# MVVM

What? and Why ?

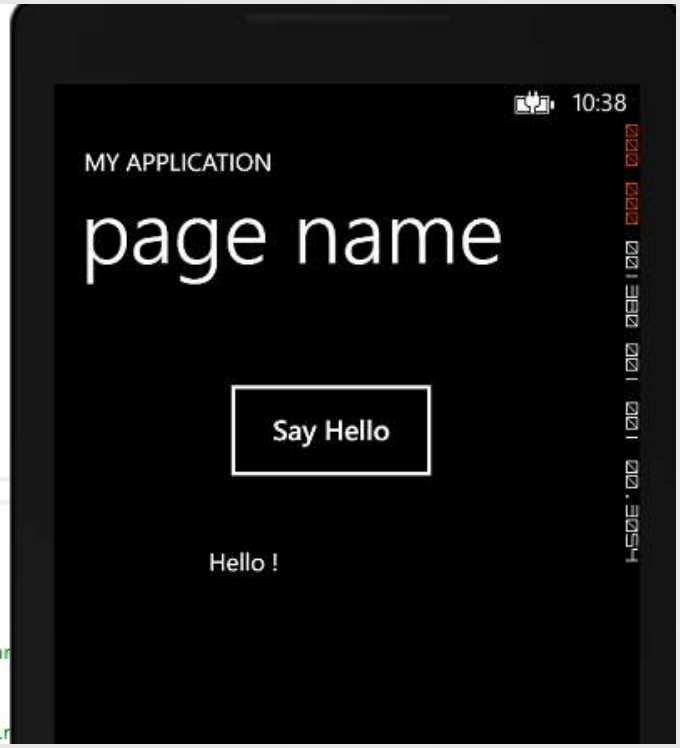
# Agenda

- Intro : Traditional UI development.
- MVVM in Deep :
  - What is MVVM?
  - Model / View Model /View.
  - Classes interaction .
- MVVM and other patterns.
- MVVM Advantages /Disadvantages.
- MVVM in Android.
- Conclusion & Recommendations.

# Traditional UI Development

Simple , no problem

```
using Microsoft.Phone.Shell;
using PhoneApp2.Resources;

namespace PhoneApp2
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            // Sample code to localize the ApplicationBar
            //BuildLocalizedApplicationBar();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            helloTxt.Text = " Hello !";
        }

        // Sample code for building a localized ApplicationBar
        //private void BuildLocalizedApplicationBar()
        //{
        //    // Set the page's ApplicationBar to a new instance of ApplicationBar
        //    ApplicationBar = new ApplicationBar();

        //    // Create a new button and set the text value to the localized strin
```

