



Odisee
DE CO-HOGESCHOOL

MVVM



Jens Baetens



Applicatie architectuur



Applicatie architecturen

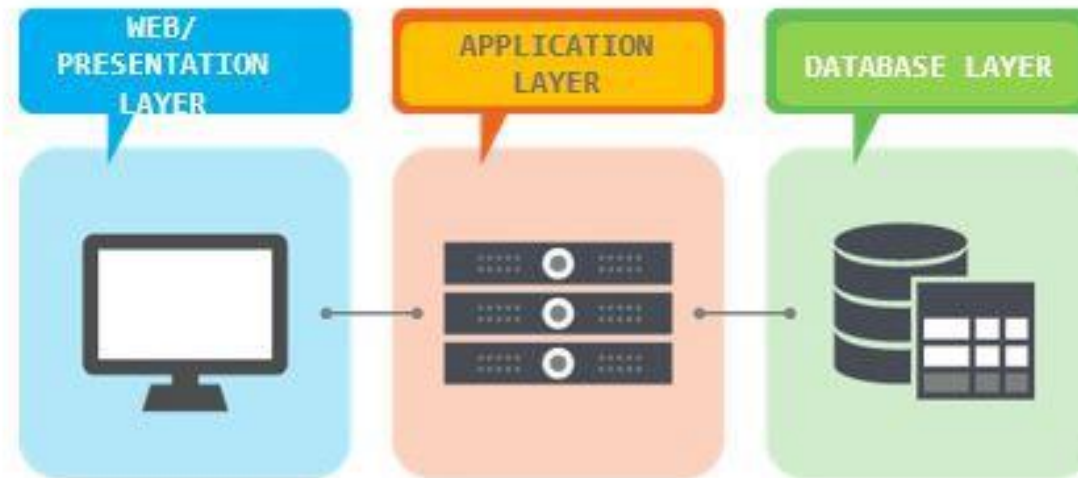
- ▣ Bouwconstructie
- ▣ Richtlijnen om communicatie tussen verschillende klassen vlot te laten verlopen
- ▣ Klasse structuren die helpen om clean code (propere code) te schrijven.

3-tier model

■ 3 lagen

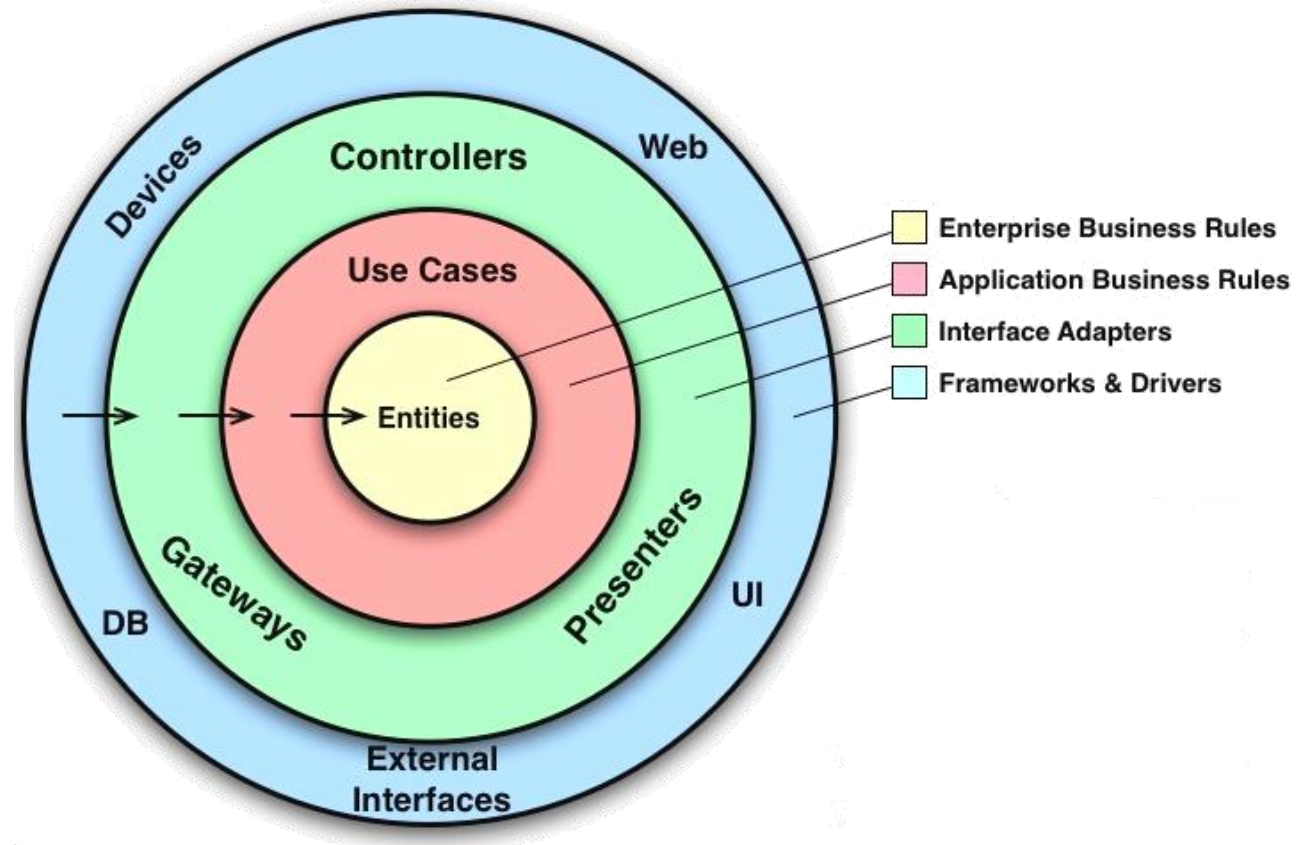
- Web/presentation Layer
- Application Layer
- Database Layer

