



**Stefan Lange** | empira Software GmbH

# WPF Anwendungsarchitektur

# Agenda

- Architektur von WPF Anwendungen
- Wozu überhaupt Architektur?
- Einfluss von WPF auf Anwendungsdesign
- Composite Application Guidance:  
Enterprise Framework (WPF / Silverlight)
- Von WPF nach Silverlight 3

# Viele neue Paradigmen in WPF

- Code + XAML
- Datenbindung
- Control-Templates & Styles
- Data-Templates
- Animationen
- Audio / Video
- Effekte
- 3D Grafik
- u.v.a.m

# XAML und Code Behind

- Applications = Code + Markup ?

**Code**  
.net Sprache /  
Code Behind

+

**Markup**  
XAML

Problem: Zweck der Code  
Behind Datei wird  
überbewertet

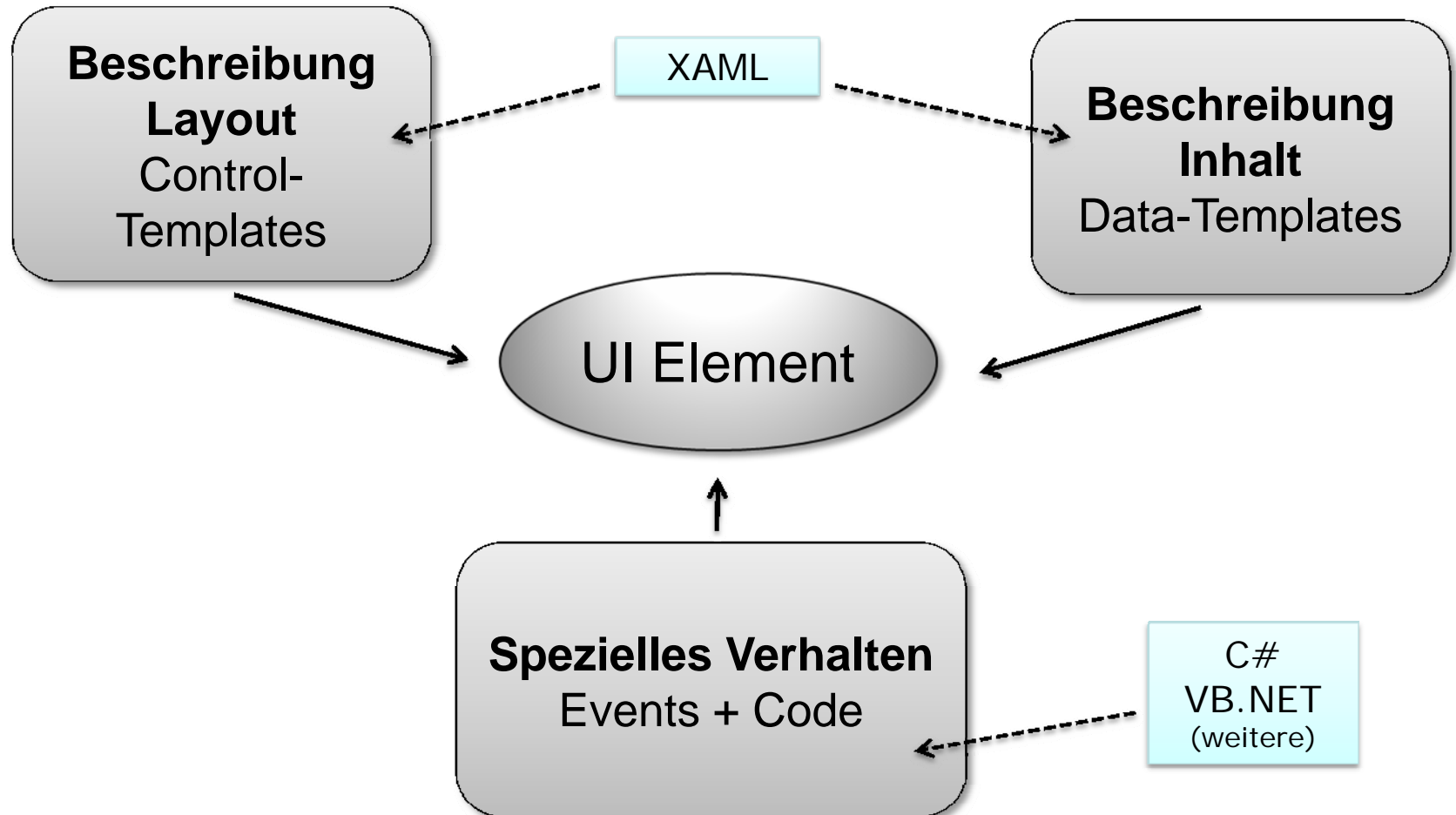
Ursprünglich:  
XML *Avalon* Markup  
Language

# XAML

- „Dynamische Sprache“ für Oberfläche
  - Gut geeignet für Designer
  - Trennt UI-Beschreibung und Code
- 
- Problem: Oftmals wandern wesentliche Teile der Applikationslogik in die Code Behind Datei ☹

