SOFTWARE DESIGN AND ARCHITECTURE
BSE: 5th Semester

18/9/24

(*) **Design** (finer details) Vs **Architecture**

Design	Architecture
-> Detailed structure	-> High-level structure.
-> Functional	-> Non-Functional
-> Algorithms, structure define.	-> Specifies design, styles and architecture.
-> Class diagram, OOP design/implementation	-> Concerned with global decisions.

(*) **Relationship**

-> Software architecture comes before software design.

(*) **Modules:**

-> software design

-> Design phase with OOP

(*) Requirements may be correct but designed may be flawed.

(*) Design: First step of development phase.

(*) Software Development Activities:

→ Requirements Elicitation, Analysis, Design, Implementation, Testing.

(*) SDLC (Software Design Life cycle).

(*) Design Process Activities:

→ Architectural design, Abstract specification, Interface Design componant design, Algorithm design, Data structure design.

(*) Smart Home Automation:

→ Lighting control sub-systems.

→ Security and surveillance sub-system

→ Climate Control sub-system.

ervices and Constraints)

Geo-Fencing (Defining boundary and if voilates boundary, it will be alarming etc....)

→ Home Automation Hub (Central control)

(*) Mobile Interface Design

→ Control Panel

→ Settings.

* : Software Architecture Foundations, Theory And Practice By Eric Dash OFY

* Levels of Software Design:
  → Architectural Design (High Level)
  → Design models.
* Quiz in Next class.

23/9/24

* Evaluate on high-level first and gradually more towards low level.

Quiz Option
* ① D ② A ③ A ④ A/D (60%) ⑤ A
  ⑥ D ⑦ D ⑧ A ⑨ D ⑩ D

* Key Aspects of Software Architecture: (Reusable)
  (i) Components (ii) Connectors (iii) Patterns (rules)
  (iv) Principles (v) Non-functional requirements
* Key Aspects of Software Design:
  (i) Class Design (ii) Data structure.
  (iii) Algorithms (iv) Design Patterns
* Common Architecture Patterns:
  (i) Layered (N-tier) Architecture
  (ii) Microservices (Purely Independent Services) (iii) Event-Driven
  (iv) Client-service (v) Service Oriented (Communication through main bus) (Common)
* Design Pattern (Low-Level of abstraction)

⊛ Structural patterns, Behaviour pattern.

⊛ Significance of design and software:

(i) Poor design not meet requirements

(ii) Adaptive future changes

(iii) Reusable.

⊛ Guidelines:

(i) Right architectural styles.

(ii) Functional and Non-Functional requirements gathering.

⊛ Software Quality Triangle:

(i) Operational

(ii) Transition (From one stage to other)

(iii) Revisional

· Modularity (Distributing modules)

: Integrity (one doesn't effect others)

25/9/24

Project

⊛ Online election system:

→ Requirements (User Stories, functional, non-functional).

: 3-4 (members) → Models (Use-Case etc.....).

→ Monday till Vision Document

:(Functional, Non Functional, User stories)

: DFD (Context Diagram) : Register candidate, View list of areas, Document verification : Stackholders list, entities : Admin Dashboard (one-time vote, verification)

⊙ CRC:

: Responsibilities (stock of items, price)

→ Class Responsibilities collaborators.

: Collaborators (customer, order_Line).

⊛ Requirements model into design model.

→ Class Diagram / Design

→ Architectural Design

→ Interface Design

⊛ Activity Diagram in classes

⊛ Sequence Diagram, Behavioral Element (State Machine, Design).

⊛ State changes into function call.

⊛ Design Concepts:

① Abstraction:

→ Hide level of data

→ Levels of abstraction can be posed.

→ Types:

(i) Procedural Abstraction

without exposing the internal workings of structure.

: Presentation

(ii) Data Abstraction:

→ Hiding Detail

② Architecture:

→ structure of organizations

→ structural Model:

Contains Diagrams etc...

