Name: Syed Muhammad Salman Khan

Id:13662

Subject:Software Design and Architecture

Submitted to: Mam Aasma Khan

Assignment:Final Term

Date 23/06/2020

Q.1:

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

Ans: Software Architecture:. The fundamental organization of a system embodied in its components, their relationships to each other, and to the environments, and the principles guiding its design and evolution.

Software architecture design important:

- A poor design may result in a deficient product that does not meet system requirements,It is not adaptive to future requirement changes,It is not reusable,

It exhibits unpredictable behaviour, or It performs badly.

b) Explain any four tasks of architect.

Ans: Tasks:

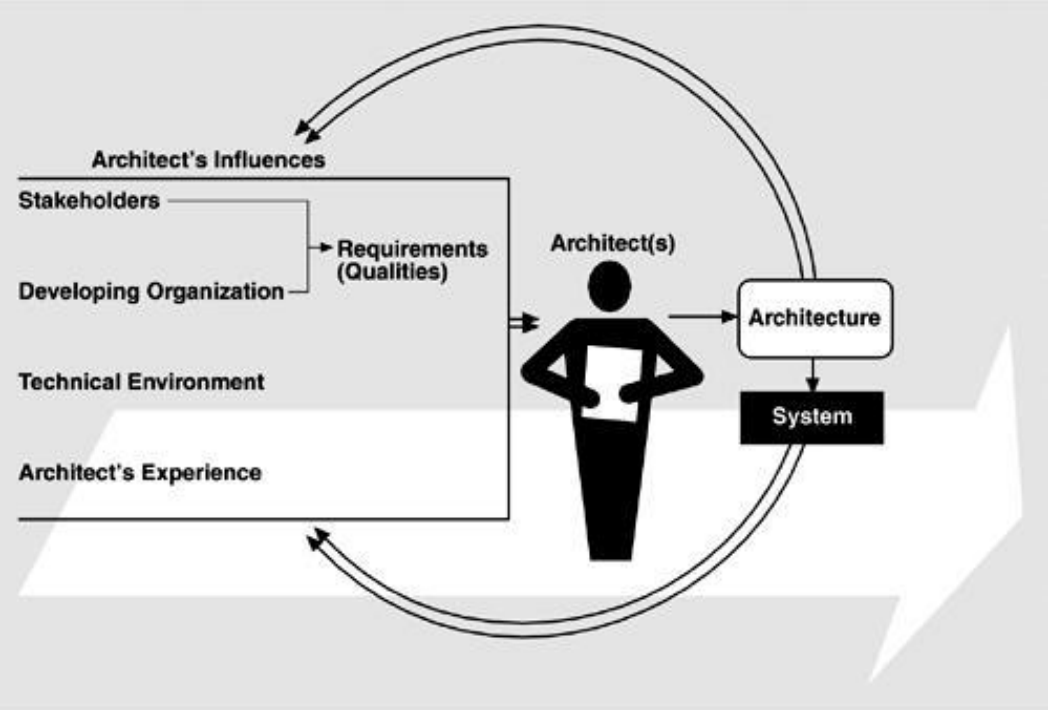
1) Perform static partition and decomposition of a system into subsystems and communications among subsystems.
2) Establish dynamic control relationships among different subsystems in terms of data flow, control flow orchestration, or message dispatching.
3) Consider and evaluate alternative architecture styles that suit the problem domain at hand.
4) Perform trade off analysis on quality attributes and other non functional requirements during the selection of architecture styles.

Q.2: Explain Architecture Business Cycle (ABC) in detail with figure.

Ans: Architecture Business Cycle (ABC):

Software architecture is a result of technical, business and social influences. These are in turn affected by the software architecture itself. This cycle of influences from the environment to the architecture and back to the environment is called the Architecture Business Cycle (ABC). 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 be get an architecture, which be gets 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:
   a. **Case studies:** Case studies of successful architectures crafted to satisfy demanding requirements, so as to help set the technical playing field of the day.
   b. **Methods:** Methods to assess an architecture before any system is built from it, so as to mitigate the risks associated with launching unprecedented designs.
   c. **Techniques:** Techniques for incremental architecture-based development, so as to uncover design flaws before it is too late to correct them.

