Project Assignment 3

Rui Zhu, Lingjing Huang, Entisar Seedi M Alshammry

1. What is software architecture? Why is software architecture needed? (10 points)

1) An architecture is the set of significant decisions about **the organization of a software system**, **the selection of the structural elements** and their **interfaces by which the system is composed**, together with their **behavior as specified in the collaborations** among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization these elements and their interfaces, their collaborations, and their composition.

2) We need the software architecture because:

1. It increases the communication among stakeholders. Whether the stakeholder is user, coder, project manager or tester, the architecture will provide a common language where they can resolve these concerns.
2. Early design decisions. When you take an early decision that will make a big difference on the rest of the project.in addition, going furthermore make it very difficult to change and will affect the cost of the project.
3. Software architecture is a transferable and reusable across systems.

During the earlier face of designing the software, Appling the re-use is helpful and not only for code but also for requirements and decisions as well.

1. Summarize each of the common architecture styles along with its description as given in case study. Point out some of the shortcomings of each architectural approach. (40 points). At least half a page per architecture.
2. **Model-View Controller**

Description:

Model–view–controller is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts: Model, View, and Controller. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Model is a synonym for the domain layer of objects. The view is a synonym for UI objects, such as windows, Web pages, applets, and reports.

The Model-View Separation Principle has at least two parts:

1. Do not connect or couple non-UI objects directly to UI objects.
2. Do not put application logic in the UI object methods. UI objects should only initialize UI elements, receive UI events, and delegate requests for application logic on to the non-UI object.

Shortcoming:

1. Increase the system architecture and implementation complexity.
2. The view and the controller of the close connection between.
3. In view of low-efficiency model of data access.
4. At present, general advanced interface tools or constructor does not support MVC mode.

**2) Data flows**

Description:

In data flow architecture, the whole software system is a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other. In this approach, the data enters the system and then flows through the modules one at a time until they are assigned to some final destination.

The connections between the components or modules may be implemented as I/O stream, I/O buffers, piped, or other types of connections. The data can be flown in the graph topology with cycles, in a linear structure without cycles, or in a tree-type structure.

The main objective of this approach is to achieve the qualities of reuse and modifiability. It is suitable for applications that involve a well-defined series of independent data transformations or computations on orderly defined input and output such as compilers and business data processing applications. There are two types of execution sequences between modules: Batch sequential and Pipe and filter or non-sequential pipeline mode.

In the batch, sequential style, processing steps, or components, are independent programs, and the assumption is that each step runs to completion before the next step starts. Each batch of data is transmitted between the steps. The typical application for this style is classical data processing.

Shortcoming:

1. Not suitable for dynamic interactions.
2. A low common denominator is needed for the transmission of data in ASCII formats.
3. Overhead of data transformation between filters.
4. Does not provide a way for filters to cooperatively interact to solve a problem.
5. Difficult to configure this architecture dynamically.
6. **Independent components**

Description:

Independent component architectures consist of a number of independent processes or objects that communicate through messages.

All of these architectures have the goal of achieving modifiability by decoupling various portions of the computations. They send data to each other but typically do not directly control each other.

These other components may register an interest in this class of data. If they do so, when the data appears, they are invoked and receive the data.

Typically, event systems make use of a message manager that manages communication among the components, invoking a component when a message arrives for it. In this publish/subscribe paradigm, a message manager may or may not control the components to which it forwards messages.

Components register types of information that they are willing to provide and that they wish to receive.

They then publish information by sending it to the message manager, which forwards the message, or in some cases an object reference, to all interested participants.

There are different types :

1. Client-server
2. Parallel communicating processes
3. Event systems
4. Service Oriented Architecture

Shortcoming:

When there are frequent simultaneous requests, the communications may severely get overloaded, forming traffic congestion.

1. **Virtual machines**

Description:

Virtual Machine architectures have the goal of achieving the quality of portability.

Virtual machines are software styles that simulate some functionality that is not native to the hardware and/or software on which it is implemented.

It can allow one to simulate (and test) platforms that have not yet been built, and it can simulate "disaster'' modes that would be too complex, costly, or dangerous to test with the real system.

Common examples of virtual machines are interpreters, rule-based systems, syntactic shells, and command language processors.

Interpretation of a particular module via a Virtual Machine may be seen as follows:

1. The interpretation engine selects an instruction from the module being interpreted;
2. Based on the instruction, the engine updates the virtual machine internal state;
3. The process above is repeated;

Shortcoming:

There is a performance cost because of the additional computation involved in the execution.

1. **Layered architectures**

Description:

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request.

One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with the logic that pertains to that layer. This type of component classification makes it easy to build effective roles and responsibility models into your architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

Shortcoming:

1. There might be a negative impact on the performance as we have the extra overhead of passing through layers instead of calling a component directly.
2. Development of user-intensive applications can sometimes take longer if the layering prevents the use of user interface components that directly interact with the database.
3. The use of layers helps to control and encapsulate the complexity of large applications but adds complexity to simple applications.
4. Changes to lower level interfaces tend to percolate to higher levels, especially if the relaxed layered approach is used.
5. **Repository**

Description:

In Repository Architecture Style, the data store is passive and the clients of the data store are active, which control the logic flow. The participating components check the data-store for changes.

A client sends a request to the system to perform actions. The computational processes are independent and triggered by incoming requests. If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is a traditional database or repository architecture, or passive repository. This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers, and CASE environments.

System has 2 components:

1. Central data unit component.
2. Set of relatively dependent components.

Central data unit called the repository shares information with all the other units. There are differences in the information exchange patterns. Thus there are 2 main control strategies to deal with these information exchange patterns.

Shortcoming:

1. High dependency between data structure of data store and its agents.

2. Changes in data structure highly affect the clients.

3. Evolution of data is difficult and expensive.

4. Cost of moving data on the network for distributed data.