2.1 Software Engineering Practices (Definition, Importance and Essence)

Definition

(Question: Give the definition and importance of software engineering practice – 4 Marks, 2 marks each)

- Consists of a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis.
- Equips managers to manage software projects and software engineers to build computer programs.
- Provides necessary technical and management "how to's" in getting the job done.
- Transforms a haphazard unfocused approach into something that is more organized, more effective, and more likely to achieve success.

Importance

- Practice helps us to understand the concepts and the principles that must be followed for development of software engineering projects and project development.
- Practice instructs us to develop the model in more systematic form and at the pace it needs to be developed for deployment.

Essence

(Question: Explain the Essence of practice of software engineering practice – 8 Marks, 2 marks each)

The practice involves problem solving, modelling, designing, code generation, testing and quality assurance listed below in four steps.

1. **Understand the problem (Communication and analysis).**
    a. Who has a stake in the solution to the problem?
    b. What are the unknowns (data, function and behavior)?
    c. Can the problems be compartmentalized?
    d. Can the problem be represented graphically?
2. **Plan a solution (modeling and software design)**
    a. Have you seen similar problems before?
    b. Has a similar problem been solved and is the solution reusable?
    c. Can sub problems be defined and are solution available for the sub problems?
    d. Can you represent a solution in a manner that leads to an effective implementation?
3. **Carry out the plan (construction, code generation)**
    a. Does the solution conform to the plan? Is the source code traceable back to the design?
    b. Is each component part of the solution probably correct?
4. **Examine the result for accuracy (testing and quality assurance)**
    a. Is it possible to test each component of the solution?
    b. Does the solution produce results that conform to the data, function, features, and behavior that are required?

2.2 Core Principles for Software Engineering (SEVEN PRINCIPLE)

(Question: Explain the core principles of software engineering practice – 8 Marks)

1. **The reason it all exists.**
    a. The software system exists to provide value for the user and satisfy the requirement.
    b. Before specifying the problem the requirement and the specifications have to be laid down.
    c. The hardware and the software platform to be decided for implementation.

2. **Keep it simple stupid (KISS).**
    a. The terms and the design used for development of the project should be kept simple and easily understandable.
    b. All design and implementation should be as simple as possible.
    c. All the terms used should be easy to facilitate the basic concept of the project.

3. **Maintain the vision.**
    a. A clear vision is important for the development of a software.
    b. Compromising the architectural vision of the project weakens the development of the software.
    c. The developer should hold the vision and ensure the successful development and deployment of the software.

4. **What you reproduce, other will consume.**
    a. Always specify, design and implement knowing that someone else will later have to understand and modify what you did.
    b. Customers for the product development is very large.
    c. Design the data structure and the implementation keeping implementation in mind and the end user.

5. **Be open to the future.**
    a. Never design yourself into a corner; build software that can be easily changed and adapted.
    b. The system designed today should be adaptable to the development and changes in the future at a low cost.
    c. There should not be much changes to the software to adopt to the new changes in the future development.

6. **Plan ahead for reuse**.
    a. The design and specifications should be developed in such a way that they can be reused for other implementations.
    b. The code and the design should be well documented for the use in future.
    c. Reuse of software reduces the long-term cost and increases the value of the program and the reusable components.

7. **Think!**
    a. Placing clear, complete thought before action will almost always produce better results.
    b. Proper data structure and the design and implementation strategy should be developed if the software needs modification in the future.

2.3 Communication Practices

(Question: Explain the communication principles of software engineering practice – 8 Marks,)

- Communication practice is two way communication between the client and the developer, hence it is also known as requirement elicitation.