Q.3: Explain ABC Activities?

Ans: ABC Activities:

1) **Creating the business case for the system**

- Why we need a new system, what will be its cost?
- Time to market, integration with existing systems?

2) **Understanding the Requirements**
   - Various approaches for requirements elicitation i.e. , object-oriented approach, prototyping etc.
   - The desired qualities of a system shape the architecture decisions- architecture defines the trade-off among requirements
3) **Creating / selecting the architecture**
4) **Communicating the architecture**
   - Inform all stakeholders (i.e.., developers, testers, managers, etc.)
   - Architecture's documentation should be unambiguous
5) **Analysing or evaluating the architecture**
   - Evaluate candidate designs
   - Architecture maps the stakeholders' requirements/ needs
6) **Implementation based on architecture .**
7) **Ensuring conformance to an architecture.**

Question No 04: (20)

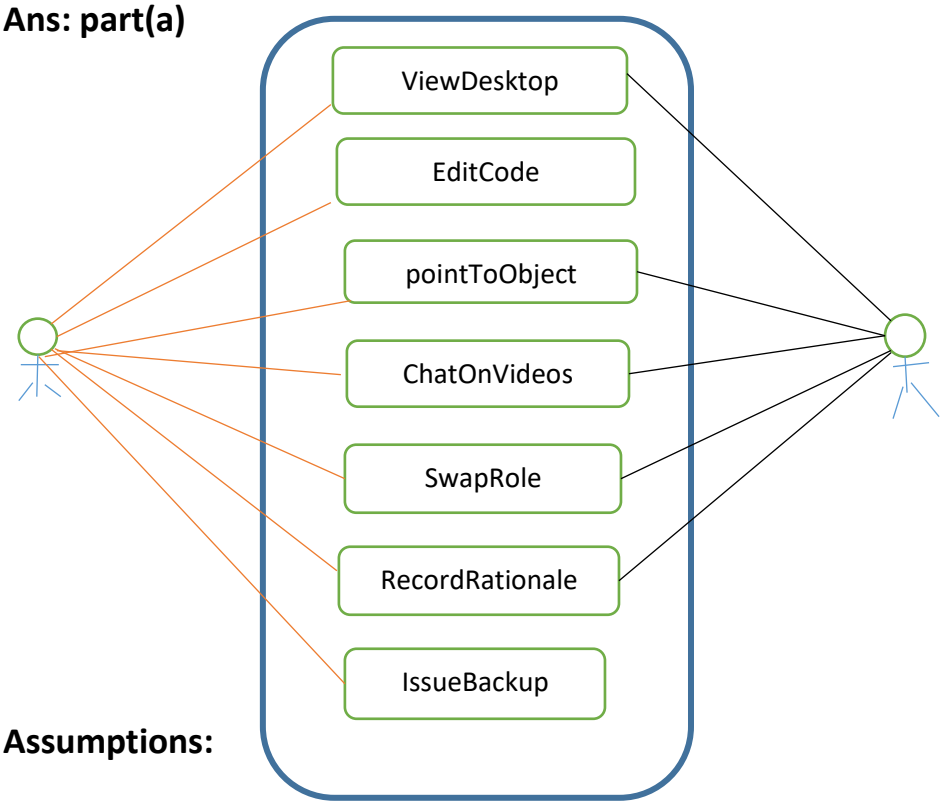
Pair programming is an agile software development technique in which two programmers work together at one work station. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer. The two 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

Ans: part(a)



Assumptions:

* when the Driver edits code, we assume that the Observer can see the changes in real

time 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.

Q no 4(b)

- Ease of use - the front-end interface must be simple and easy to use.
- Security - the backup code should be kept securely and be protected from unauthorized

access.

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.

Q no 4(c)

Ease of Use:

Ease of use always ranks highly when it comes to what customers want from a product or service. Whether it's something they intend to use in their personal lives, like a mobile phone, or their work lives, like office equipment or business software, simplicity is crucial when attempting to make your product or service stand out to potential customers.

