Final-Term – Semester Assignment

- Subject : Software Design and Architecture
- Submitted To : Ma'am Aasma khan
- Submitted by : Muhammad Sohail
- Degree : BS(SE)
- ID # 14071
- Semester : 6th
- Date : 23/06/2020

Question No: 01

a) What is Software Architecture? Why is software architecture design so important?

b) Explain any four tasks of architect.

Answer a):

Software Architecture:

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. The *architecture* of a software system is a metaphor, analogous to the architecture of a building. It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.

Importance:

We focus on why architecture matters from a technical perspective. In this context, there are fundamentally three reasons for software architecture's importantance.

1. Communication among stakeholders. Software architecture represents a common abstraction of a system that most if not all of the system's stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication.
2. Early design decisions. Software architecture manifests the earliest design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its

maintenance life. It is also the earliest point at which design decisions governing the system to be built can be analyzed.

3. Transferable abstraction of a system. Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its elements work together, and this model is transferable across systems. In particular, it can be applied to other systems exhibiting similar quality attribute and functional requirements and can promote large-scale re-use.

b) **Explain any four tasks of architect.**

Answer:

- Perform static partition and decomposition of a system into subsystems and communications among subsystems.
    - ❖ A software element can be configured, delivered, developed, and deployed, and is replaceable in the future.
    - ❖ Each element's interface encapsulates details and provides loose coupling with other elements or subsystems.
    - ❖ Establish dynamic control relationships among different subsystems in terms of data flow, control flow orchestration, or message dispatching.
    - ❖ Consider and evaluate alternative architecture styles that suit the problem domain at hand.
        - – For example, in order to increase a distributed system's extensibility, portability, or maintainability, software components and Web services may be the best choice of

element types, and a loose connection among these elements may be most appropriate.

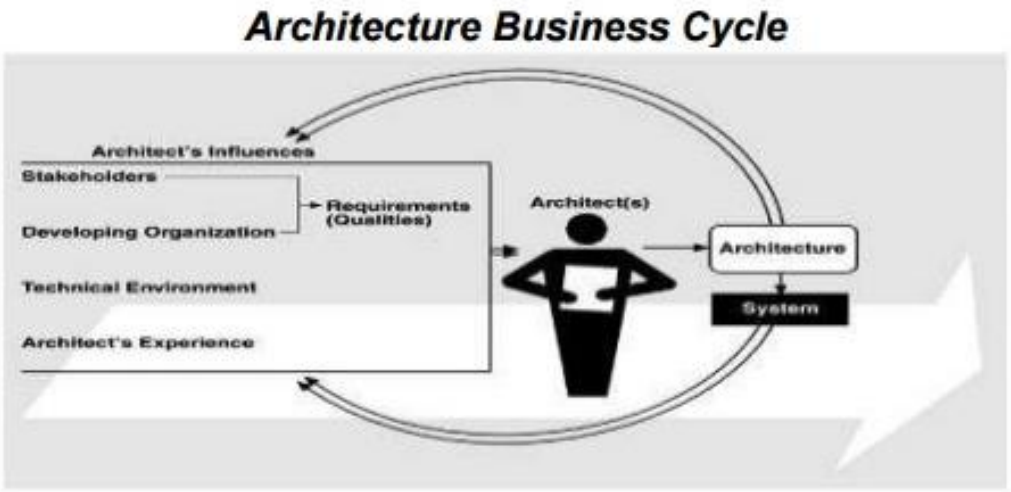
Question No: 02

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

Answer:

Architecture Business Cycle (ABC):

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).



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.

The architecture affects the –

- Structure of the developing organization.
- Goals of the developing of the organization.
- Customer requirements with reusability.
- The process of the system building will affect the architect's experience with subsequent systems.

Architecture business cycle changes- ◦

- Org. goals to req.
- Req. to arch.

- Arch. to systems.
- Systems to org

Influences

- Technical, business, social.
- Stakeholders, other source.

ABC activities include

- Create the business case.
- Understand the requirement.
- Create the architecture.
- Document & communicate the architecture.
- Analyse the architecture.
- Implement the system based on architecture
- Confirms the implementation.

Question No: 03

Explain ABC Activities?

