NAME: HASSAN KAMAL

ID#: 12925

SUBJECT NAME: SOFTWARE DESIGN AND ARCHITECTURE.

Question 1:

a: What is software architecture? Why is software architecture design so important?

Answer:

Software architecture:

refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.

Software architecture is the foundation of a software system. Like other types of engineering, the foundation has a profound effect on the quality of what is built on top of it. As such, it holds a great deal of importance in terms of the successful development, and eventual maintenance, of the system.

Software architecture is a series of decisions. Some of the earliest decisions come from designing the architecture, and these carry a high degree of importance because they affect the decisions that come after it.

Another reason software architecture is important is because all software systems have an architecture. Even if it comprised just one structure with one element, there is an architecture. There are software systems that don't have a formal design and others that don't formally document the architecture, but even these systems still have an architecture.

The greater the size and complexity of a software system, the more you will need a well-thought-out architecture in order to succeed. Software architecture provides a number of benefits when done properly, which greatly increase the chances that the software system will succeed.

A proper foundation laid down by a software system's architecture yields a number of benefits. Let's take a deeper look at those benefits.

b: Explain any four tasks of architect?

Answer:

Customer Service and Retention

Architects work closely with their clients. Before drafting plans, they meet with their clients several times to learn their clients' objectives, budget and any specific requirements for their project. Architects also work with other related professionals, such as engineers, urban planners, landscape architects, construction representatives and interior designers. Since they deal with customers and professionals directly, architects must have excellent oral and written communication skills.

Design, Plan and Develop

Designing, planning and developing are integral tasks in an architect's daily routine. Architects may be required to provide predesign figures such as an environmental impact or feasibility study, cost analysis and land-use study. Final construction plans are created by architects and used by builders as a step-by-step guide on how the look and details of the building will play out, including plumbing, communication and heating, electrical, ventilation and structural systems.

Research and Knowledge

Architects must follow building codes, fire regulations, zoning laws and city ordinances when creating their plans. For public buildings, architects must be aware of disabled access laws. Since these laws change regularly, architects need to stay up-to-date on policy, zoning and regulation changes. Architects must be in on the latest energy-efficient products, building styles and must research the area they are building to ensure their design matches current building structures. Some states require architects to enroll in continuing education courses to keep their license, while other states may require attendance at seminars, workshops and conferences to maintain a license.

Application of Technology

Today's architects use a lot of technology. Architects must be trained and familiar with computer-aided drafting systems, building modeling and other relevant technologies. They must conceptualize and experiment with different construction approaches through software and must also be familiar with basic office software, such as word processing, spreadsheets and accounting.

Question 2:

Explain Architecture Business Cycle (ABC) in detail with figure.

Answer:

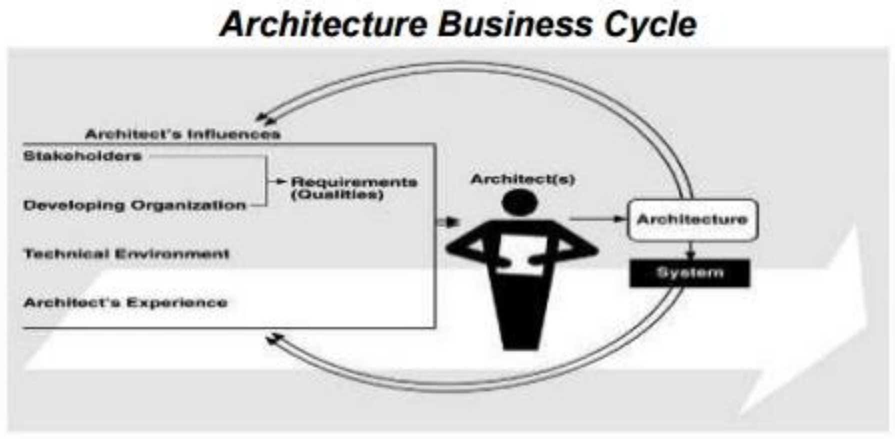
"Software architecture is a result of technical, business, and social influences. Its existence in turn affects the technical, business, and social environments that subsequently influence future architectures. We call this cycle of influences, from the environment to the architecture and back to the environment, the Architecture Business Cycle (*ABC*)."