# Vollkommene Trennung

- UI Elemente haben 3 Aspekte

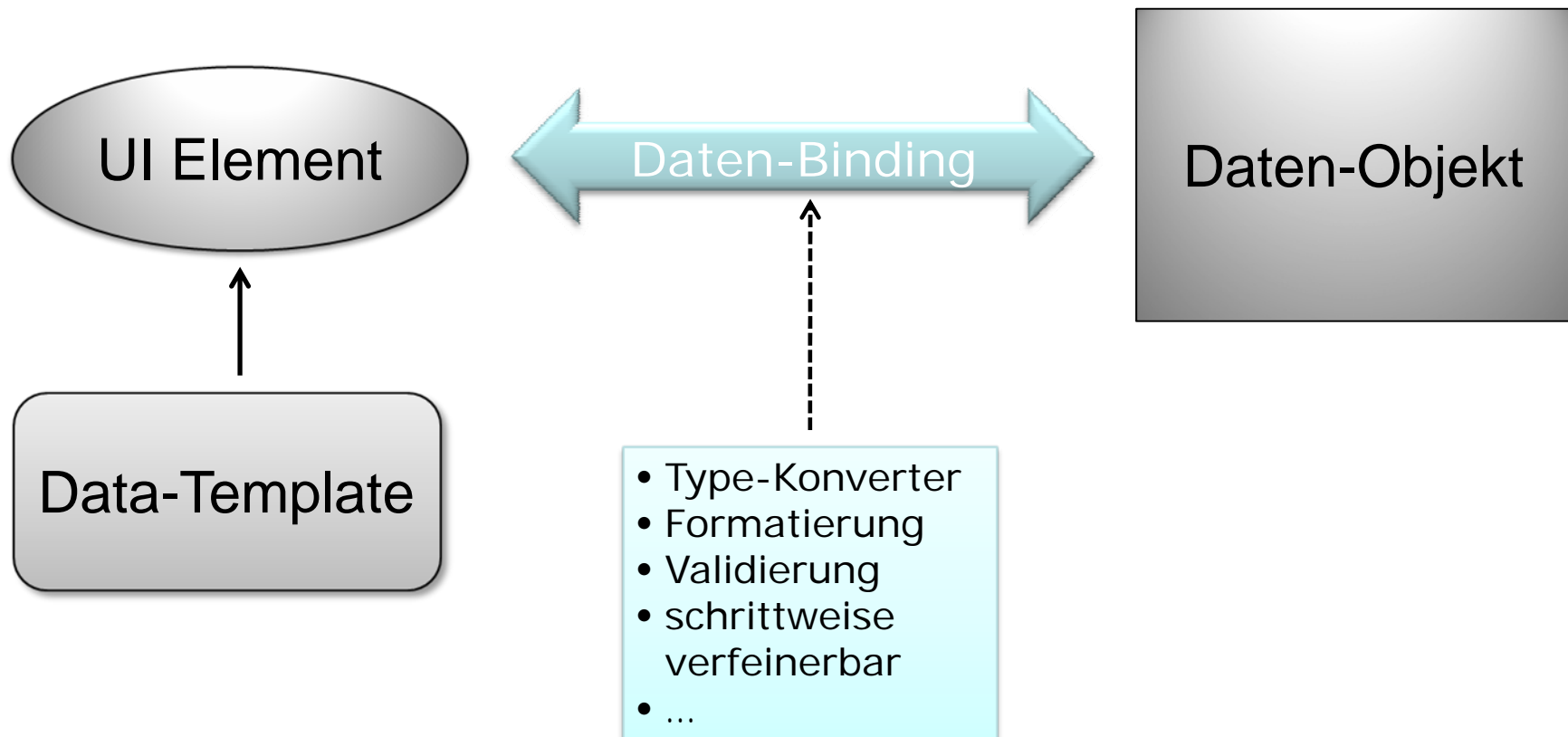


# Push vs. Pull

- Code „drückt“ Daten in die Oberfläche
  - z.B.: `Listbox.AddItem(...)`
  - enge Kopplung von Code und UI
  - sollte in WPF nicht verwendet werden
  
- UI „zieht“ die Daten selbst
  - d.h. Databinding
  - lockere Kopplung von UI und Daten
  - UI kann leicht geändert werden
  - bevorzugte Methode in WPF

