Final-Term – Semester Assignment

Software Design & Architecture

Submitted By: Mudassir Ahmad Khan
ID# 14086
BSSE (6th Semester)

Submitted To: Mrs. Aasma Khan
(Lecturure)

Date: 23/06/2020

Question : 01

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

Ans:
The software architecture of a system depicts the system's organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions. In other words, the software architecture provides a sturdy foundation on which software can be built.

A series of architecture decisions and trade-offs impact quality, performance, maintainability, and overall success of the system. Failing to consider common problems and long-term consequences can put your system at risk.

There are multiple high-level architecture patterns and principles commonly used in modern systems. These are often referred to as architectural styles. The architecture of a software system is rarely limited to a single architectural style. Instead, a combination of styles often make up the complete system.

there are fundamentally three reasons for software architecture's importantance.

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.

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.

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.

Part(b) Explain any four tasks of architect.

1. Software Architect job title

A great job title typically includes a general term, level of experience and any special requirements. The general term will optimize your job title to show up in a general search for jobs of the same nature. The level of experience will help you attract the most qualified applicants by outlining the amount of responsibility and prior knowledge required. And if your position is specialized, consider including the specialization in the job title as well. But avoid using internal titles, abbreviations or acronyms to make sure people understand what your job posting is before clicking.

Examples of Software Architect job titles

- Software Architect
- Senior Software Architect

- Software Architect (with Java proficiency)
- Technical Architect

2. Software Architect job summary

A great job description starts with a compelling summary of the position and its role within your company. Your summary should provide an overview of your company and expectations for the position. Outline the types of activities and responsibilities required for the job so job seekers can determine if they are qualified, or if the job is a good fit.

Example of a Software Architect job summary

Our HR applications firm is currently seeking an experienced Software Architect. The successful candidate will be responsible for designing, developing and implementing software solutions to address complex business issues, and providing technical leadership within the IT department. The ideal applicant will able to thrive in a highly collaborative workplace and actively engage in the development process. This is an excellent career opportunity for a professional with an impressive architectural design background and excellent interpersonal skills.

3. Software Architect responsibilities and duties

The responsibilities and duties section is the most important part of the job description. Here you should outline the functions this position will perform on a regular basis, how the job functions within the organization and who the job reports to.

Examples of Software Architect responsibilities

- Design, develop and execute software solutions to address business issues
- Provide architectural blueprints and technical leadership to our IT team
- Evaluate and recommend tools, technologies and processes to ensure the highest quality product platform
- Collaborate with peer organizations, quality assurance and end users to produce cutting-edge software solutions
- Interpret business requirements to articulate the business needs to be addressed
- Troubleshoot code level problems quickly and efficiently

4. Software Architect qualifications and skills

Next, outline the required and preferred skills for your position. This may include education, previous job experience, certifications and technical skills. You may also include soft skills and personality traits that you envision for a successful hire. While it may be tempting to include a long list of skills and requirements, including too many could dissuade qualified candidates from applying. Keep your list of qualifications concise, but provide enough detail with relevant keywords and terms.