Let us walk through a three tier architecture :



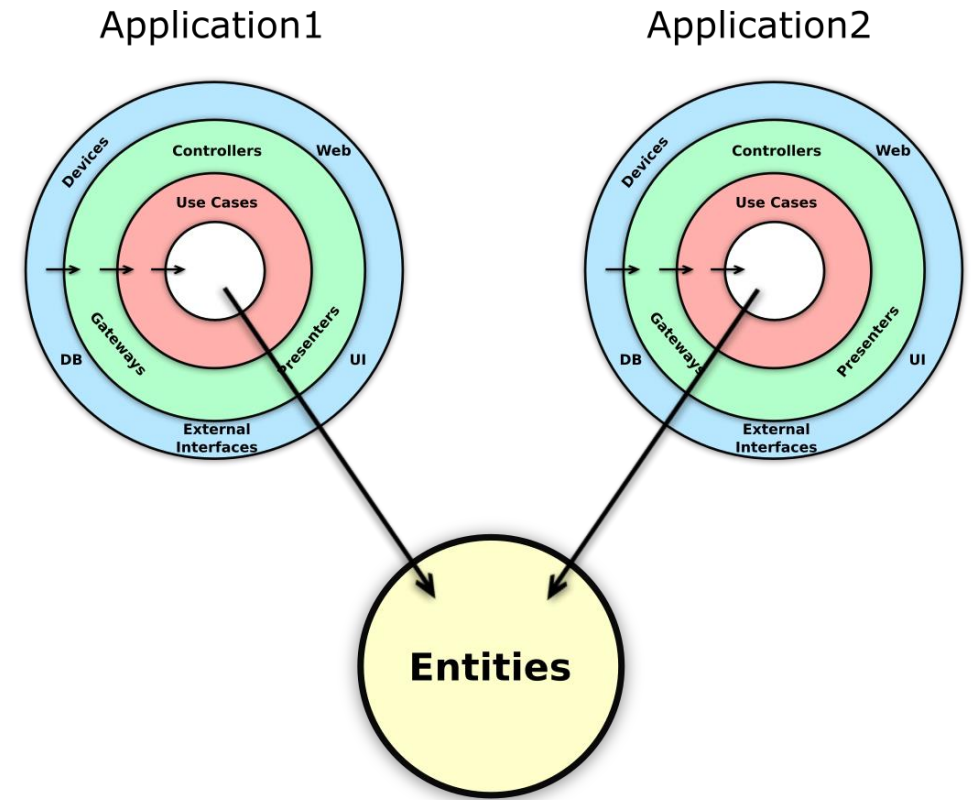
Onion Model

- ▣ Meerdere schillen
- ▣ Buitenste schillen hebben kennis van de binnenste
- ▣ Binnenste schillen kennen buitenste schillen **niet**
- ▣ Veel andere architecturen komen hiermee overeen