# Daten ins UI

- UI Elemente holen ihre Daten selber

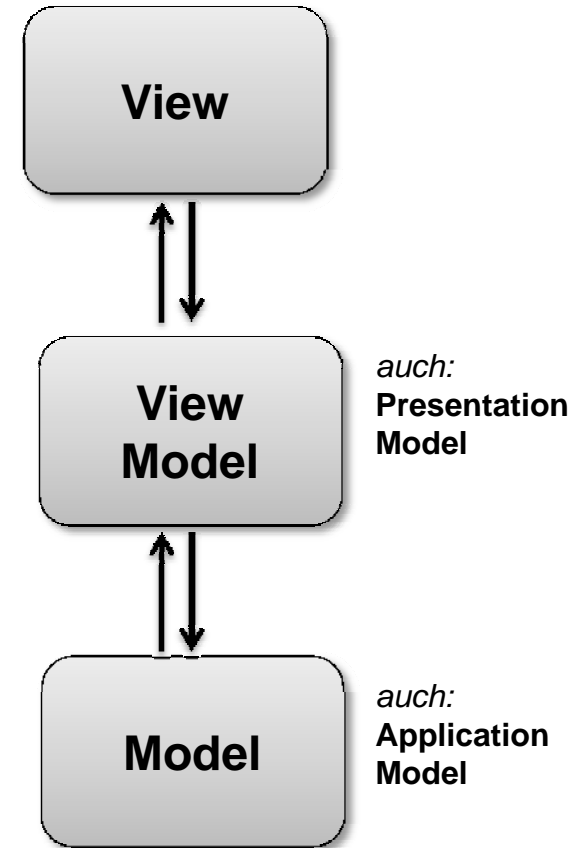
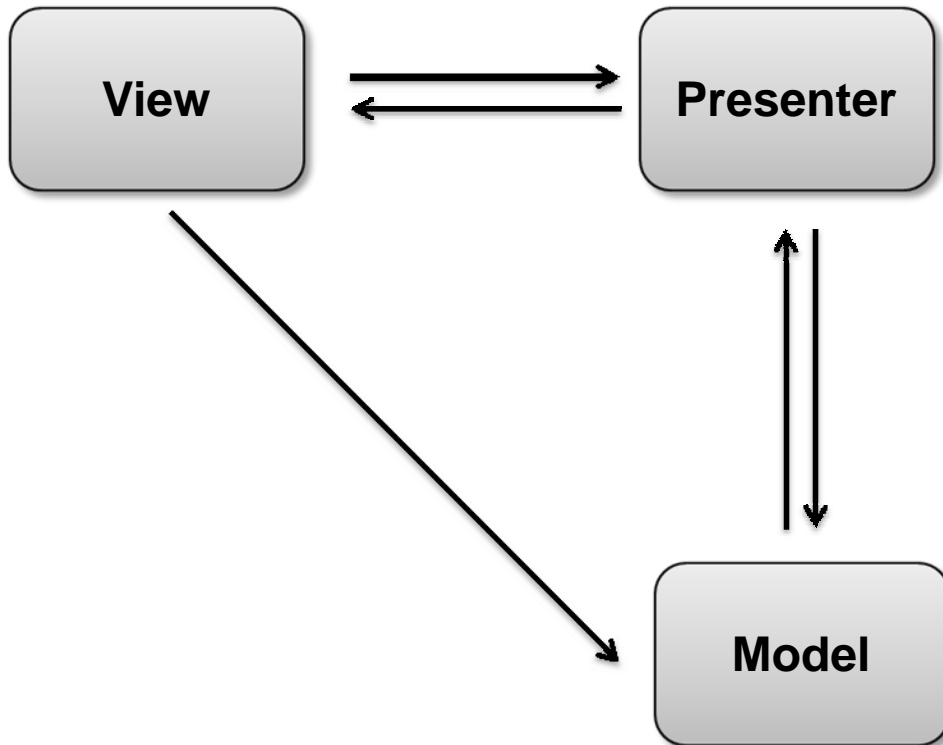




# CODE BEISPIEL 1

# MVP vs. MVVM

- Beides möglich; empfohlen MVVM



- Auch möglich: MVVM + Presenter

# CODE BEISPIEL 2

# Model - View - ViewModel

- Bevorzugtes Paradigma für WPF
- Modellierung der Daten passend zur Präsentation
- ViewModel kann wiederverwendet werden
- ViewModel unabhängig vom UI
- Code Behind Datei ist **kein** ViewModel

# Eigenschaften View Model

- Klassen eines ViewModels
  - Implementieren INotifyPropertyChanged
  - Ggf. abgeleitet von DependencyObject
  - Implementieren ggf. Dependency Properties
  - Verwenden ObservableCollection
  - Lösen in Settern PropertyChanged Event aus
  - Laufen ausschließlich im UI-Thread
  - Verarbeiten Events vom UI
  - Validierung des UI
  - Lesen und schreiben Daten aus/ins Model
  - ...

# Eigenschaften Application Model

- Klassen des Application Models
  - Modellieren die Funktionalität der Anwendung unabhängig von einem konkreten UI
  - Verwenden keine UI Funktionalität
  - Lesen und schreiben Daten von/in Data Provider (Datenbank, Web-Service, ...)
  - Laufen ggf. in Background-Thread (asynchron)
  - Können automatisiert getestet werden
  - Validieren die Daten im Gesamtkontext
  - ...

# Tipps zum Application Model

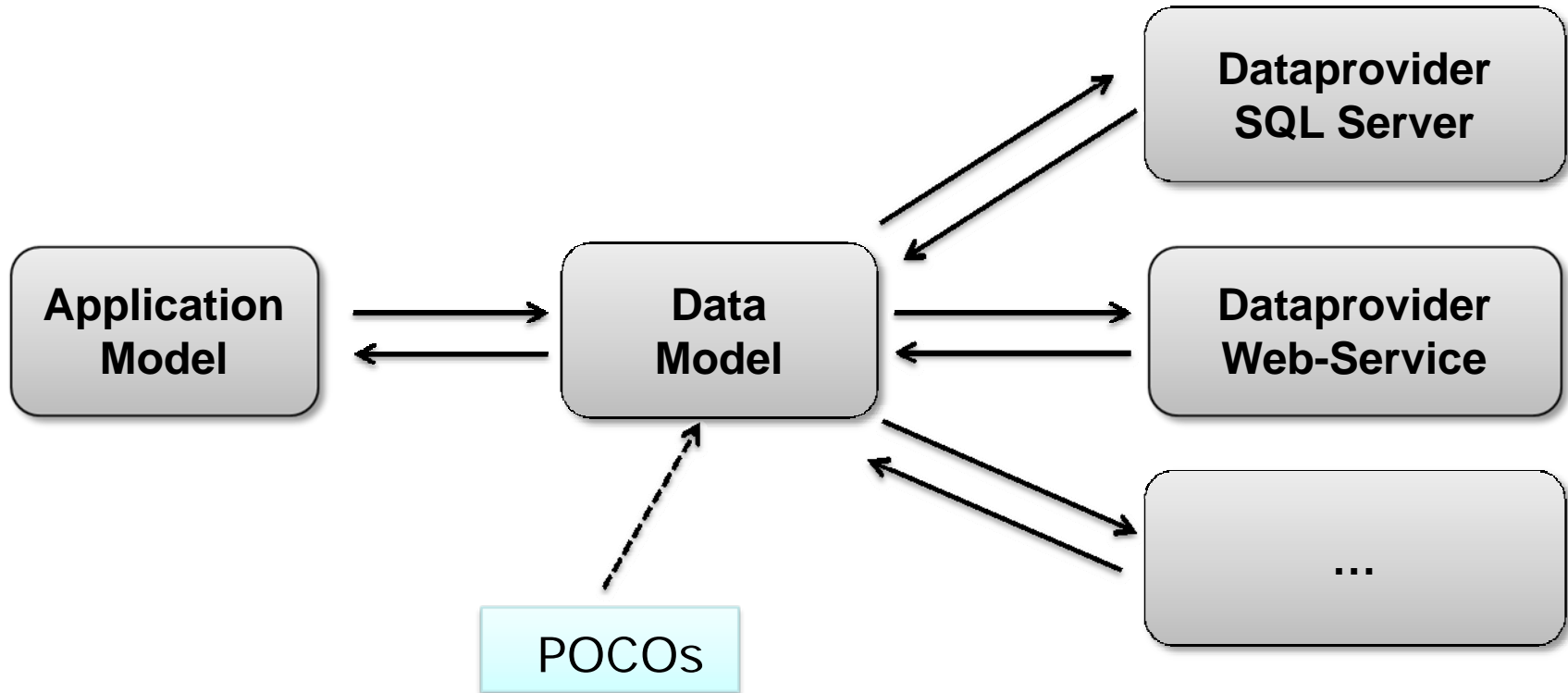
- Direkt bei Projektbeginn
  - Application Model in eigene Assembly
  - Tests auf dieser Assembly schreiben
- Application Model wiederverwenden
- Keinen UI oder Datenbank Code verwenden

