II. SOFTWARE Architecture Patterns

- They are several Software stakeholders
- Each stakeholder views the software architecture following their point of interest.
- The 4+1 view model shows the various stakeholder architectural perspectives.

II. SOFTWARE Architecture Patterns

1. The logical view of the architecture
2. The process view of the architecture
3. The deployment view of the architecture
4. The implementation view of the architecture
5. The user case view of the architecture.

II. SOFTWARE Architecture Patterns

1. The logical view of the architecture

- ➢ This perspective deals with decomposing the software architecture system into logical components that represent the structural integrity that supports functional and non-functional(quality) requirements.

II. SOFTWARE Architecture Patterns

1. The logical view of the architecture

- ➢ When designing logical architecture, its important to use past experiences to discover overall strategies that have been used successfully in the development of software systems.

II. SOFTWARE Architecture Patterns

1. The logical view of the architecture

- ➢ The concepts of architecture styles and patterns have emerged as an approach for achieving reuse at the architectural level.

II. SOFTWARE Architecture Patterns

- One of the most useful tools in designing a software architecture

II. SOFTWARE Architecture Patterns

- Some of the design issues faced by software architects already have a documented proven solution.

II. SOFTWARE Architecture Patterns

- Once a pattern is identified, architects can always refer to a pattern catalog to find documented details about it.

II. SOFTWARE Architecture Patterns

- Knowing how to apply architectural pattern means you are knowledgeable about available patterns and can recognize when they apply to a given design scenario.

II. SOFTWARE Architecture Patterns

- ***Lesson major questions to answer***

- ✓ What software architecture patterns are?
- ✓ How the patterns can be used?
- ✓ Illustration of some commonly used architecture patterns

II. SOFTWARE Architecture Patterns

- ***Lesson major questions to answer***

- ✓ What software architecture patterns are?
- ✓ How the patterns can be used?
- ✓ Illustration of some commonly used architecture patterns

II. SOFTWARE Architecture Patterns

➢ ***What is software architecture pattern?***

✓ A solution to a recurring problem that is well understood, in a particular context.

✓ *Contain of each Pattern*

1. A context
2. A problem
3. A solution

II. SOFTWARE Architecture Patterns

- ***What is software architecture pattern?***

- ✓ *Contain of each Pattern*

The problem may be to overcome some challenge, take advantage of some opportunity, or to satisfy one or more quality attributes.

II. SOFTWARE Architecture Patterns

- ***What is software architecture pattern?***

- ✓ Patterns codify knowledge and experience into a solution that we can reuse.

II. SOFTWARE Architecture Patterns

- ***Importance of using Patterns***

✓It simplifies design and allows us to gain the benefits of using a solution that is proven to solve a particular design problem.

II. SOFTWARE Architecture Patterns

➢ ***Design Patterns vs Architecture Patterns***

✓Both are similar except that they are broader in scope and are applied at the architectural level.

✓Architecture patterns focus on architectural problems , while design patterns focus on solving problems that occur during implementation.

II. SOFTWARE Architecture Patterns

- ***Design Patterns vs Architecture Patterns***

- ✓ Architecture pattern(AP) provide a high-level structure and behavior for software systems.
- ✓ AP is a grouping of design decisions that have been repeated and used successfully for a given context.

II. SOFTWARE Architecture Patterns

- ***Using Software architectural patterns***

- ✓ It can be applied to the entire system or one of the subsystems.
- ✓ More that one pattern can be used in to a single software system.
- ✓ Patterns can be combined to solve problems.

II. SOFTWARE Architecture Patterns

- ***Overusing architecture patterns***

- ✓ Though a pattern is an important tool, don't force the use of a particular pattern

II. SOFTWARE Architecture Patterns

➢ ***Architectural Style Vs Architecture pattern***

✓ Architectural style is a set of elements, and the vocabulary to be used for those elements, that is available to be used in an architecture.

II. SOFTWARE Architecture Patterns

- ***Architectural Style Vs Architecture pattern***

- ✓ Architectural style contains an architecture design by restricting the available design choices.
- ✓ When a software adheres to a particular style, it is expected to exhibit certain properties.

II. SOFTWARE Architecture Patterns

- ***Architectural Style Vs Architecture pattern***

- ✓ If we were to follow microservice architecture style for an application, which comes with the constraint that each service should be independent on the others, we can expect such a system to have certain properties.

