CS 3354 Reading Activity #1

Written by Zach Leach, February 12, 2024

1 What is Software Architectural Design, and why is it important?

Software architectural design is the process of defining a structured solution that meets both technical and operational requirements. It involves making important decisions about the overall structure of the software, such as the relationships between components, data flow patterns, and the mechanism for communication between different parts of the system. A good software architecture helps define performance, quality, scalability, maintainability, manageability, and usability. [1]

It is important because it provides a solid foundation for the software project, ensuring that the software is flexible, extensible, and can evolve as new requirements emerge. Additionally, software architecture introduces constraints on implementation and restricts design choices, reducing the complexity of a software system. [2]

2 Provide the main characteristics of, and find an example for, each type of system:

a. Interactive system

b. Event-Driven system

c. Transformational system

d. Object-Persistence/Database system

2.1 Interactive system

The main characteristics of an interactive system include interleaved input and output, significant human-computer interaction, and real-time feedback to the user based on their actions. Examples of interactive systems range from graphical user interfaces like Macintosh or Windows operating systems, web browsers, and Integrated Development Environments (IDEs) to specific applications such as word processors, spreadsheet software, and games. [1] [2] [3]

2.2 Event-Driven system

The main characteristics of an event-driven system are loose coupling, asynchronous communication, reactivity to events. An example of an event-driven system is an e-commerce website where various events occur, such as items being added to a shopping cart, a purchase being made, or a user posting a review. Each of these actions generates an event that the system can respond to. For instance, when a customer places an order, the event is published to an event broker, which then informs the inventory service to update stock levels, the payment service to process the payment, and the shipping service to handle the delivery. [1] [2]

2.3 Transformational system

The main characteristics of a transformational system are input-output transformation, batch processing, deterministic processing, and limited user interaction. An example of a transformational system is a compiler. A compiler takes source code written in a programming language (the input) and transforms it into machine code or bytecode (the output), which can then be executed by a computer or a virtual machine. The process is deterministic, and the interaction with the user is limited to providing the source code and receiving the compiled output. [1] [2]

2.4 Object-Persistence/Database system

The main characteristics of a object-persistence/database system are object lifetimes, storage and retrieval, reliability, and cross-platform access. An example of an object-persistence/database system is an object-relational database management system (ORDBMS) that stores and manages persistent data in the form of objects. For instance, a system using ORDBMS can store customer information as persistent objects, ensuring their durability and accessibility across different applications and processes [1] [2] [3]

3 Why perform Custom Architectural Design

It's common for a company to have a set of software that they distribute to customers. Custom architectural design allows developers to not have to write new software from scratch everytime.

4 What is a package diagram

A package diagram in software engineering is a type of UML (Unified Modeling Language) diagram that depicts the organization and arrangement of

various model elements, such as classes, documents, and other packages, in a hierarchical structure. [1]

5 What are software design princples? Explain each briefly. [1] [2] [3]

5.1 Single Responsibility (SRP)

Classes should focus on a single task, and have only one reason to change.

5.2 Open-Closed (OCP)

Software entities should be open for extension but closed for modification.

5.3 Liskov Substitution (LSP)

Objects should be replaceable with instances of their subtypes without altering the correctness of the program.

5.4 Interface Segregation (ISP)

Many client-specific interfaces are better than one general-purpose interface.

5.5 Dependency Inversion (DIP)

High-level modules should not depend on low-level modules, and both should depend on abstractions.

5.6 Don't Repeat Yourself (DRY)

Avoid duplication to reduce maintenance and potential inconsistencies.

5.7 Keep It Simple, Stupid (KISS)

Avoid unnecessary complexity by keeping code and design simple.

5.8 You Aren't Going To Need It (YAGNI)

Deliver features that are needed instead of wasting time over-engineering, premature optimization, or anticipating future requirements.

5.9 Composition Over Inheritance

Build larger objects from smaller object components, encouraging flexibility and loose coupling instead of large complex abstraction hierarchieshierarchies.

6 Which agile principles should be applied during Architectural Design?

Teams that code the system design the system, pick the simplest architecture that can possibly work, system architecture role collaboration. Promoting flexibility, simplicity, and collaboration, which are essential elements for developing adaptable and effective system architectures. [1] [2] [3]

7 What is a MVC pattern?

The Model-View-Controller (MVC) is an architectural design pattern that organizes an application's logic into distinct layers, each of which carries out a specific set of tasks. The layers also interact with each other to ensure that the application's functionality is delivered in a coordinated and efficient manner. Model manages data and business logic, view handles layout and display, controller routes commands to the model and view. [1] [2]