# CODE BEISPIEL 3



# Jenseits des Application Models

- Datenbankschicht



# Data Model

- Klassen des Data-Models
  - „Plain old CLR Objects“ (POCO)
  - Modellieren beispielsweise Datenbank Entitäten
  - Lassen sich serialisieren
  - ...
- Code wird oft generiert (OR-Mapping)
  - Nhibernate
  - ADO.NET Entity Framework
  - diverse weitere OR-Mapper

# Unterschiede in den Schichten

- Die einzelnen Schichten haben unterschiedliche Zuständigkeiten
- Sie ähneln sich nur am Anfang eines Projekt und „wachsen später auseinander“

# Beispiel

- Hotelreservierungssystem:  
Ausstattung eines Hotelzimmers
  - Hotel bietet verschiedene Zimmerkategorien
  - Zimmer haben verschiedene Extras

# Datenbank

- Tabellenstruktur

**Tabelle:  
Zimmer**

Tabelle mit  
allen Zimmern

**Tabelle:  
Extras**

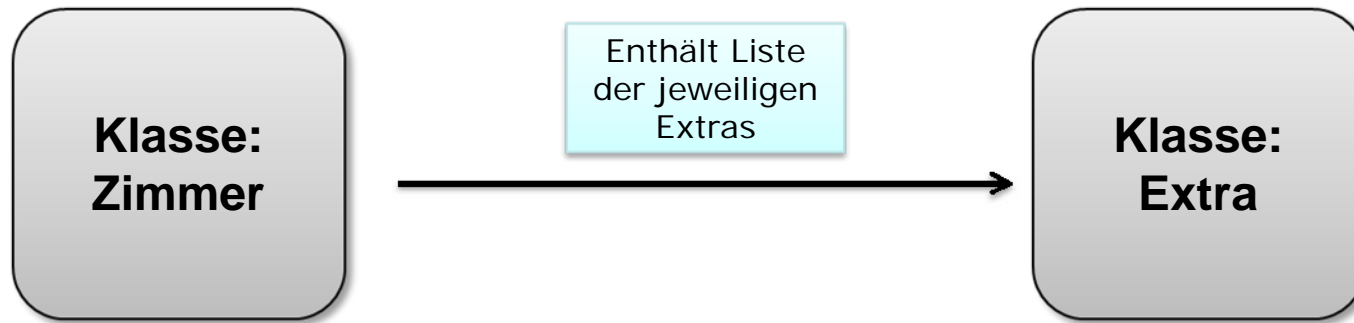
Tabelle mit  
allen  
Ausstattungs-  
merkmalen