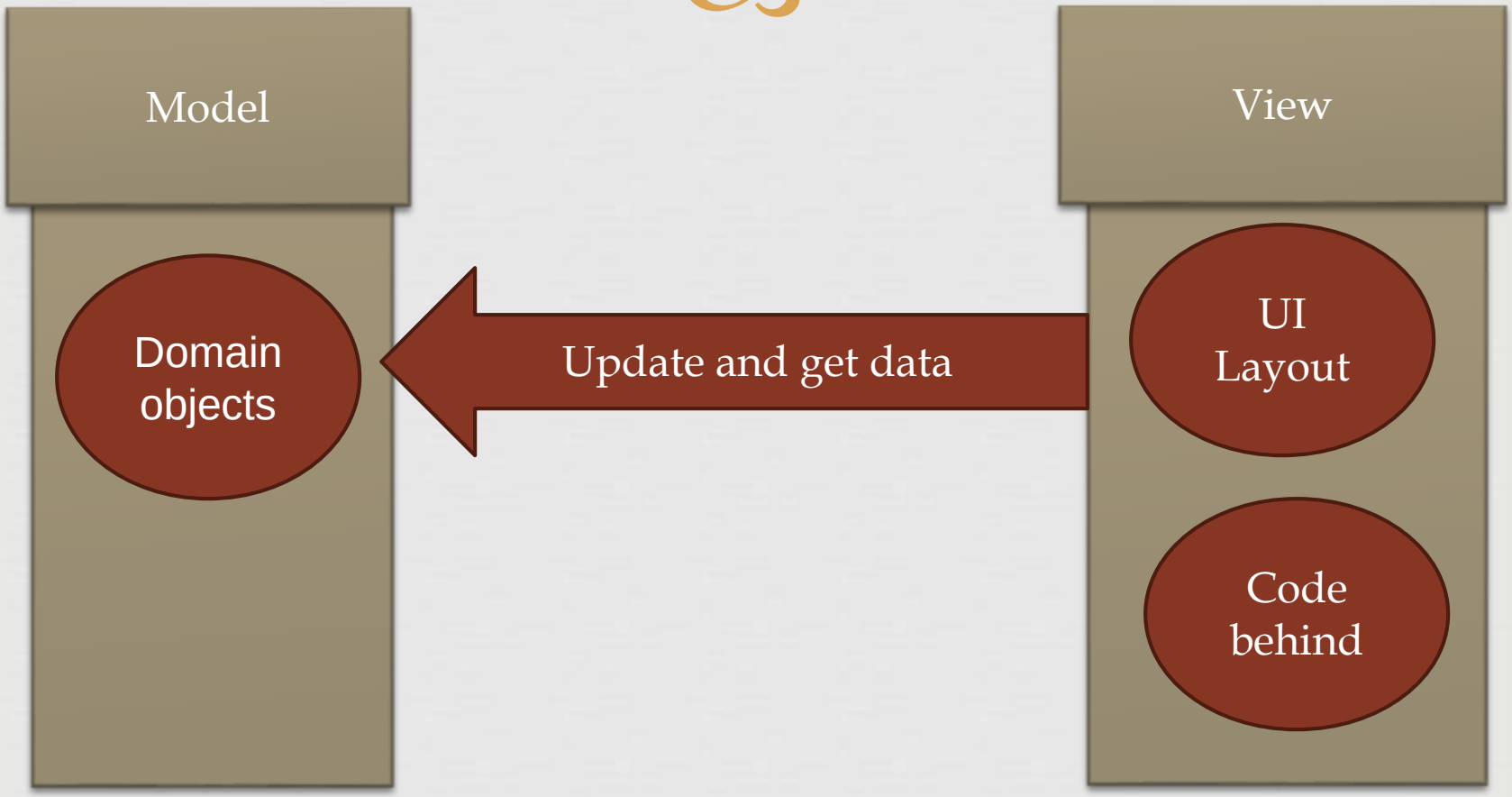


# Traditional UI Development

More work , more problems

```
CountryServie countryService;
// Constructor
public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
    countryService = new CountryServie();
}

void MainPage_Loaded(object sender, RoutedEventArgs e)
{

    List<CountryModel> Countries = countryService.GetCountries();
    foreach (CountryModel country in Countries)
    {
        countriesList.Items.Add(country.Country);
    }
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    string CountryName = (countriesList.SelectedItem ) as string;
    capitaltxt.Text = countryService.GetCapitalName(CountryName);
}
```

# Traditional UI Development



# Traditional UI Development

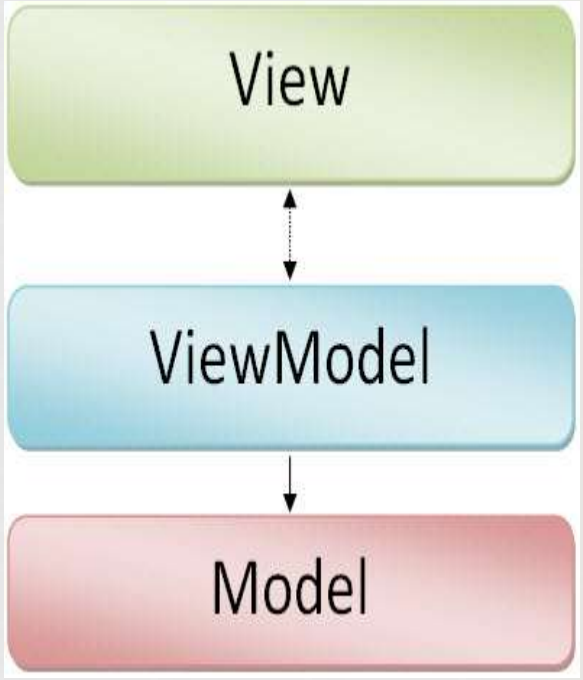
- Constraints:
  - Difficult to Test
  - Encourages using UI as data storage
  - Complex classes and a huge mount of code.
  - No code sharing .
  - very strong dependency between UI & logic.
  - What if you need to redesign UI?