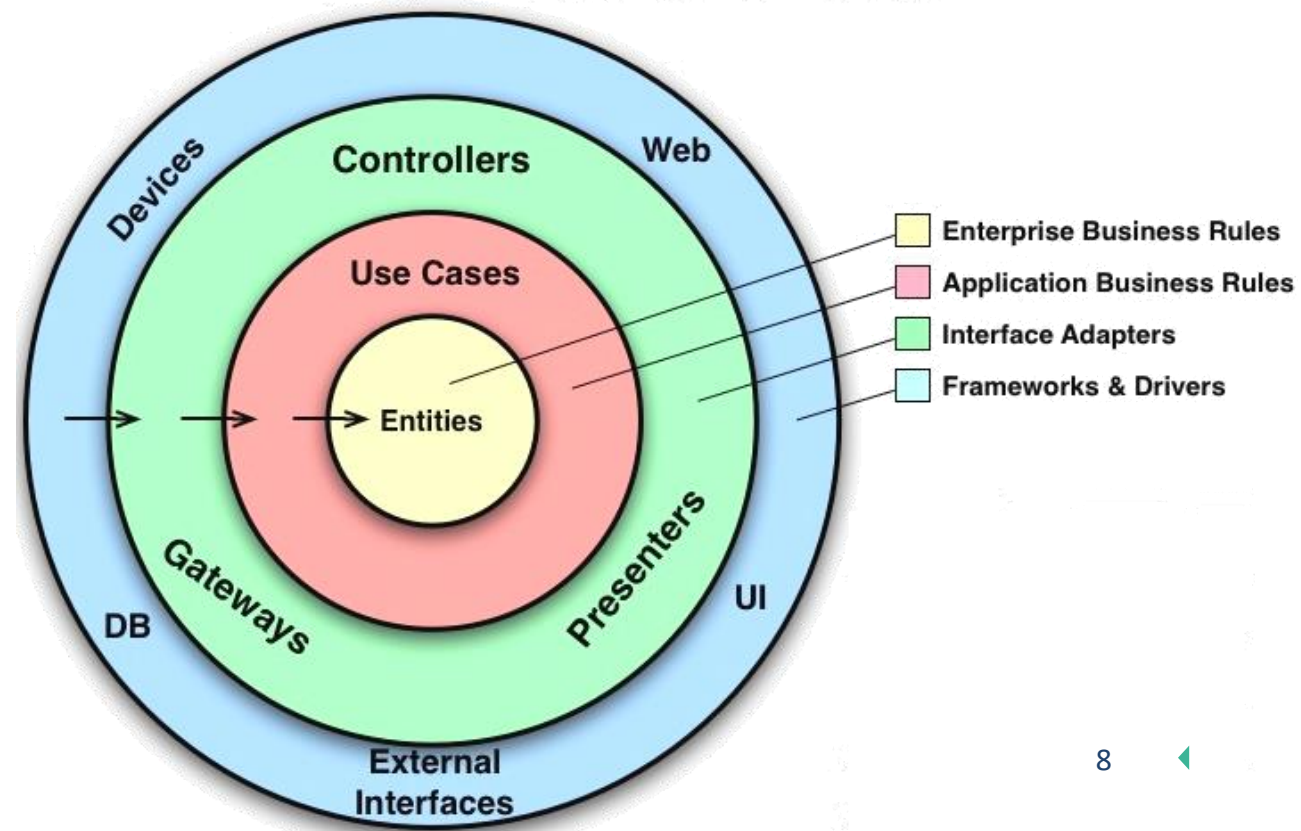
Onion Model herbruikbaarheid

- 2 applicaties kunnen gebruik maken van dezelfde entity set zonder gebonden te zijn aan andere applicatiespecifieke zaken
- Use Cases kunnen rechtstreeks communiceren met entities
- Entities kunnen niet rechtstreeks communiceren met Use Cases



Onion Model Simplified

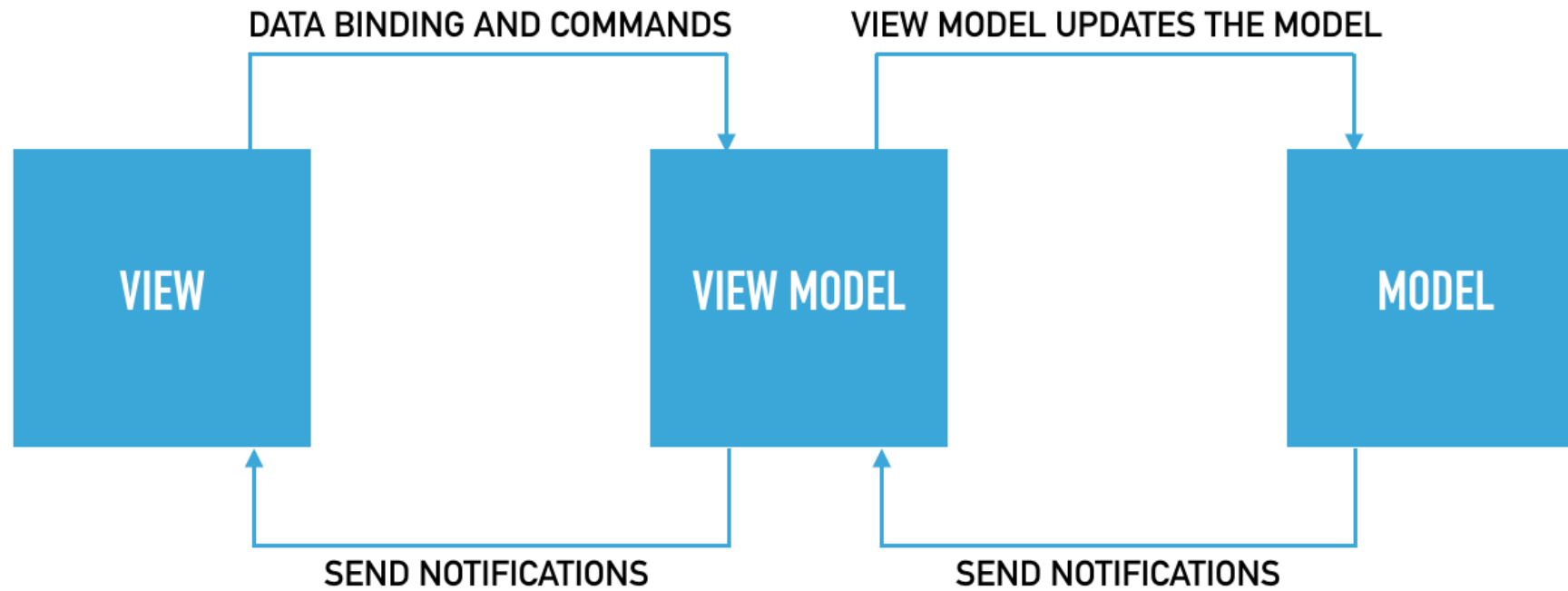
- ▣ Het onion model is behoorlijk complex
- ▣ Inspiratie voor
 - Model-View-Controller (MVC)
 - Model-View-Presenter (MVP)
 - Model-View-ViewModel (MVVM)