**Tabelle:  
Zimmer  
x  
Extras**

Kreuztabelle  
mit den Extras  
der Zimmer

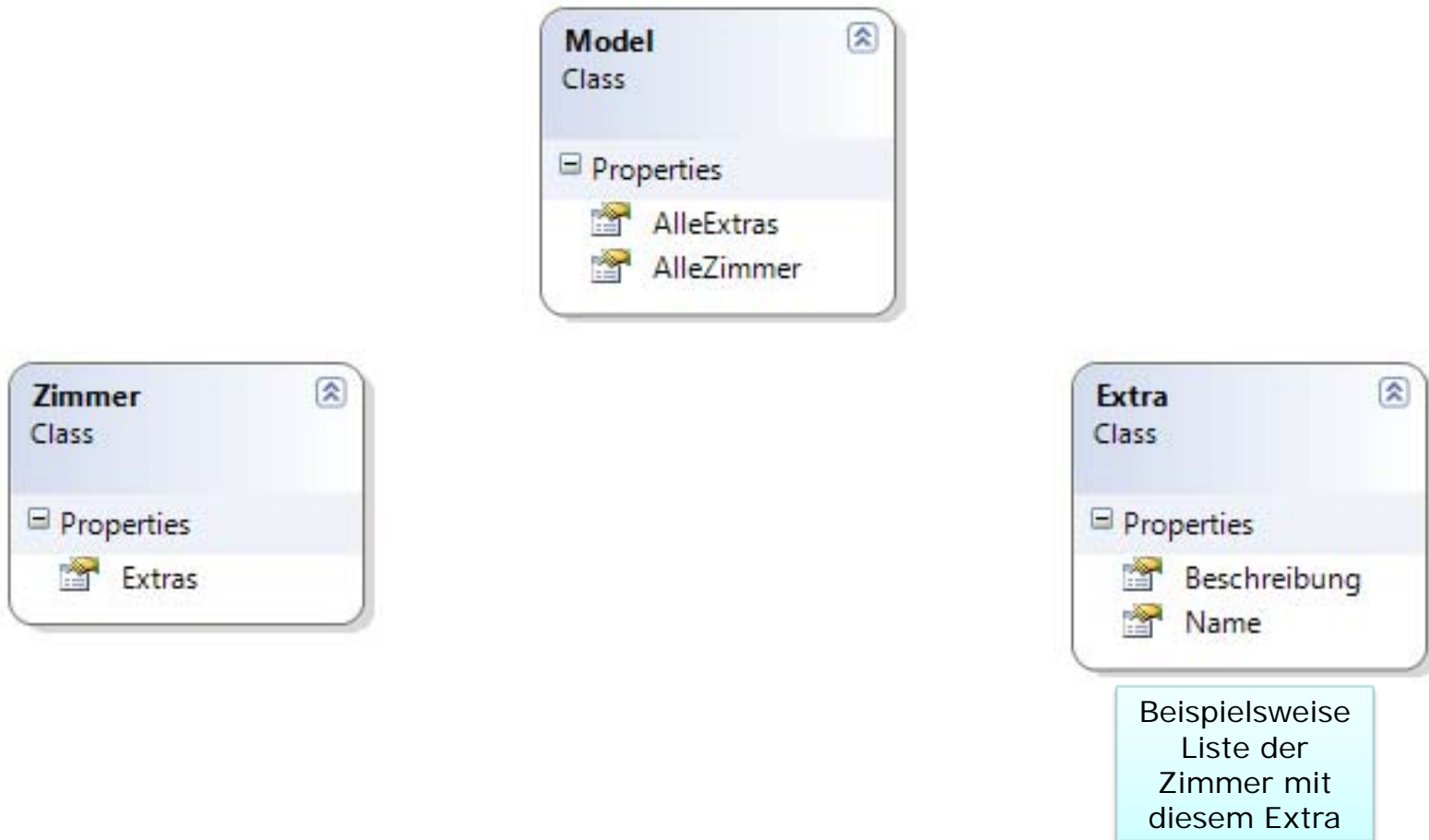
# Data Model

- POCOs (Plain old CLR Objects)



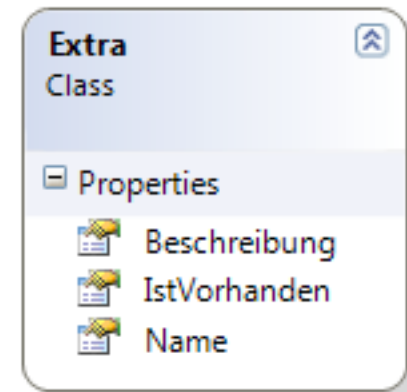
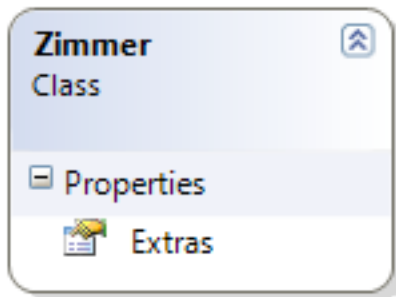
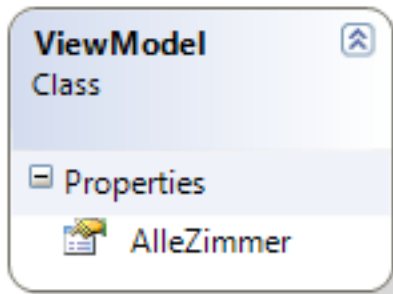
# Application Model

- Modellierung aus Sicht der Businesslogik



# View Model

- Modellierung aus Sicht des UI



Liste aller möglichen Extras,  
„IstVorhanden“ kennzeichnet  
die für diesen Zimmer  
verfügbaren



# Beispiel: Neuanforderung

- Farbliche Kennung der Zimmerkategorie:
  - Feld in Data Model ergänzen
  - Feld in Application Model ergänzen
  - Color Property im View Model
  - Property im View an Farbe binden
- Durch Trennung der Zuständigkeiten sind Änderungen einfach und nachvollziehbar
- Oberfläche wird durch Daten gesteuert, weniger durch Code

# Aufteilung des Gesamtprojekts

- „Horizontale“ Aufteilung in Schichten
- „Vertikale“ Aufteilung in Module (Teilanwendungen)