→ Framework Model:

MVC (controller), Template
(Model) (View)

→ Dynamic Model:

(Flow of control between components)

→ Process Model:

Sequence of Activities.
Activity Diagram.

→ Functional Model:

(Functionalities) (Attendence, Registration etc.)
(Functional Requirements)

* ADLs (Architectural Description language).

⊙ Patterns:

→ Architecture pattern (organize and structure an entire system)

→ Design Pattern { Functional requirements etc.... }.

⊙ Separation of Concerns:

→ subdivided into components
→ Specific aspect such as feature, functionality etc.....
→ Goal of SoC to make easier to develop, test, maintain, reuse.
→ Example: Web Application:
(i) User Interface (UI)
(ii) Business Logic
(iii) Data management.

⊙ Modularity:

→ Breaking down into independent, interchangeable modules.
→ SoC is used.

⊙ Information Hiding:

→ Encapsulation, Abstraction
Interfaces.

⊙ Functional Independence:

→ Two criteria:
(i) Cohesion (strength of module)
(ii) Coupling (relative interdependence)

→ High · Ideal, Low · Ideal
(little interaction) (Error propagate through a system)

30/9/24

⊙ **Refinement:**

→ Abstraction complimentary concept (refinement).

⊙ **Aspects:**

→ Implementation of cross-cutting process. (Functionality or behaviour that affects)

→ Logging (Keep record) (Not dedicated or separate module but using different aspects).

→ Security (Layer of protection).

→ Transaction Management · (Atomicity) (Updating in single transaction) (Account)

→ AOP (Aspect oriented Programming).

→ In Java (AspectJ or AOP).

⊙ **Refactoring:**

→ Changing in software without changing external behaviour.

→ Improve Integrity, Extendability etc...

→ Refactoring process using Eclipse with a Java. (Extract Method) (Technique) refactoring

Install it (May be in Quiz or Exam)

⊙ **Move:**

→ Move the type to a different package or source folder.

→ Packages must not be miss to properly work code.

⊙ **Extract Method:**

→ Relative values will be extracted together.

⊙ **Extract Variable:**

→ Make variable / functions separately and we it in better way.

⊙ **InLine Method:**

→ Not make more methods or functions, if not needed: Make one function or method.

⊙ **Pull-up and push-up:**

→ Method of refactoring used to get functionalities of superclass and sub-classes.

⊙ **Extract Interface:**

→ If you want to create interface from class, then use "Extract Interface" classes.

⊙ Extract Superclass

⊙ **Delete:**

If you want to remove type.

⊙ **Replace Temp with Query:**

→ Make separate Method.
Minimize local variables.

⊛ Bad refactor Example (next class topic)

2/10/24

⊛ Do ellipse downloading and perform.

⊛ **Object oriented Design:**

→ Inheritance, Polymorphism, Abstraction etc...

⊛ Design classes:

Business

→ User interface, Domain classes
(Interaction) (core Business logic)
(Login Form)
(Dash Board)

→ Process Class
(Procedure or Flow of information)
(Order Processes) (Payment Process).

→ Persistent Classes
(Files or Databases) (Customer Repository)

→ System classes
(System-Level) (Interaction with Hardware) (File manager Email Service)

⊙ Dependency Inversion Principle (DIP):

→ High-level modules
(Concrete and handle details).

→ Changing into Low-Coupling.

⊙ Solid Principles:

↗ Liskov

(Single) Inter face Dependancy Principle
(much dedicated

Open Close

(*) Interface Segregation
(Divided into sub-problems).

(*) Benefits
-> Easy to understand
-> Maintable
-> scalability
-> Testing component easy
-> Low Coupling | Loose Coupling.

① Design Test:
-> system testing,
Design Testing.