MVVM

MVVM – Model View ViewModel



Model

▣ Bevat alle informatie

- Alle data objecten
- Reeds gekend (Entity Framework, ...)

Quote
+Id: int +Author: string +Text: string
+Quote() +Quote(string text, string author)

```
21 references
public class Quote
{
    0 references
    public int Id { get; set; }
    2 references
    public string Text { get; set; }
    2 references
    public string Author { get; set; }

    1 reference
    public Quote(string text, string author)
    {
        Text = text;
        Author = author;
    }

    0 references
    public Quote()
    {
    }
}
```

View

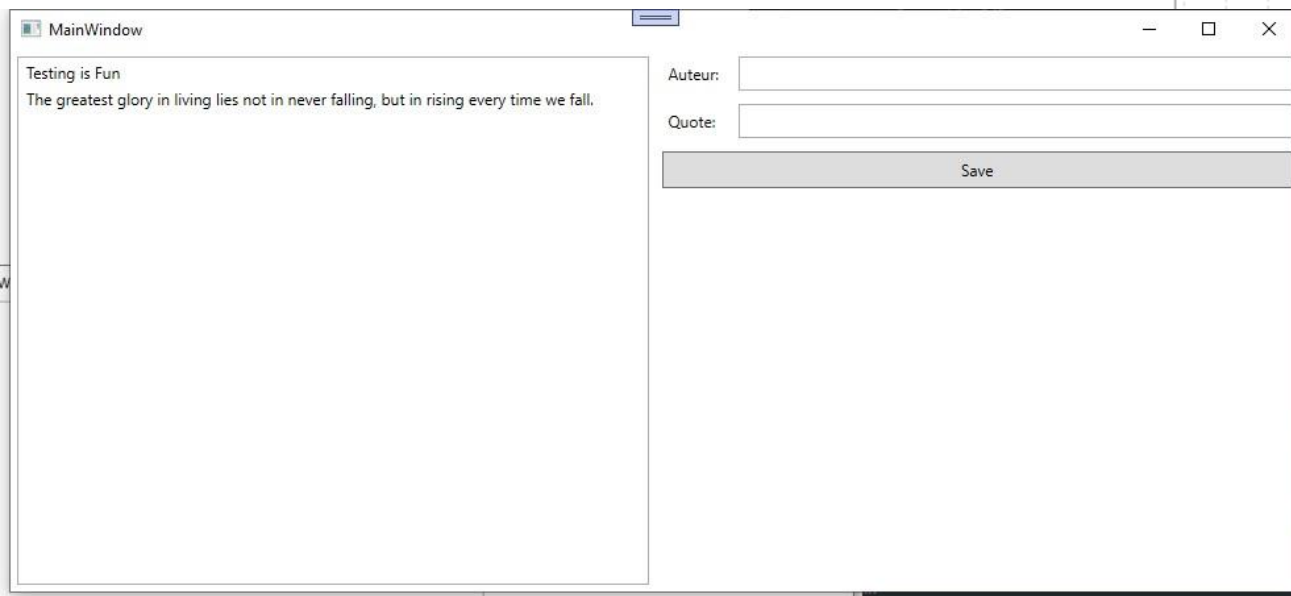
- Bevat alle logica voor de view
- Hoe staan de elementen geordend
- Welke knoppen/acties zijn er

```
<Window x:Class="Quotes.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Quotes"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<ListBox x:Name="quotesListBox" Margin="5" SelectionChanged="quotesListBox_SelectionChanged"/>
<Grid Grid.Column="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition Height="auto"/>
<RowDefinition Height="auto"/>
</Grid.RowDefinitions>
<Label Margin="5">Auteur:</Label>
<TextBox x:Name="authorTextBox" Grid.Column="1" Margin="5"/>

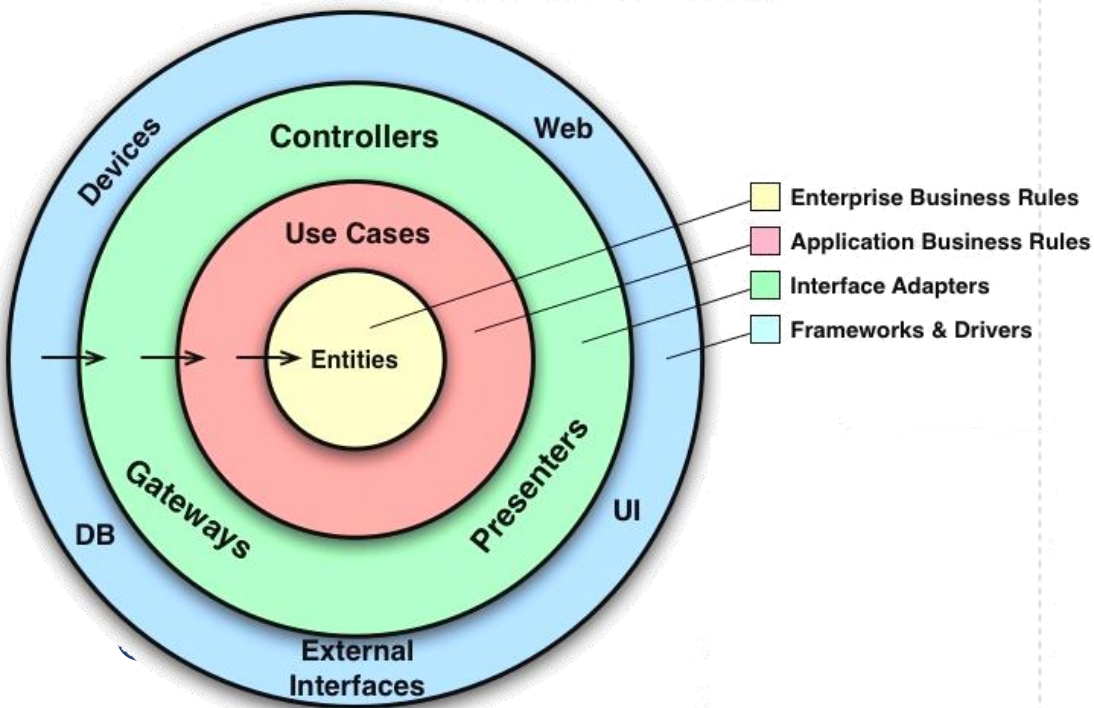
<Label Grid.Row="1" Margin="5">Quote:</Label>
<TextBox x:Name="quoteTextBox" Grid.Column="1" Grid.Row="1" Margin="5"/>

<Button Grid.Row="2" Grid.ColumnSpan="2" Name="saveButton" Margin="5" Padding="5" Click="saveButton_Click">Save</Button>
</Grid>
</Grid>
```