Answer:

ABC includes the following activities

a. Create the business case.
b. Understand the requirement.
c. Create the architecture.
d. Document & communicate the architecture.
e. Analyse the architecture.
f. Implement the system based on architecture
g. Confirms the implementation.

Creating the business case for the system

It is simple to create a business case than understanding the needs of market How much should be the product cost? What is the Targeted market? What is the targeted time to market? Will it need to interface other system? Are there system limitations

Understanding the requirements

There are variety of techniques to understand requirements from stakeholders. Object oriented analysis: use cases & scenarios Safety Critical Systems: Finite state machine models Formal specification languages Quality attributes Prototypes Regardless of technique used, -

- the desired qualities of the system to be constructed determine the shape of architecture. | Website for Students

Creating the architecture

Conceptual integrity A small no. of minds coming together to design the system's architecture.

Communicating the architecture

For effective architecture It must be communicated clearly and unambiguously to all stakeholders. Developers must understand work assignments. Testers must understand the task structures Management must understand the scheduling implications

Analyzing the architecture

Out of multiple designs, after analyzing, some design will be accepted or some are rejected. Evaluating an architecture for the qualities it supports is essential to ensure the stakeholders satisfaction (needs). Scenario- based techniques are for evaluation of architecture. | Website for Students

Implementing based on the architecture

Concerned with keeping the developers faithful to the structures. Should have an environment that assists developers in creating the architecture. Ensuring conformance to an architecture Finally, when an architecture is created and used, it goes into maintenance phase. Constant vigilance is required to ensure that actual architecture and its implementations remain faithful to each other.

Confirming the implementations

The final step in the cycle is to confirm the implementations and reviewed by a single architect or small group of architects. gather both

the functional requirements and a well specified, prioritized list of quality attributes. be well documented, with at least one static view and one dynamic view. be reviewed by the system's stakeholders. be analyzed for applicable quantitative measures and formally evaluated for quality measures.

Question No 04: (20)

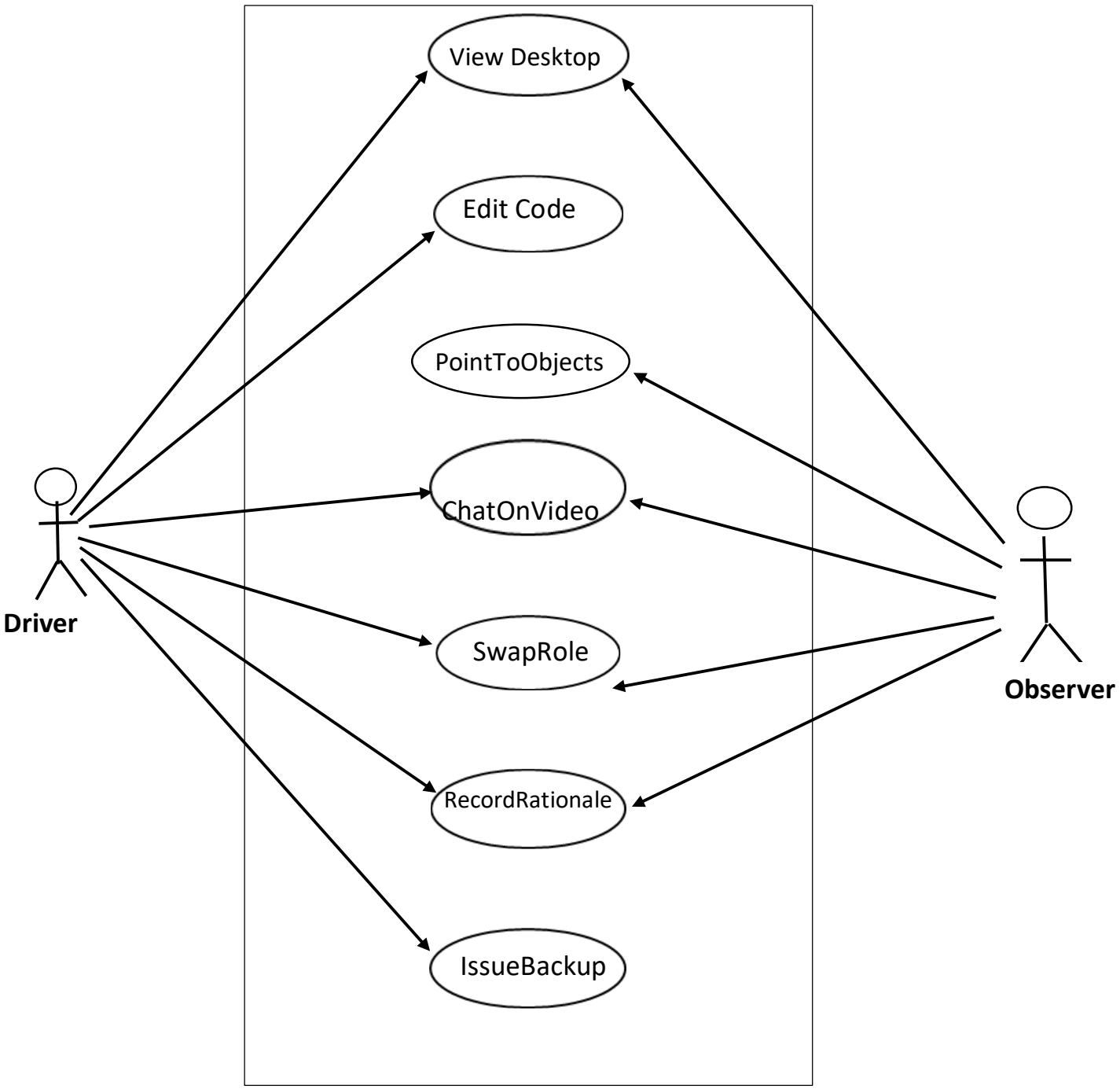
Pair programming is an agile software development technique in which two programmers work together at one work station. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer. The two programmers switch roles frequently (possibly every 30 minutes or less).