II. SOFTWARE Architecture Patterns

➢ ***Architectural Style Vs Architecture pattern***

✓ A system that follows microservice architectural style will be able to deploy services independently, isolate faults for a particular service, and use a technology of the teams choosing for a given service..

II. SOFTWARE Architecture Patterns

➢ ***Architectural Style Vs Architecture pattern***

✓ A software architecture pattern is a particular arrangement of the available elements into a solution for a recurring problem given a certain context.

II. SOFTWARE Architecture Patterns

- ***Architectural Style Vs Architecture pattern***
- ✓ Given a particular architecture style, we can use the vocabulary of that style to express how we want to use the elements available in that style in a certain way.
- ✓ When this arrangement is a known solution to a common, recurring problem in a particular context, it's a software architecture pattern.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture**

- ✓ Useful in complicated system partitioning
- ✓ Software application is divided into various horizontal layers, with each layer located on top of a lower layer.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(1)**

- ✓ Each layer is dependent on one or more layers below it but independent of the layers above it.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Layered Architecture(1)**

✓ **Open Vs Closed Layers**

○ Layered architecture could to be design to be close or open.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(2)**

  - **Close Layer:** Request that are flowing down the stack from the layer above must go through it and cannot bypass it.

Eg: Three –layer application architecture with Presentation, business and data layers

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(2)**

- **Close Layer: Eg:** Three –layer application
- if the business layer is closed, the presentation layer must send all request to the business layer and cannot bypass it to send a request directly to the data layer.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(3)**

- **Close Layer:**

- It provides layer isolation, which makes code easier to change, write and understand.
- It makes the layers independent of each other, such that changes made in one should not affect others.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(4)**

- **Open Layer:**
- Increase of complexity
- Maintainability is lowered because multiple layers can now call into another layer, increasing the number of dependencies and making changes more difficult.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Layered Architecture(5)**

○ **Open Layer:**

- Could also be use, take our previous example of three-layer architecture introducing a shared service layer between the business and data layers.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Layered Architecture(6)**

- **Open Layer:**

- Could also be use, take our previous example of three-layer architecture introducing a shared service layer between the business and data layers.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Tiers Vs Layers**

  - Layers are logical separations of a software application and tiers are physical ones.
  - When partitioning application logic, layers are a way to organize functionality and components.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Tiers Vs <u>Layers</u>**

- In a three-layered architecture, the logic may be separated into presentation, business and data layers.

- If more than one layer, its known as multi-layer architecture.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- Tiers Vs Layers

  - Different layers do not necessarily have to be located on different physical machines but on the same machine.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***
- Tiers Vs Layers

- Tiers concern themselves with the physical location of the functionality and components.
- A three-tiered architecture with presentation, business , and data tier implies that those three have been physically deployed to three separate machines and are each running on those separate machines.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Tiers Vs Layers**

- When a software is partitioned into multiple tiers, its known as multi-tier architecture.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Advantages of Layered architecture**
  - It reduces complexity by achieving a separation of concerns (Soc)
  - Dependencies between layers can be minimized that further reduces complexity
  - Make development easy
  - Increase testability
  - Increase reusability

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Disadvantages of Layered architecture**

- In spite that the layers are designed to be independent, a requirement change requires changes in multiple layers.

✓ This type of coupling lowers the agility of the software application.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

  - **Disadvantages of Layered architecture**

    - More code will be required for layered applications so as the provide the interfaces and other logic necessary for the communication between the multiple layers.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Disadvantages of Layered architecture**

- Development team has to be diligent about placing code in the correct layer so as not to leak logic to a layer that belongs in another layer.

Example :

✓ Placing business logic in the presentation layer or putting data-access logic in the business layer.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Disadvantages of Layered architecture**

- Application may experience low performance if you need to design a high-performance application, because it will be inefficient for the request to go through multiple layers.

Example :

- Moving from one layer to another sometimes requires data representations to be transformed.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***
- **Disadvantages of Layered architecture**

- Greater monetary cost associated with having a multi-tier architecture; more machines are used by the application, the greater the overall cost.
- Unless the hosting of the software app is handled by a cloud provider , an internal team will be needed to manage the physical hardware of a multi-tier app

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

▪ **Disadvantages of Layered architecture**

- Example : Adding a new field will require changes to multiple layers.

