Chapter-4

- Software Architecture

→ It is the process of converting software characteristics such as flexibility, scalability, feasibility, reusability and security into a structured solution that meets the technical and the business expectations

→ It is the process of designing the global organization of a software system, including dividing software into subsystems, deciding how these will interact, and determining their interfaces.

→ Core of design, so all software engineers need to understand it.

→ The architectural model will often constraint the overall efficiency, reusability and maintainability of the system.

→ Poor decisions made while creating this model will constrain subsequent design.

- Reasons needed to develop software architectural model

1) To enable everyone to better understand the system

→ A good architectural model allow people to understand how the system as a whole works; it also defines the terms that people use when they communicate with each other about lower-level details

2) To allow people to work on individual pieces of the system in isolation.

→ Allows planning & coordination of this distributed work

→ Architecture should provide sufficient information so that the work of the individual people or teams can later on be integrated to form the final system

3) To prepare for extension of the system

→ it beromes easier to plan the evolution of the system

→ Subsystems that are envisioned to be part of a future release can be included in the architecture, even though they are not to be developed immediately.

→ It is then possible to see how the new elements will be integrated, & where they will be connected to the system

4 To facilitate reuse and reusability

→ The architectural model makes each system component visible.
→ This is an important benefit since it encourages reuse.
→ By analyzing the architecture identify components that have high potential reusability.

- **Contents of good architectural model**

→ An important challenge in architectural modeling is to produce a relevant and synthetic picture of a large & complex system.

→ To ensure the maintainability and reliability of a system, an architectural model must be designed to be stable.

→ The architecture should be expressed clearly enough that it can be used to communicate effectively with clients.

→ A system's architecture will often be expressed in terms of several different views. These can include:

- The logical breakdown into subsystems
- The interfaces among the "
- The dynamics of the interaction among components at run time
- The data that will be shared among the subsystems.
- The components that will exist at run time, and the machines or devices on which they will be located.

- Developing an architecural model

1. Start by sketching an outline of the architecture
   → Based on the poincipal requirements and use cases
   → Determine the main components that will be needed
   → choose among the various architectural patterns

2. Refine the architecture
   → Identify main ways in which the components will interact and the interfaces between them
   → Decide how each piece of data & functionality will be distributed among the various components
   → Determine ; if you can re-use an existing framework, if you can build a framework.
3. Consider each use case and adjust the architecture to make it realizable

4. Mature the architecture

- Architecture Centric Process

- Describing an architecture using UML

→ All UML diagrams can be useful to describe aspects of the architectural model.

→ Four UML diagrams are particularly suitable for architecture modeling:

- Package diagram
- Subsystem diagrams
- Component diagrams
- Deployment diagrams.

- **Architectural Patterns**

→ An architectural pattern is a general, reusable solution to a commonly occuring problem in software architecture within a given context.

→ Similar to Software design pattern but have a broader scope.

→ allows us to design flexible systems using components where the components are as independent of each other as possible.

- The Model View Controller Architectural Patterns

→ also known as MVC pattern, divides an interactive application in to 3 parts as:

- model -
→ Contains the core functionality and data, contains the underlying classes whose instances are to be viewed & manipulated.

- view
→ displays the information to the user (more than one view may be defined), contains objects used to render the appearance of the data from the model in the user interface.

- Controller
→ handles the input from the user, contains the objects that control & handle the user's interaction with the view and the model.

→ This is done to separate internal representation of information from the ways information is presented to, and accepted from the user.

→ It decouples components and allows efficient code reuse.

- Usage
  → Architecture for world wide web applications in major programming languages.
  → Web frameworks such as Django & Rails.

- The MVC Architecture & design Principles

1 Divide & conquer
→ The 3 components can be somewhat independently designed.

2. Increase cohesion:

→ The components have stronger layer cohesion than if the view and controller were together in a single UI layer.

3 Reduce Coupling

→ The communication channels between the three components are minimal.

