MODULE-1

B1. What is software?

software is a set of instructions, data, or programs used to operate a computer and execute specific tasks. In simpler terms, software tells a computer how to function. It's a generic term used to refer to applications, scripts, and programs that run on devices such as PCs, mobile phones, tablets, and other smart devices. Software contrasts with hardware, which is the physical aspects of a computer that perform the work.

==========

B2. Types of software?

there are different types of software based on their classification however, if we broadly classify the types of software in a computer, there are two major types as listed below:

**system softwere*

**application softwere*

------SYSTEM SOFTWARE----------

System software helps the user, the computer or mobile device, and an application all work together seamlessly. This makes system software crucial to running any kind

Think about when your laptop or phone has an update. This is system software in action: there is a tweak made to the system software that helps your computer or phone continue to work well and keep applications running. Apple's iOS is an example of system software, as is Microsoft Windows. System software is always running in the background of your device, but it is never something you will use directly. In fact, the only time most people remember it's there is when it is time for an update.

----------APPLICATION SOFTWERE----------

An application program (software application, or application, or app for short) is a computer program designed to carry out a specific task other than one relating to the operation of the computer itself,[1] typically to be used by end-users.[2] Word processors, media players, and accounting software are examples. The collective noun "application software" refers to all applications collectively.[3] The other principal classifications of software are system software, relating to the operation of the computer, and utility software ("utilities").

Applications may be bundled with the computer and its system software or published separately and may be coded as proprietary, open-source, or projects.[4] The term "app" usually refers to applications for mobile devices such as phones

==========

B3. What is Software Development Methodology?

Software development methodology is a process or series of processes used in software development. Again, quite broad but that it is things like a design phase, a development phase. It is ways of thinking about things like waterfall being a non iterative kind of process. Generally it takes the form of defined phases. It is designed to describe the how of the life cycle of a piece of software.

It is also codified communication. So you're actually setting a set of norms between a group of people that say this is how you're going to work and this is how you're going to pass information between each of you in certain ways; whether that is documentation, whether that is discussion, whether that is drawings on paper.

there are 7 steps software devlopment methodology

(1)requirement

(2)analysis

(3)designing

(4)devlopment

(5)testing

(6)deployment

(7)maintanance

==========

B4. What is DesignPattern?

a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply mutable state may be unsuited for functional programming languages. Some patterns can be rendered unnecessary in languages that have built-in support for solving

the problem they are trying to solve, and object-oriented patterns are not necessarily suitable for non-object-oriented languages.

Design patterns may be viewed as a structured approach to computer programming intermediate between the levels of a programming paradigm and a concrete algorithm.

==========

==========

Intermediate

1.What is

the difference between Application software and system software

	System software	Application software
(1)	This acts as an interface between the system and the applications.	This is designed directly from the user perspective.
(2)	it is the platform that allows the various application software to run on the system	These are indendent applications which can be download and installed in the system.
(3)	system software is generally developed in low-level languages. This is so that the interaction between the software and hardware can be simplified and made more compatible	Each application has a specific purpose and thus is developed with high-level languages so that the purpose can be fulfilled.
(4)	is working is more automated. once a system is turned on. the system software starts working.	User action is required to start application software. These applications can only be work when the user commands the system to do so.
(5)	These are responsible for the working of the system.	They have minimum involvement in the processing and functioning of the computer device

(6)	The system software are installed at the time of installing the operating system. A computer device cannot work without its Presence	the application software can be installed as and when the user requires them.
(7)	Example for system software includes Android, mac, operating system, ms windows etc.	Examples of Application software includes world, Processor, games, media Player, etc....

==========

2 Explain the SDLC Each faceProcess

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

**Requirement analysis*

**Planning*

**Software design such as architectural design*

**Software development*

**Testing*

**Deployment*

**maintanance*

This article will explain how SDLC works, dive deeper in each of the phases, and provide you with examples to get a better understanding of each phase.