Suppose that you are asked to build a system that allows Remote Pair Programming. That is, the system should allow the driver and the observer to be in remote locations, but both can view a single desktop in real-time. The driver should be able to edit code and the observer should be able to "point" to objects on the driver's desktop. In addition, there should be a video chat facility to allow the programmers to communicate. The system should allow the programmers to easily swap roles and record rationale in the form of video chats. In addition, the driver should be able to issue 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 for the system.
- Give a prioritized list of design constraints 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:

Use-Case Diagram



Assumptions: when the Driver edits code, we assume that the Observer can see the changes in realtime through the ViewDesktop use case, thus there is no arrow pointing back to the Observer for the EditCode use case. A similar assumption is made for the PointToObjects use case, so no arrow points back to the Driver.

we assume that both the Driver and Observer can initiate the ViewDesktop, ChatVideo, SwapRole, and RecordRationale use cases.

Nonfunctional:

- **Ease of use** - the front-end interface must be simple and easy to use.
- **Real-time performance** - the Observer should be able to see the changes made by the Driver immediately without delay; the video chat should be smooth without delay also.
- **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.

Give a prioritized list of design constraints for the system and justify your list and the ordering.

Answer:

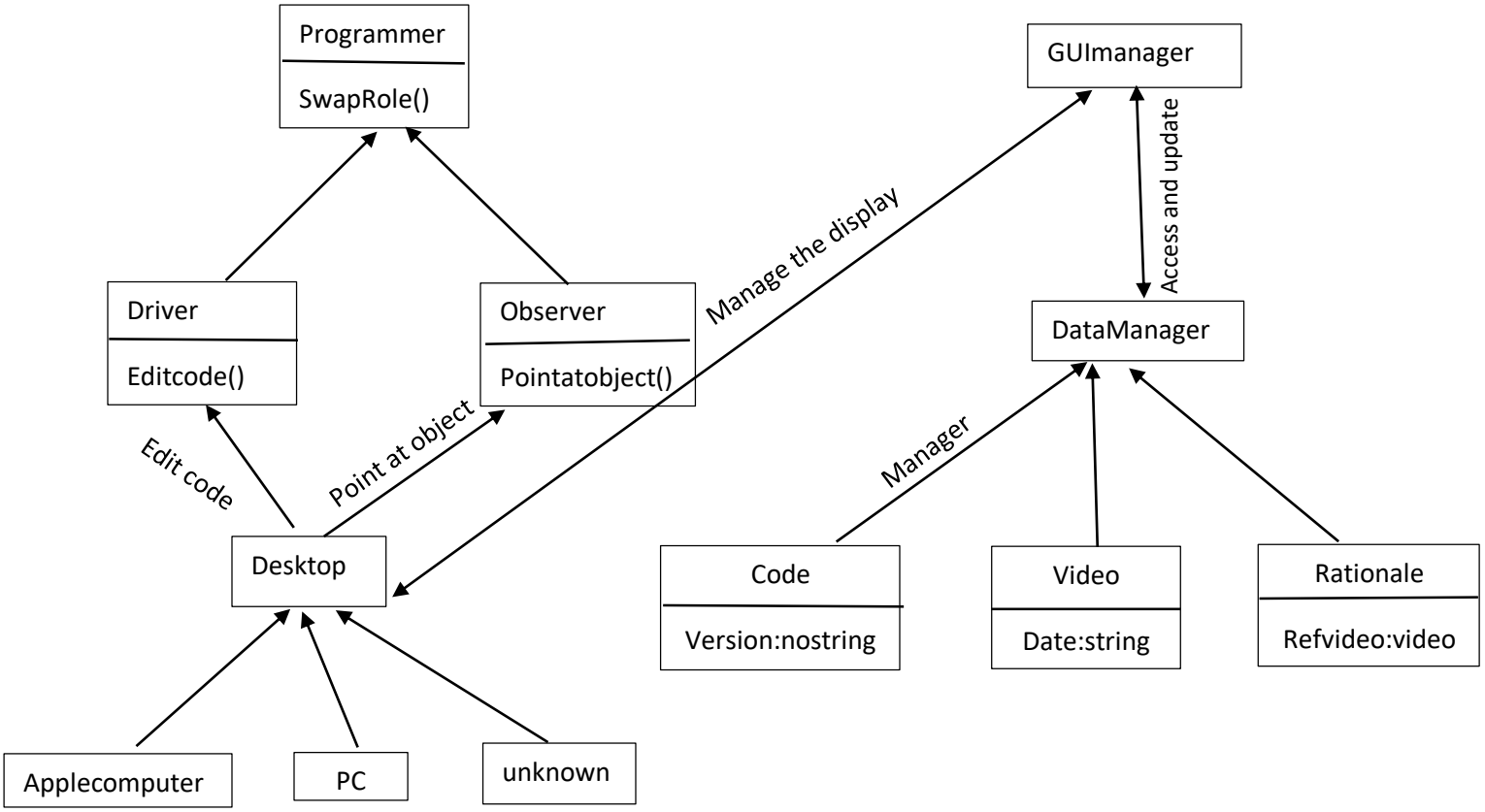
Example 1: "**Portability-** the system should be portable" is a NFR. This NFR may lead to a constraint on the programming language used for the implementation of the system (e.g., the programming language Java (rather than C and C++) might be preferred in order to meet this NFR).