4 Increase reuse:

→ The view and controller normally make extensive

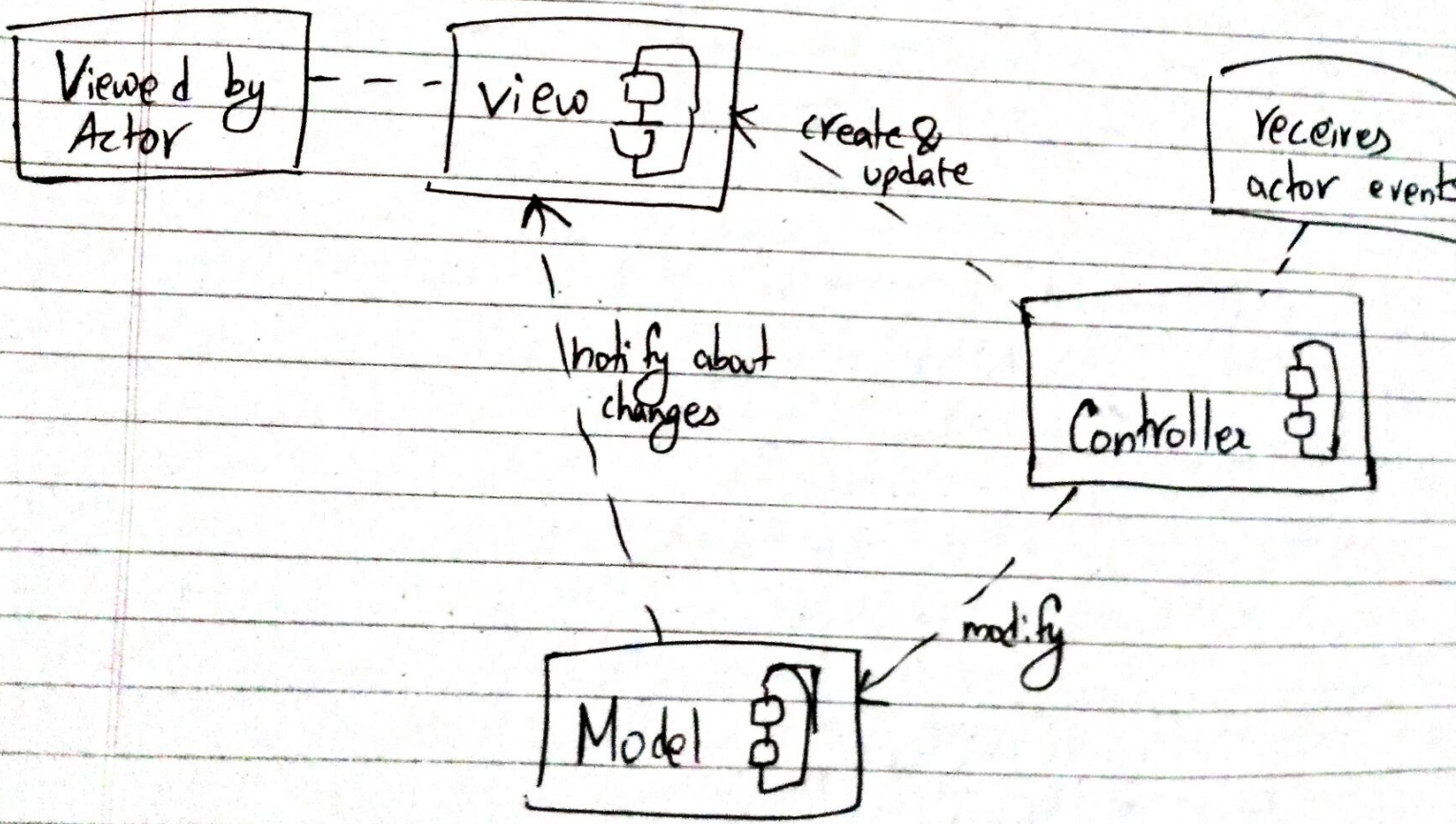
use of reusable components for various kinds of UI controls.

