

# Model-View-ViewModel (MVVM)

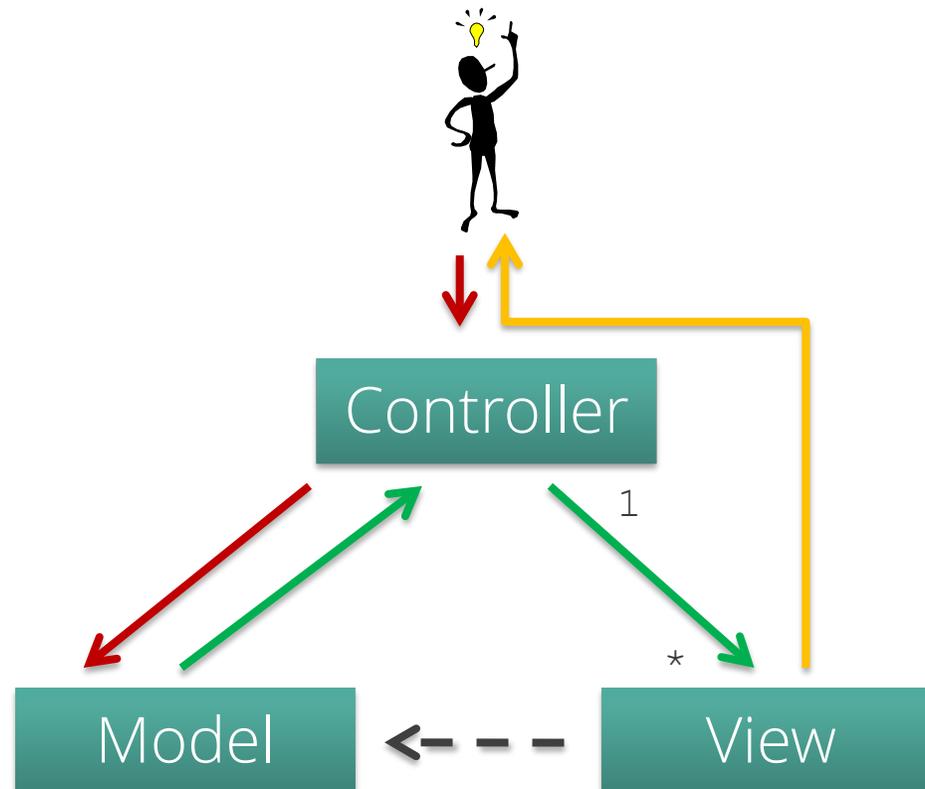
Grundlagen und Einsatz des GUI-  
Architekturmusters

Phillip Conrad

- **Motivation**
  - Architekturmuster
- **MVVM-Pattern**
  - Aufbau & Komponenten
  - Technische Grundlagen
    - Databinding
    - ICommand
  - Entwicklung und Frameworks
  - Praxisbeispiel (WP7)
  - Einsatzgebiete
  - Anti-Pattern
- **Fazit**

- **Was sind Architekturmuster?**
  - Ein Architekturmuster beschreibt die grundlegende Struktur und Organisation einer Anwendung
  - Architekturmuster werden in unterschiedliche Kategorien eingeteilt
  - → **Interaktive Systeme:** Umgang mit Benutzereingaben
    - Model-View-Controller (MVC)
    - Model-View-Presenter (MVP)
    - Model-View-ViewModel (MVVM)

