ASSIGNMENT-1 SECTION A:

Q1. What are the reasons of a successful and unsuccessful software project?
ANS:
Reasons of an unsuccessful software project:

1. **Poor planning**: One of the most common reasons why software projects fail is due to the lack of goals and proper planning. Without a plan, it is hard to know exactly what and when something needs to be done. This can lead to project delays and ultimately results a failure.
2. **Lack of leadership**: leadership is one of the most crucial factors in a software project's success or failure. A lack of proper leadership can lead to missed opportunities and unforeseen problems that could have been avoided if proper moves had been undertaken from the outset.
3. **Poor communication**: when teams don't share information or co-ordinate their efforts, they can't produce a high quality error-free product.
4. **Inadequate use of resources**: Inadequate use of resources or their improper placement can lead to several problems including schedule delays, cost overruns and quality compromise.

5. **Not Understanding The Needs Of The Business:** Lack of understanding of the business' needs leads to an improper mapping of features and functions to the business' needs. Eventually, the project fails to satisfy the client requirement.
6. **Unclear requirement:** One of the most common reasons software projects fail is unclear requirements and the lack of a detailed explanation. Very often clients themselves are not sure exactly what they want to see, and as a result, the project cannot move forward.
7. **Expecting one solid solution**: proper solutions are rarely so simple—they are a blend of methodology, strategy and team support, not the result of a single action, technology or idea.
8. **Narrow scope**: No project can be considered successful if it does not adapt to changing business requirements during development. Unfortunately, some tech teams still insist on hitting the original goal, thus rendering their effort ineffective or even a failure.
9. **Miscalculated Time and Budget Frames:** Clients are always eager to have their projects rolled-out on time and even before the stipulated time at throw away prices. In most cases, this keenness of the client leads to developers agreeing to a rather shorter or unrealistic and non-negotiable time frame for the project delivery at meagre rates. As a result, programmers are not able to deliver the project on time.
10. **Limited review work of project progress:** As project progresses, things change, significantly impacting the project. It is important to keep examining the project progress to overcome challenges early and warn stakeholders of possible delays and outcomes changes the failure of which have a disastrous effect in the end result.
11. **Inadequate testing:** Casual testing, testing under non-real time environments contribute to testing failures .Low number of testing due various factors result is a product filled with bugs and an unsatisfied client.
12. **Testing in the production environment:** organisations often test products in their production environment. Using the production environment is a high-risk strategy that

can lead to security breaches and accidental release without testing, disrupting the production systems.

13. **Lack of quality assurance:** Often in the haste to deliver the software, quality assurance suffers. Documentation is incomplete for code changes, the design contains flaws, and implementations can be unfinished.
14. **Not conforming to industry standards:** Conforming to industry standards in your software projects can prove beneficial by ensuring good accessibility, portability, usability, robustness, and reducing current and future problems. Tech teams and developers often neglect this and fail to reach up to the standard resulting a poor and failed project.
15. **Substandard engineers:** Handling projects by incapable substandard engineers is also one of the main reasons of a project failure.
16. **Lack of adequate resources:** Cross-training and multi-tasking are the two common mistakes that many companies make to save organisational resources and cost-cutting. These practices affect the project's success and could fail the project prematurely.
17. **Unrealistic timeline:** When tech teams are given a colossal project and an unrealistic timeline to complete the project, the project failure rate increase exponentially.
18. **No end user involvement:** If the user's point of view is not taken into consideration while developing the IT project, the project result takes a disastrous turn.
19. **Chasing technology:** Some managers are lured into the benefits of the latest available technology, and try to use it for their on-going projects. This forces them to gravitate from the planning done at the design stage. This leads to the failure of the whole system, change in objectives, or failing to complete the project on time.