# MVVM in Deep

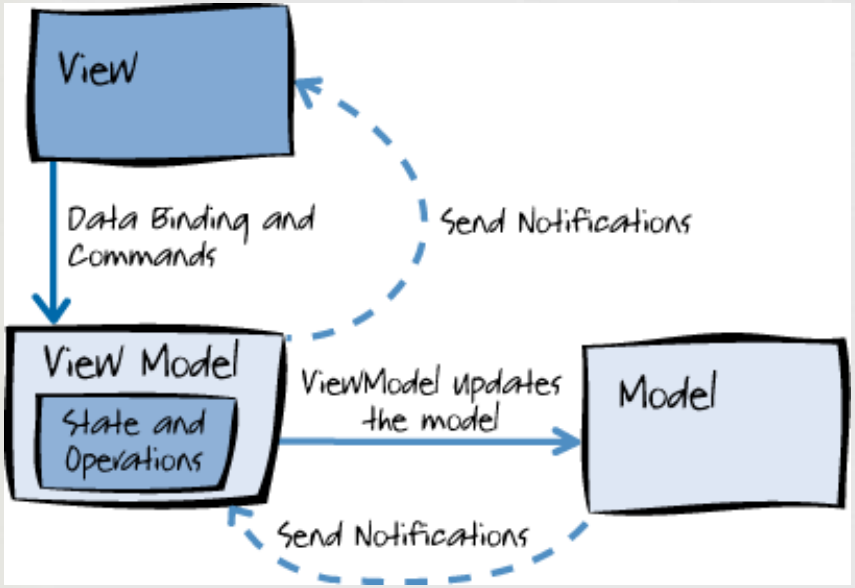
What ? And How?

# What is MVVM?

- The **Model View ViewModel.**
- **Introduced by Microsoft in 2006.**
- Not a design pattern it is an architectural pattern.
- It is a specialization of the presentation model pattern.