## Model-View-Controller



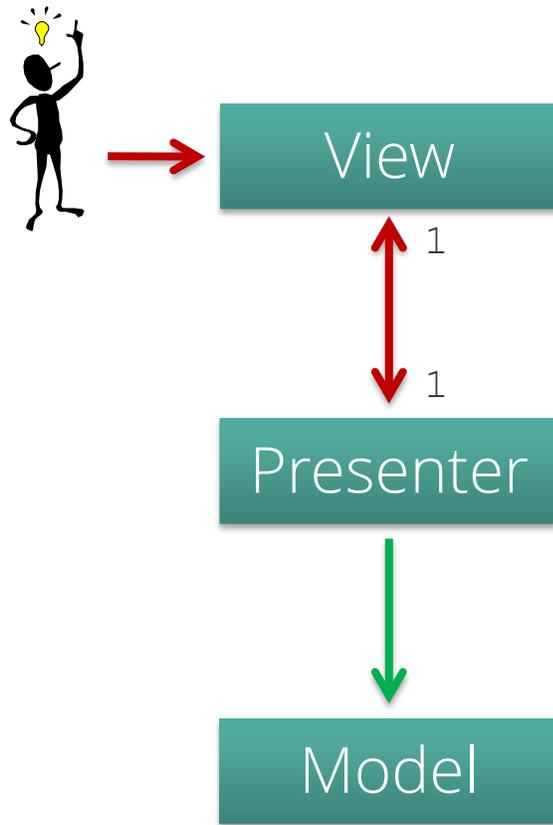
- **Vorteile**

- Aufteilung von Zuständigkeiten
- Feste Kommunikationswege

- **Probleme**

- Die View kennt das Model
- Controller sind schwer austauschbar
- Unit-Tests sind kritisch
- Schwergewichtig: Overkill

## Model-View-Presenter



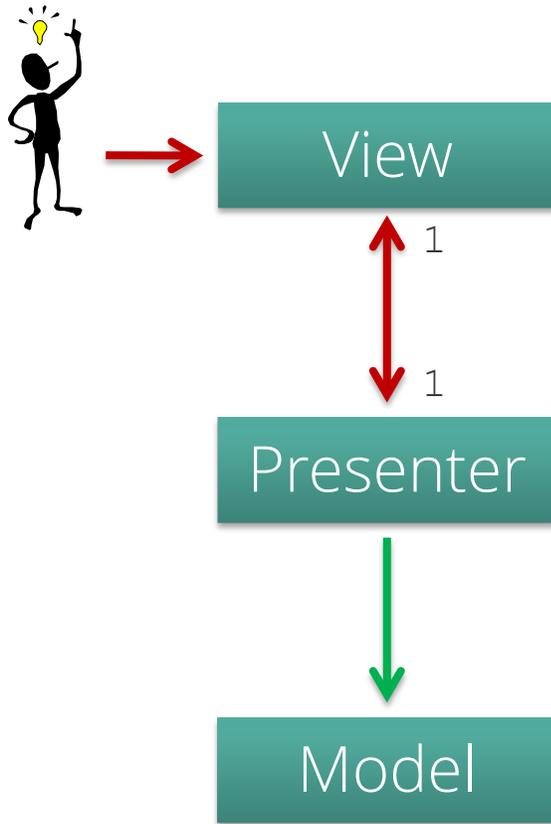
- **Verbesserung**

- Views sind passiv, keinerlei steuernde Logik
- Bessere Testbarkeit → Präsentationslogik im Presenter

- **Probleme**

- Veränderungen in der View haben i.d.R. Auswirkungen auf den Presenter (enge Kopplung)
- Schwergewichtige Presenter

## Model-View-Presenter



- **Verbesserung**

- Views sind passiv, keinerlei steuernde Logik
- Bessere Testbarkeit → Präsentationslogik im Presenter