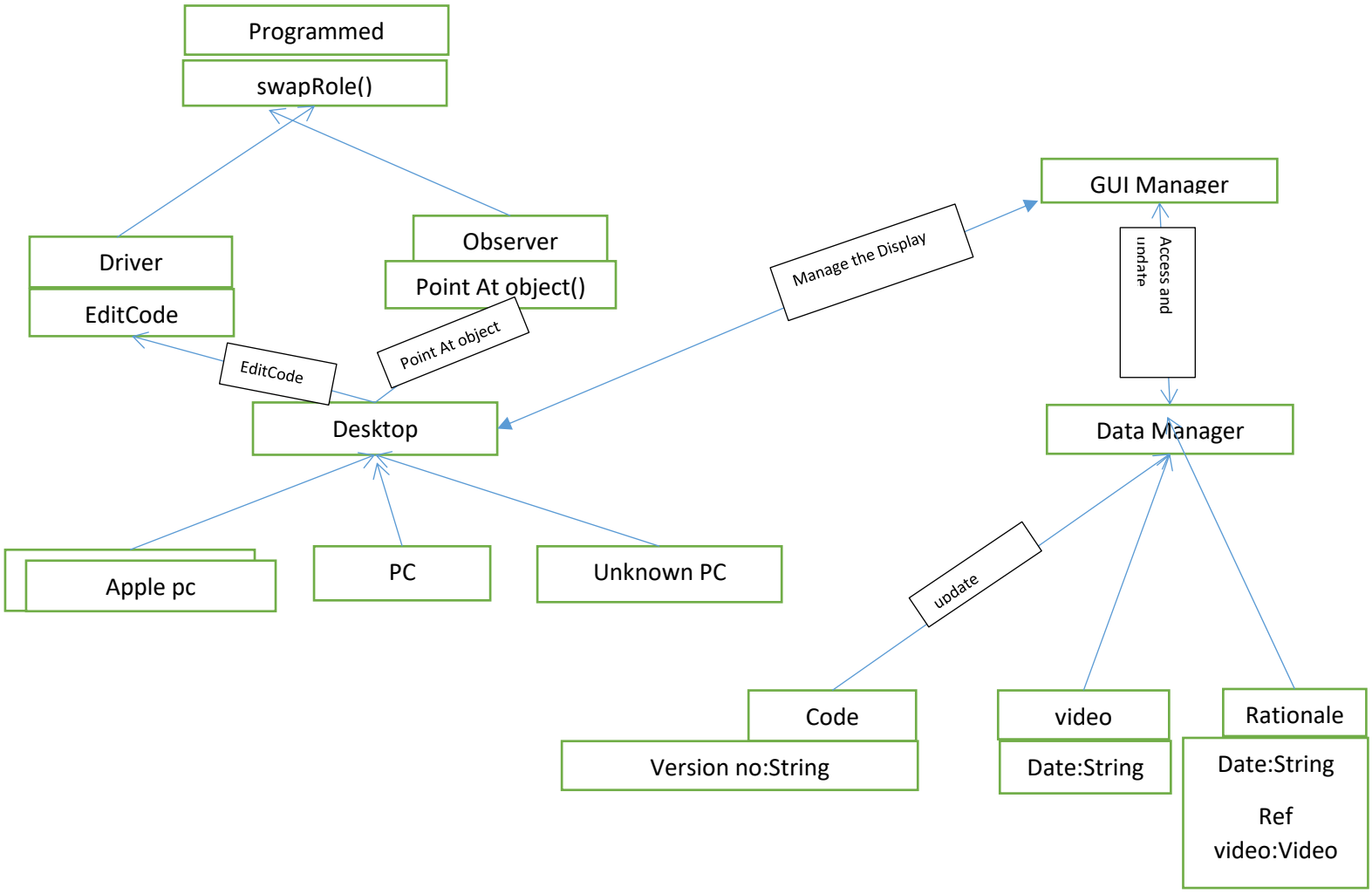
Think of Apple products. Our trusty iPhones, iPads and MacBooks are as popular as ever, and, arguably, that may not necessarily be because they are the best gadgets on the market. Instead, Apple has always made its products easy to use, with simple buttons, accessible screens and clear set up instructions, meaning they have appealed to a wide range of customers over the years.

Availability: How often does the system experience critical failures? and how much time is it available to users against downtimes?

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)

Security: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.

Question no 4(c)



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 Abstract a Factory design pattern in my mind.In our class diagram we 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. 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.