Code Behind van Mainwindow

- ▣ Waar zou dit moeten in het onion model?



4 references

```
public partial class MainWindow : Window
{
```

```
    IQuoteRepository quoteRepository;
```

0 references

```
    public MainWindow() : this(new QuoteRepository())
```

```
{
```

```
}
```

1 reference

```
    public MainWindow(IQuoteRepository quoteRepository)
```

```
{
```

```
        InitializeComponent();
```

```
        this.quoteRepository = quoteRepository;
```

```
        LoadQuotes();
```

```
}
```

1 reference

```
    private void saveButton_Click(object sender, RoutedEventArgs e)
```

```
{
```

```
        if (authorTextBox.Text != String.Empty && quoteTextBox.Text != String.Empty)
```

```
{
```

```
            Quote quote = new Quote(quoteTextBox.Text, authorTextBox.Text);
```

```
            quoteRepository.AddQuote(quote);
```

```
}
```

```
        else
```

```
{
```

```
            MessageBox.Show("error");
```

```
}
```

```
        LoadQuotes();
```

```
}
```

2 references

```
    private void LoadQuotes()
```

```
{
```

```
        quotesListBox.Items.Clear();
```

```
        List<Quote> quotes = quoteRepository.GetAllQuotes();
```

Codebehind

- ▣ Doordat in de logica UI-componenten aangesproken worden is dit moeilijk te testen
- ▣ Daarnaast moet er ook voor gezorgd worden dat het als een Single Threaded Applicatie uitgevoerd worden (standaard wordt de GUI op meerdere threads uitgevoerd)

```
[TestFixture]
[RequiresThread(ApartmentState.STA)]
0 references
public class MainWindowTests
{
    [Test]
    0 references
    public void Ctor__LoadData()
    {
    }
```