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.

Propose a set of classes that could be used in your system and present them in a class diagram

Answer is on next page….

Class Diagram



The sample class diagram above captures most of the classes and relationships. I have inserted an UnknownDesktop class for future extension of RPP. Obviously when I designed this class diagram I already have the AbstractaFactory design pattern in my mind.

In your class diagram you don't need to mention attributes and operations except for those really important ones. Some of the

operations can be extracted from the use case diagram (Q11a)). For instance, you can probably spot very quickly that, under the Driver class, I should have the issueBackup() operation. Also, under the Programmer class, I should have the viewDesktop(), chatOnVideo(), and recordRationale() operations. Maybe we should have Session as a class as well and associated with DataManager? I left it out because it is not explicitly mentioned in the description. Note that since AppleComputer and PC are not explicitly mentioned in the description, if you don't have any subclasses under Desktop it is fine as well.

A second version is to have the Programmer class associated with the Desktop class instead. However, in this second version we cannot label the association whether it is 'edit' or 'point at objects'.

The GUIManager class and the DataManager class (both can be interfaces) are inserted to help manage the display and the access of data. The model-view-controller (MVC) software architecture pattern is adopted here: DataManager takes care of the 'model', the GUIManager, which interacts with DataManager and Desktop, takes care of the 'view', and the operations carried out by the two programmer compose the 'controller' part.

☆ **THE END** ☆