1. **Listen**
    a. To collect lots of data from the client, the developer team has to listen carefully.
    b. Maximum information with respect to requirement and the specifications should be collected before the implementation and the designing of the software.
2. **Prepare before you communicate**
    a. A proper agenda or the guidelines for the meetings should be prepared before the start of the meeting.
    b. Complete detail and the description about the clients and their work area should be gathered to deliver the software up to the best expectation.
3. **Someone should facilitate the activity**
    a. The requirement gathering and the specification are important for any software development, hence the communication should continue till the requirement gathering is over.
4. **Face-to-face communication is best**
    a. It is always better to sit across the table and have discussion on the requirement on the software development by the client and the developer.
5. **Take notes and document decisions**
    a. The important points discussed should also be recorded.
    b. Proper notes and the documentation is important for the successful completion and deployment of the project.
6. **Strive for collaboration** ***(make great efforts to achieve or obtain something.)***
    a. Collaboration in terms of teamwork is required for the successful completion of the software.
    b. The collective knowledge of the team members should be implemented in the development.
7. **Stay focused, modularize your discussion**
    a. As the development is the working of many team members, so the possibility of the discussion going from one topic to the other topic is quite possible.
    b. As a good software developer it is required that the discussion remains focused on the specified area.
8. **If something is unclear, draw a picture**
    a. Drawing flowcharts, E-R diagrams and other supporting graphical representations give clarity to the discussion and the documentation.
9. **(A) Once you agree to something, move on.**
   (B) If you can't agree to something, move on.
   (C) If a feature or function is unclear and cannot be clarified at the moment, move on.
    a. Healthy discussion leads to the final conclusion of successful implementation of the software.
    b. Once reached to final statement recorded should move to the next step.
    c. If no conclusion is reached than that point should be left and move ahead with new implementation which is cost effective.
10. **Negotiation is not a contest or game. It work best when both parties win.**
    a. Negotiation should be mutual not to put someone down or make them feel to be the loser.

2.4 Planning Practices

(Question: Explain the planning practices of software engineering practice – 8 Marks,)

- The planning activity encompasses a set of management and technical practices that enable the software team to define a road map as it travel its strategic goal and tactical objectives.

1. **Understand the scope of the project.**
    a. To plan the development of the software after the communication principles are been laid down, analyze the scope of the software.
    b. The implementation strategy and its effectiveness is considered and scope is decided.

2. **Involve the customer in planning activity.**
    a. The client should be involved in the continuous development of the software.
    b. The requirement and the specifications should not change frequently.

3. **Recognize that planning is iterative.**
    a. Planning is iterative, but the specification should not change continuously.
    b. After the analysis phase the requirement should be fixed and should not change.

4. **Estimate based on what you know.**
    a. The knowledge of the developer is used for the final estimation of the planning phase of the software.
    b. New techniques if unknown should not be considered for planning.

5. **Consider the risk as you define the plan.**
    a. The risk in the development should be considered for the software to be with the designing standard and for future scope and development.

6. **Be realistic.**
    a. Considering the latest trends in the development and not choosing something which is not practically possible to implement.

7. **Adjust granularity as you define the plan *(the scale or level of detail)*.**
    a. The specification should be refined as the plan is defined at each stage of development.

8. **Define how you intend to ensure quality.**
    a. The quality of the software should be maintained by providing help at each level of the software.
    b. Proper documentation should be maintained and given at the time of deployment to the client.
    c. The end user license and agreement (EULA).should be provided.

9. **Describe how you intend to accommodate change.**
    a. The up gradation option to the software should be provided to the client in the future with the regular maintenance policy.

10. **Track the plan frequently and make adjustments as required.**
    a. The continuous communication with the client to be done to make the required changes to the planning and the development phase.

2.5 Modeling Practices

(Question: Explain the modeling practices of software engineering practice – 8 Marks)

- The models are created to gain better understanding of actual entity to be built.
- When the entity is software, our model must take a different form.
- It must be capable of representing the information that software transforms, the architecture and functions of a software, the features that user's desire, and the behavior of the system.
- Two classes of models are crated: **Analysis models** and **Design models**.
- Analysis models represent the customer requirements by describing the software in three different domains: the information domain, the functional domain, and the behavioral domain.
- Design models represent characteristics of the software that help software engineering team to construct it effectively.

2.5.1 Analysis Modeling Principles

(Question: Explain the analysis modelling principles of software engineering practice – 8 Marks)

- A large number of analysis modeling methods have been developed. Each analysis methods has unique point of view. However, all analysis methods are related by a set of operational principles.

- **Principle #1: The information domain of a problem must be represented and understood.**
  - The data that flow into the system and the data stores into the system must be clearly represented.

- **Principle #2: The functions that the software performs must be defined.**
  - Software functions provide direct benefit to visible end-user.
  - Some functions transform data that flow into the system.

- **Principle #3: The behavior of the software must be represented.**
  - The behavior of software is driven by its interaction with the external environment.
  - Input provided by end-users, control data provided by an external system all cause the software to behave in a specific way.

- **Principle #4: The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered fashion.**
  - Analysis modeling is the first step in software engineering problem solving.
  - It allows the developer and end-user to understand the problem better and establishes a basis for the solution (design).

