1. What is Software Engineering? (2 points)

Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance. The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product. Four fundamental activities are common to all software processes: Software specification (define the software that is to be produced and the constraints on its operation), Software development (software is designed and programmed), Software validation (software is checked to ensure that it is what the customer requires), Software evolution (software is modified to reflect changing customer and market requirements).

2. Why is the Waterfall method described as a plan-driven development methodology? (2 points)

It presents the software development process as a number of stages. Because of the cascade from one phase to another, this model is known as the waterfall model or software life cycle. The waterfall model is an example of a plan-driven process. In principle at least, you plan and schedule all of the process activities before starting software development.

3. What do you think that some companies opening software product beta testing public? (2 points)

When a system is to be marketed as a software product, a testing process called beta testing is often used. Beta testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors that may not have been anticipated by the product developers. After this feedback, the software product may be modified and released for further beta testing or general sale.

4. What is the most commonly used V&V activity in testing? (2 points)

A description of the V & V procedures used and, where appropriate, the test plans for the system. Summaries of the test results showing defects that have been detected and corrected. If formal methods have been used, a formal system specification and any analyses of that specification. Records of static analyses of the source code.

5. What is the difference between Functional Requirements and Non-Functional Requirements? (2 points)

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: *Functional* and *Non-functional requirements*.
Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

6. What are the advantages of Test-driven development in Software Engineering? (2 points)

Test-driven development helps programmers clarify their ideas of what a code segment is actually supposed to do. To write a test, you need to understand what is intended, as this understanding makes it easier to write the required code. Of course, if you have incomplete knowledge or understanding, then TDD won't help. If you don't know enough to write the tests, you won't develop the required code. For example, if your computation involves division, you should check that you are not dividing the numbers by zero. If you forget to write a test for this, then the checking code will never be included in the program.

As well as better problem understanding, other benefits of test-driven development are:
1. Code coverage
In principle, every code segment that you write should have at least one associated test. Therefore, you can be confident that all of the code in the system has actually been executed. Code is tested as it is written, so defects are discovered early in the development process.

2. Regression testing

A test suite is developed incrementally as a program is developed. You can always run regression tests to check that changes to the program have not introduced new bugs.

3. Simplified debugging

When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified. You do not need to use debugging tools to locate the problem. Reports of the use of TDD suggest that it is hardly ever necessary to use an automated debugger in test-driven development (Martin 2007).

4. System documentation

The tests themselves act as a form of documentation that describe what the code should be doing. Reading the tests can make it easier to understand the code.

One of the most important benefits of TDD is that it reduces the costs of regression testing. Regression testing involves running test sets that have successfully executed after changes have been made to a system. The regression test checks that these changes have not introduced new bugs into the system and that the new code interacts as expected with the existing code. Regression testing is expensive and sometimes impractical when a system is manually tested, as the costs in time and effort are very high. You have to try to choose the most relevant tests to re-run and it is easy to miss important tests.

7. What is the main difference between Requirement Elicitation and Requirements Specification? (2 points)

The aims of the requirements elicitation process are to understand the work that stakeholders do and how they might use a new system to help support that work. During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and so on.

Requirements specification is the process of writing down the user and system requirements in a requirements document. Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent. In practice, this is almost impossible to achieve. Stakeholders

interpret the requirements in different ways, and there are often inherent conflicts and inconsistencies in the requirements.

8. What are Software Design Patterns? (2 points)

Design patterns were derived from ideas put forward by Christopher Alexander (Alexander 1979), who suggested that there were certain common patterns of building design that were inherently pleasing and effective. The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings. The pattern is not a detailed specification. Rather, you can think of it as a description of accumulated wisdom and experience, a well-tried solution to a common problem. A quote from the Hillside Group website (hillside.net/patterns/), which is dedicated to maintaining information about patterns, encapsulates their role in reuse:

Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience†.