✓ The presentation layer so that it can be displayed…
✓ The business layers so that I can be validated/saved/processed…
✓ The data layer because it will need to be added to the database.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

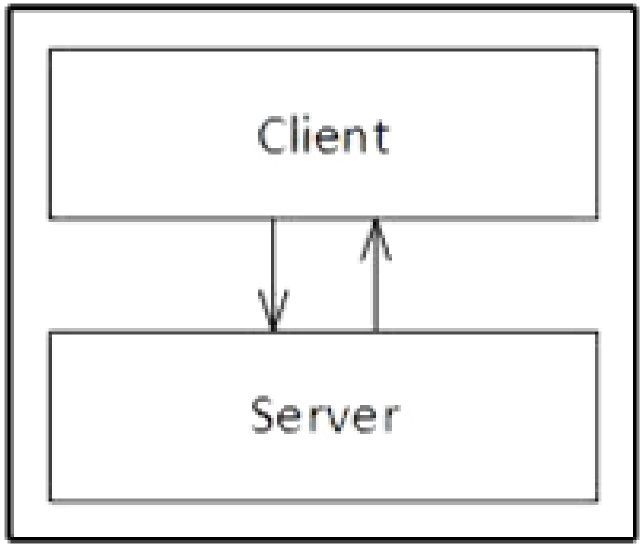
❑ **Client-server Architecture (Two-tier architecture)**

- Clients and servers communicate with each other directly
- A client request some resource or calls some service provided by a server and the server responds to the request of client.
- The can be multiple clients connected to a single server.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**



II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture**

- Usually in a distributed application
- Also known as the two-tier architecture
- The client part-contains the user interface code
- The server contains database, which is like the RDBMS- Relational database management system.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

Client-server Architecture (Two-tier architecture)

- Majority of application logic in a client-server architecture is located on the server, but some of its could be located in the client.
- Application logic located on the server might exist in software components in the database , or both

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**

- **Thick or fat client :**

✓ When the client contains a significant portion of the logic and is handling a large share of the workload.

- **Thin client :**

✓ When the server is doing that instead

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**

○ **Business logic spread on client and server**

✓ Consistency need to be applied and maintenance become more difficult

○ **The same business logic code on client and server :**

✓ Modification need to be done on several places

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Client-server Architecture (Two-tier architecture)**

- **Using stored procedure for application logic**

- ✓ Situation where application logic exist in the database, commonly found in stored procedures.
- ✓ The stored procedure is a grouping of one or more SQL statements that forms a logical unit to accomplish some task.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Client-server Architecture (Two-tier architecture)**

- **Using stored procedure for application logic**

- ✓ Can be use to do a combination of retrieving , inserting , updating and deleting data.
- ✓ Their use reduces the amount of network traffic between the client and the server

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**

○ **Using stored procedure for application logic**

✓ A single call from the client to the server is all that is needed to execute a stored procedure.

✓ If the logic was not inside the store procedure, multiple calls would need to be made over the network between the client and the server to execute the logic.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**

○ **Using stored procedure for application logic**

✓ Stored procedures are compiled the first time they are executed.

✓ They are security advantages to using stored procedure because users and applications do not need to grand permission to the database objects for the procedure to run.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***
- **Client-server Architecture (Two-tier architecture)**
- **Using stored procedure for application logic**
- ✓ The drawback is there are limited coding constructs , as compared to high-level programming languages.
- ✓ Having business logic stored in the procedure means your business logic is not centralized.
- ✓ In modern applications, application logic should not be stored in procedures.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***
- **Client-server Architecture (Two-tier architecture)**
  - **Using stored procedure for application logic**

✓ Application logic belongs outside the data layer and independent and decoupled from the mechanism used to store the data.

✓ Some engineers have limit store procedures in CRUD operations which limit the stored procedures power.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Client-server Architecture (Two-tier architecture)**

- **Using stored procedure for application logic**

✓ You can use stored procedure for complex queries with complex joins and where clauses and queries requiring multiple statements and large amount of data, the performance advantages of stored procedure can be useful.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Client-server Architecture (Two-tier architecture)**

- **Using stored procedure for application logic**

✓ You can use stored procedure for complex queries with complex joins and where clauses and queries requiring multiple statements and large amount of data, the performance advantages of stored procedure can be useful.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **N-tier Architecture**