- **Principle #5: The analysis task should move from essential information toward implementation detail.**
  - Describing the problem from the end-user's perspective.
  - The "essence" or "key objective" of a problem is described without any consideration of how a solution will be implemented.

2.5.2 Design Modeling Principles

- The design model created for software provides a variety of different views of system.

- **Principle #1: Design should be traceable to the analysis model.**
  - The analysis model describes the information domain of the problem.
  - The design model translates this information into an architecture, a set of subsystems that implement major functions.

- **Principle #2: Always consider the architecture of the system to be built.**
  - Software architecture is the skeleton of the system to be built.
  - It effects interfaces, data structures, program control flow and manner in which testing can be conducted and the maintainability of resultant system.

- **Principle #3: Design of data is as important as design of processing functions.**
  - Data design is an essential element of architectural design.
  - The data should be designed as the flow of algorithm is designed for the implementation and deployment of the software.

- **Principle #4: Interfaces (both internal and external) must be designed with care.**
  - Data flows between the modules should be designed with simplicity and processing efficiency.
  - A well designed interface makes integration easier.

- **Principle #5: User interface design should be tuned the needs of the end-user.**
  - The user interface is the visible implementation of the software.
  - The poor interface always leads to the perception that the software is bad.

- **Principle #6: Components should be functionally independent.**
  - Each sub function or the module developed should focus on the perfect working functionality of that module.

- **Principle #7: Components should be loosely coupled to one another and to the external environment.**
  - Coupling is accomplished through many ways like message passing, component interface and global data.
  - When the level of coupling increases, error propagation increases with overall maintainability of software decreases.
  - Component coupling should be kept as low as possible.

- **Principle #8: Design representation (model) should be easily understandable.**
  - The design models are created for easy communication of information to the users about the code
  - The design model should be easily understandable by the tester.
  - If the software design model is difficult to understand and communicate than it is not the effective model.

- **Principle #9: The design should be developed iteratively.**
  - The design work should be open for improvement at the first level.
  - At the final stages it should be easy and creative.

2.6 Construction Practices

(Question: Explain the construction practices of software engineering practice – 8 Marks)

- Construction practices comprises of coding and testing principles.
- The initial focus is on the component level known as unit testing and the other testing are listed below:
  - **i.** **Integration testing:** Conducted as the system is constructed.
  - **ii.** **Validation testing:** That assesses whether requirements have been met for the complete system.
  - **iii.** **Acceptance testing:** That is conducted by the customer in an effort to exercise all required features and functions.

2.6.1 Coding Principles

(Question: Explain the coding principles of software engineering practice – 3 Marks)

- The principles which guide the coding tasks are programming languages, programming styles and programming methods.

1. **Preparation principles: Before you write one line of code, be sure you**
   a. Understand of the problem you are trying to solve
   b. Understand basic design, principles & concepts.
   c. Pick a programming language that meets the needs of the software to be build and the environment in which the software will operate.
   d. Select a programming environment that provides tools that will make you work simpler and easier.
   e. Create a set of units that will be applied once the component you code is completed.

2. **Coding principles: As you begin writing code, be sure you:**
   a. Constrain your algorithms by following structured programming practice.
   b. Consider the use of pair programming.
   c. Select data structures that will meet the needs of the design.
   d. Understand the software architecture and create interfaces that are consistent with it.
   e. Keep conditional logic as simple as possible.
   f. Create nested loops in a way that makes them easily testable.
   g. Select meaningful variable names and follow other local coding standards.
   h. Write code that is self-documenting.
   i. Create a visual layout that aids understanding.

3. **Validation Principles: After you have completed your first coding pass, be sure you:**
   a. Conduct a code walkthrough when appropriate.
   b. Perform unit tests and correct errors when you have uncovered.
   c. Refactor the code.

2.6.2 Testing Principles

(Question: Explain the testing principles of software engineering practice – 4 Marks)

- Testing is the process of executing programs with the intent of finding errors.

1. **All tests should be traceable to customer requirements:**
    a. The main objective of software testing is to uncover errors.
    b. The most server defects are those that cause the program to fail to meet its requirements.

2. **Tests should be planned long before testing begins**
    a. Test planning can begin as soon as the requirements model is complete.
    b. Detailed definition of test cases can begin as soon as the design model has been solidified.
    c. For this reason all tests can be planned and designed before any code has been generated.