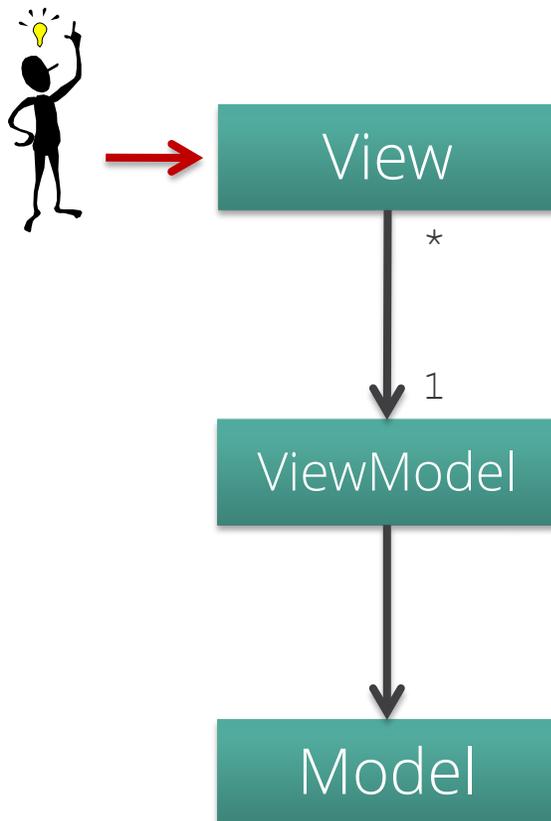
- **Probleme**

- Veränderungen in der View haben i.d.R. Auswirkungen auf den Presenter (enge Kopplung)
- Schwergewichtige Presenter

Mögliche Lösung der Probleme von MVC & MVP

# Model-View-ViewModel

## Model-View-ViewModel



- **Vorteile gegenüber MVC & MVP**

- Schwache Kopplung zwischen View und View-Model
- Command-basierte Interaktion
- Testbarkeit
- Austauschbare Views und View-Models

- **Probleme**

- Speziell für Silverlight & WPF
- Schwierigkeiten mit Databinding

- **MVVM-Triade**

- **Model – die Datenhaltung**

- Repräsentiert die Daten
    - Abstrahiert von der Datenquelle (WCF-Dienst; RIA-Dienst, REST/JSON-Quelle,...)
    - Kann Validierungsmechanismen beinhalten