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier architecture**

✓ Also known as multitier architecture

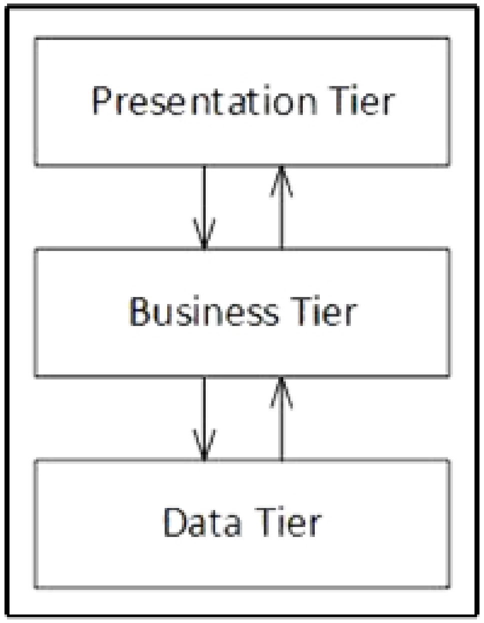
✓ The widely use N-tier architecture is the 3-tier architecture.

✓ The 3-tier application separates logic into ***presentation***, ***business*** and ***data layers***.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier architecture : 3-tier architecture**



Three-tier architecture separated the logic into Presentation, business and data layers

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

○ **Presentation tier**

✓ The application part that :-

❖ Users see and interact with

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

○ **Presentation tier**

✓ Provides functionality for application's user interface

❖ Data is presented to users in this tier

❖ Inputs are received from users in this tier too.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **N-tier Architecture – three-tier architecture**

- **Presentation tier – some logic available in this tier**
- ✓ Logic to render the user interface
- ✓ Logic to place data in the appropriate UI components
- ✓ Logic for formatting data
- ✓ Logic to hid/show UI components
- ✓ Logic for basic data validation – help users input

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

- **Presentation tier – Be aware**

✓ Not to introduce business logic into the validation, which should be handled by the business tier.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

○ **Presentation tier – things to note**

✓ It should provide users with useful feedback

Example : *Friendly and informative messages, tooltips, visual feedback such as progress bar for long-running process, and notifications to inform users about the completion or failure of asynchronous operations.*

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

- **Business tier (Application tier)**

✓ Provide implementation for business layer

- *Business rules validation*
- *Calculation logic*

✓ It coordinates the app and execute logic

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

- **Business tier (Application tier)**

✓ Can perform detailed processes and make logical decisions

✓ Serves as the intermediary between the presentation and data tiers.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

N-tier Architecture – three-tier architecture

- **Presentation tier**

- ✓ This tier has to consider some aspects of software quality attributes

- Usability quality attribute

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

○ **Data tier**

✓ Provide functionality to access and manage data

✓ It contains a datastore for persistence storage, such as RDBMS

✓ It provide services and data for the business tier

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **N-tier Architecture – three-tier architecture**

○ **Data tier**

✓ Some systems share the data layer into two other layers – the data access or persistence layer and the database layer

✓ **Data access layer – contains ORM tool**

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture**

- **An event** is the occurrence of something significant in a software application.

✓ The change of state of a component may interest other applications or other components within the same application. State change like purchase order

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

- It's a distributed , asynchronous software architecture pattern
- It integrate app and components through the production and handling of events.
- It is loosely coupled, event producer does know the event subscribers

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

- SOA can be called based on events being triggered
- Service operations can also give rise to an event
- Issues may occur dues to lack of responsiveness, performance issues, or failures with event mediators and event brokers.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

○ **Event Channels**

- An event producer creates an event
- An event is recognize by the message is produces
- The event message contains data that is send to the event processor through an event channel which are streams of event messages.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

- **Event Channels**
  - Event channels are implemented as message queues which use the point-to-point channel pattern or message topics, which use the publish-subscribe pattern.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

- **Message queues**
  - Ensures there is one, and only one receiver for a message meaning only one event processor will receive an even from a event channel
  - Point-to point is use for message queues implementations

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA)**

○ **Message queues**

▪ Point-to-point channel pattern

✓ A pattern use that makes sure only one receiver receive a given message

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

Event-driven Architecture (EDA)

- **Message topics**
  - Allows multiple event consumers to receive an event message
  - The publish-subscribe pattern is used for the implementation of message topics.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***
- **Event-driven Architecture (EDA)**
  - **Message topics**
    - The publish-subscribe pattern
      - ✓ Also referred to as pub/sub is a messaging pattern that provides a way for a sender(publisher) to broadcast a message to interested parties(subscribers)