3. **The Pare to principle applies to software testing**
    a. The Pareto principle, also known as the 80/20 rule.
    b. 20 percent of software bugs cause 80 percent of the software's failures.

4. **Testing should begin "in the small" and progress towards testing "in the large"**
    a. The first tests planned and executed generally focus on individual components.
    b. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

5. **Exhaustive testing is not possible**
    a. The number of path permutations for even a moderately sized program is exceptionally large.
    b. For this reason, it is impossible to execute every combination of paths during testing.

2.7 Software Deployment

- **Software deployment** is all of the activities that make a software system available for use.

2.7.1 Deployment Activities:

- **Delivery Cycle:** Each delivery cycle provides the customer and end users with an operational software increment that provides usable functions and features.
- **Support Cycle:** Each support cycle provides documentation and human assistance for all functions and features introduced during all deployment cycles to date.
- **Feedback Cycle:** Each feedback cycle provides the software team with important guidance that results in modifications to the function, features and approach taken for the next increment.

2.7.2 Deployment Principles:

(Question: Explain the deployment principles of software deployment – 4 Marks)

1. **Manage customer's expectations or requirements**
    a. In most cases customer wants more than he/she has stated earlier as his requirements or expectations.
    b. In many cases customer is disappointed, even after getting all his/her requirements satisfied.
    c. Hence, at the time of software delivery, developer must have skills to manage the customer's expectations and requirements.

2. **Record-keeping mechanism must be established for customer support**
    a. 'Customer Support' is essential and important factor in deployment phase.
    b. The support should be well planned and with proper record keeping mechanism.

3. **Provide essential instructions, documentations and manual**
    a. Actual software project delivery includes all documentations, help files and guidance for handling the software by user.

4. **Assemble and test complete delivery package**
    a. The customer side must get all supporting and essential help from developer's side.
    b. For this reason CD with complete assembled and tested delivery package with the following support should be delivered:
        i. Necessary supported operational features and help.
        ii. Essential manuals required for software troubleshooting.
        iii. All executable file of developed software.
        iv. Necessary installation procedures.

5. **Do not deliver any defective or buggy software to the customer**
    a. The software should be tested before deployment to the customer.
    b. The specification should be with the requirement to the customer.

2.8 Requirement Engineering

- Requirement describe how a system should act, appear or perform.
- Requirement is a condition possessed by the software component in order to solve a real world problems.
- IEEE defines a requirement as: "A condition that must be possessed by a system to satisfy a contract specification, standard or other formally imposed document.

2.8.1 Principles of Requirement Engineering

(Question: Explain the principles of requirement engineering – 4 Marks)

1. **Understand the problem before you start to create the analysis model**
   a. There is a tendency to rush to a solution, even before the problem is understood.
   b. This often leads to elegant software that solves the wrong problem.

2. **Develop prototypes that enable a user to understand how human-machine interaction will occur**
   a. Since the perception of the quality of software is often is based on perception of time "friendliness" of the interface, prototyping (and the interaction that results) is highly recommended.

3. **Record the origin and the reason for every document**
   a. This is the step in establishing traceability back to the customer.

4. **Use multiple views of requirement**
   a. Building data, functional and behavioral models provides software engineer three different views.
   b. This reduces the chances of missing errors.