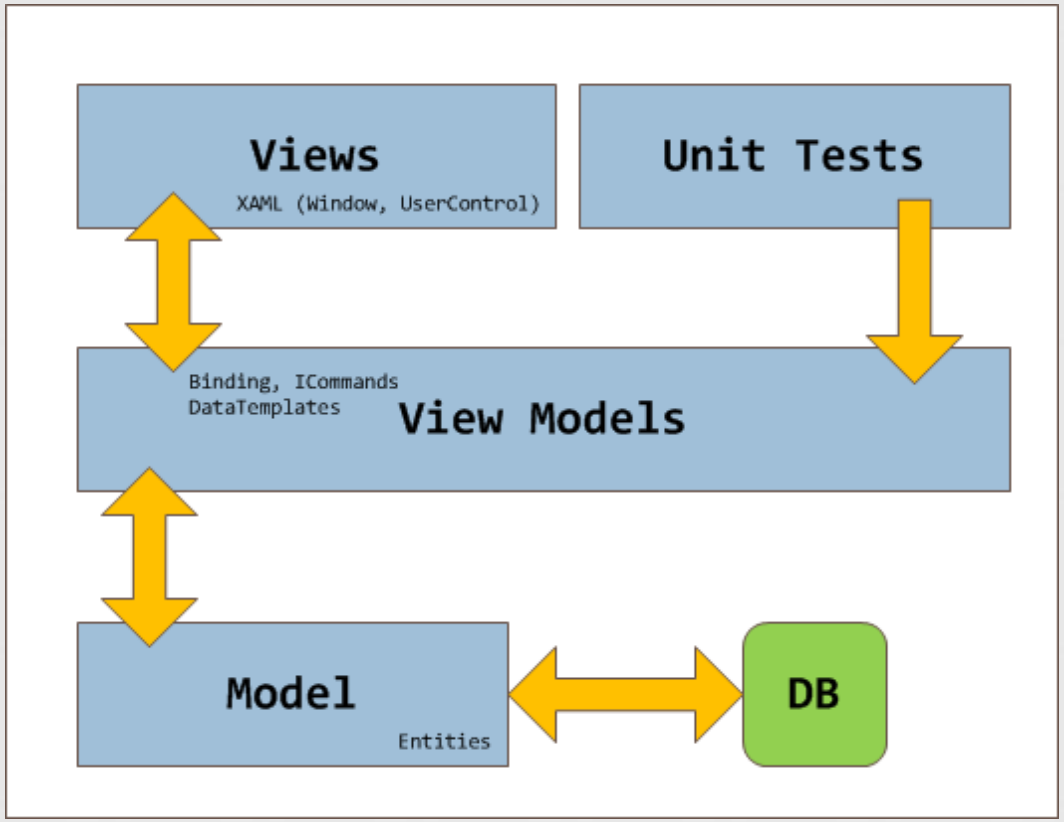


# MVVM In Deep



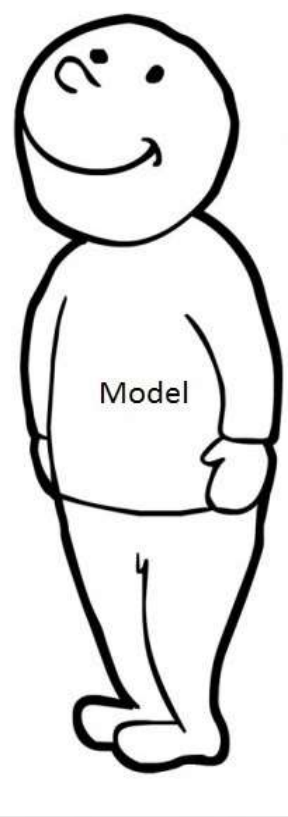

- Components to be swapped
- Internal implementation to be changed without affecting the others
- Components to be worked on independently
- Isolated unit testing

# MVVM In Deep


Views
XAML (Window, UserControl)
Unit Tests
Binding, ICommands
DataTemplates
View Models
Model
Entities
DB

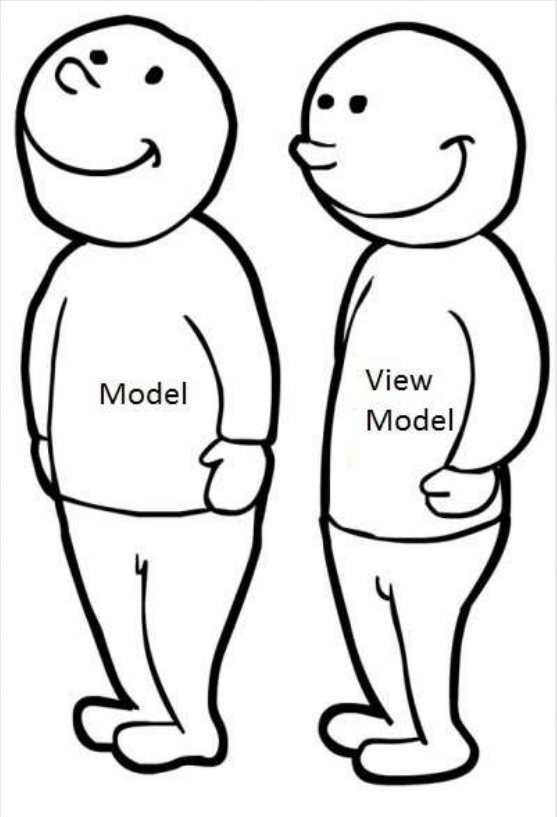
# Model

- Non-visual classes that encapsulate the application's data and business logic.
- Can't See View Model or View.
- Should not contain any use case–specific or user task–specific behavior or application logic.
- Notifies other components of any state changes.
- May provide data validation and error reporting