1.The organization goals of **Architecture Business Cycle** are beget requirements, which beget an architecture, which begets a system. The architecture flows from the architect's experience and the technical environment of the day.

2.Three things required for ABC are as follows:

i. Case studies of successful architectures crafted to satisfy demanding requirements, so as to help set the technical playing field of the day.

ii. Methods to assess an architecture before any system is built from it, so as to mitigate the risks associated with launching unprecedented designs. **iii.Techniques** for incremental architecture-based development, so as to uncover design flaws before it is too late to correct them.



Question 3:

Explain ABC activities.

Answer:

1. The architecture affects the structure of the developing organization. An architecture prescribes a structure for a system; as we will see, it particularly prescribes the units of software that must be implemented (or otherwise obtained) and integrated to form the system. These units are the basis for the development project's structure. Teams are formed for individual software units; and the development, test, and integration activities all revolve around the units. Likewise, schedules and budgets allocate resources in chunks corresponding to the units. If a company becomes adept at building families of similar systems, it will tend to invest in each team by nurturing each area of expertise. Teams become embedded in the organization's structure. This is feedback from the architecture to the developing organization.

In the software product line case study, separate groups were given responsibility for building and maintaining individual portions of the organization's architecture for a family of products. In any design undertaken by the organization at large, these groups have a strong voice in the system's decomposition, pressuring for the continued existence of the portions they control.

2. The architecture can affect the goals of the developing organization. A successful system built from it can enable a company to establish a foothold in a particular market area. The architecture can provide opportunities for the efficient

production and deployment of similar systems, and the organization may adjust its goals to take advantage of its newfound expertise to plumb the market. This is feedback from the system to the developing organization and the systems it builds.

3. The architecture can affect customer requirements for the next system by giving the customer the opportunity to receive a system (based on the same architecture) in a more reliable, timely, and economical manner than if the subsequent system were to be built from scratch. The customer may be willing to relax some requirements to gain these economies. Shrink-wrapped software has clearly affected people's requirements by providing solutions that are not tailored to their precise needs but are instead inexpensive and (in the best of all possible worlds) of high quality. Product lines have the same effect on customers who cannot be so flexible with their requirements. A Case Study in Product Line Development will show how a product line architecture caused customers to happily compromise their requirements because they could get high-quality software that fit their basic needs quickly, reliably, and at lower cost.

4. The process of system building will affect the architect's experience with subsequent systems by adding to the corporate experience base. A system that was successfully built around a tool bus or .NET or encapsulated finite-state machines will engender similar systems built the same way in the future. On the other hand, architectures that fail are less likely to be chosen for future projects.

5. A few systems will influence and actually change the software engineering culture, that is, the technical environment in which system builders operate and learn. The first relational databases, compiler generators, and table-driven operating systems had this effect in the 1960s and early 1970s; the first spreadsheets and windowing systems, in the 1980s. The World Wide Web is the example for the 1990s. J2EE may be the example for the first decade of the twenty-first century. When such pathfinder systems are constructed, subsequent systems are affected by their legacy.