Completing projects successfully and within schedule and budget can be a very challenging endeavour. Main reasons of successful software projects are:

1. Product is designed and developed with focus on satisfying the end user requirements
2. Tech team are supervised under the leadership of a capable project manager.
3. Proper project timeline, planning and budgeting are allocated
4. Able engineers are assigned and engaged in proper area of the development phase
5. Right technologies and process model is implemented

Q2. What types of problems may arise if a software project is developed on ad hoc basis?

ANS:

Software that is developed on ad-hoc basis, Ad-hoc testing is carried out without following any formal process like requirement documents, test plan, test cases, etc. Some of downside of software projects developed on ad-hoc basis is listed below:

- Since testing is done without any planning and in unstructured way, the recreation of bugs sometime becomes a big trouble.
- The test scenarios executed during the ad-hoc testing are not documented so the tester has to keep all the scenarios in their mind which he/she might not be able to recollect in future.
- Ad-hoc testing is very much dependent on the skilled tester who has thorough knowledge of the product and it cannot be done by any new joiner of the team.

Q3. Provide three examples of software projects that would be amenable to the waterfall model. Be specific.

Ans:

Three examples of software projects that would be amenable to the waterfall model are:

1. Accounting software that has been mandated because of changes to government regulations.
2. Predominantly used in military standard software development of DoD.
3. Small software products can be developed using the waterfall method where business requirements are clear and business objectives are fixed. For example, if a software company has experience in developing payroll systems, and a business is looking to get one, then a waterfall model would suffice to build a new payroll system for the company.

Q4. . Provide three examples of software projects that would be amenable to the prototyping model. Be specific.

ANS:

Three examples of software projects that would be amenable to the prototyping model are

1. Online systems, web interfaces that have a very high amount of interaction with end users, are best suited for Prototype model.
2. They are excellent for designing good human computer interface systems like fingerprint voting system.
3. Certain classes of mathematical algorithms, subset of command-driven systems and other applications where results can be easily examined without real-time interaction. e.g. video editor software

Q5. What process adaptations are required if the prototype will evolve into a delivery system or product?

Ans:

If a prototype is evolved into a delivery system or product, it begins to incorporate communication and network element. The software engineer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to make use of existing program fragments or applies tools (e.g., report generators, window managers, etc.) that enable working programs to be generated quickly.

Q6. Provide three examples of software projects that would be amenable to the incremental model.

ANS:

Three examples of software projects that would be amenable to the incremental model are

1. word--processing software developed using the incremental paradigm might deliver basic file management, editing and document production functions in the first increment; more sophisticated editing and document production capabilities in the

second increment; spelling and grammar checking in the third increment, and advanced page layout capability in the fourth increment.
2. Application software such as communication apps are amenable to incremental model as new features can be added or extended with user demands.
3. Antivirus software are also amenable to incremental model as new security features can be incremented with time.

Q7. As you move outward along the spiral process flow, what can you say about the software that is being developed or maintained?

ANS:

As we moves outward along the spiral process flow, the product moves toward a more complete state and the level of abstraction at which work is performed is reduced (i.e., implementation of specific phase accelerates as we move further from the origin).

Q8. What is a Product & Service based company?

ANS:

Product-based companies are the ones that are focused on producing or introducing products that have a high market value and are capable enough to satisfy the customer. The important aspect of these companies is to produce top-quality products. These companies constantly improve the products with new features and upgrades using different and new technologies that are at their disposal.

Service-Based Companies are the ones that give customers or clients a solution in the form of amenities (any desirable feature), skills, and/or expertise based on their needs. The important aspect of this type of company is the client. They do not prepare anything specific before the problem arises on the client's end. Their sole aim is to ensure that the service provided to the client is of the best quality.

Q9. What is a Process framework & framework activities?
ANS: Software Process Framework is an abstraction of the software development process. It details the steps and chronological order of a process. Since it serves as a foundation for them, it is utilized in most applications. Task sets, umbrella activities, and process framework activities all define the characteristics of the software development process.

Software process includes:
- Tasks – focus on a small, specific objective.
- Action – set of tasks that produce a major work product.
- Activities – group of related tasks and actions for a major objective.