5 Design for flexibility:

→ It is usually quite easy to change the UI by changing the view, the controller or both

6 Design for testability

→ Test application separately from the UI



- Example:

→ The view component generates the HTML code to be displayed by the browser

→ The controller is the component that interprets 'HTTP post' transmissions coming back from the browser

→ The Model is the underlying system that manages the information

- **The Service Oriented Architectural Patterns**

→ It organizes an application as a collection of services that communicate with each other through well defined interfaces

→ In the context of the internet, the services are called web services

→ A web service is an applicable, accesible through the internet, which can be integrated with other services to form a complete system.

→ The different components generally communicate with each other using open standards such as XML.

→ Web services can perform a wide range of tasks.

→ Some may handle simple requests for information while others can perform more complex business processing

→ Enterprises can use Web services to automate and improve their operations.

- For example:
  An electronic commerce application can make use of web services to:
  - access the product database of several suppliers;
  - process credit card using a web service offered by a bank
  - arrange for delivery using a web service offered by a shipping company

→ The highest challenge facing the developer of a web service is security

→ Other important considerations are reliability, availability and scalability of the web services

- The service oriented architecture and design principles

1) Divide and Conquer
→ The application is made of independently designed services

2) Increase cohesion:
→ Web-based applications are loosely coupled built by binding together distributed components.

2. Increase cohesion:
→ The web services are structured as layers and generally have good functional cohesion.

3. Reduce coupling:
→ Web-based applications are loosely coupled built by binding together distributed components

4. Increase reusability:
→ A web service is a highly reusable component

5. Increase reuse:
→ web based application are built by reusing existing web services

6. Anticipate obsolescence
→ Obsolete services can be replaced by new implementation without impacting the application that use them.

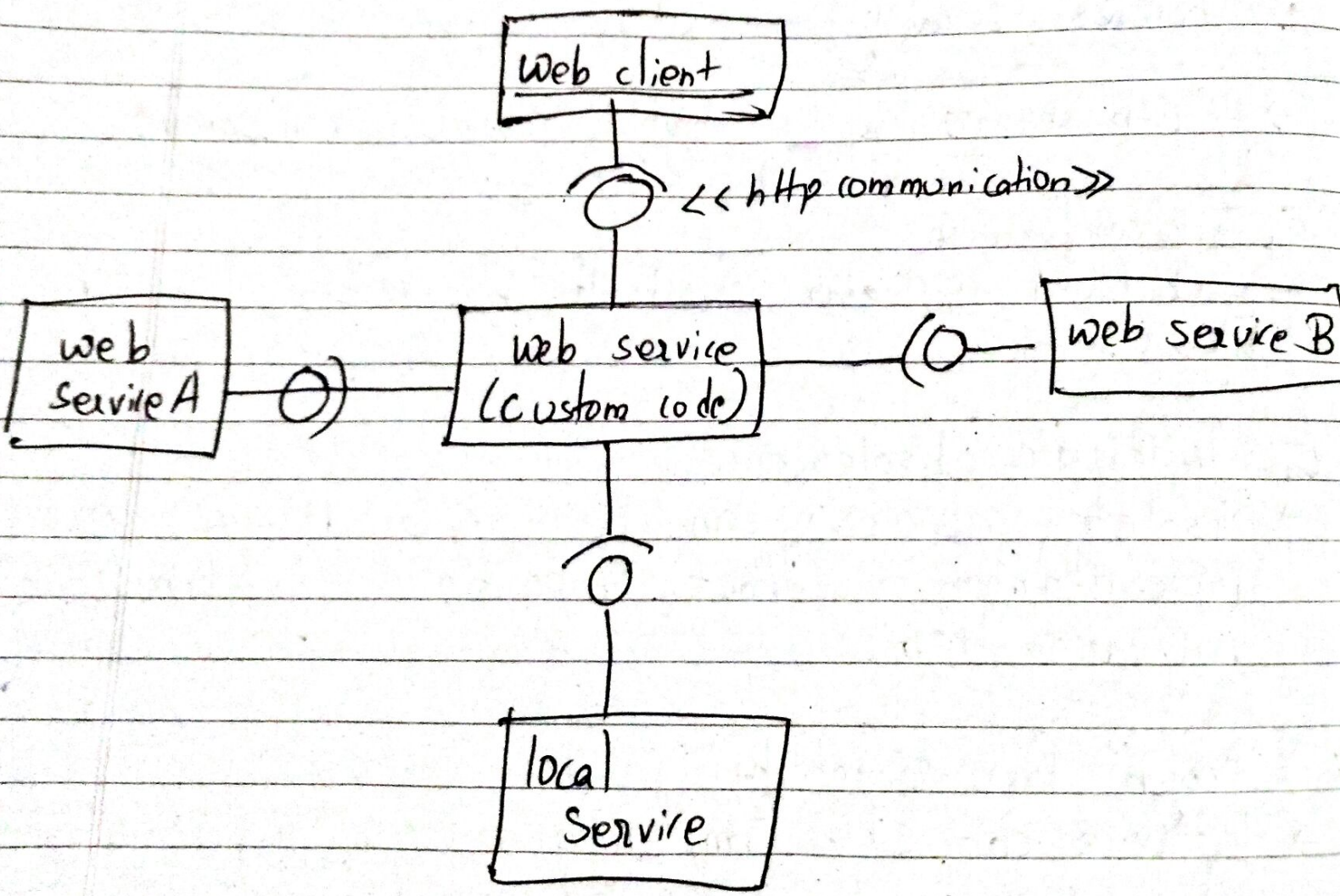
7) Design for portability
→ A service can be implemented on any platform that supports the required standards.

