Fundamental of software engineering MCQ with Answer

1. Software engineering is a \_\_\_\_\_\_\_\_\_ technology.

- A) Layered

- B) Linear

- C) Circular

- D) None of the above

- Answer: A) Layered

2. Software engineering encompasses a \_\_\_\_\_\_\_\_, a collection of \_\_\_\_\_\_\_ and an array of \_\_\_\_\_\_\_ that allow professionals to build high-quality computer software.

- A) Process, methods, tools

- B) Principles, tools, engineering

- C) Quality, testing, features

- D) None of the above

- Answer: A) Process, methods, tools

3. Who is the father of software engineering?

- A) Margaret Hamilton

- B) Watts S. Humphrey

- C) Alan Turing

- D) Boris Beizer

- Answer: B) Watts S. Humphrey

4. What are the features of good software code?

- A) Simplicity

- B) Accessibility

- C) Modularity

- D) All of the above

- Answer: D) All of the above

5. What does SDLC stand for?

- A) System Design Life Cycle

- B) Software Design Life Cycle

- C) Software Development Life Cycle

- D) System Development Life Cycle

- Answer: C) Software Development Life Cycle

6. What is a functional requirement?

- A) Specifies the tasks the program must complete

- B) Specifies the tasks the program should not complete

- C) Specifies the tasks the program must not work

- D) All of the above

- Answer: A) Specifies the tasks the program must complete

7. What is agile scrum methodology?

- A) Project management that emphasizes incremental progress

- B) Project management that emphasizes decremental progress

- C) Project management that emphasizes neutral progress

- D) Project management that emphasizes no progress

- Answer: A) Project management that emphasizes incremental progress

8. CASE stands for \_\_\_\_\_\_\_\_\_.

- A) Computer-Aided Software Engineering

- B) Control Aided Science and Engineering

- C) Cost Aided System Experiments

- D) None of the mentioned above

- Answer: A) Computer-Aided Software Engineering

9. Who proposed the spiral model?

- A) Barry Boehm

- B) Pressman

- C) Royce

- D) IBM

- Answer: A) Barry Boehm

10. Software engineering enables us to build complex systems in a specified \_\_\_\_\_\_ period with high \_\_\_\_\_\_\_.

- A) Time, quality

- B) Testing, debugging

- C) None of the mentioned above

- D) All of the above

- Answer: A) Time, quality

11. Software is defined as \_\_\_\_\_\_\_\_\_\_\_.

- A) Set of programs, documentation & configuration of data

- B) Set of programs

- C) Documentation and configuration of data

- D) None of the mentioned

- Answer: A) Set of programs, documentation & configuration of data

12. What is software engineering?

- A) Designing a software

- B) Testing a software

- C) Application of engineering principles to the design a software

- D) None of the above

- Answer: C) Application of engineering principles to the design a software

13. \_\_\_\_\_\_\_\_\_\_\_\_ is a software development activity that is not a part of software processes.

- A) Validation

- B) Specification

- C) Development

- D) Dependence

- Answer: D) Dependence

14. \_\_\_\_\_\_\_\_ is defined as the process of generating analysis and designing documents?

- A) Re-engineering

- B) Reverse engineering

- C) Software re-engineering

- D) Science and engineering

- Answer: B) Reverse engineering

15. The activity that distributes estimated effort across the planned project duration by allocating the effort to specific software developing tasks is \_\_\_\_\_\_\_\_\_\_\_\_.

- A) Project scheduling

- B) Detailed schedule

- C) Macroscopic schedule

- D) None of the mentioned

- Answer: A) Project scheduling

16. Why do bugs and failures occur in software?

- A) Because of developers

- B) Because of companies

- C) Because of both companies and developers

- D) None of the mentioned

- Answer: C) Because of both companies and developers

17. Attributes of good software are \_\_\_\_\_\_\_\_\_\_\_\_.

- A) Development

- B) Maintainability & functionality

- C) Functionality

- D) Maintainability

- Answer: B) Maintainability & functionality

18. The Cleanroom philosophy was proposed by \_\_\_\_\_\_\_\_\_.

- A) Linger

- B) Mills

- C) Dyer

- D) All of the mentioned

- Answer: D) All of the mentioned

19. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ is not among the eight principles followed by the Software Code of Ethics and Professional Practice.