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Event-driven Architecture (EDA) – topologies**

- **The mediator topology**

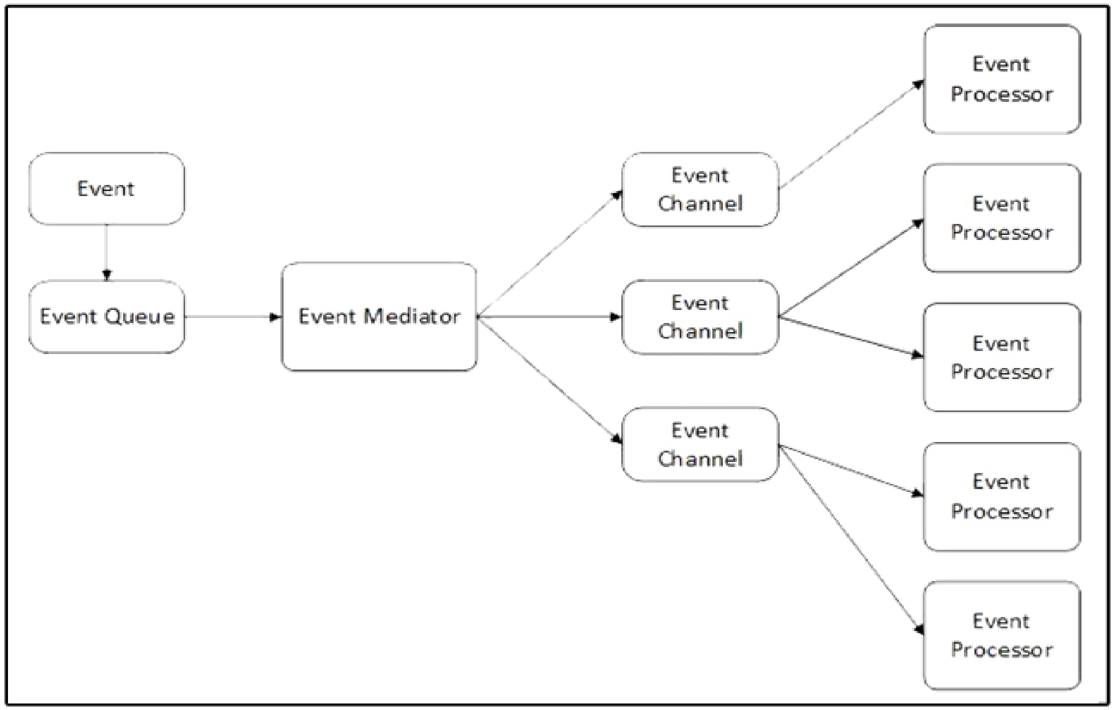
  - Uses a single event queue and an event mediator to route events to the relevant event processors.
  - Use when multiple steps are required to process an event

