**1. Why is software architecture a major issue for software engineering?**

Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system. The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity. Architecture “constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together”

**2. What is a software architecture? What does it describe? How is it used? By whom? And for what purpose?**

Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

Software architecture encompasses the set of significant decisions about the organization of a software system: Selection of the structural elements and their interfaces by which a system is composed; Behavior as specified in collaborations among those elements; Composition of these structural and behavioral elements into larger subsystems; Architectural style that guides this organization Skills and development teams; Stakeholders

**3. Why are class interfaces so important in software architecture?**

A component is an encapsulated part of a software system. A component has an interface. Selection of the structural elements and their interfaces by which a system is composed. Software architecture represents a structure in which entities are connected by a set of define relationships. Properties describe the nature of the components and the organization of the interfaces.

**4. How do we perform quality control of a software architecture?**

At each stage, software design work products are reviewed for clarity, correctness, completeness, and consistency with requirements and with one another

**5. Describe the layered architecture. What are the principles behind a layered architecture?**

A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions.

Once requirements engineering find the characteristics and constraints of the system to be built, the architectural style and/or combination of patterns that best fits those characteristics and constraints can be chosen. In many cases, more than one pattern might be appropriate and alternative architectural styles can be designed and evaluated

**6. How does a typical client-server architecture relate to the layered architecture?**

Client-server is a 2-tiered layered architecture. Clients and servers communicate using request/reply protocol. Clients initiate communication by issuing a request, such as a message or a remote-procedure call, and servers respond by fulfilling the request and replying with a result.

The relationship between the clients and servers is asymmetrical. Clients know the identity of the server from which they request information, but servers know nothing about which, or even how many, clients they serve. Presentation layer is part of the business layer

**7. How does a typical** **client-server architecture relate to the domain model in requirements?**

Client-server is a 2-tiered layered architecture. Client-server architecture contains the whole domain model which is loaded on client start-up. It therefore can build up queries and cases based on the domain model and sends requests to the server only when needed. Presentation layer is part of the business layer

**8. Describe the Model-View-Controller architecture. What are the principles behind the MVC architecture?**

It is a software design pattern for developing web applications. Model - The lowest level of the pattern which is responsible for maintaining data. View - This is responsible for displaying all or a portion of the data to the user. Controller - Software Code that controls the interactions between the Model and View. MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows. Improves maintainability. Model notifies the views of state changes.

**9. How is the Observer design pattern used in MVC?**

Observer pattern – define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The subject of the observer pattern is the Model with states that can be modified. The observer is the View which is attached/registered to the subject, such that it is automatically notified when changes in the model occur, consequently showing these changes to the user.

**10.What is the relationship between a typical client-server architecture and the MVC?**

The controller and model classes on the server. Controller is the component that's tricky. One may be easily tempted to keep it on the client, reasons to put it on the server would be: interactions with Model classes, error handling and control flow. A controller on the client side may prove to be easy to develop but ultimately you need to pass changes to the server. Further, a controller on the client side would then slowly start pushing you towards more and more classes on the client side.