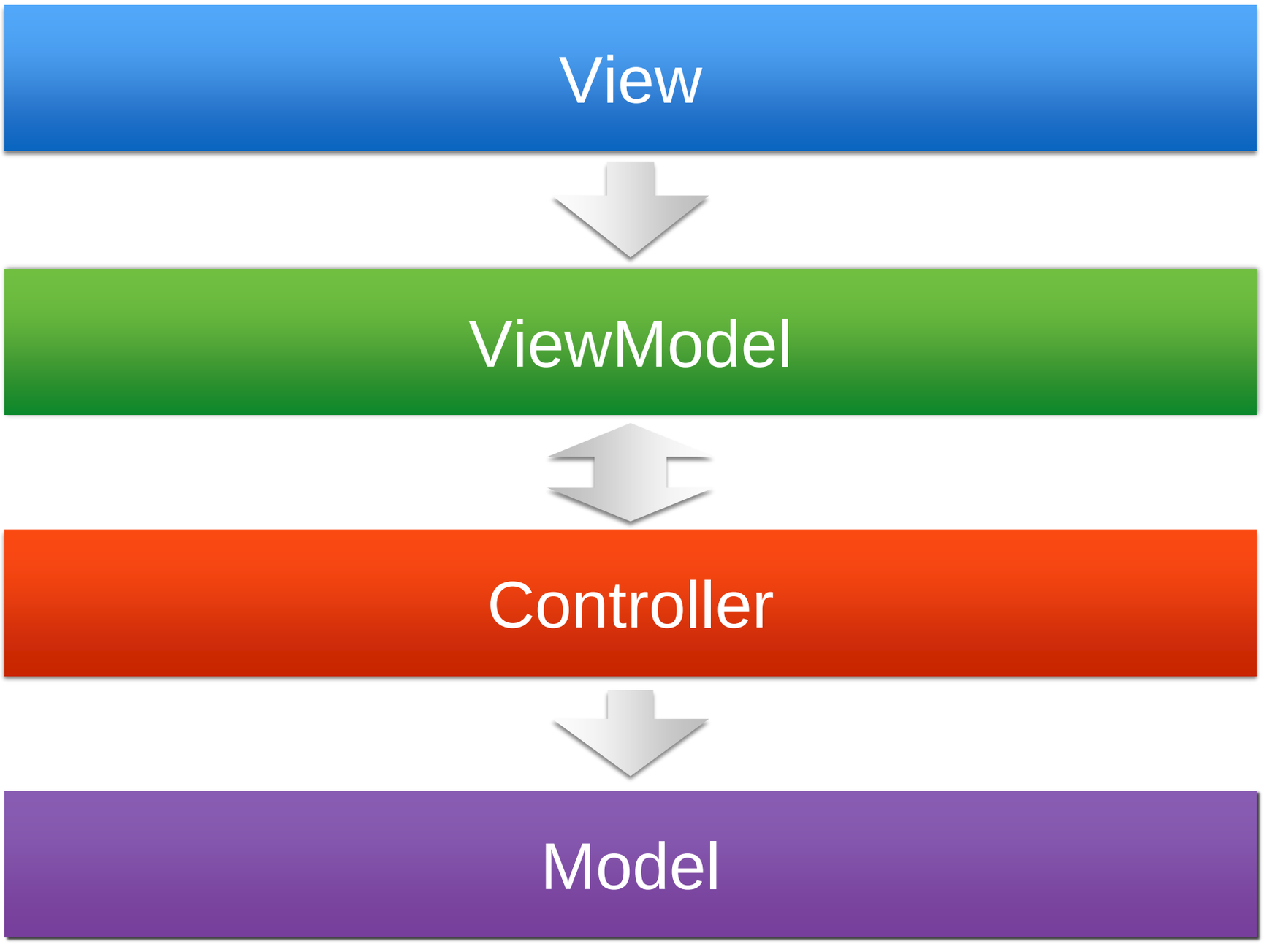
MVVM in practice

How to write easily testable UI code

Agenda

- Separation of layers
- Interaction between layers
- Testing
- Base classes
- Q&A

Separation of layers



View

- Defined mainly in XAML
- Declarative
- Operating on ViewModel's contract
- Blendable

ViewModel

- State of the application
- Unaware of the view definition
- Operating on ViewModel's contract
- Injected to Controller
- Utilising Controller

Controller

- UI logic
- Lightweight business logic
- Coupled with ViewModel by its interface
- Using ViewModel as state
- Highly asynchronous code

Model

- Business layer of application
- Usually defined as service interface
- Not aware of who is the actual consumer

Interaction between layers

Interaction between layers View - ViewModel

- View is aware of ViewModel's interface
- ViewModel doesn't know anything about view
- Interaction with user is handled by Commands
- ViewModel is notifying View through INotifyPropertyChanged, IDataErrorInfo and other WPF interfaces

Interaction between layers ViewModel - Controller

- They depend on each other
- ViewModel holds view / application state
- Controller is responsible for handling interaction, notifications and other events
- This allows Controller to be almost function / reactive code

Interaction between layers Controller -Model

- Model is unaware of who is its consumer
- Controller depends on Model's interface
- Controller subscribes to Model's streams and calls its methods

Testing

Testing - Why it's so important

- Lets developers sleep at night
- Reduces need for overtime
- Reduces stress and other health issues
- Allows developer to produce more code
- Gives developer +1 and Like It! from coworkers

Testing - Interfaces

- Each layer (except from view) should be hidden behind its interface

- This allows classes to be tested independently (unit testing) and simplifies multi-tier testing (integration testing)

- All public methods and properties should be exposed in interface, so developers are sure that classes are interchangeable through their interfaces without hard referencing to concrete types

Testing - How?

- Create all required mocks
- Register everything in test container, including system under test
- SUT is resolved from container through its interface - this forces correct class interfaces
- Model test should implement AAA pattern - Arrange-Act-Assert

Base classes

Show us the code!

Q&A

If anything is left unclear...