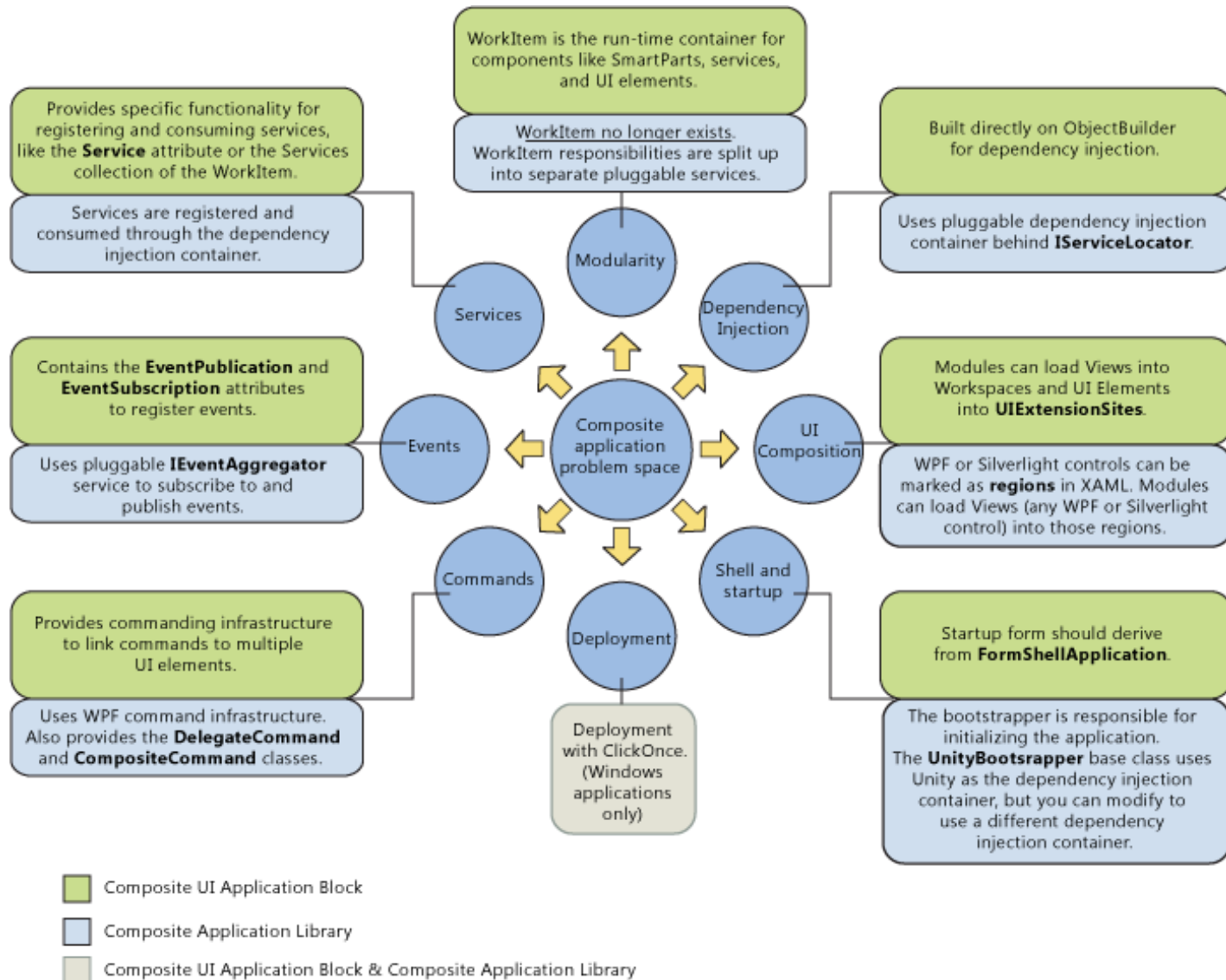
CAG

# Composite Application Guidance

# CAG – Was ist das?

- Composite Application Guidance
  - Framework für „Enterprise-Level“ WPF und Silverlight Anwendungen
  - Von „Patterns and Practices“, Microsoft
- CAG ⇔ „Library/Framework plus Leitfaden“
- Quellcode und diverse Community Erweiterungen auf Codeplex
- Gute Quick-Start Beispiele (VB und C#)