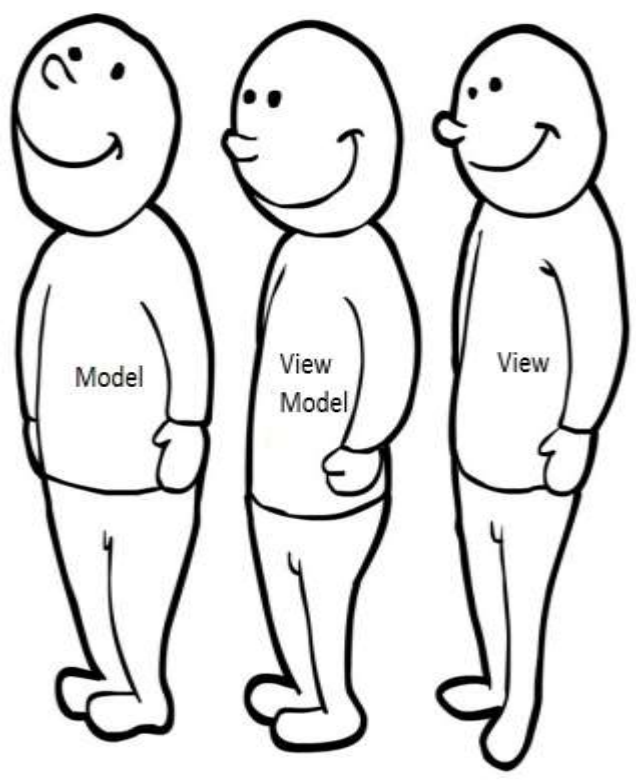
# View Model

- Non-visual class encapsulates the presentation logic required.
- Can't See View (no direct Reference).
- Coordinates the view's interaction with the model.
- May provide data validation and error reporting.
- Notifies the view of any state changes.

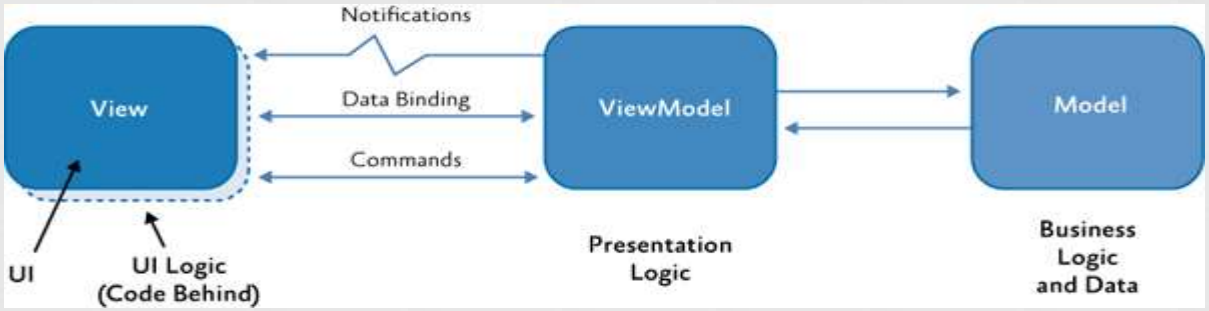


# View

- Visual element defines the controls and their visual layout and styling.
- Can see all others components.
- Defines and handles UI visual behavior, such as animations or transitions.
- Code behind may contain code that requires direct references to the specific UI controls.