- A) PRODUCT

- B) ENVIRONMENT

- C) PUBLIC

- D) PROFESSION

- Answer: B) ENVIRONMENT

20. Software is not affected by the same \_\_\_ that cause hardware to fail.

- A) Environmental factors

- B) Mechanical factors

- C) Electrical factors

- D) None of the mentioned above

- Answer: A) Environmental factors

21. Software is developed or engineered; it is not \_\_\_\_\_\_\_.

- A) Manufactured

- B) Programmed

- C) Tested

- D) None of the mentioned

- Answer: A) Manufactured

22. What are the benefits of software engineering?

- A) Reduces complexity

- B) Improves reliability

- C) Enhances productivity

- D) All of the above

- Answer: D) All of the above

23. What are the types of software products?

- A) Generic and customized

- B) Standard and customized

- C) Generic and specific

- D) None of the above

- Answer: A) Generic and customized

24. What are the phases of software development life cycle (SDLC)?

- A) Planning, analysis, design, implementation, testing, maintenance

- B) Requirement, design, coding, testing, deployment, maintenance

- C) Specification, development, validation, evolution

- D) All of the above

- Answer: D) All of the above

25. What are the models of software process?

- A) Waterfall model

- B) Incremental model

- C) Spiral model

- D) All of the above

- Answer: D) All of the above

26. What are the characteristics of a good software process model?

- A) Predictability

- B) Manageability

- C) Adaptability

- D) All of the above

- Answer: D) All of the above

27. What are the activities involved in software specification?

- A) Requirement engineering

- B) System modeling

- C) System analysis

- D) All of the above

- Answer: D) All of the above

28. What are the types of software requirements?

- A) Functional and non-functional

- B) User and system

- C) Quality and performance

- D) All of the above

- Answer: D) All of the above

29. What are the techniques for eliciting requirements from stakeholders?

- A) Interviews

- B) Questionnaires

- C) Scenarios

- D) All of the above

- Answer: D) All of the above

30. What are the principles of software design?

- A) Abstraction

- B) Modularity

- C) Coupling and cohesion

- D) All of the above

- Answer: D) All of the above

Software engineering is the systems engineering approach for software product/application development. It is an engineering branch associated with analyzing user requirements, design, development, testing, and maintenance of software products².

Some basic principles of software engineering include:

- \*\*Better requirement analysis\*\*: This gives a clear vision of the project and provides value to its users by delivering a good software product that meets their requirements².

- \*\*KISS (Keep it Simple, Stupid)\*\*: This means that all designs and implementations should be as simple as possible, which makes code easy to debug and maintain².

- \*\*Modularity\*\*: This means breaking the software into smaller, reusable components that can be developed and tested independently³.

- \*\*Abstraction\*\*: This means hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software³.

- \*\*Think then act\*\*: This means that before starting developing functionality, it requires to think about application architecture, as good planning on the flow of project development produces better results².

- \*\*Never add extra\*\*: This means that the developer should implement what is actually needed and avoid adding unnecessary features or functionalities that may increase complexity and effort².

- \*\*Documentation\*\*: This means providing clear and concise explanations of the code and design decisions at required steps, which helps other developers to understand and work with the software².

- \*\*Law of Demeter\*\*: This means that each component should have minimal knowledge of other components and reduce connections and interdependencies between them, which is called coupling².

- \*\*Generality\*\*: This means that the software should not be limited or restricted to some cases or functions, but rather it should be free from unnatural restrictions and able to provide service to customers what they actually need or want².

- \*\*Consistency\*\*: This means that the coding style and the graphical user interface (GUI) should follow a uniform pattern and standard, which makes the code easier to read and the GUI easier to learn and use².

Apart from the principles I mentioned earlier, there are some other principles of software engineering that are widely followed by software engineers. Some of them are:

- \*\*The SOLID principles\*\*: These are five principles that help to create software that is easy to understand, extend, and maintain. They are¹:

- \*\*Single Responsibility Principle (SRP)\*\*: This means that each class or module should have only one reason to change and should be responsible for only one aspect of the software.

- \*\*Open-Closed Principle (OCP)\*\*: This means that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. In other words, new functionality should be added by adding new code rather than changing existing code.

- \*\*Liskov Substitution Principle (LSP)\*\*: This means that subclasses should be able to replace their base classes without breaking the functionality of the software. In other words, the behavior of a subclass should be consistent with the behavior of its base class.

- \*\*Interface Segregation Principle (ISP)\*\*: This means that clients should not be forced to depend on interfaces that they do not use. In other words, interfaces should be small and focused rather than large and general.

- \*\*Dependency Inversion Principle (DIP)\*\*: This means that high-level modules should not depend on low-level modules; both should depend on abstractions. In other words, software should depend on interfaces rather than concrete implementations.

- \*\*The YAGNI principle\*\*: This stands for You Aren't Gonna Need It and it means that you should not add functionality or complexity to your software until you actually need it. In other words, you should avoid over-engineering and premature optimization and focus on delivering the minimum viable product².

- \*\*The KISS principle\*\*: This stands for Keep It Simple, Stupid and it means that you should always strive for simplicity and clarity in your software design and code. In other words, you should avoid unnecessary complexity and confusion and make your software easy to understand and maintain².

- \*\*The DRY principle\*\*: This stands for Don't Repeat Yourself and it means that you should avoid code duplication and redundancy in your software. In other words, you should create reusable components and functions and avoid repeating the same logic or data in multiple places².

A software process is the set of activities for designing, implementing, and testing a software system. A software process model is an abstract representation of the software process that specifies the stages and order of the activities¹.