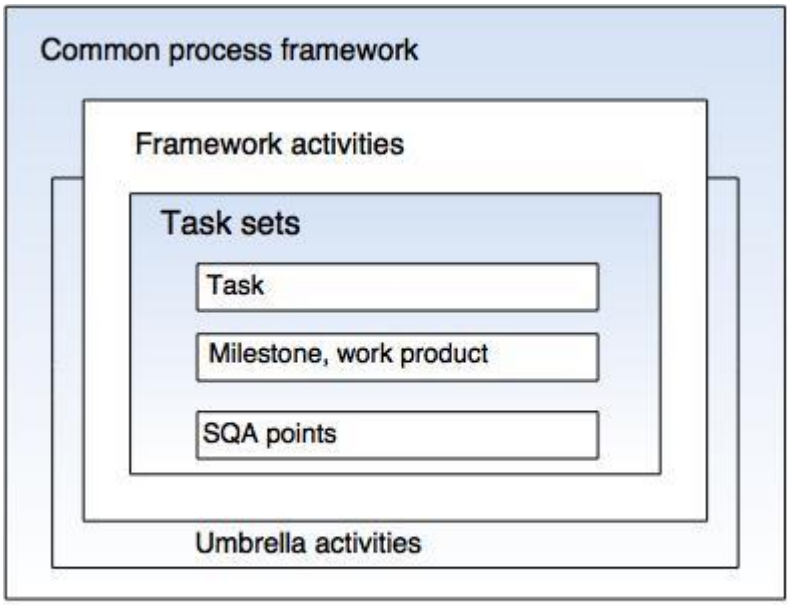

Fig.- A software process framework

The process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering. Communication, planning, modelling, construction, and deployment are all examples of framework activities. Each engineering action defined by a framework activity comprises a list of needed work outputs, project milestones, and software quality assurance (SQA) points.

- **Communication:** By communication, customer requirement gathering is done. Communication with consumers and stakeholders to determine the system's objectives and the software's requirements.
- **Planning:** Establish engineering work plan, describes technical risk, lists resources requirements, work produced and defines work schedule.
- **Modeling:** Architectural models and design to better understand the problem and for work towards the best solution. The software model is prepared by:
  -Analysis of requirements

  - Design
  - **Construction:** Creating code, testing the system, fixing bugs, and confirming that all criteria are met. The software design is mapped into a code by:
    -Code generation
    -Testing
  - **Deployment:** In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for the supply of better products.

Umbrella activities:

Umbrella Activities are that take place during a software development process for improved project management and tracking.

1. **Software project tracking and control:** This is an activity in which the team can assess progress and take corrective action to maintain the schedule. Take action to keep the project on time by comparing the project's progress against the plan.

2. **Risk management:** The risks that may affect project outcomes or quality can be analysed. Analyse potential risks that may have an impact on the software product's quality and outcome.
3. **Software quality assurance:** These are activities required to maintain software quality. Perform actions to ensure the product's quality.
4. **Formal technical reviews:** It is required to assess engineering work products to uncover and remove errors before they propagate to the next activity. At each level of the process, errors are evaluated and fixed.
5. **Software configuration management:** Managing of configuration process when any change in the software occurs.
6. **Work product preparation and production:** The activities to create models, documents, logs, forms, and lists are carried out.
7. **Reusability management:** It defines criteria for work product reuse. Reusable work items should be backed up, and reusable software components should be achieved
8. **Measurement:** In this activity, the process can be defined and collected. Also, project and product measures are used to assist the software team in delivering the required software.

Q10. What are principles of software engineering?

Ans: Software engineering principles are a collection of approaches, styles, philosophies, and best practices recommended by world-renown software engineers and authors. As part of software development, these principles serve as guidelines to ensure the final version of a piece of software fulfils its purpose. The following are the Software engineering principles and tactics we must employ to stay grounded and to make reasonable technical choices based upon requirements, budgets, timelines, and expectations:

1. KISS (Keep It Simple, Stupid)

The principle of simplicity states that codes should be as simple as possible without a complicated structure, otherwise debugging and maintenance might be more difficult.