10. Design for testability

→ Each service can be tested independently

11 Design defensively

→ Web services enfore defensive design since different applications can access the service.



- The Message Oriented Architectural Patterns

→ Also known as Message oriented Middleware (MOM)

→ It is based on the idea that since humans can communicate and collaborate to accomplish some task by exchanging emails or instant messages, then software applications should also be able to operate in a similar manner i.e the different sub-systems communicate and collaborate to accomplish some task only by exchanging messages.

→ The core of this architecture is an application to application messaging system.

→ Senders and receivers need only to know what are the messaging formats.

→ the communicating applications do not have to be available at the same time (i.e messages can be made persistent)

→ The self-contained messages ~~do not~~ are sent by one component (the publisher) through virtual channels (topics) to which other interested software components can subscribe (subscribers)

→ The application can choose to ignore a received message or react to it by, for instance, sending a reply containing requested information.

→ The exchange of these messages is governed by two important principles.

→ First, message delivery is completely asynchronous; that means the exact moment at which a given message is delivered to a given subscribing application is unknown

→ Secondly, reliability mechanisms are in place such that the messaging system can offer the guarantee that a given message is delivered once and only once.

- The Message oriented Architecture & design Principles

1: Divide & Conquer

→ The application is made of isolated software component

2. Reduce Coupling
→ The components are loosely coupled since they share only data format

3. Increase abstraction
→ The prescribed formats of the messages are generally simple to manipulate all the application details being hidden behind the messaging system.

