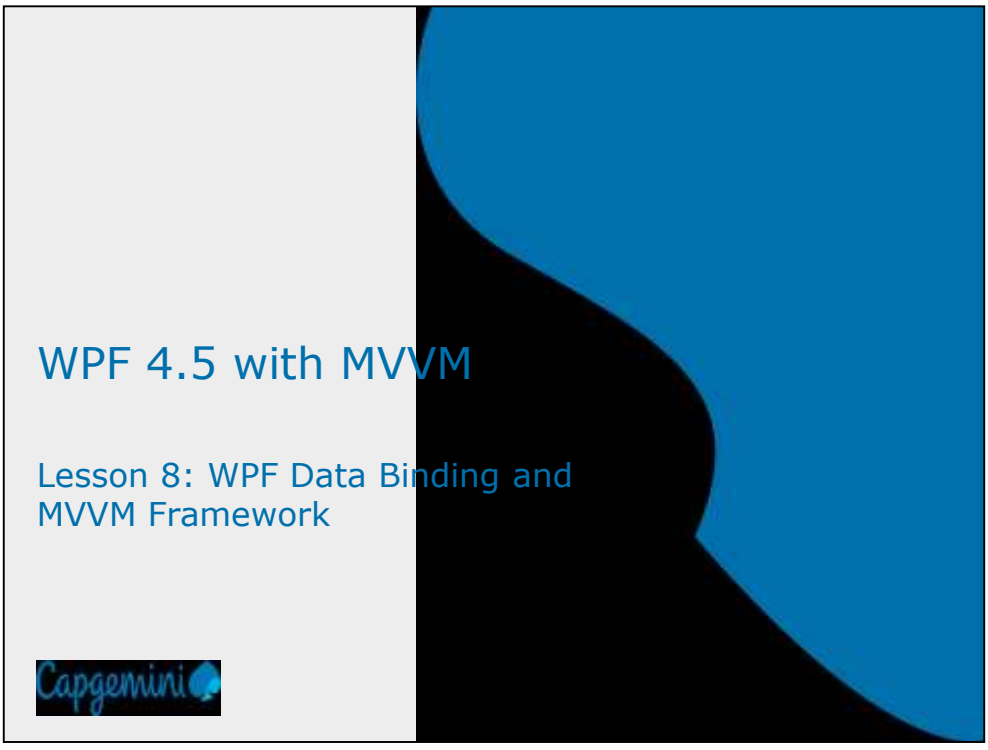
Instructor Notes:



Instructor Notes:

Explain the lesson coverage

Lesson Objectives

- In this lesson, you will learn:
  - MVVM Framework
    - Developing WPF Applications using MVVM Framework

Instructor Notes:

6.1: MVVM Framework

Introduction

- With the increasing popularity of WPF and Silverlight as an application development framework the discussion of patterns has grown widely.
- The majority of developers building WPF and applications have agreed on a pattern that fits well in the WPF world called Model-View-ViewModel (MVVM).
- The MVVM pattern allows applications to be divided up into separate layers.

MVVM is one of several separated presentation patterns that helps you to achieve a separation of concerns between the way things appear on the screen and the supporting data manipulation and logic that supports the screen. MVVM is defined specifically in the context of Silverlight and WPF to fully leverage the capabilities of the framework. Specifically data binding, commands, and behaviors are facilitators for hooking up views and view models in a loosely coupled way, helping to achieve clean separation of concerns.

Instructor Notes:

6.1: MVVM Framework

Introduction (contd..)