Codebehind

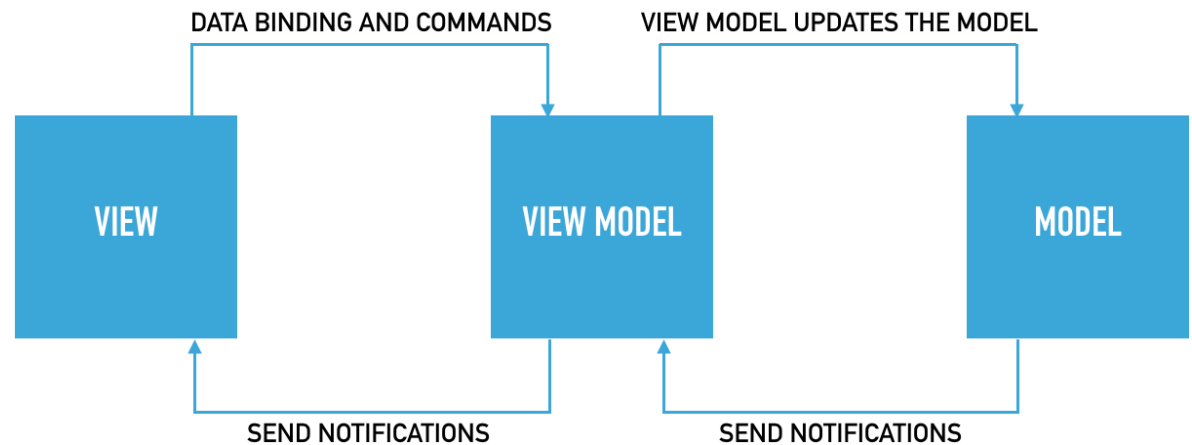
- ▣ Doordat in de logica UI-componenten aangesproken worden is dit moeilijk te testen
- ▣ Daarnaast moet er ook voorgezorgd worden dat het als een Single Threaded Applicatie uitgevoerd worden (standard wordt de GUI op meerdere threads uitgevoerd)
- ▣ Click events kunnen niet getest worden (of met vreemde publieke functies)

Code-behind vs ViewModel

- ▣ CodeBehind past niet in Onion Model
- ▣ Moeilijk testbaar
 - Tests duren ook lang door aanmaken GUI
- ▣ Teveel verantwoordelijkheden
 - UI interacties
 - UI tonen
 - Logica

ViewModel

- ▣ Een model voor views
- ▣ Link tussen model en view
- ▣ Bepaalt in welke vorm bepaalde informatie zichtbaar wordt
 - ▣ Kleur/String concatenations, ...
- ▣ Vangen interactie op van de view
- ▣ Business logica
- ▣ ViewModel kent het model niet
- ▣ ViewModel kent de UI niet



ViewModel – Stap 1

- We maken een klasse die alle properties bevat die getoond of gelezen worden door de GUI
- Enkel kennis van models, niet van views

```
6 references
public class QuoteViewModel
{
    0 references
    public List<Quote> Quotes { get; set; }
    0 references
    public string Author { get; set; }
    0 references
    public string Text { get; set; }

    1 reference
    public QuoteViewModel()
    {
        Quotes = new List<Quote>();
        Author = "";
        Text = "";
    }
}
```

ViewModel stap 2: Koppel view en viewmodel

- ▣ Databinding techniek

- ▬ Properties en commands van het viewmodel binden aan elementen van de view

- ▣ Zet het viewmodel als de datacontext van de MainWindow (View-klasse)

```
4 references
public partial class MainWindow : Window
{
    1 reference
    public MainWindow()
    {
        InitializeComponent();
        DataContext = new QuoteViewModel();
    }
}
```

Viewmodel step 3: Bind properties en gui-elements

```
6 references
public class QuoteViewModel
{
    0 references
    public List<Quote> Quotes { get; set; }
    0 references
    public string Author { get; set; }
    0 references
    public string Text { get; set; }

    1 reference
    public QuoteViewModel()
    {
        Quotes = new List<Quote>();
        Author = "";
        Text = "";
    }
}
```

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <ListBox x:Name="quotesListBox" Margin="5" ItemsSource="{Binding Quotes, Mode=TwoWay}" SelectedItem="{Bin
    <ListBox.ItemTemplate>
      <DataTemplate>
        <Label Content="{Binding Text}" />
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>

  <Grid Grid.Column="1">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="auto" />
      <RowDefinition Height="auto" />
      <RowDefinition Height="auto" />
    </Grid.RowDefinitions>
    <Label Margin="5">Auteur:</Label>

    <TextBox x:Name="authorTextBox" Grid.Column="1" Margin="5" Text="{Binding Author}" />

    <Label Grid.Row="1" Margin="5">Quote:</Label>
    <!-- Add binding part-->
    <TextBox x:Name="quoteTextBox" Grid.Column="1" Grid.Row="1" Margin="5" Text="{Binding Text}" />
  </Grid>
```

Databinding properties

- ▣ Veel opties mogelijk bij Databinding
- ▣ Bvb: Mode
 - TwoWay
 - Van en naar het viewmodel, Get + Set
 - OneTime
 - Get, slechts 1 maal uitgelezen
 - OneWay
 - Binding wordt enkel gelezen uit het viewmodel
 - OneWayToSource
 - Set, binding wordt enkel door het viewmodel gelezen

Quotes is een read only binding