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

○ **The mediator topology**

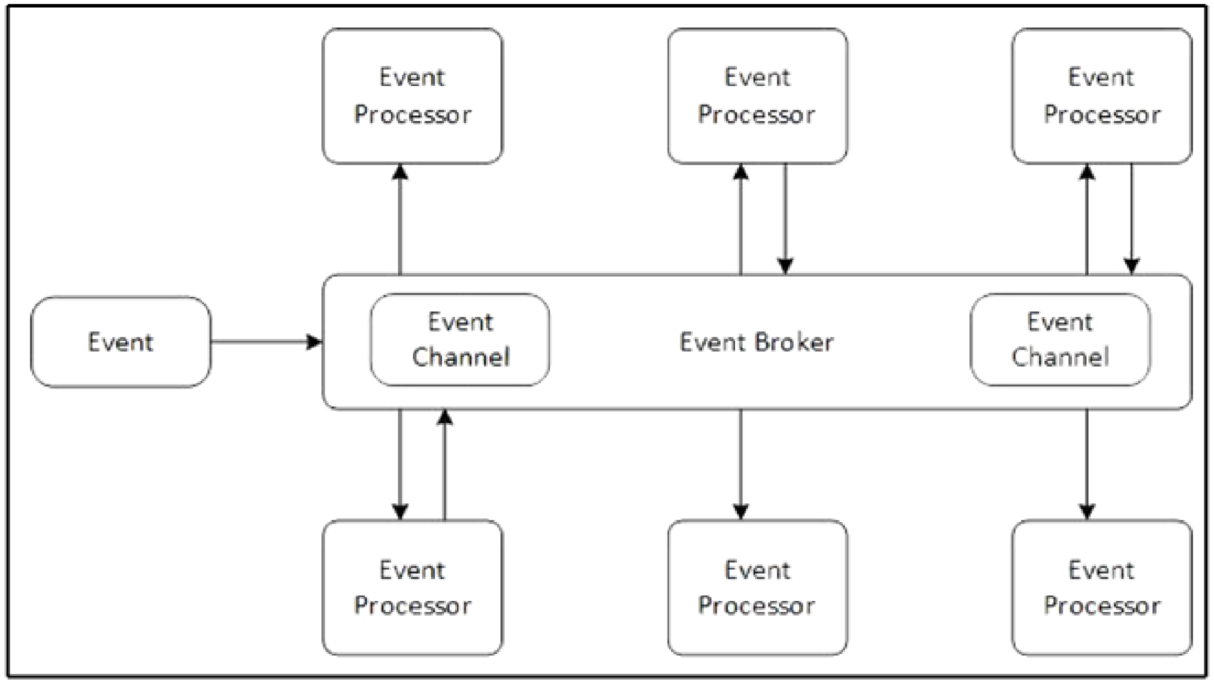


II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

○ **The broker topology**



II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Types of event-driven functionality**

✓ Event notification

✓ Event-carried state transfer

✓ Event sourcing

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-driven functionality – Event notification**
  - A functionality where a software system sends a message when an event takes place
  - The mediator and broker topologies allow its implementation

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-carried state transfer functionality**
  - Used in the case where, when and even consumer receives an event notification, it may need more information from the event producer in order to take the action that they want to take.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-carried state transfer functionality**
  - Example : A sales system may send a new order event notification and a shipping system may subscribe to this type of event message.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-carried state transfer functionality**
  - Example : .. The system now needs additional information about the order, such as the quantity and type of line items that are in the order

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

❑ Event-driven Architecture (EDA) – topologies

- **Event-carried state transfer functionality**
  - Example : ..

This requires shipping system to query the sales system in some way, such as through an API for this information

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-sourcing functionality**

▪ The need to know the details of the state changes that got to the state of the event

▪ With event-sourcing, the events that take place in a system, such as state changes are persisted in an event store.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture (EDA) – topologies**

- **Event-sourcing functionality**
  - Having a complete record of all the events that took place allows it to serve as a source of truth
  - Replaying events from an event log can be used to recreate an application's state.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

- Widely use for the UI of an application
- Well suited for web application but also use for desktop applications
- Provides a structure for building user interfaces and provides a separation of different responsibilities involved.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

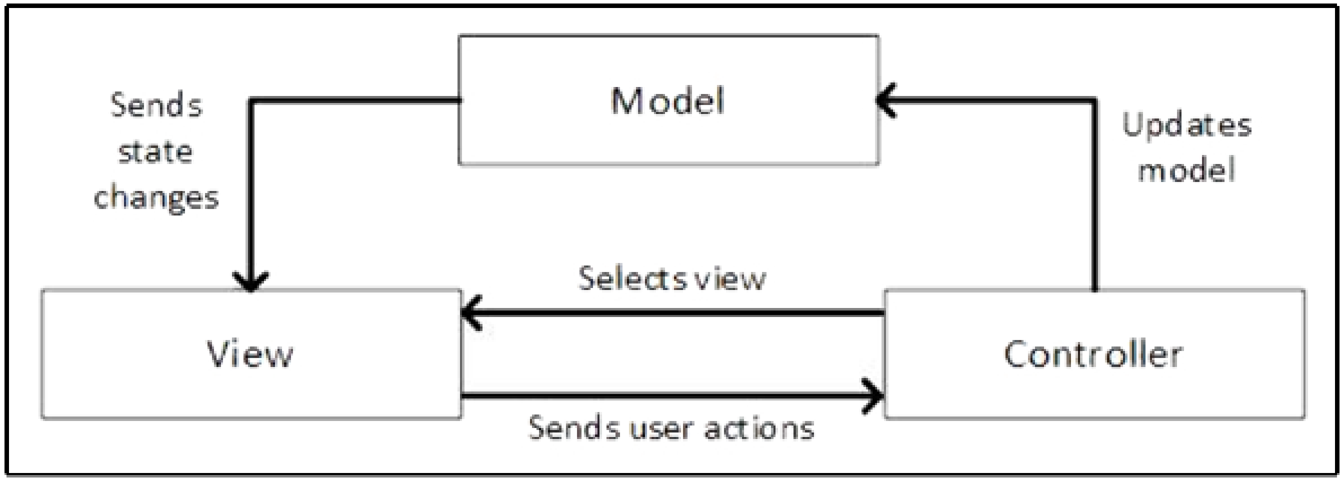
- Popular development frameworks make use of this pattern such as Ruby on Rails, ASP.NET MVC, and Spring MVC.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