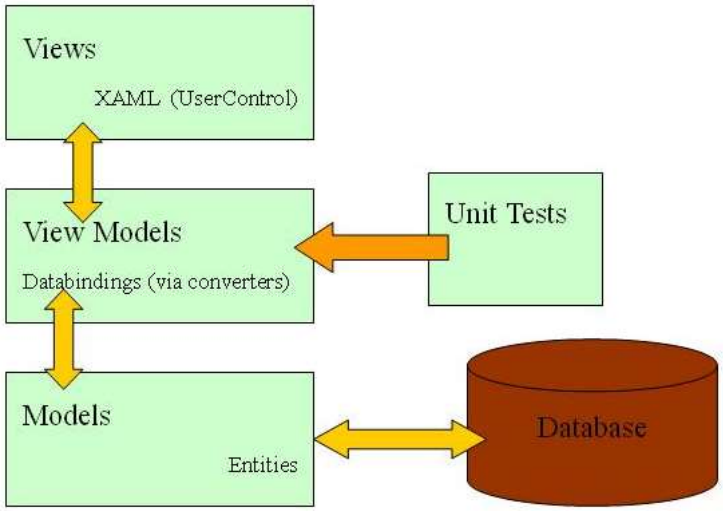
- MVVM pattern provide multiple benefits ranging from better code re-use to enhanced testing capabilities.
- The MVVM Pattern consists of three parts:
  - View - User Interface (WPF Screens)
  - ViewModel - Middle-man between View and Model
  - Model - Business Domain (Business rules, data access, model classes)

You have clean separation of view structure and appearance (the view) from supporting data and logic (the view model), allowing a developer and designer to work concurrently and collaboratively on the same view. And you have better potential for re-use because the view model can be used across both WPF and Silverlight, as well as using a single view model with more than one view definition or vice versa. The logic in the view model is also more testable because it is not coupled to execution in a particular UI context.

Instructor Notes:

6.1: MVVM Framework

Basics



The Model

- •The Model is unaware of anything except itself
- •The Model could be used by any form of program, and is not tied to a specific framework or technology

The View

- •The View is defined entirely in XAML
- •The View only needs to know what to bind to, by name. It does not need to know anything about what will happen when properties change or commands are executed.
- •The View's state is completely based upon data binding

The ViewModel

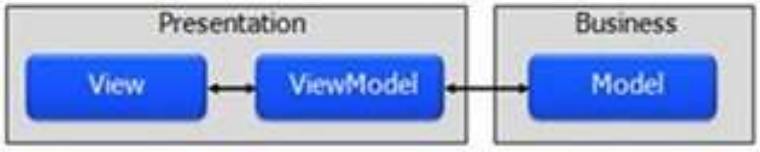
- •The ViewModel knows nothing about the View
- •The ViewModel directly interacts with the Model, in order to expose it for data binding
- •The ViewModel manages the Application-specific information

Instructor Notes:

6.1: MVVM Framework

Basics (contd..)

- The View is bound to the Viewmodel and any change in the Model will automatically be reflected in the View.
- The ViewModel will handle any changes to the Model and receive the events triggers on the View.



MVVM is just more suited to the WPF and Silverlight frameworks because it takes better advantage of the core features of the frameworks to achieve even looser coupling between the view definition and that-other-chunk-of-code-that-supports-the-view (view model / presenter / controller).

Instructor Notes:

6.1: Developing WPF Applications

Overview

- Model represents the business domain which includes the model classes used (Customer, Order, etc.), data access code and business rules.
- The View in MVVM represents the WPF screens that you build.
  - This includes the XAML files and the code-beside files that are responsible for showing data to end users.
- The ViewModel acts as the middle-man between the View and the Model. It is responsible for aggregating and storing data that will be bound to a View.

In general you can think of the Model as representing the entities that live on the server as well as the objects that are responsible for interacting with the data store your application uses and filling entities with data.

The View's responsibilities include displaying data and collecting data from end users. A given View isn't responsible for retrieving data, performing any business rules or validating data.

The ViewModel- A ViewModel may contain a List<State> property and a List<Person> property that may be bound to two ComboBox controls in a View. The ViewModel will retrieve the values held by these two properties from the Model. By using a ViewModel the View doesn't have to worry about retrieving data and knows nothing about where data comes from

Instructor Notes:

6.1: Developing WPF Applications

MVVM and the Service Agent

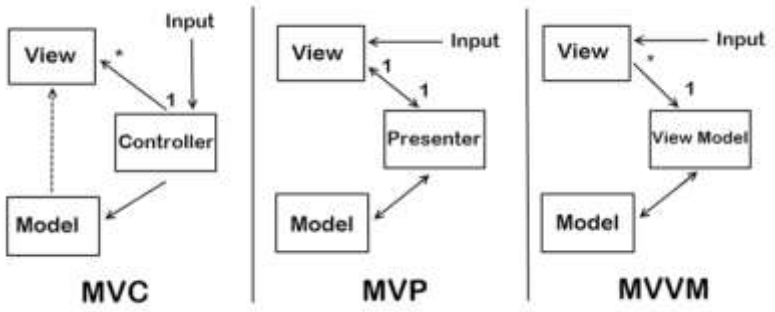
- Additional players may be added into the Model-View-ViewModel mix to help segregate code even further.
  - For example, a service agent class that is responsible for making calls from Silverlight to remote services.
- The service agent is responsible for initiating the service call, capturing the data that's returned and forwarding the data back to the ViewModel.

By doing this the ViewModel classes can delegate data gathering responsibilities to the service agent. A given service agent can also be re-used across multiple ViewModel classes as needed.

Instructor Notes:

6.1: Developing WPF Applications

MVVM compared to MVC and MVP



MVC – Model View Controller

The input is directed at the Controller first, not the view. That input might be coming from a user interacting with a page, but it could also be from simply entering a specific url into a browser. In either case, it is a Controller that is interfaced with to kick off some functionality.

There is a many-to-one relationship between the Controller and the View. That's because a single controller may select different views to be rendered based on the operation being executed.

Note the one way arrow from Controller to View. This is because the View doesn't have any knowledge of or reference to the controller.

The Controller does pass back the Model, so there is knowledge between the View and the expected Model being passed into it, but not the Controller serving it up.

MVP – Model View Presenter

It looks very similar to MVC, except for some key distinctions:

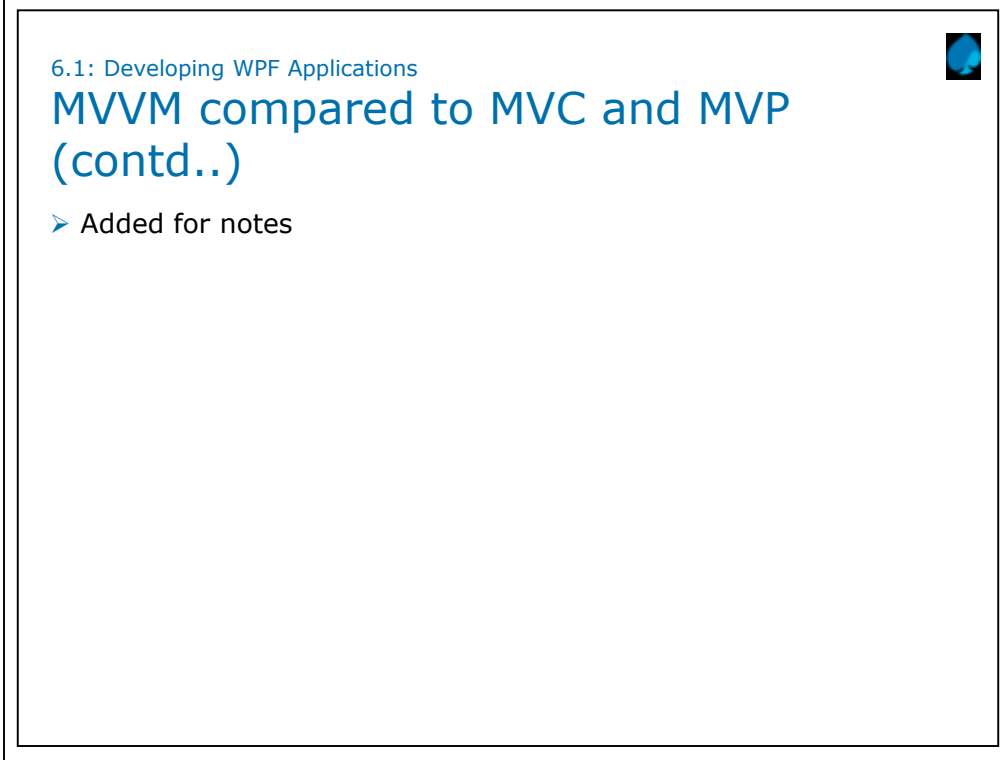
The input begins with the View, not the Presenter.