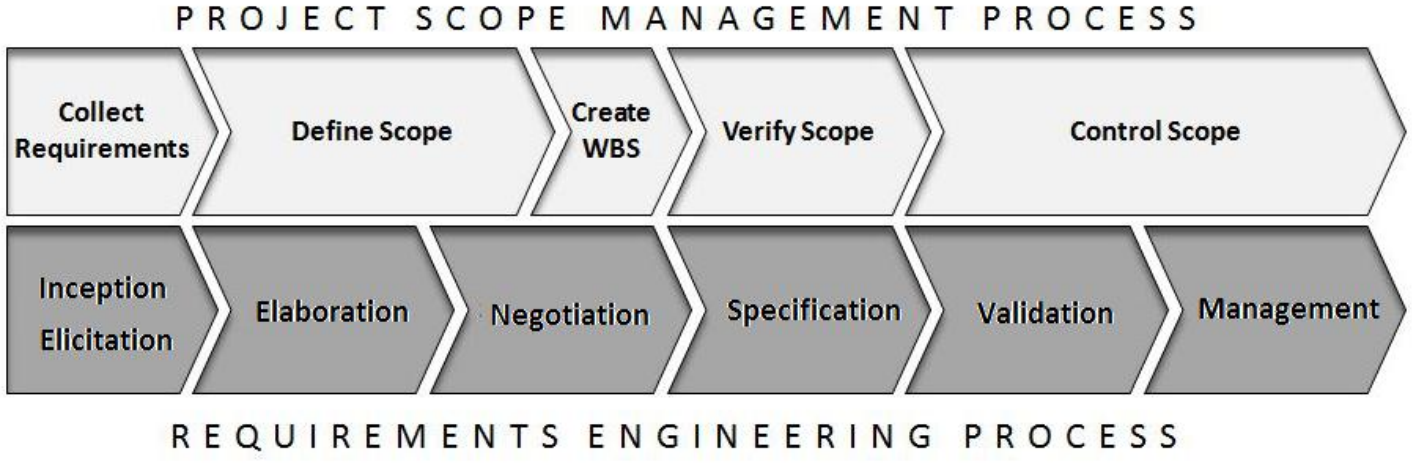
5. **Prioritize the requirements**
   a. Requirements should be followed for the tight implementation and delivery of the product.

6. **Work to eliminate ambiguity**
   a. The use of more technical reviews should be used for no ambiguity.

2.8.2 Requirement Engineering Task

(Question: Explain the various requirement engineering tasks – 8 Marks)

- The requirement engineering process tasks are achieved through seven distinct functions as shown in Figure 1



1. **Inception**
    a. Inception is also known as the beginning.
    b. Inception needs various questions to be answered.
    c. How does a software project get started?

2. **Elicitation**
    a. This is the process by which users of the system are interviewed in order to reveal and understand their requirements.
    b. This phase involves a high level of client input.

3. **Elaboration**
    a. The most relevant, mission critical requirements are emphasized and developed first.
    b. This ensures the final product satisfies all of the important requirements, and does so in the most time and cost efficient manner possible.

4. **Negotiation**
    a. A milestone within that phase may consist of implementing Use Case x, y and z.
    b. By scheduling the project in this iterative way, our client has a good understanding of when certain things will be delivered and when their input will be required.

5. **Specification**
    a. This is the process by which requirements and their analyses are committed to some formal media.
    b. For this project, we would generate four documents during this sub-phase:
    c. **Use Case Document**: A use case is a description of the system's response as a result of a specific request from a user.
    d. **Interface Prototypes**: An interface prototype is a rough representation of certain critical functionality to elaborate HTML.
    e. **Architecture Diagram**: An architecture diagram will be generated for developer use as it is a high level view of how the software will be structured.
    f. **Database Diagram**: As with the architecture diagram, a database diagram is generally created for developer use.

6. **Validation**
    a. This is the final phase of the requirements engineering process.
    b. This must be done by all interested parties so they are in agreement with the proposed system's behavior.

7. **Management**
    a. As requirements are elicited, analyzed, specified and validated, we will be estimating most features.
    b. This is an on-going process that will culminate in the delivery of the Project Estimation and Timeline document.

2.9 Software Requirement Specification

2.9.1 Concept of SRS

(Question: Explain the concept of SRS with general format – 4 Marks)

- A software requirement specification (SRS) is a complete description of the intended purpose and environment for software under development.
- The SRS fully describes what the software will do and how it will be expected to perform.
- An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost.
- A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations.
- Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

2.9.2 General Format of SRS

1. **Functional Requirements Definition**
    a. Functional requirements are a subset of the overall system requirement.
    b. These requirements are used to consider system behavior.
    c. Trade-offs *(a balance achieved between)* may be between hardware and software issues.

2. **Non-functional Requirements Definition**
    a. It measures the documentation and the inputs for the various requirements in the system.

3. **System Evolution**
    a. Starting from the specification and the data base schema, on how the system should be designed.

2.9.3 Need/Importance of SRS

1. **What will be the business impact of the software?**
    a. The development of the software will help the company to grow on the larger perspective.

2. **What the customer wants exactly?**
    a. The communication principles to be followed of what exactly is the requirement of the customer.

3. **How end user will interact with the system?**
    a. Proper documentation and step by step procedure of working module of the project or the software.

4. **Developed as a joint effort between the developer and the customer**
    a. By understanding the complete requirement the project is developed.