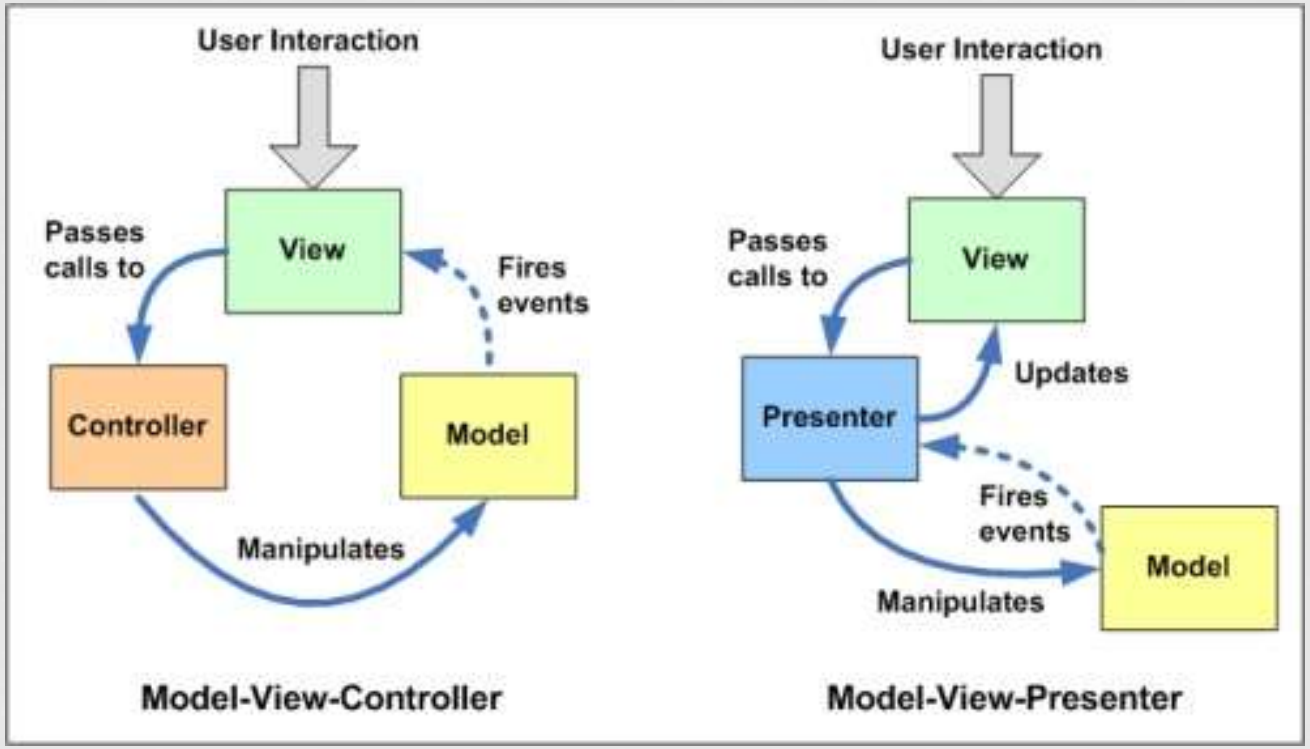
# Classes Interaction




# MVVM and other patterns

What about MVP & MVC ?

# MVP vs. MVC vs. MVVM



# MVP vs. MVC vs. MVVM

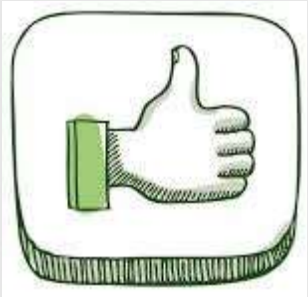
| MVP (Presenter) | MVC (Controller) | MVVM (View Model) |
|---|---|---|
| view communicates with the presenter by directly calling functions on an instance of the presenter. | view sends input events to the controller via a callback or registered handler | View binds directly to the View Model. |
| The presenter communicates with the view by talking to an interface implemented by the view | View receives updates directly from the model without having to go through the controller | changes in view are automatically reflected in View Model and changes and Vice versa. |
| Use where binding via a data context is not possible | Use where the connection between view and the rest of the program is not always available | Use where binding via a data context is possible |
| | | |

# MVVM Advantages & Disadvantags

Why ? And Why not?