# Von CAB zu CAG



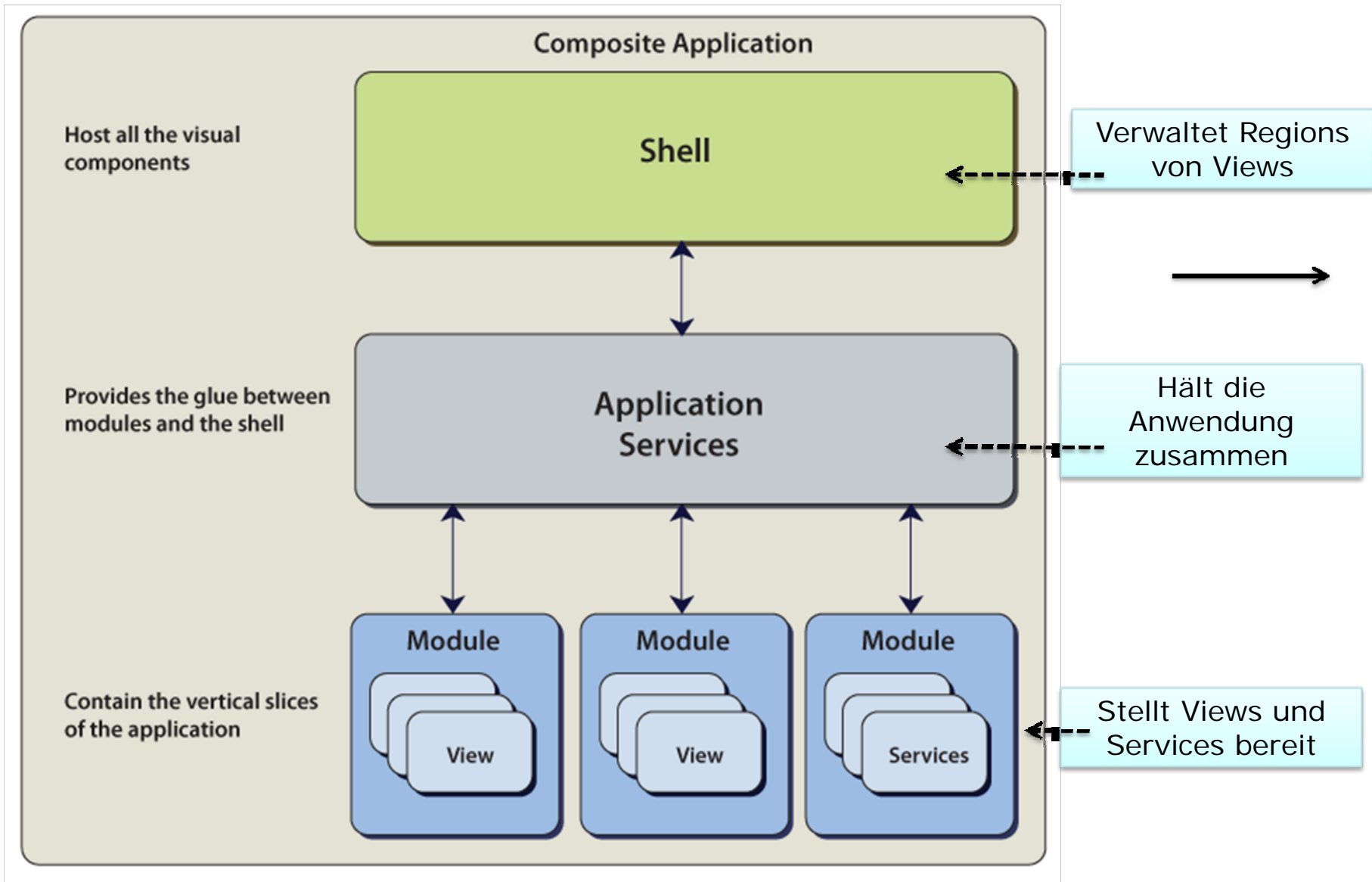
# Ziele und Nutzen von CAG

- Zerlegbarkeit
- Leichtere Erlernbarkeit
- Erweiterbarkeit/Konfigurierbarkeit
- Kompatibilität
- Einfachheit
- Testbarkeit
- Performance
- Sowohl für WPF als auch für Silverlight

# Vorteile des modularen Aufbaus

- Module können getrennt entwickelt und getestet werden
- Verringerung der Download Zeit durch Nachladen (ClickOnce / XBAB / Silverlight)
- Module könnten in unterschiedlichen Programmiersprachen geschrieben werden
- Abhängig von der Rolle des Benutzers konfigurierbar

# Grundstruktur einer CA

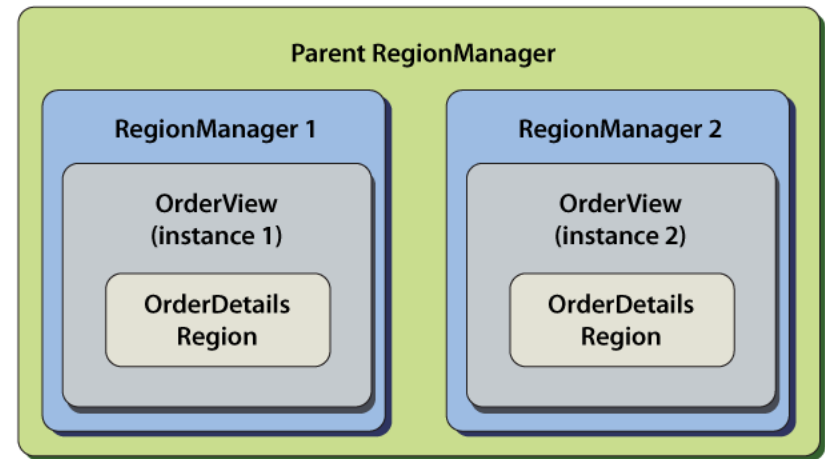
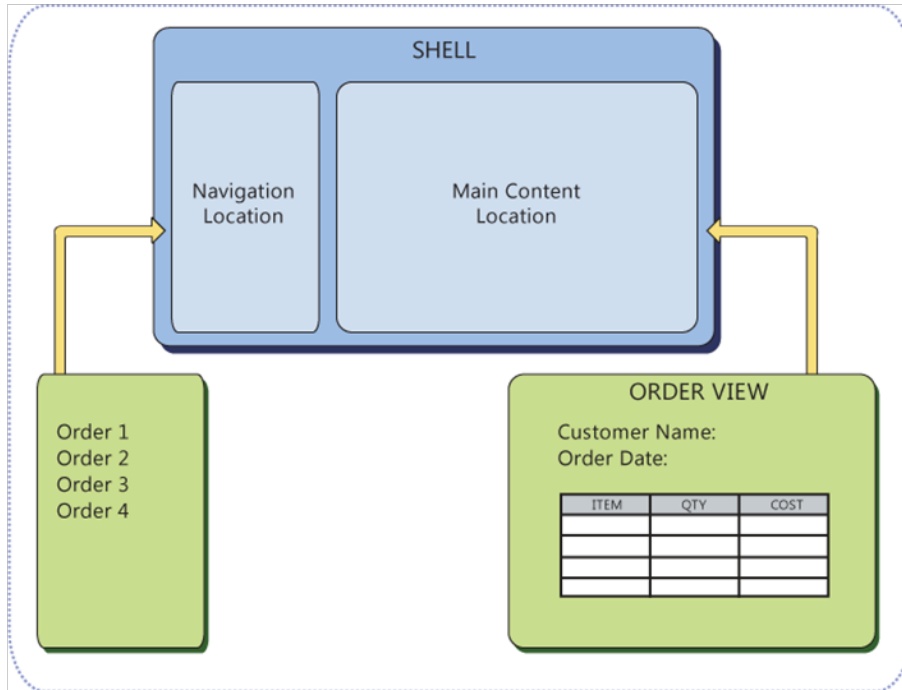