Examples of Software Architect skills

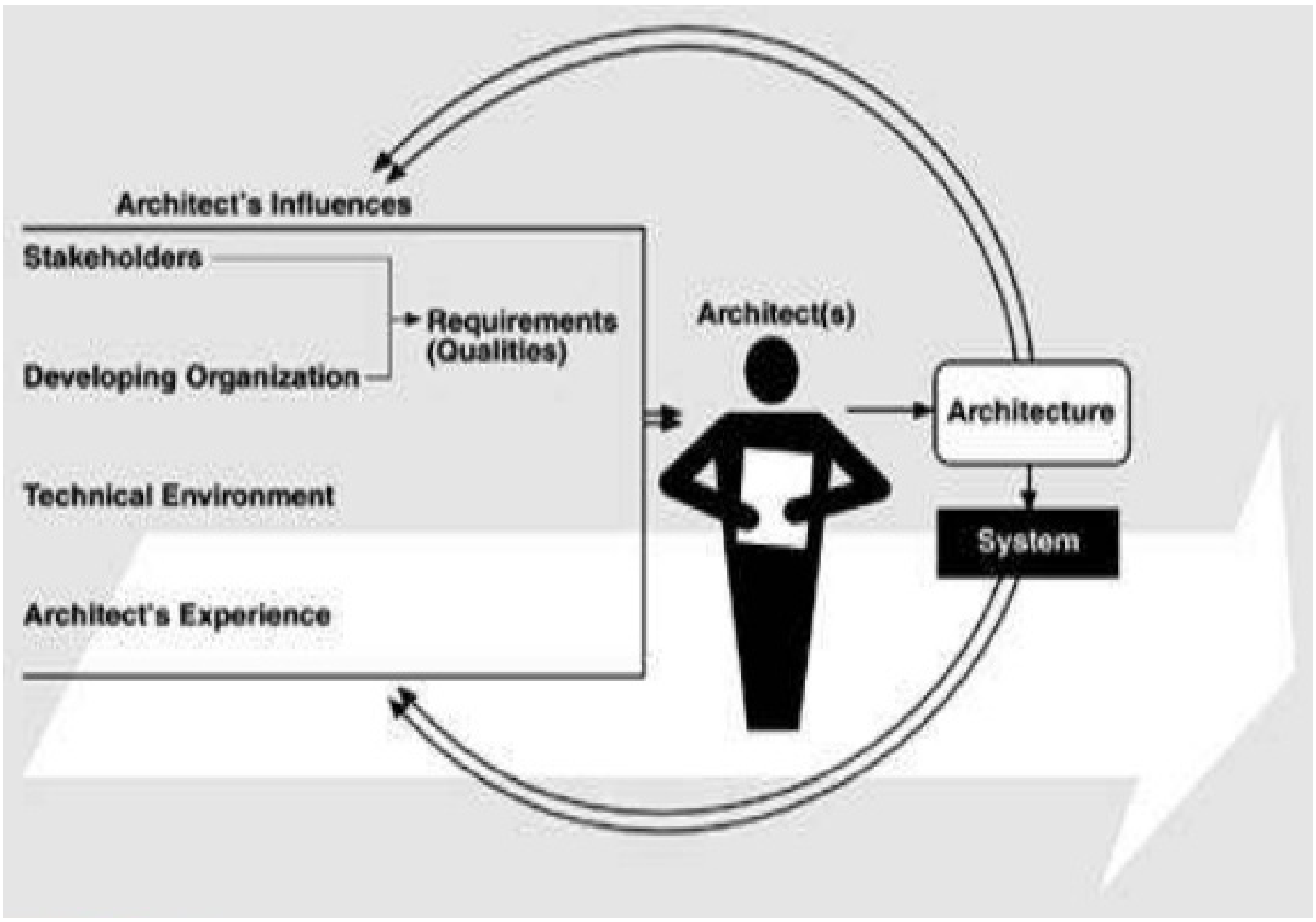
- Master’s degree in Computer Science or Computer Engineering
- 5+ years’ experience designing and building software applications
- Proficiency with Java
- Experience working on complex software projects
- Knowledge of C++/object oriented programs, SQL, web application development, security and open source technologies

Question : 02

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. 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.

How the ABC Works :

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.

These and other feedback mechanisms form what we call the ABC, illustrated in Figure , which depicts the influences of the culture and business of the development organization on the software architecture. That architecture is, in turn, a primary determinant of the properties of the developed system or systems. But the ABC is also based on a recognition that shrewd organizations can take advantage of the organizational and experiential effects of developing an architecture and can use those effects to position their business strategically for future projects.

Question : 03

Explain ABC Activities?

Ans:

1. Creating the Business Case for the System

Creating a business case is broader than simply assessing the market need for a system. It is an important step in creating and constraining any future requirements. How much should the product cost? What is its targeted market? What is its targeted time to market? Will it need to interface with other systems? Are there system limitations that it must work within?

These are all questions that must involve the system's architects. They cannot be decided solely by an architect, but if an architect is not consulted in the creation of the business case, it may be impossible to achieve the business goals.

2. Understanding the Requirements

There are a variety of techniques for eliciting requirements from the stakeholders. For example, object-oriented analysis uses scenarios, or "use cases" to embody requirements. Safety-critical systems use more rigorous approaches, such as finite-state-machine models or formal specification languages.

One fundamental decision with respect to the system being built is the extent to which it is a variation on other systems that have been constructed. Since it is a rare system these days that is not similar to other systems, requirements elicitation techniques extensively involve understanding these prior systems' characteristics.