# MVVM Advantages

- Separation of concerns.
- Can use unit tests.
- Designers and developers can work synchronously.
- Easy to redesign the UI .
- Can share code easily.



# MVVM Disadvantages

- Harder to debug.
- May affect performance.
- More files to serve the architecture.

# MVVM in Android

- MVVMCross for Xamarin.Android.
- Dot42 Framework. (C# for android).
- Android-binding.
- Robo Binding

# Conclusion & Recommendations

- MVVM is a suitable architectural pattern for business apps.
- Use MVVM only when you need it.
- Using MVVM is highly recommended in Cross-Platform development.
- MVVM is a tool not a purpose.

# References


- http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained
- http://blogs.adobe.com/paulw/archives/2007/10/presentation_pa_3.html
- http://en.wikipedia.org/wiki/Model_View_ViewModel
- http://aspnet.blogershub.com/Archive/2013/12/what-is-difference-Application-Architecture-and-Design-Patterns
- http://msdn.microsoft.com/en-us/library/hh848246.aspx
- http://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh758320.aspx
- http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx
- http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/

# Design Patterns VS. Architectural Patterns

## Design Patterns

- Guidelines to avoid/solve common problems in application development.
- Tell us how we can achieve a solution in terms of implementation.
- Implementation Level.

## Architectural Patterns

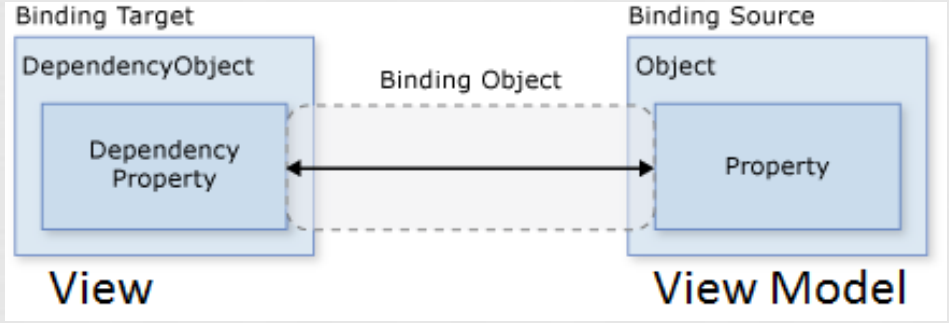
- Guidelines to construct the structure and behavior of applications.
- Tell us how to arrange the interaction between application packages, databases, and middleware systems .
- Abstract Level.

# Presentation Model Pattern

- Introduce by Martin Fowler in 2004.
- An abstract of the view that is not dependent on a specific GUI framework.
- State is in the presentation model.
- Logic is in the presentation model.
- View observes the model and updates accordingly.
- The view “knows” about the presentation model.
- The presentation model does not “know” about the view.
- presentation model Also Known as application model, view state or logical View.

# Data Binding (State)

- Process that establishes a connection between the application UI and application logic.
- When the data changes its value, the elements that are bound to the data reflect changes automatically.



# Data Binding (State)

- **Direction of the data flow**:
  - **One Time** bindings update the target with the source data when the binding is created.
  - **One Way** bindings update the target with the source data when the binding is created, and any time the data changes. This is the default mode.
  - **Two Way** bindings update both the target and the source when either changes.

# Commands (Behavior)

- Provide a way to represent actions or operations that can be easily bound to controls in the UI.
- Encapsulate the actual code that implements the action or operation .
- Examples : Button Click , List selection changed.

# Notifications

- Implementing **INotifyPropertyChanged** and **INotifyCollectionChanged** interface.
- **OnPropertyChanged** method is required to notify binding target properties with change in source.
- **Observable Collection** needed for binding lists source.