their requirement, instead of just depending on documentation. Customer collaboration - As the requirements cannot be gathered completely in the

• beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.. Responding to change - agile development is focused on quick responses to change and

• continuous development.

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.

• Transfer of technology to new team members may be quite challenging due to lack of

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Draw usecase on Online shopping product using COD.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Draw usecase on Online shopping product using payment gateway.•