The MVC pattern consists of the **Model**, **View**, and **Controller**:



II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

- **Model**

✓ Manages application data and state

✓ It processes the data to and from a data store

✓ Independent of controllers and views so that reused can be applied with different UI

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

○ **Model**

✓ Receives directives from controllers to retrieve and update data

✓ Provides application state updates

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

○ **View**

✓ The part of the application visible to the user

✓ Display data to the user in an appropriate interface based on information received from the controller.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

○ **View**

✓ If model is providing application state updates directly to views, the views may also be updated based on these notifications

✓ As users manipulate the view, the view will send the information to the controller.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

○ **Controller**

✓ Requests are routed to the appropriate controller based on routing configuration

✓ It acts as an intermediary between the model and the view

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

○ **Controller**

✓ Controllers executes application logic to select the appropriate view and sends the information for it to be rendered in the user interface.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

- **Advantages of the MVC pattern**

✓ It allows separation of concerns

✓ It makes presentation objects more reusable – separating the user interface from the data allows UI components to be reused.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

- **Advantages of the MVC pattern**

✓ Separation of presentation from the business logic and data allows developers to specialize in either frontend or backend development.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Controller Pattern**

- **Disadvantages of the MVC pattern**

✓ If developers team is not set up so that developers are focused on either frontend or backend development, this does require developers to be skilled in full stack.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

- A variation of the MVC pattern
- It provides a separation between UI login and business logic
- The presenter takes the place of the controller in the MVP pattern

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

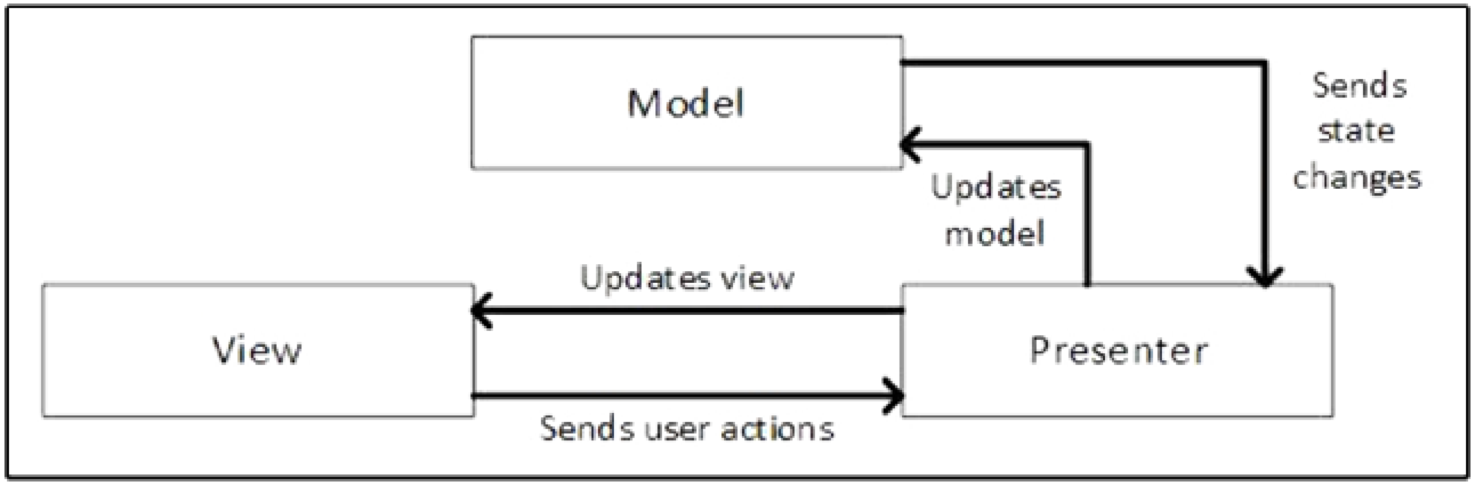
❑ **The Model-View-Presenter Pattern**

- Each view typically has a corresponding interface (view interface)
- Presenters are coupled with the view interfaces
- In MVC the view is closely coupled to the Model