1. Identify the Current Problems

"What are the current problems?" This stage of the SDLC means getting input from all stakeholders, including customers, salespeople, industry experts, and programmers. Learn the strengths and weaknesses of the current system with improvement as the goal.

2. Plan

"What do we want?" In this stage of the SDLC, the team determines the cost and resources required for implementing the analyzed requirements. It also details the risks involved and provides sub-plans for softening those risks.

In other words, the team should determine the feasibility of the project and how they can implement the project successfully with the lowest risk in mind.

3. Design

“How will we get what we want?” This phase of the SDLC starts by turning the software specifications into a design plan called the Design Specification. All stakeholders then review this plan and offer feedback and suggestions. It’s crucial to have a plan for collecting and incorporating stakeholder input into this document. Failure at this stage will almost certainly result in cost overruns at best and the total collapse of the project at worst.

4. Build

“Let’s create what we want.”

At this stage, the actual development starts. It’s important that every developer sticks to the agreed blueprint. Also, make sure you have proper guidelines in place about the code style and practices.

For example, define a nomenclature for files or define a variable naming style such as camelCase. This will help your team to produce organized and consistent code that is easier to understand but also to test during the next phase.

5. Code Test

“Did we get what we want?” In this stage, we test for defects and deficiencies. We fix those issues until the product meets the original specifications.

In short, we want to verify if the code meets the defined requirements.

Try Stackify's free code profiler, Prefix, to write better code on your workstation. Prefix works with .NET, Java, PHP, Node.js, Ruby, and Python.

6. Software Deployment

"Let's start using what we got."

At this stage, the goal is to deploy the software to the production environment so users can start using the product. However, many organizations choose to move the product through different deployment environments such as a testing or staging environment.

This allows any stakeholders to safely play with the product before releasing it to the market. Besides, this allows any final mistakes to be caught before releasing the product.

(7) Software Maintenance

"Let's get this closer to what we want." The plan almost never turns out perfect when it meets reality. Further, as conditions in the real world change, we need to update and advance the software to match.

The DevOps movement has changed the SDLC in some ways. Developers are now responsible for more and more steps of the entire development process. We also see the value of shifting left. When development and Ops teams use the same toolset to track performance and pin down

defects from inception to the retirement of an application, this provides a common language and faster handoffs between teams.

Application performance monitoring (APM) tools can be used in a development, QA, and production environment. This keeps everyone using the same toolset across the entire development lifecycle.

Read More: 3 Reasons Why APM Usage is Shifting Left to Development & QA

**Examples*

The most common SDLC examples or SDLC models are listed below.

**Waterfall Model*

This SDLC model is the oldest and most straightforward. With this methodology, we finish one phase and then start the next. Each phase has its own mini-plan and each phase "waterfalls" into the next. The biggest drawback of this model is that small details left incomplete can hold up the entire process.

**Agile Model*

The Agile SDLC model separates the product into cycles and delivers a working product very quickly. This methodology produces a succession of releases. Testing of each release feeds back info that's incorporated into the next version. According to Robert Half, the drawback of this

model is that the heavy emphasis on customer interaction can lead the project in the wrong direction in some cases.

**Iterative Model*

This SDLC model emphasizes repetition. Developers create a version very quickly and for relatively little cost, then test and improve it through rapid and successive versions. One big disadvantage here is that it can eat up resources fast if left unchecked.

**V-Shaped Model*

An extension of the waterfall model, this SDLC methodology tests at each stage of development. As with waterfall, this process can run into roadblocks.