```
<ListBox x:Name="quotesListBox" Margin="5" ItemsSource="{Binding Quotes, Mode=OneWay}" SelectedItem="{Binding SelectedQu
  <ListBox.ItemTemplate>
    <DataTemplate>
      <Label Content="{Binding Text}"/>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
<Grid Grid.Column="1">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="auto"/>
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="auto"/>
    <RowDefinition Height="auto"/>
    <RowDefinition Height="auto"/>
  </Grid.RowDefinitions>
  <Label Margin="5">Auteur:</Label>

  <TextBox x:Name="authorTextBox" Grid.Column="1" Margin="5" Text="{Binding Author, Mode=OneWayToSource}"/>

  <Label Grid.Row="1" Margin="5">Quote:</Label>
  <!-- Add binding part-->
  <TextBox x:Name="quoteTextBox" Grid.Column="1" Grid.Row="1" Margin="5" Text="{Binding Text, Mode=OneWayToSource}"/>
```

Author is een write only binding

Text is een write only binding

Databinding – Custom ListBoxItems

- Door objecten te binden aan een listbox, combobox met een itemSource krijg je vreemde output te zien
 - Override de ToString() methode om het correct om te zetten in tekst
 - Een andere optie voor complexere zaken is gebruik te maken van DataTemplates

```
<ListBox x:Name="quotesListBox" Margin="5" ItemsSource="{Binding Quotes}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <Label Content="{Binding Text}"/>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

```
<ListBox x:Name="quotesListBox" Margin="5" ItemsSource="{Binding Quotes, Mode=OneWay}"
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Label Content="{Binding Text}"/>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

23 references

```
public class Quote
```

```
{
```

0 references

```
public int Id { get; set; }
```

2 references

```
public string Text { get; set; }
```

2 references

```
public string Author { get; set; }
```

1 reference

```
public Quote(string text, string author)
```

```
{
```

```
    Text = text;
```

```
    Author = author;
```

```
}
```

0 references

```
public Quote()
```

```
{
```

```
}
```

```
}
```

```
public class QuoteViewModel
```

```
{
```

```
    public List<Quote> Quotes { get; set; }
```

```
    public string Author { get; set; }
```

```
    public string Text { get; set; }
```


ViewModel – stap 4: Handle events

- ▣ Gebeurt aan de hand van commands
 - Stuur events door naar het viewmodel
 - Deze klasse implementeert ICommand
 - Bvb RelayCommand
 - Bind deze commands aan de events die afgevuurd moeten worden

RelayCommand

- Implementeert ICommand
- Action is delegate
 - ▬ voorzien door .NET framework
- CanExecuteChanged & CanExecute
 - ▬ Valideren of het event kan uitgevoerd worden
 - ▬ Gebruiken we niet in deze cursus
- Execute
 - ▬ Methode om de actie uit te voeren
 - ▬ Wordt automatisch aangeroepen door de binding
 - ▬ Moeten we bij tests wel zelf aanroepen

```
//Command class for sending command to the viewmodel
3 references
class RelayCommand : ICommand
{
    //Func<> delegate (object van een functie oproep)
    private Action action;

    2 references
    public RelayCommand(Action action)
    {
        this.action = action;
    }

    public event EventHandler CanExecuteChanged;

    0 references
    public bool CanExecute(object parameter)
    {
        return true;
    }

    0 references
    public void Execute(object parameter)
    {
        action();
    }
}
```

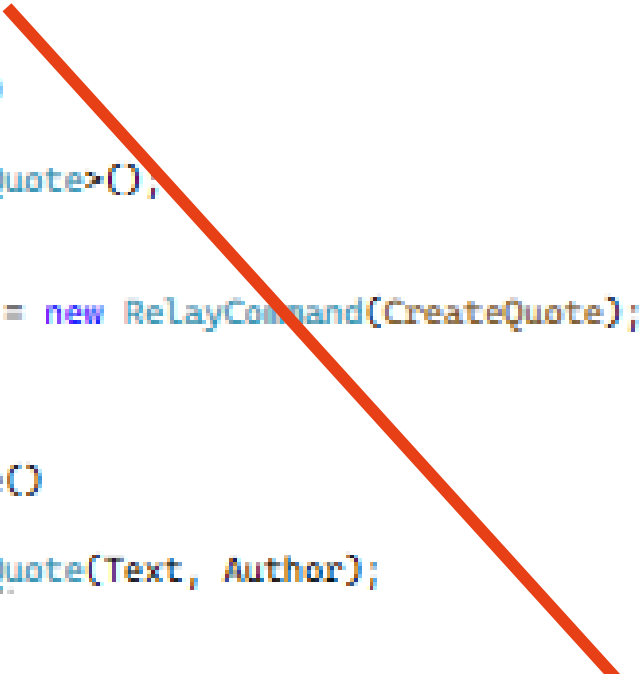
Binding commands