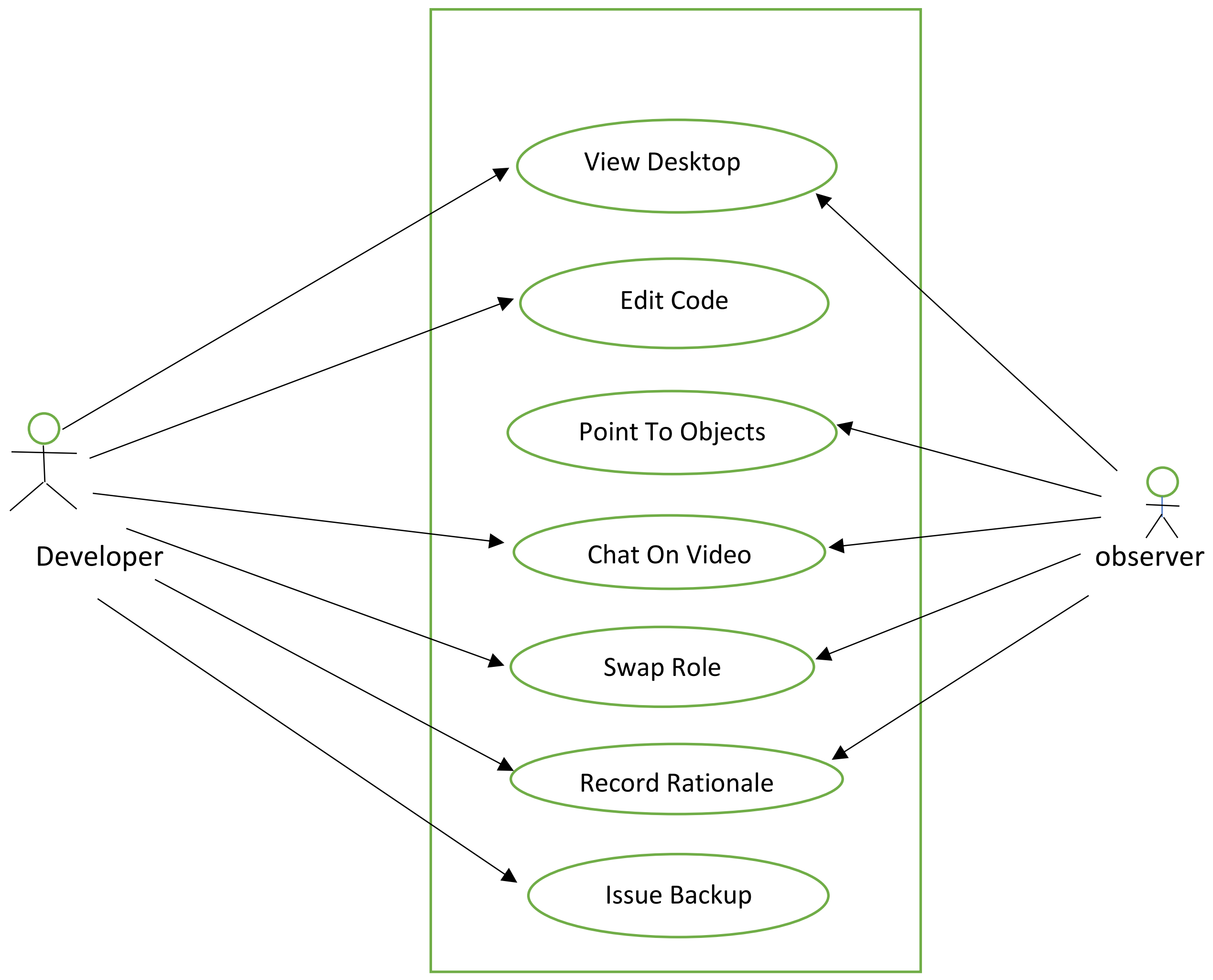
Question 4:

Pair Programming is an agile software development technique……… ……………………………………………………………………………… the system to backup old work.

- **Draw a use case diagram to show all the functionality of the system.**
- **Describe in detail four non-functional requirements of the system.**
- **Give a prioritized list of design constraint for the system and justify your list and the ordering.**
- **Propose a set of classes that could be used in your system and present them in a class diagram.**

Answer:

A:



B:

- Availability - the system should be available to both programmers all the time.
- Portability - the programmers should be able to use the system regardless of what computer and operating system used by the programmers.
- Security - the backup code should be kept securely and be protected from unauthorized access.
- Cost - users should pay no more than $5 per month to use RPP

C:

Example 1: "the system should be portable" is a NFR. This NFR may lead to a constraint on the programming language used for the implementation of the system (e.g., the programming language Java (rather than C and C++) might be preferred in order to meet this NFR).

Example 2: "security - the system must be secured" is a NFR. The design constraints could be a user authentication must be in place, the communication protocol must be encrypted, and/or the data must be stored on a server behind firewall.

D:

One obvious subsystem decomposition is to have 3 subsystems separating 'model', 'view', and 'controller'. So,

 * Programmer and its two subclasses in subsystem

1. * Desktop and all its subclasses together with GUIManager in subsystem

2. * Data Manager and its subclasses in subsystem 3. An alternative is to have 4 subsystems with GUIManager in one subsystem by itself.