ADVANCED SOFTWARE ENGINEERING

CSE 422

ARCHITECTURAL DESIGN

ARCHITECTURAL PATTERNS

PRESENTED BY

DR. ABDALLA MOUSTAFA

1

- **Are you an entrepreneur planning to launch your IT application?**
- **are you an enterprise IT manager in charge of building a new enterprise IT application?**
- you need to decide the most appropriate software architecture

2

Software Architecture Pattern: What is it?

- Software architecture is a pictorial representation of the IT system that contains the following:
  - The organization of the system.
  - Different components and their functions.
  - How the components interact with each other.

3

Why Should You Focus on Software Architecture?

- Your business stakeholders need to know about your progress. Software architecture acts as a ***visual communication tool***, therefore they can understand the system you are building.
- Software architecture models make it easy to ***reuse*** them in other projects since you now know the decisions you made and the various trade-offs.

4

How do You Know if Your Software Architecture is Good?

- Indications that you have a good software architecture:
  - The software is easy to maintain;
  - Business stakeholders can understand it easily;
  - Good software architectures are usable over the long-term;
  - Such architecture patterns are flexible, adaptable, and extensible;
  - It should facilitate scalability;
  - The team can easily add features, moreover, the system performance doesn't diminish due to this;
  - There is no repetition of the code;
  - The system can be refactored easily.

5

COMMON SOFTWARE ARCHITECTURE PATTERNS

6

PATTERN #1: LAYERED ARCHITECTURE

7

- one of the most used patterns.
- The code is arranged in layers.
- Key characteristics of this pattern are as follows:
  - The outermost layer is where the data enters the system.
  - The data passes through the subsequent layers to reach the innermost layer, which is the database layer.
  - Simple implementations of this pattern have at least 3 layers (presentation layer, an application layer, and a data layer.)
  - Users access the presentation layer using a GUI,
  - The application layer runs the business logic.
  - The data layer has a database for the storage and retrieval of data.

8

- **Advantages:**
  - Maintaining the software is easy since the tiers are segregated.
  - Development teams find it easy to manage the software infrastructure, therefore, it's easy to develop large-scale web and cloud-hosted apps.
  - Popular frameworks like Java EE use this pattern.
- **Disadvantages**
  - The code can become too large.
  - A considerable part of the code only passes data between layers instead of executing any business logic, which can adversely impact performance.

9

PATTERN #2: CLIENT-SERVER

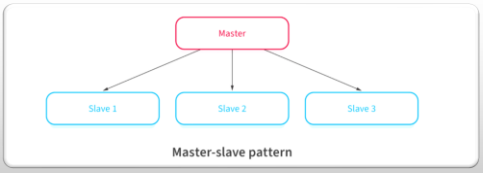
10

- another commonly used one where there are 2 entities.
- It has a set of clients and a server.
- The following are key characteristics of this pattern:
  - Client components send requests to the server, which processes them and responds back.
  - When a server accepts a request from a client, it opens a connection with the client over a specific protocol.
  - Servers can be stateful or stateless. A stateful server can receive multiple requests from clients. It maintains a record of requests from the client, and this record is called a 'session'.
  - Email applications are good examples of this pattern.

11

- **Advantages:**
  - Clients access data from a server using authorized access, which improves the sharing of data.
  - Accessing a service is via a 'user interface' (UI), therefore, there's no need to run terminal sessions or command prompts.
  - Client-server applications can be built irrespective of the platform or technology stack.
  - This is a distributed model with specific responsibilities for each component, which makes maintenance easier.
- **Disadvantages**
  - The server can be overloaded when there are too many requests.
  - A central server to support multiple clients represents a 'single point of failure'.

12



PATTERN #3: MASTER - SLAVE

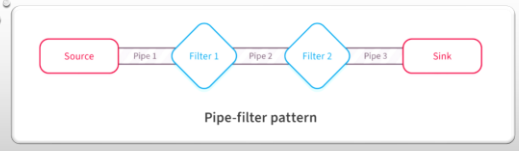
13

- "Master-slave architecture pattern" is useful when clients make multiple instances of the same request.
- The requests need simultaneous handling.
- The following are key characteristics of this pattern:
  - The master launches slaves when it receives simultaneous requests.
  - The slaves work in parallel, and the operation is complete only when all slaves complete processing their respective requests.

14

- **Advantages:**
  - Applications read from slaves without any impact on the master.
  - Taking a slave offline and the later synchronization with the master requires no downtime.
  - Any application involving multi-threading can make use of this pattern, e.g., monitoring applications used in electrical energy systems.
- **Disadvantages**
  - This pattern doesn't support automated fail-over systems since a slave needs to be manually promoted to a master if the original master fails.
  - Writing data is possible in the master only.
  - Failure of a master typically requires downtime and restart, moreover, data loss can happen in such cases.

15



PATTERN #4: PIPE-FILTER

16

- Suppose you have complex processing in hand.
- You will likely break it down into separate tasks and process them separately.
- This is where the "Pipe-filter" architecture pattern comes in use.
- The following are key characteristics of this pattern:
  - The code for each task is relatively small. You treat it as one independent 'filter'.
  - You can deploy, maintain, scale, and reuse code in each filter.
  - The stream of data that each filter processes pass through 'pipes.
  - Compilers often use this pattern.