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**



II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

- Both web and desktop applications can use the MVP pattern.
- The main components are the Model, View and Presenter

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

- **The Model**

  - It represent the business model and the data.
  - It interacts with the database to retrieve and update data
  - It receives messages form the presented for updates and reports state changes back to the presenter.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

○ **The View**

- Responsible for displaying user interface and data.
- Each view in the MVP implements an interface (view interface)
- As the user interacts with the view, the view will send messages to the presenter to act on the events and data.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

○ **The View**

- Presenters are loosely coupled with views through the view interface
- Views are more passive in the MVP model and rely on the presenter to provide information on what to display.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

- **The Presenter**
  - Intermediary between the model and the view
  - Each view has a presenter and the view notifies the presenter of user actions.
  - The presenter updates the model and receives state changes from the model.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-Presenter Pattern**

- **The Presenter**
  - It receives data from the model and format it for the view to display, having active role in the presentation logic.
  - In MVC, a controller can interact with multiple views, in the MVP, each presenter typically handles one, and only one , view.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

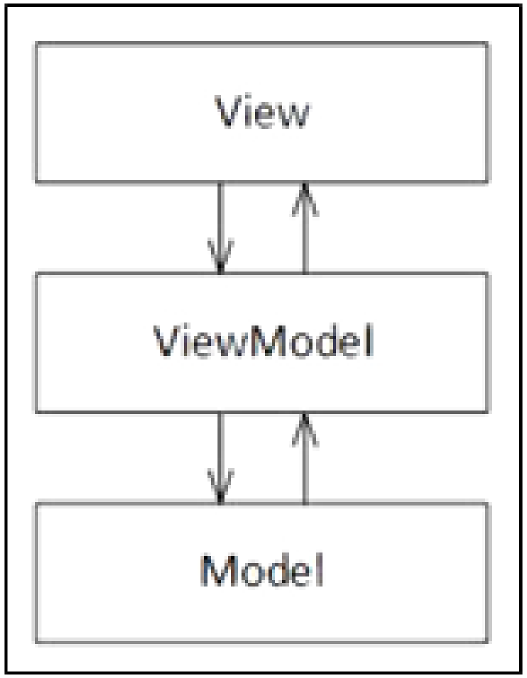
❑ **The Model-View-ViewModel Pattern (MVVM)**

- Shares similarities with MVC and MVP
- Partitioning the various responsibilities makes an application easier to maintain, extend, and test.
- It separates the UI from the rest of the app.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-ViewModel Pattern (MVVM)**



II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-ViewModel Pattern (MVVM)**

- It works well for rich desktop applications and also other types of application such as web application and mobile applications.
- Main components are Model , View and ViewModel.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-ViewModel Pattern (MVVM)**

○ **The Model**

- It represents the business domain object and the data.
- It uses the database to retrieve and update data.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-ViewModel Pattern (MVVM)**

○ **The View**

- Responsible for user interface visible to users
- The view here is active
- If passive, it means the view is completely manipulated by a controller or a presenter, and does not have knowledge of the model,
- MVVM views are aware of the model and viewModel

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **The Model-View-ViewModel Pattern (MVVM)**

○ **The ViewModel**

- They coordinate between the view and the model.
- Provides data for views for display and manipulation, and also contain interaction logic to communicate with views and models.
- Must be able to handle user actions and data input send from views.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **The command Query responsibility Segregation pattern (CQRS)**

- Models used to read information is separated from the model that is used to update information.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Service-oriented architecture (SOA)**

- A service is a part of a software application that performs a specific task, providing functionality to other parts of the same app or to other apps.

II. SOFTWARE Architecture Patterns

- ***Architecture patterns (and styles)***

- **Service-oriented architecture (SOA)**

- Exampleinclude web applications, mobile applications , desktop application and other services.

II. SOFTWARE Architecture Patterns

➢ ***Architecture patterns (and styles)***

❑ **Event-driven Architecture**

- **Event channels**
- **Message queues**
- **Message topics**

Welcome!

This course is design for you to understand the ways software architectures are represented, both in UML and other visual tools.

QUESTIONS