There are many types of software process models, each with its own advantages and disadvantages. Some of the most common software process models are¹²:

- \*\*Waterfall model\*\*: This is a linear and sequential model that follows a fixed set of phases: requirements analysis, design, implementation, testing, deployment, and maintenance. Each phase must be completed before moving to the next one. This model is simple and easy to follow, but it does not accommodate changing requirements or customer feedback well.

- \*\*V model\*\*: This is an extension of the waterfall model that emphasizes verification and validation at each phase. For each development phase, there is a corresponding testing phase. This model ensures high quality and reliability, but it also suffers from the same drawbacks as the waterfall model.

- \*\*Incremental model\*\*: This is an iterative model that divides the software into small increments that are developed and delivered separately. Each increment adds some functionality to the software until the final product is complete. This model allows for early feedback and testing, but it also requires careful planning and coordination.

- \*\*RAD model\*\*: This stands for Rapid Application Development and it is a model that focuses on delivering software quickly and with minimal planning. It uses prototyping, reuse, and user involvement to speed up the development process. This model is suitable for fast-changing and low-risk projects, but it may compromise quality and scalability.

- \*\*Agile model\*\*: This is a family of models that embrace change, collaboration, and customer satisfaction. Agile models use short iterations, frequent deliveries, continuous feedback, and self-organizing teams to produce software that meets changing needs and expectations. Agile models are flexible and adaptive, but they also require high commitment and discipline from the developers and customers.

- \*\*Iterative model\*\*: This is a model that repeats the same set of activities over several iterations until the software meets the desired quality and functionality. Each iteration produces a working version of the software that can be evaluated and improved. This model allows for learning and refinement, but it also requires clear goals and metrics to measure progress.

- \*\*Prototype model\*\*: This is a model that uses a prototype or a mock-up of the software to elicit requirements and feedback from the customers. The prototype is then refined or discarded based on the feedback. This model helps to clarify ambiguous or incomplete requirements, but it also may create unrealistic expectations or confusion among the customers.

Sure, I can give you some general guidelines on when to use and when not to use these models. However, you should keep in mind that the choice of a software process model depends on many factors, such as the project requirements, size, complexity, budget, timeline, customer expectations, team skills, etc. Therefore, you should always evaluate the pros and cons of each model for your specific project and context.

Here are some possible scenarios for using or avoiding these models:

- \*\*Waterfall model\*\*: You can use this model when you have clear and stable requirements, well-defined specifications, and predictable outcomes. You should avoid this model when you have changing or uncertain requirements, frequent customer feedback, or complex and dynamic environments.

- \*\*V model\*\*: You can use this model when you need to ensure high quality and reliability of the software, when you have well-defined specifications and test cases, and when you have enough time and resources for testing. You should avoid this model when you have changing or uncertain requirements, frequent customer feedback, or complex and dynamic environments.

- \*\*Incremental model\*\*: You can use this model when you want to deliver software quickly and incrementally, when you have modular and independent features or functions, and when you can get early feedback and testing. You should avoid this model when you have interdependent or complex features or functions, or when you have difficulty in planning and coordinating the increments.

- \*\*RAD model\*\*: You can use this model when you need to deliver software rapidly and with minimal planning, when you have low-risk and fast-changing projects, and when you can reuse existing components or frameworks. You should avoid this model when you need high quality and scalability of the software, when you have high-risk or long-term projects, or when you have limited resources or expertise.

- \*\*Agile model\*\*: You can use this model when you need to accommodate changing requirements and customer expectations, when you have collaborative and self-organizing teams, and when you value customer satisfaction over documentation. You should avoid this model when you need to follow strict standards or regulations, when you have large or distributed teams, or when you have low commitment or discipline from the developers or customers.

- \*\*Iterative model\*\*: You can use this model when you need to learn and improve from each iteration, when you have unclear or incomplete requirements, and when you have flexible timelines and budgets. You should avoid this model when you need to deliver software quickly and with minimal changes, when you have clear and stable requirements, or when you have difficulty in measuring progress or quality.

- \*\*Prototype model\*\*: You can use this model when you need to elicit requirements and feedback from the customers, when you have ambiguous or incomplete requirements, and when you have innovative or experimental projects. You should avoid this model when you need to deliver software quickly and with minimal changes, when you have clear and stable requirements, or when you have unrealistic or confused customers.

The spiral model is a risk-driven software development process model that combines the iterative and incremental approach with the systematic and controlled aspects of the waterfall model⁵. The spiral model is based on the idea of a spiral, with each loop of the spiral representing a complete software development cycle, from planning and risk analysis to engineering and evaluation¹.

The spiral model is suitable for complex and large-scale software projects that have high levels of uncertainty or risk. It allows for flexibility and adaptation to changing requirements and customer feedback. It also enables early prototyping and testing of the software¹⁵.

You can use the spiral model when you need to manage risk effectively, when you have unclear or incomplete requirements, when you have innovative or experimental projects, or when you have flexible timelines and budgets²⁴.

You should avoid the spiral model when you need to deliver software quickly and with minimal changes, when you have clear and stable requirements, when you have low-risk or simple projects, or when you have limited resources or expertise²⁴.