7/10/24 (SS)

(*) Case-Study:
-> Title -> Introduction
-> Objectives -> Planning
-> Questions:
① Source of water ② Quality
③ Quantity

7/10/24

* (*) Segregate and Interface
Use in Quiz (can be part of exam)

* **Component Diagram:**
→ Visualize structure relationship.
(i) Component (ii) Interface (→ Provided interface, → Required interface (Half cut circle) (—C)
(iii) Dependencies. (Dashed-arrow)
→ Eg: Shopping Cart —C Inventory Service
—C O— (Required) (Provide)

* (*) Create a component diagram for CMS (Do it till Next Lecture).

9/10/24

→ CMS:
→ User Interface
→ Student Management
→ Administration Management
→ Fee Management
→ Library Management
→ Notification
→ Report Generation

→ Security.

⊙ **Collaboration Diagram:** (Flow of messages and interaction)/(structure)

(Weather Forecast system)

→ Key Components:

(i) User (ii) Forecasting System
(iii) Weather Data Provider
(iv) Database (v) Notification service

→ Sequence, Activity, Collaboration have little difference:

(i) Focus/Emphasis (ii) Best for
(iii) Key Elements

⊙ **Deployment Diagram:**

→ Depicts the Hardware components also.

→ Key Elements:

(i) Nodes (① Device Nodes ② Execution Environment)
(ii) Artifacts (① Executable etc.....)

→ **Clarity in Complex System:**

(i) Execution Environment
(ii) Physical Hardware Nodes
(iii) Artifacts

→ Virtualization and Cloud System (Dockers and VM).

→ Admin Panel and Checkout Panel

→ Scalability and Maintenance

* Diagram Content Reading and Activity in Next class on Deployment Diagram. (Graded Activity)

* Component Diagram and Collaboration Diagram.

14/10/24

(*) Class Activity

(*) Single Point Failure

16/10/24

(*) Data Flow architectures:

→ Batch Sequential (Chunks flow)

→ pipes and filers (Processing unit)

→ Event-Driven Architecture. (Data flow in form of event)

→ Stream Processing (Continuous flow of data)

(*) Data Source, Processing Unit, Data Sink.

(*) **Batch Sequential Architecture:**

→ Sequential Processing, Batch oriented, Non-Real Time.

→ Simplicity, State Independence

→ Structure:

(i) Input Data source
(ii) Batch Process 1
(iii) Intermediate Data
(iv) Batch Process 2
(v) Output Data sink

→ Pros of Batch:

(i) Efficiency in Large Scale Processing
(ii) Predictability and Control
(iii) Simplicity
(iv) Fault (Processing) Tolerance
(v) Offline Processing

: Not for Real-Time (Beneficial)

→ Cons:

(i) Latency
(ii) Lack of Real Time Feedback
(iii) Resource overhead
(iv) Inflexibility
(v) Error Handling.

→ University, Banking Transaction, Payroll, End of Day Reporting, Data

2000 sleeping time

Image and Video Processing, Log Analysis, ETL (Extract_Transform_Load) (Hadoop etc.)

⊛ Pipes and Fillers:

→ Filter (Input into Output)

→ Pipes (Flow source).

→ Details:

(i) Data Flow

(ii) Flexibility

(iii) Reusability

→ Pros:

→ Modularity, Testing, Maintability Scalability.

→ Cons:

⊙ Performance overhead, Data Format Dependency, Error Handling, Latency.

→ Use cases:

(i) Data-Processing Pipelines, Compilers, stream Processing systems.

: Runs in Pre-Processing Stages (ii) Audio and video and Text Processing tools.

→ Scenarios.

⊛ Event-Driven Architecture:

→ Based on events

→ Key components:

(i) Event Producers
(ii) Event Consumers
(iii) Event Channels
(iv) Event Brokers | Message Brokers

→ Types:

(i) Simple Event Processing
(ii) Complex Event Processing

→ Pros:

(i) Loose coupling
(ii) Real-Time Processing
(iii) Scalability
(iv) Resilience (Easy to Manage)
(v) Asynchronous (No dependency) communication
(vi) Flexability

: All things are independent : Autonomous Event

: Multiple Events are difficult to access the underlined issue

→ Cons:

(i) Complex Debugging and monitor
(ii) Event Ordering (iii) Latency
(iv) Data Consistency
(v) Overhead. (vi) Increased Complexity

→ Use Cases:

(i) E-Commerce System
(ii) Real-Time Analysis
(iii) Microservice Architecture
(iv) Financial Transactions
(v) IOT (vi) Logistics and Supply chain
(vii) HealthCare System
(viii) Streaming Platforms.