Another technique that helps us understand requirements is the creation of prototypes. Prototypes may help to model desired behavior, design the user interface, or analyze resource utilization. This helps to make the system "real" in the eyes of its stakeholders and can quickly catalyze decisions on the system's design and the design of its user interface.

3. Creating or Selecting the Architecture

In the landmark book The Mythical Man-Month, Fred Brooks argues forcefully and eloquently that conceptual integrity is the key to sound system design and that conceptual integrity can only be had by a small number of minds coming together to design the system's architecture.

4. Communicating the Architecture

For the architecture to be effective as the backbone of the project's design, it must be communicated clearly and unambiguously to all of the stakeholders. Developers must understand the work assignments it requires of them, testers must understand the task structure it imposes on them, management must understand the scheduling implications it suggests, and so forth. Toward this end, the architecture's documentation should be informative, unambiguous, and readable by many people with varied backgrounds.

5. Analyzing or Evaluating the Architecture

In any design process there will be multiple candidate designs considered. Some will be rejected immediately. Others will contend for primacy. Choosing among these competing designs in a rational way is one of the architect's greatest challenges.

Evaluating an architecture for the qualities that it supports is essential to ensuring that the system constructed from that architecture satisfies its stakeholders' needs. Becoming more widespread are analysis techniques to evaluate the quality attributes that an architecture imparts to a system. Scenario-based techniques provide one of the most general and effective approaches for evaluating an architecture.

6. Implementing Based on the Architecture

This activity is concerned with keeping the developers faithful to the structures and interaction protocols constrained by the architecture. Having an explicit and well-communicated architecture is the first step toward ensuring architectural conformance. Having an environment or infrastructure that actively assists developers in creating and maintaining the architecture (as opposed to just the code) is better.

7. Ensuring Conformance to an Architecture

Finally, when an architecture is created and used, it goes into a maintenance phase. Constant vigilance is required to ensure that the actual architecture and its representation remain faithful to each other during this phase. Although work in this area is comparatively immature, there has been intense activity in recent years.

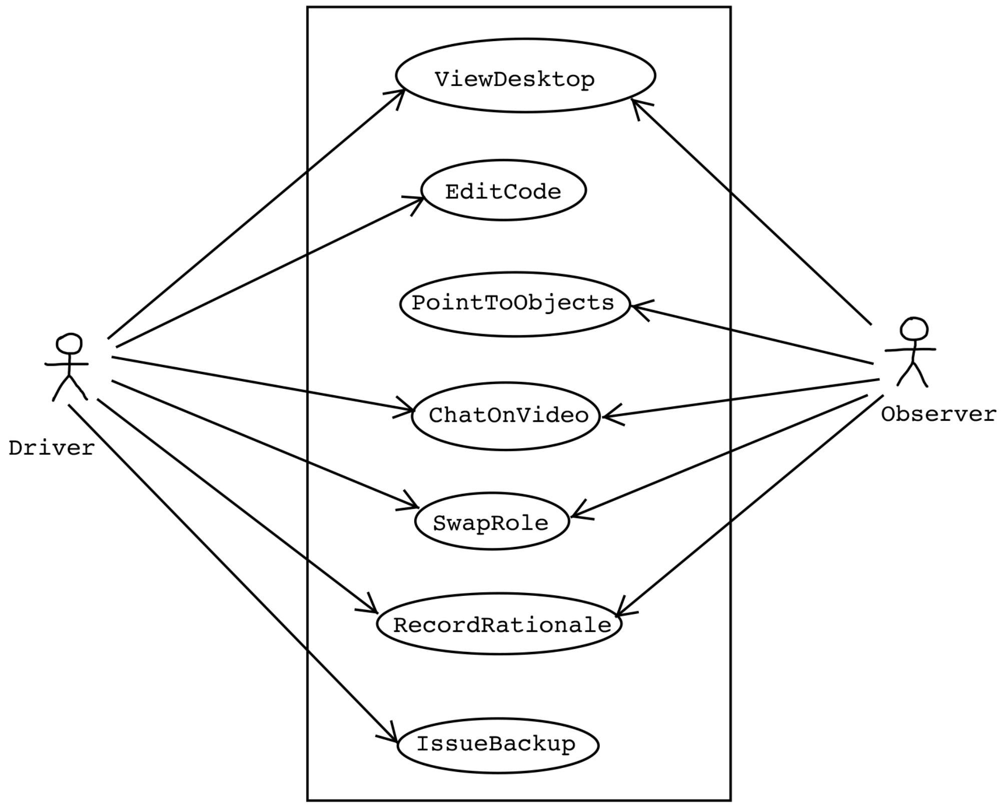
Question No 04:

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.

a) Draw a use case diagram to show all the functionality of the system.