# Bietet diverse Entwurfsmuster an

- Module
- Services
- Commands
- Inversion of Control
- Event Handling

# UI Composition



- Shell ⇔ Hauptfenster
- View ⇔ UserControl
- Region ⇔ benannte Bereiche für Views

# Beispiel für ein Composite View

The screenshot shows a Windows application window titled "View Discovery Composition QuickStart". The window has a blue header bar with the text "View Discovery Composition QuickStart". Below the header, there is a section titled "Select Employee:" containing a table with two columns: "First Name" and "Last Name". The table has two rows: "John" and "Smith" (highlighted), and "Bonnie" and "Skelly". Below the table, there are three tabs: "General", "Location", and "Current Projects". The "General" tab is selected, showing a form with four fields: "First Name:" (John), "Last Name:" (Smith), "Phone:" (+1 (425) 555-0101), and "Email:" (john.smith@example.com).

First Name	Last Name
John	Smith
Bonnie	Skelly

General Location Current Projects

**First Name:** John **Last Name:** Smith

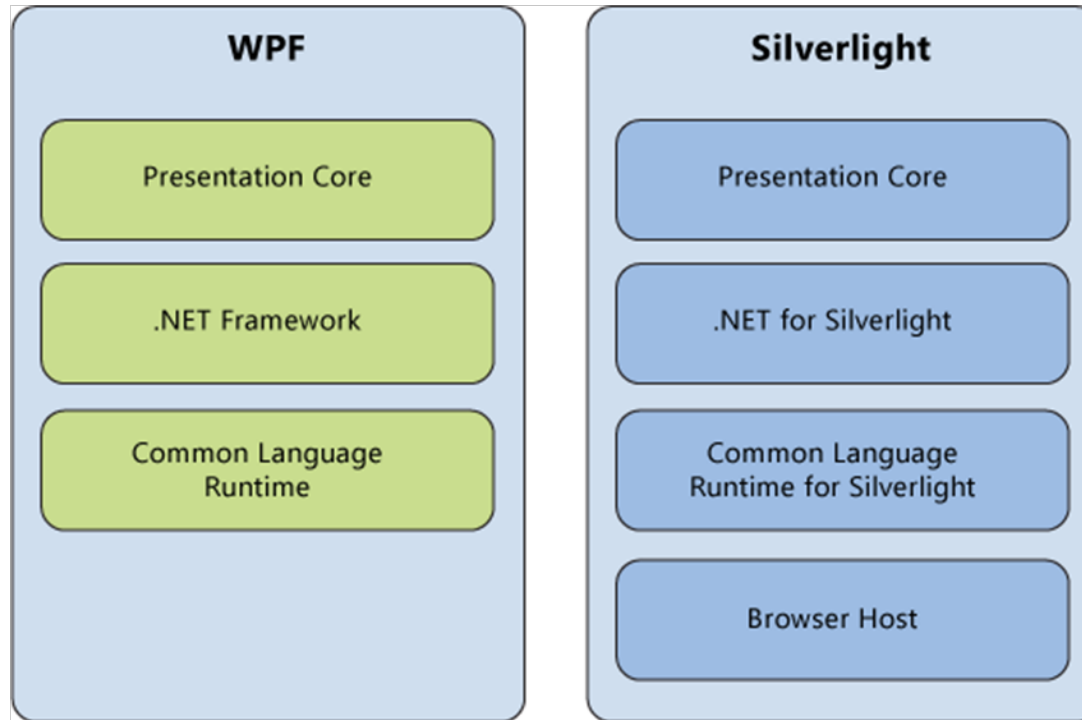
**Phone:** +1 (425) 555-0101 **Email:** john.smith@example.com

# CODE BEISPIEL

# WPF & Silverlight

# WPF und Silverlight

- Von der Struktur her sehr ähnlich...



- ... aber viele Unterschiede in den Details

# WPF und Silverlight

- Szenario:
  - Anwendung als WPF Desktop konzipiert
  - Kunde will noch Web-Variante mit Silverlight
- Code-Behind Programmierer:
  - Fängt praktisch wieder von vorne an
  - Maximal „Cut & Paste“ Wiederverwendung ☹
- Software Architekt:
  - Wiederverwendung großer Teile des Codes 😊

# Von WPF nach Silverlight

- Es kann wiederverwendet werden
  - Das eigene WPF Knowhow
  - Application Model
  - View Model
  - CAG Infrastruktur Code
- Nicht wiederverwendet werden kann
  - XAML Code (nur Cut & Paste möglich)  
(UI, Themes, Styles, ...)



# Silverlight erzwingt Schichten

- Silverlight kommuniziert über Servies
- Horizontale Schichten sind notwendig

# CODE BEISPIELE

# Fazit

- Das Design von WPF ist optimal für moderne Anwendungsarchitektur
- Programmiersprache spielt praktisch keine Rolle
- Mit CAG gibt es eine gute Grundlage für große modulare Anwendungen

# Kontakt

- Danke für´s zuhören!
- [Stefan.Lange@empira.de](mailto:Stefan.Lange@empira.de)
- [www.st-lange.net](http://www.st-lange.net) (in Kürze)

# Fragen / Diskussion