(*) Activity in Next Class.

21/10/24

⊙ Stream Processing:

→ Continuous or Real-Time Processing.

→ Stream Processing Models:

(i) Stateless
(ii) Stateful

→ Pros:

(i) Real-Time
(ii) Low-Latency
(iii) Continuous Insights
(iv) Scalability.

→ Use Cases:

→ IOT Devices

→ Real-Time Analysis.

(*) Cases Studies:

→ Google Route Redvie

→ Facebook News Feed

→ Netflix.

23/10/24 (4 layers)

(*) Presentation Layer:

UI/UX Layer

→ User Login

(*) Application Layer:

Credentials of Presentation layer are passed to it.

→ Coordinate User Operations.

→ User Authentication

(*) Domain Layer:

: Communication / Networks

→ Validate User Credentials

→ Precisely Business Logic Responsibilities.

④ Data Layer (Persistence layer):

→ Responsible for managing data.

⑤ Additional layers:

→ Integration layer
→ security layer
→ Caching layer
→ Logging and Monitoring layer.
→ Workflow layer

⑥ Use Cases:

→ Web Application
→ Enterprise systems
→ Microservice Architecture
→ Banking systems.

⑦ Benefits:

→ separation
→ Scalability
→ Maintenance
→ Testability
→ Reusability.

(*) Cons:

→ Performance Overhead
→ Complexity in Small System.
→ Tigut coupling Between Layers
→ Difficulty in Modification

⊙ Real-Time Smart Traffic Management System (SEMS):

→ Smart Route Detection
→ GPS, Real Time Data processing
→ Sensors, Cameras etc....

(*) Scalability, Testing, Modularity.

(*) When to Consider Layered Architecture:

→ Historical Data Analysis
→ User-Facing

28/10/24

⊙ Data Centric Architectural Style:

→ Access, Update, Read, Write through central repository.

→ Key characteristics:

(i) Centralized Data

(ii) Loose Coupling

→ Benefits:

(i) Data Consistency

(ii) Reusability (iii) Maintainability.

→ Version Control System (VCS)

MVC

⊙ **Model:**

→ Contains business Logic

⊙ View:

→ Interface of Logic

⊙ Controller:

→ Middle man between model and view

→ Overall control

⊙ Key Benefits:

(i) Separation of concerns

(ii) Reusable

⑤ new Model, view etc....

① Model path

* MVP (Model View Presenter)

* Example MVP : Case Study

30/10/24

* MVP :-
  - Presenter (All Implementation work will be included in it)
  - Controller (Only calling work)

* MVM :-
  - → Binding (UI change will be reflected in model)
  - → View changes and it will be reflected in model.

4/11/24

* Multiple Tier: Peers :

- → Responsibility Distribution
- → Can run on different dedicated servers
- → For Large scale and complex scope.
- → For Durability, Scalability advantage

: Bonus Quiz in next Class

→ Security wise (server and independently managed).

(+) Redundency can occur

(*) Distributed servers (Concurrency control Management)

(*) Layered vs multier Architecture
→ Deployment strategy
→ Purpose and Usability

(*) Monolithic Architecture:
→ single code
→ simple, Easy to test, purpose Maintainability.

(•) **Client-Server Architecture:**
(Request) (Responds)
→ Centralized control

(•) Microservices Architecture.
→ Any Responsibilities / function is called services.
→ Food delivery app.

(*) SOA (Service oriented Architecture) Vs Microservices
(coarse grain)

(till syllabus of Mid).