**Big Bang Model*

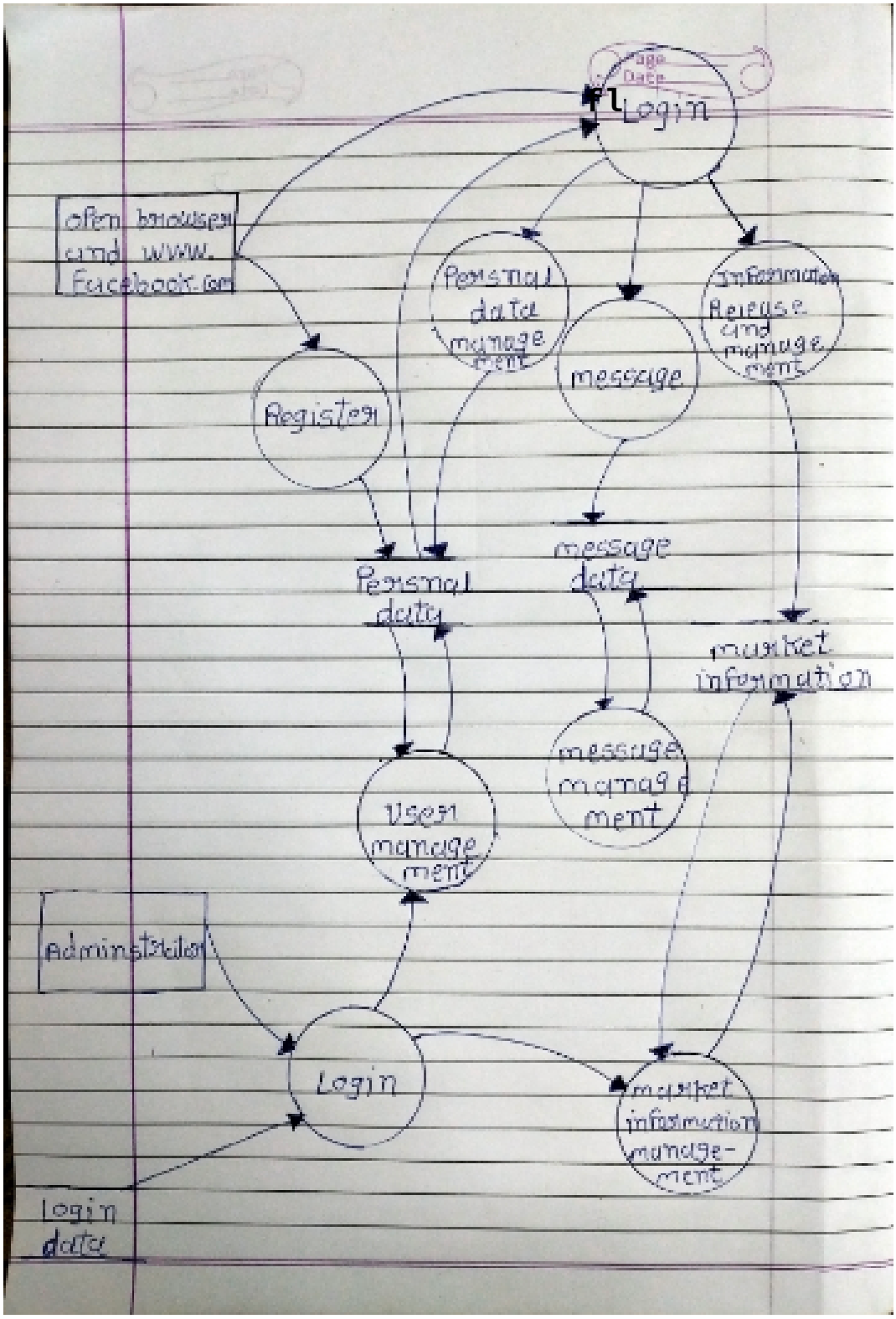
This high-risk SDLC model throws most of its resources at development and works best for small projects. It lacks the thorough requirements definition stage of the other methods.

**Spiral Model*

The most flexible of the SDLC models, the spiral model is similar to the iterative model in its emphasis on repetition. The spiral model goes through the planning, design, build and test phases over and over, with gradual improvements at each pass.

==========

3.crrate a dfd flowchart login process for Facebook.com



complete assignment

==========

==========