2. DRY (Don't Repeat Yourself)

The principle of DRY states that we shouldn't repeat the same thing too many times in too many places. In software systems, it aims to reduce repetitive code and effort.

3.**YAGNI (You Aren't Going to Need It)**
In accordance with this principle, programmers should not include functionality unless it is absolutely necessary. It states that we shouldn't introduce things to solve future problems that don't exist yet.

BDUF (Big Design Upfront)

According to this principle, a developer should design the project first, create the flow of the diagram, and then implement it later. Before developing functionality, we should first think about the architecture and design of the whole system to the smallest details, and then follow the steps outlined in our plan to implement it. This helps uncover issues at the requirements stage and resolve them quickly.

SOLID

SOLID is an acronym for a collection of object-oriented design principles. Each letter in the"SOLID" stands for one of the following principles:

S – SRP (Single Responsibility Principle): According to the Single Responsibility Principle, a class, function, module, or service must have only one reason to change, i.e., it must have only one responsibility.

O – OCP (Open Closed Principle): In software development, we work in phases. As a team, we implement a bunch of functionalities, test them, and then deliver them to the users. We then move on to implementing the next set of functionalities. When it comes to developing new functionality, the last thing we want to do is to change the existing functionality, which has been tested and is working. Therefore, we try to add new functionality on top of existing ones. This idea is facilitated by the Open-Closed principle. According to it, our functions, classes, and modules should be designed in such a way that they're open for extension, but closed for modification.

- **Open for Extension:** New functionality can be added to classes and modules without breaking the existing code. Composition and inheritance can be used to accomplish this.
- **Closed for Modification:** It's ideal not to make changes that break current functionality, as doing so would require refactoring a lot of existing code and writing several tests to ensure the changes work.

L – LSP (Liskov Substitution Principle): According to the Liskov Substitution Principle, all child/derived classes should be replaceable for their parent/base classes without affecting or breaking the program's correctness. Thus, objects in our subclass (derived/child class) should behave similarly to objects in our superclass (parent/base class). Therefore, we should use inheritance carefully in your projects.

I – ISP (Interface Segregation Principle): According to the Interface Segregation Principle, clients shouldn't be forced to depend or rely on methods that they don't use.

D – DIP (Dependency Inversion Principle): This Principle seeks to eliminate tight coupling between software modules. According to this principle, high-level modules should not depend on lower-level modules, but rather on their abstractions. We can break it down into two parts:

- A high-level module must be independent of a low-level module. Both should rely on abstractions
- The abstraction should be independent of the details, while the details should be dependent upon the abstractions.

Occam's Razor

This is a very common philosophy-based idea found in programming. The principle is named after English monk William of Ockham. The principle is interpreted as follows: don't create extra entities unless they're needed. It's crucial to consider the benefits of adding a new method/class/tool/process before implementing it.

Law of Demeter

The Law of the Demeter principle states that an object should never know the internal details of another object. It is intended to promote loose coupling among software components. The more tightly coupled the software components are, the more difficult it is to modify them.

Measure Twice and Cut Once

The golden rule in engineering is to measure twice and cut once. In essence, this principle says we must plan and prepare thoroughly and carefully before we take an action. Whenever we are creating a new system, be sure it is useful, desirable, easily accessible, and credible. The requirement stage of the development life cycle can cause more than 50 percent of coding issues if not done correctly. Thus, we should adopt a systematic approach to the coding process.

Principle of Least Astonishment

It is also called the Principle of Least Surprise. Ideally, a feature should not have a high-astonishment factor, according to the principle of least astonishment. It means that we should write code that is intuitive and obvious so that it doesn't surprise others when they review it. Components of our system should behave as expected by end-users.