There is a one-to-one mapping between the View and the associated Presenter.

The View holds a reference to the Presenter. The Presenter is also reacting to events being triggered from the View, so it is aware of the View it is associated with.

The Presenter updates the View based on the requested actions it performs on the Model, but the View is not Model aware.

Instructor Notes:



MVVM – Model View View Model

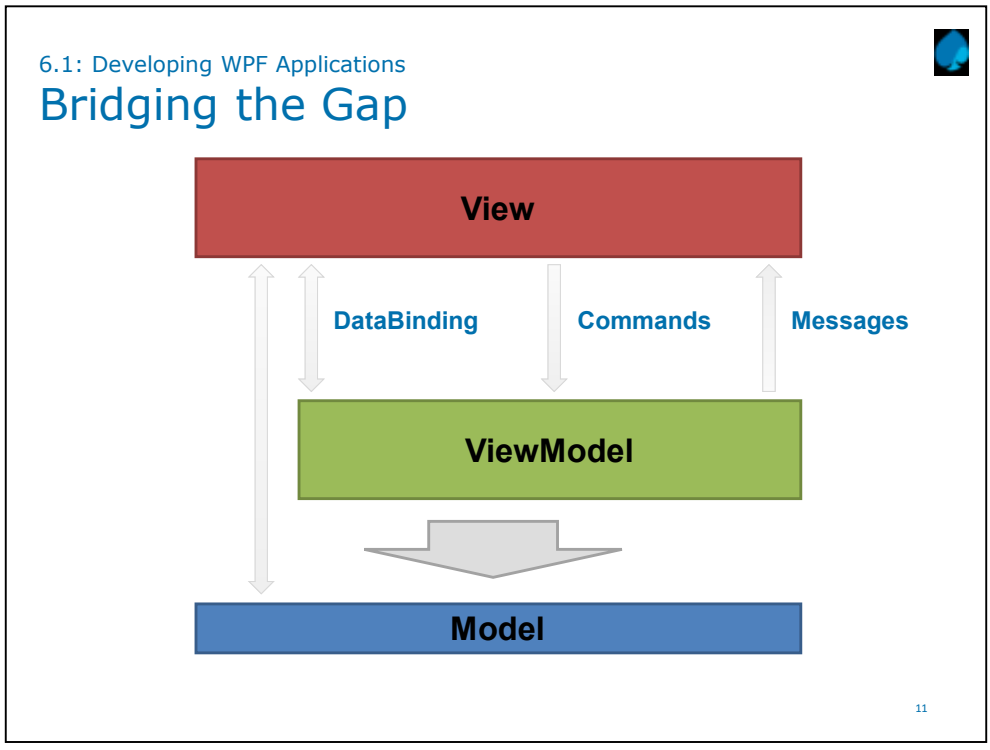
So with the MVC and MVP patterns in front of us, let us look at the MVVM pattern and see what differences it holds:

The input begins with the View, not the View Model.

While the View holds a reference to the View Model, the View Model has no information about the View. This is why its possible to have a one-to-many mapping between various Views and one View Model…even across technologies. For example, a WPF View and a Silverlight View *could* share the same View Model. However, this is a bad practice and creates Franken-ViewModels that have too many responsibilities. It is better to keep it as a one-to-one mapping instead.

You'll also notice that the View has no idea about the Model in the MVVM pattern. This is because, as far as the View knows, it is "Model" IS the View Model (hence its name). Because of how data-binding and other features like commanding work in WPF and Silverlight, there is rich communication between the View and View Model, isolating the View from having to know anything about what's really happening behind the scenes.

Instructor Notes:



Instructor Notes:

6.1: Developing WPF Applications

Demo

- Developing Applications using MVVM Pattern.

Instructor Notes:

Summary

- In this lesson, you have learnt:
  - How to Develop WPF Applications using MVVM Framework.