9. Why is the scrum (agile) method convenient in modern software engineering? (2 points)

Scrum is an agile development methodology used in the development of Software based on an iterative and incremental processes. Scrum is adaptable, fast, flexible and effective agile framework that is designed to deliver value to the customer throughout the development of the project. The primary objective of Scrum is to satisfy the customer's need through an environment of transparency in communication, collective responsibility and continuous progress. The development starts from a general idea of what needs to be built, elaborating a list of characteristics ordered by priority (product backlog) that the owner of the product wants to obtain.

10. What is the difference between Release testing & User testing? (2 points)

Release testing, where a separate testing team tests a complete version of the system before it is released to users. The aim of release testing is to check that the system meets the requirements of the system stakeholders.

User testing, where users or potential users of a system test the system in their own environment. For software products, the "user" may be an internal marketing group that decides if the software can be marketed, released and sold. Acceptance testing is one type of user testing where the customer formally tests a system to decide if it should be accepted from the system supplier or if further development is required.

11. What kind of methods/activities are there in Requirements Elicitation? (2 points)

The process activities are:

1. Requirements discovery and understanding
This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.

2. Requirements classification and organization
This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.

3. Requirements prioritization and negotiation
Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

4. Requirements documentation
The requirements are documented and input into the next round of the spiral. An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.

12. How do companies get benefits and earn money from open-source software solutions? (2 points)

The open-source approach is one of several business models for software. In this model, companies release the source of their software and sell add-on services and advice in association with this. They may also sell cloud-based software services— an attractive option for users who do not have the expertise to manage their own open-source system and also specialized versions of their system for particular clients. Open-source is therefore likely to increase in importance as a way of developing and distributing software.

13. Why is the Activity Diagram widely used between software developers and non-software developers (salespeople, logistics people, finance people, etc)? (2 points)

UML activity diagrams may be used to show the business processes in which systems are used UML activity diagrams show the activities in a process and the flow of control from one activity to another. The start of a process is indicated by a filled circle, the end by a filled circle inside another circle. Rectangles with round corners represent activities, that is, the specific subprocesses that must be carried out. You may include objects in activity charts. Figure 5.2 shows the systems that are used to support different subprocesses within the involuntary detection process. I have shown that these are separate systems by using the UML stereotype feature where the type of entity in the box between chevrons is shown. Arrows represent the flow of work from one activity to another, and a solid bar indicates activity coordination.

14. Please draw UML diagrams to describe WSP (University-based information exchange application) (2 points)

15. Where do we use State Machine Diagrams? (2 points)

State machine diagrams are **usually applied to objects but can be applied to any element that has behavior to other entities such as: actors, use cases, methods, subsystems systems and etc**. and they are typically used in conjunction with interaction diagrams (usually sequence diagrams).

16. What is the availability of the principal dependability properties of software engineering? (2 points)

Availability
Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services to users at any given time.

17. What is the 4 + 1 view model of software architecture? (2 points)

four fundamental architectural views, which can be linked through common use cases or scenarios (Figure 6.3). He suggests the following views:

1. A logical view, which shows the key abstractions in the system as objects or object classes. It should be possible to relate the system requirements to entities in this logical view.

2. A process view, which shows how, at runtime, the system is composed of interacting processes. This view is useful for making judgments about non-functional system characteristics such as performance and availability. 3. A development view, which shows how the software is decomposed for development; that is, it shows the breakdown of the software into components that are implemented by a single developer or development team. This view is useful for software managers and programmers.

4. A physical view, which shows the system hardware and how software components are distributed across the processors in the system. This view is useful for systems engineers planning a system deployment

18. What are Context models in System Modelling? (2 points)

Context models show how a system that is being modeled is positioned in an environment with other systems and processes. They help define the boundaries of the system to be developed.

19. **Why do you think that modern software operation system turns off automatically when processors heat up (at a higher temperature than the allowed temperature)? What are the principles behind this dependability property? (2 points)**

Resilience
Informally, the resilience of a system is a judgment of how well that system can maintain the continuity of its critical services in the presence of disruptive events, such as equipment failure and cyberattacks. Resilience is a more recent addition to the set of dependability properties that were originally suggested by Laprie.

20. Please describe how GitHub helps software developers (2 points)

Group work. Commit. Check the Performance. Copy. Subscribe. Add. Share