**Basics of MVVM Pattern**

**0. Intro of MVVM Pattern**

De-facto pattern that achieves goal of SoC & UIs.

One of the major presentation patterns.

**1. History of Presentation Pattern**

**[ Monolithic Design ]**

UI Widget

Other UI Widget

Application

Data Access

Business Logic

UI Logic & States

**Old problems of monolithic design**

Coupling – too high

Cohesion – too low

Reusability – too low

Extensibility – too low

Productivity – too low

When the widgets operates the same business logic, too many duplicated codes exist.

Maintenance was also a main problem.

**[ Rapid Application Development ]**

Providing developers to use UI components by simple drag & drop.

Developers became easier to build a project but still the app had fundamental problems of monolithic design.

**[ Model View Controller ]**

Providing developers to use UI components by simple drag & drop.

Controller

Controller

View

View

Controller

Model

View

Application

**View**

Component that display data & collect user input.

Notification can be included in view by being implemented in a type of observer pattern.

View interact with user & forward then to controller.

**Controller**

Takes user input and communicate with model.

Most of the user input are not directly passed through the model. Independent business logic is processed and matches to the current model.

This makes the application more extensible and not affected from model/architecture change.

**Model**

A in-memory representation of data which is stored/loaded from persistence storage.

This is generally done in an observer pattern.

**[ Layered Design ]**

Enterprise applications are generally separated into three logical layers.

Data Layer

Business Layer

Application

Presentation Layer

This kind of layers provides scalability for deploying many different layers and servers.

It has a great ability to swap layers with alternative implementations.

Design becomes more extensible.

This becomes the simple form of enterprise level design.

**Presentation Layer**

This layer displays data & provide feedback to user.

This collects user input which will be proceeded to business layer.

This layer separation provides benefits of easy UI change with minimal duplication.

**Business Layer(as well as Application Layer)**

The core functionality of system lives.

Business logics or domain logics are implemented here.

This supports multiple types of UIs & data stores and this increases App`s extensibility.

**Data Layer**

This layer pull & push data to store such as database, service or XML.

This allows change in data store without modifying other layers.

**[ MVC with Layered Design ]**

This is similar to MVC but has improved architecture by layering.

Business Layer

Data Layer

Presentation Layer

Model

Controller

Controller

View

View

Controller

View

Application

This was a remarkable jump compared to pure monolithic design.

However, there were still some problems like coupling between view logic & view state, danger of memory leaks, and difficulty of reusing view logic & view state.

+) The memory leakage occurred from old .NET because there were no support of weak references.

The problem was the observer object which subscribes a model. The garbage collector were not able to collect objects that were referenced strongly. So when the views become many of it, the memory leaked because garbage collector just look on it. There were some ways to prevent this such as finalizer.

And one day.. MVP pattern showed up.

**[ Model View Presenter ]**

IView

IView

IView

View

View

View

Presenter

Presenter

Presenter

Model

Application

**…. what`s different with MVC?**

The presenter is quite similar with controller.

This updates the views and make do some operation before passing through model.

This allows a interface system. Models can be replaced by fake objects and this makes the application more flexible and easier to test.

Views can only interact with presenters.

Presenter act like a mediator between view & model.

So all the problems are set?

NO!! There`s still some problems like memory leaks.

**[ Model View View Model ]**

MVVM pattern emerged to save the developers from hell of MVC&MVP.

This idea is from MVP pattern, especially the idea of humble objects.

Humble objects are left objects from external class or minimal interface, that are difficult to test/reuse. These kinds of objects is classified by view humble to make the design simpler. This object became a concept of property, that can be changed but being observed in real time. This idea even expanded to two-way binding which is one of the major concept of MVVM.

Two-way data binding allows presentation pattern no need to require overhead of implementing view.

This made GUI framework more rich and flexible.

Property Object

Ob

Property Object

Ob

Property Object

Ob

Property Object

Ob

Property

Application Model

View

Property

Property

Property

Data Layer

Model

Application

There`s no need for coupling between view & presenter.

This was the initial structure of MVVM design.

**[ Pure MVVM ]**

The modern Style of MVVM pattern.

View

Data Binding

View Model

View Model

View Model

Data Binding

View

Application