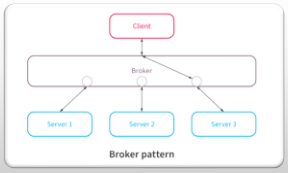
17

- **Advantages:**
  - There are repetitive steps such as reading the source code, parsing, generating code, etc. These can be easily organized as separate filters.
  - Each filter can perform its' processing in parallel if the data input is arranged as streams using pipes.
  - It's a resilient model since the pipeline can reschedule the work and assign to another instance of that filter.

Disadvantages

- This pattern is complex.
- Data loss between filters is possible in case of failures unless you use a reliable infrastructure.

18



Broker pattern

PATTERN #5: BROKER

19

- Consider distributed systems with components that provide different services independent of each other.
- Independent components could be heterogeneous systems on different servers.
- However, clients still need their requests serviced.
- The following are key characteristics of this pattern:
  - A broker component coordinates requests and responses between clients and servers.
  - The broker has the details of the servers and the individual services they provide.
  - The main components of the broker architectural pattern are clients, servers, and brokers. It also has bridges and proxies for clients and servers.
  - Clients send requests, and the broker finds the right server to route the request to.
  - It also sends the responses back to the clients.

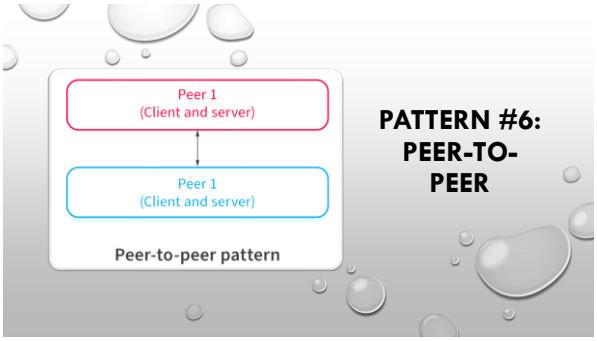
20

- **Advantages:**
  - Developers face no constraints due to the distributed environment. they simply use a broker.
  - This pattern helps using object-oriented technology in a distributed environment.

Disadvantages

- ?

21



PATTERN #6: PEER-TO-PEER

22

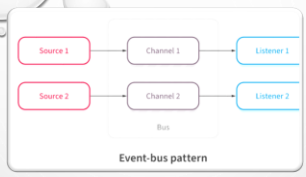
- "Peer-to-peer (P2P) pattern" is markedly different from the client-server pattern since each computer on the network has the same authority.

- The following are key characteristics of this pattern:
  - There isn't one central server, with each node having equal capabilities.
  - Each computer can function as a client or a server.
  - When more computers join the network, the overall capacity of the network increases.
  - File-sharing networks are good examples of the P2P pattern.
  - Bitcoin and other cryptocurrency networks are other examples.

23

- **Advantages:**
  - P2P networks are decentralized; therefore, they are more secure. You must have already heard a lot about the security of the Bitcoin network.
  - Hackers can't destroy the network by compromising just one server.
- **Disadvantages**
  - Under heavy load, the P2P pattern has performance limitations, as the questions surrounding the Bitcoin transaction throughout show.

24



Event-bus pattern

PATTERN #7: EVENT-BUS PATTERN

25

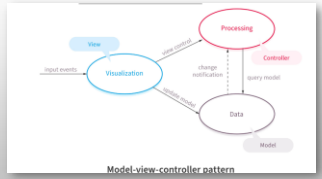
- There are applications when components act only when there is data to be processed.
- At other times, these components are inactive.
- The following are key characteristics of this pattern:
  - A central agent, which is an event-bus, accepts the input.
  - Different components handle different functions, therefore, the event-bus routes the data to the appropriate module.
  - Modules that don't receive any data pertaining to their function will remain inactive.
  - Think of a website using JavaScript. Users' mouse clicks and keystrokes are the data inputs.
  - The event-bus will collate these inputs and it will send the data to appropriate modules.
  - This software architecture pattern is also used in Android development.

26

- **Advantages:**
  - This pattern helps developers handle complexity.
  - It's a scalable architecture pattern.
  - This is an extensible architecture; new functionalities will only require a new type of events.
- **Disadvantages**
  - Testing of interdependent components is an elaborate process.
  - If different components handle the same event require complex treatment to error-handling.
  - Some amount of messaging overhead is typical of this pattern.
  - The development team should make provision for sufficient fall-back options in the event the event-bus has a failure.

27

PATTERN #8: MODEL-VIEW-CONTROLLER (MVC)



Model-view-controller pattern

28

- involves separating an applications' data model, presentation layer, and control aspects.
- The following are key characteristics of this pattern:
  - There are three building blocks here, namely, model, view, and controller.
  - The application data resides in the model.
  - Users see the application data through the view; however, the view can't influence what the user will do with the data.
  - The controller is the building block between the model and the view.
  - View triggers events, subsequently, the controller acts on it.
  - The action is typically a method call to the model.
  - The response is shown in the view.

29

- **Advantages:**
  - Using this model expedites the development.
  - Development teams can present multiple views to users.
  - Changes to the UI is common in web applications, however, the MVC pattern doesn't need changes for it.
  - The model doesn't format data before presenting to users, therefore, you can use this pattern with any interface.
- **Disadvantages**
  - With this pattern, the code has new layers, making it harder to navigate the code.
  - There is typically a learning curve for this pattern, and developers need to know multiple technologies.

30