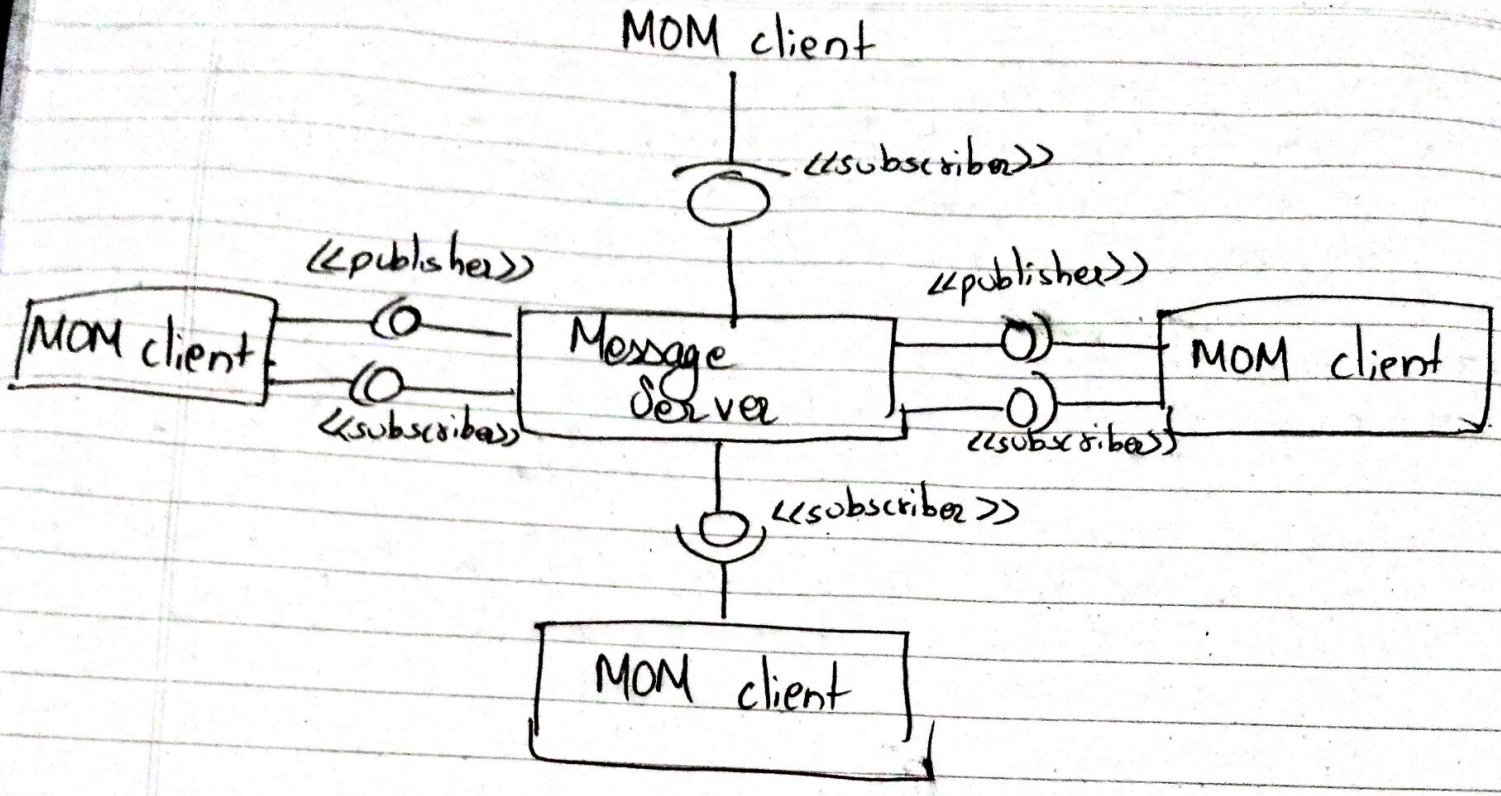
4. Increase reusability
→ A component will be reusable is the message formats are flexible enough

5 Increase reuse:
→ The components can be reused as long as the new system adheres to the proposed message formats.

6. Design for flexibility
→ The functionality of a message-oriented system can be easily updated or enhanced by adding or replacing components in the system.

7. Design for testability
→ Each component can be tested independently

8 Design defensively
→ Defensive design consists simply of validating all received messages before processing them.


MOM client
<<subscriber>>
<<publisher>>
<<publisher>>
MOM client
Message Server
MOM client
<<subscriber>>
<<subscriber>>
<<subscriber>>
MOM client