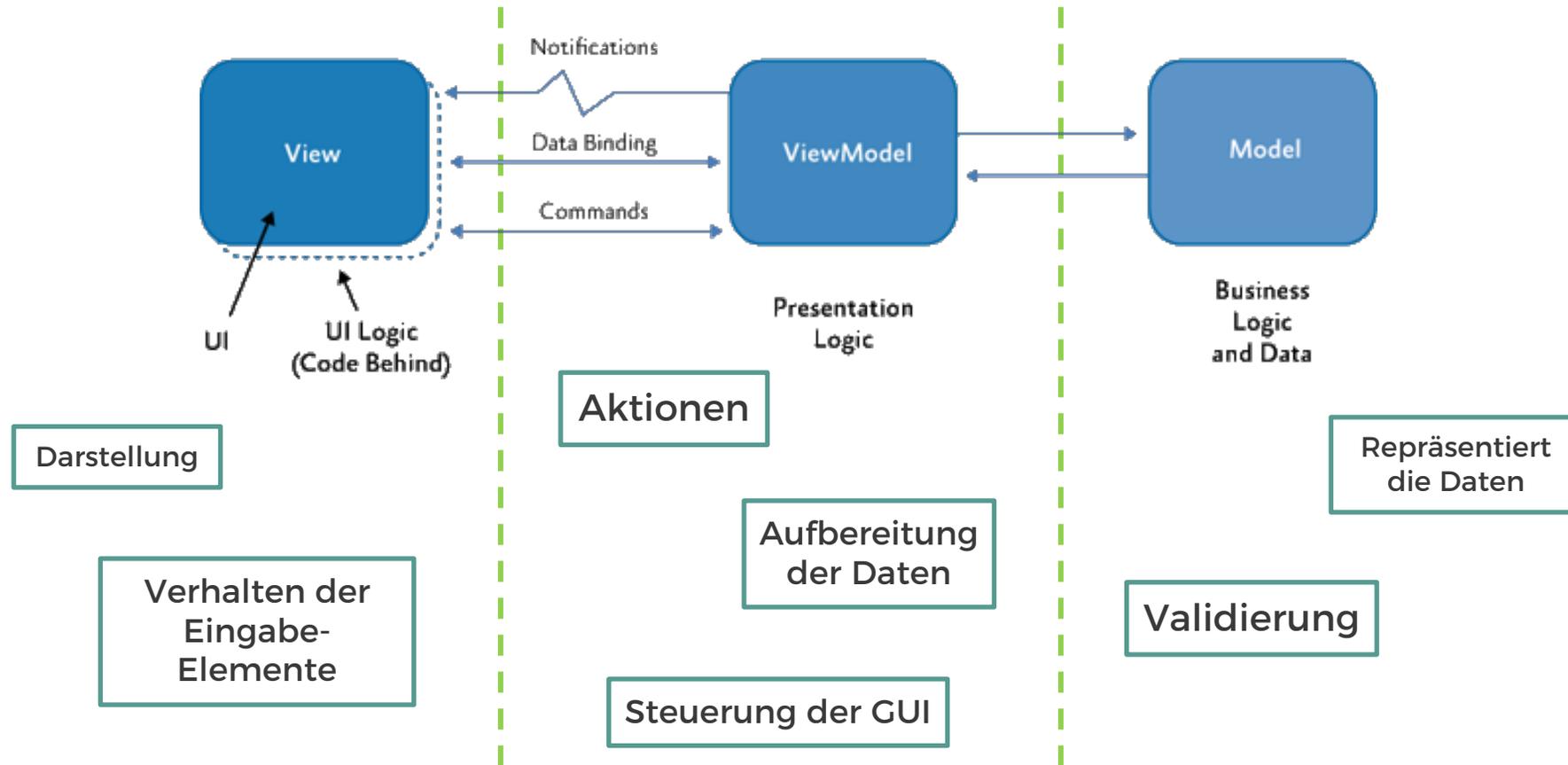
- **View – das Userinterface**

- Realisiert das Look & Feel
    - Stellt die Informationen dar
    - Kommuniziert mit dem ViewModel bei Veränderungen
    - Umsetzung mit XAML und Code-Behind

- **ViewModel – die Brücke zwischen View und Model**

- Beinhaltet die Präsentationslogik
    - Informiert die View über Veränderung der Daten (INotifyPropertyChanged)
    - Reagiert auf Benutzeraktionen (ICommand)

## Separation of Concerns



- **Grundlagen von MVVM: Databinding**

- Verbindung zwischen View und ViewModel
  - Die View referenziert das ViewModel
  - Dies erfolgt über die DataContext-Eigenschaft von der View

Variante 1

```
this.DataContext = new MainViewModel();
```

Variante 2

```
<phone:PhoneApplicationPage.DataContext>  
    <viewModels:MainViewModel />  
</phone:PhoneApplicationPage.DataContext>
```

Variante 3

```
this.DataContext = ViewModelLocator.Find<IMainViewModel>();
```

- **Grundlagen von MVVM: Databinding**

- In Silverlight und WPF wird ein Databinding-Konzept angeboten
- Durch Databinding können Objekte an GUI-Elemente gebunden werden

```
<TextBox Text="{Binding Path=Username, Mode=TwoWay}"  
        Height="25" TextWrapping="NoWrap" Margin="1" />
```

- **Binding**

- Path
  - Gibt den Namen der zu bindenden Property an
- Mode
  - OneWay: View ← ViewModel
  - TwoWay: View ↔ ViewModel

- **Grundlagen von MVVM: Änderungsbenachrichtigung**

- Einsatz des Observer-Musters
- Das ViewModel stellt dabei das „Subject“ dar
- In .NET wird hierfür das INotifyPropertyChanged-Interface implementiert

```
public class LoginViewModel : INotifyPropertyChanged
{
    public ObservableCollection<UserViewModel> Users;

    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(string propertyName) {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    private String _Username;
    public String Username {
        get { return _Username; }
        set {
            _Username = value;
            NotifyPropertyChanged("Username");
        }
    }
}
```