Ans:



b) Describe in detail four non-functional requirements for the system.

Ans:

- **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 whatcomputer and operating system used by the programmers.

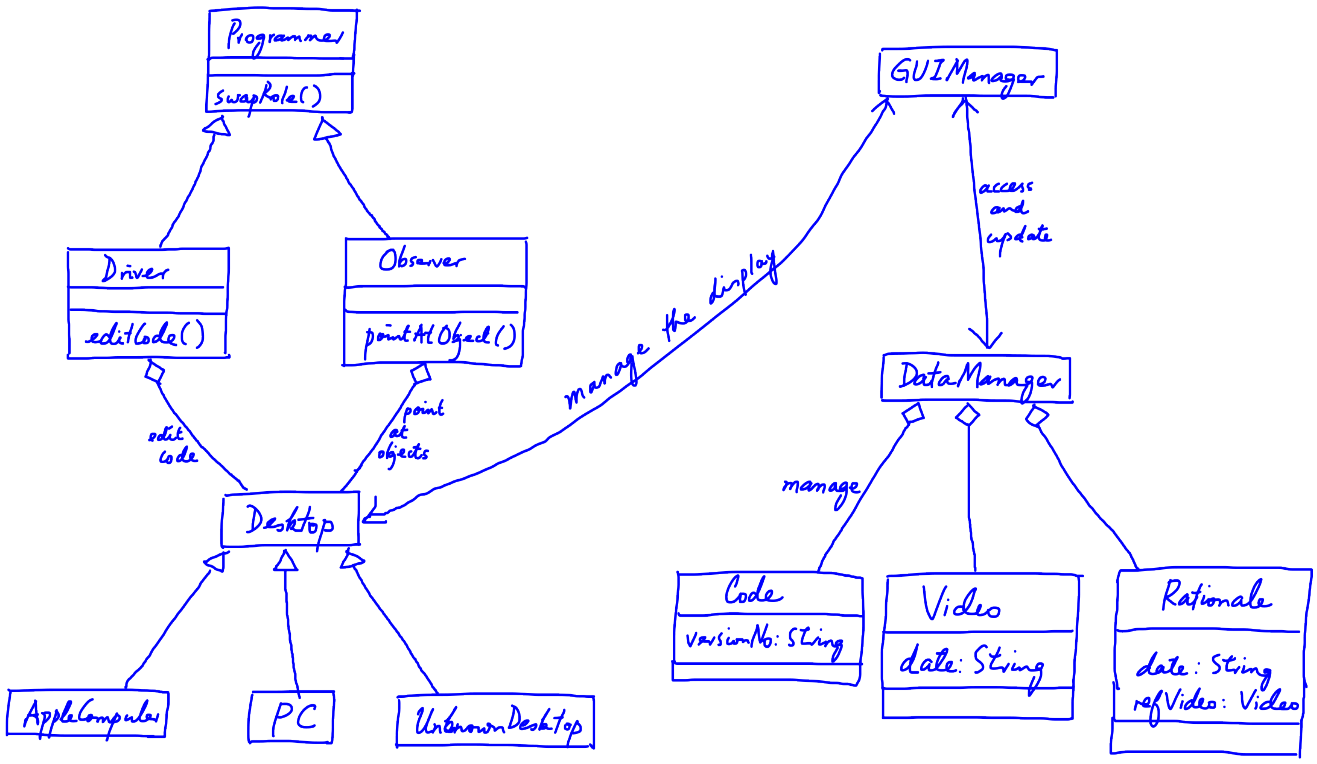
c) Give a prioritized list of design constraints for the system and justify your list and the ordering.

Ans:

The system should be portable; 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 - 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 se rver behind firewall.

d) Describe in detail four non-functional requirements for the system.

Ans:



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.

Than You!