- **Grundlagen von MVVM: Command-Binding**
  - Über die ICommand-Schnittstelle ist es möglich, Aktion zu binden
  - Mögliche Implementierung:

```
public class Command : ICommand /* vereinfacht */
{
    private readonly Action _handler;
    private bool _isEnabled;
    public event EventHandler CanExecuteChanged;

    public Command(Action handler)
    {
        _handler = handler;
        _isEnabled = true;
    }

    public bool CanExecute(object parameter) {
        return _isEnabled;
    }

    public void Execute(object parameter) {
        _handler();
    }
}
```

- **Grundlagen von MVVM: Command-Binding**
  - Einsatzmöglichkeiten von ICommand

## Command-Binding in der View

```
<Button Command="{Binding LoginCommand}" Content="Login"/>

//Eine Alternative mit Code-Behind (WP7)
<Button Content="Login" Click="Button_Click"/>

private void Button_Click(object sender, RoutedEventArgs e)
{
    _viewModel.LoginButton.Execute(null);
}
```

## Ereignis-Senke im ViewModel

```
public ICommand LoginButton { get; set; }
public LoginViewModel()
{
    LoginButton = new Command(OnClickLoginButton);
}

private void OnClickLoginButton()
{
    //Do Something
}
```

- **Entwicklung von MVVM**

- Ursprung

- Presentation-Model Design-Pattern von Martin Fowler (2004)

- Einsatz in WPF und Silverlight

- John Gossman spricht erstmals vom MVVM-Pattern (2005)

- Erste Anleitung für eine MVVM-basierte Architektur

- Prism: Patterns for Building Composite Applications (2008)

- **MVVM-Frameworks**

- MVVM Light Toolkit (Laurent Bugnion)

- Caliburn Micro (Rob Eisenberg)

- MVVM Foundation (J. Smith)

# Model-View-ViewModel



- **Einsatz von MVVM**
  - Speziell für WPF-basierten Techniken
    - Silverlight; Silverlight for Windows Phone; Window Presentation Foundation
  - Abhängig vom Databinding-Konzept
    - DataContext (Property-Binding)
    - Command-Binding (DelegateCommand)
    - Event-basierten Benachrichtigung
- **Model-View-ViewModel mit anderen Technologien?**
  - Presentation-Model-Pattern + Observer-Pattern
  - Knockout.js: MVVM-Framework für JavaScript
    - → <http://learn.knockoutjs.com>

- Einsatz von Value-Converter

```
<UserControl.Resources>  
    <converter:VisibilityConverter x:Key="VisibilityConverter"/>  
</UserControl.Resources>
```

```
<Image Source="{Binding Logo}"  
    Visibility="{Binding IsLogoSet, Converter={StaticResource VisibilityConverter}}" />
```

- Instanzieren von View-Elementen aus dem ViewModel

```
public void OnClickLoginButton()  
{  
    MessageBox.Show("Hello World");  
}
```

- **Stärken von MVVM**

- Präsentation und Präsentations-Logik sind strikter getrennt
- Bessere Aufteilung der Arbeit zwischen Entwickler und Designer möglich
- Einfache Unit Tests gegen das ViewModel möglich
- Hohe Wartbarkeit

- **Probleme von MVVM**

- Databinding stellt eine Blackbox dar
- Höherer Kommunikationsaufwand
- Höherer Entwicklungsaufwand

<http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>

<http://martinfowler.com/eaaDev/PresentationModel.html>

<http://msdn.microsoft.com/en-us/magazine/ff798279.aspx>

<http://www.wpftutorial.net/MVVM.html>

<http://compositewpf.codeplex.com/releases/view/55580>

<http://knockoutjs.com/>



### Hauptsitz in Deutschland:

🏠 SMF GmbH  
Paul-Henri-Spaak-Straße 5  
44263 Dortmund

✉️ [info@smf.de](mailto:info@smf.de)

☎️ +49 231 9644-0

🌐 [smf.de](http://smf.de)

### Weitere Standorte:

Großbritannien | Polen | Slowenien | Serbien