```
4 references
public class QuoteViewModel
{
    2 references
    public List<Quote> Quotes { get; set; }
    2 references
    public string Author { get; set; }
    2 references
    public string Text { get; set; }

    0 references
    public ICommand CreateQuoteCommand { get; set; }

    2 references
    public QuoteViewModel()
    {
        Quotes = new List<Quote>();
        Author = "";
        Text = "";
        CreateQuoteCommand = new RelayCommand(CreateQuote);
    }

    0 references
    public void CreateQuote()
    {
        Quote quote = new Quote(Text, Author);
        Quotes.Add(quote);
    }
}
```



```
<!-- Select command to execute-->
```

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="saveButton" Margin="5" Padding="5" Command="{Binding CreateQuoteCommand}">Save</Button>
```

Next slide

ObservableCollection

- Merk op dat het commando wel uitgevoerd wordt maar er komt niets bij in de lijst
 - ▬ View moet gemeld worden dat er wijzigingen zijn
 - ▬ Vervang hiervoor List door ObservableCollection
 - ▬ Wijzigingen hieraan worden gemeld aan het view
- Zie: <https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1?view=netframework-4.8>



ObservableCollection - Voorbeeld

```
4 references
public class QuoteViewModel
{
    2 references
    public ObservableCollection<Quote> Quotes { get; set; }
    2 references
    public string Author { get; set; }
    2 references
    public string Text { get; set; }

    1 reference
    public ICommand CreateQuoteCommand { get; set; }

    2 references
    public QuoteViewModel()
    {
        Quotes = new ObservableCollection<Quote>();
        Author = "";
        Text = "";
        CreateQuoteCommand = new RelayCommand(CreateQuote);
    }

    1 reference
    public void CreateQuote()
    {
        Quote quote = new Quote(Text, Author);
        Quotes.Add(quote);
    }
}
```



SelectedItem

- ▣ Niet alle events kunnen gedaan worden door Commands
 - Vaak op te lossen door een property te maken en in de setter de methode aan te roepen
 - Dit is zo voor selectedItem van Listbox

SelectedItem - Voorbeeld

```
4 references
public class QuoteViewModel
{
    2 references
    public ObservableCollection<Quote> Quotes { get; set; }
    3 references
    public string Author { get; set; }
    3 references
    public string Text { get; set; }
    0 references
    public Quote SelectedQuote
    {
        set
        {
            ShowSelectedQuote(value);
        }
    }

    1 reference
    public ICommand CreateQuoteCommand { get; set; }

    2 references
    public QuoteViewModel()
    {
        Quotes = new ObservableCollection<Quote>();
        Author = "";
        Text = "";
        CreateQuoteCommand = new RelayCommand(CreateQuote);
    }

    1 reference
    public void CreateQuote()
    {
        Quote quote = new Quote(Text, Author);
        Quotes.Add(quote);
    }

    1 reference
    private void ShowSelectedQuote(Quote selectedQuote)
    {
        Text = selectedQuote.Text;
        Author = selectedQuote.Author;
    }
}
```

```
<ListBox x:Name="quotesListBox" Margin="5" ItemsSource="{Binding Quotes, Mode=OneWay}" SelectedItem="{Binding SelectedQuote}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Label Content="{Binding Text}"/>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

INotifyPropertyChanged

- Wanneer we op een quote klikken moeten de velden ernaast de details tonen
- Enkel de properties aanpassen niet voldoende
 - De view moet nog genotified worden
 - Kan door een PropertyChanged event te sturen
 - Implementeer hiervoor de INotifyPropertyChanged interface
- Zie: <https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.inotifypropertychanged?view=netframework-4.8>

INotifyPropertyChanged

■ Interface verplicht ons het PropertyChanged Event te implementeren

- View ontvangt dit event en update de nodige elementen
- Aangezien nu ook de listbox schrijft naar het viewmodel
 - plaats de binding op two-way mode

```
4 references
public class QuoteViewModel: INotifyPropertyChanged
{
    2 references
    public ObservableCollection<Quote> Quotes { get; set; }
    3 references
    public string Author { get; set; }
    3 references
    public string Text { get; set; }

    public event PropertyChangedEventHandler PropertyChanged;
    0 references
    public Quote SelectedQuote
    {
        set
        {
            ShowSelectedQuote(value);
        }
    }

    1 reference
    public ICommand CreateQuoteCommand { get; set; }
}
```

INotifyPropertyChanged

- ClickEvents hebben een sender en argumenten
- PropertyChanged dus ook
 - Sender => this
 - EventArgs => PropertyChangedEventArgs met string van de gewijzigde property

```
private string text;  
3 references  
public string Text  
{  
    get => text;  
    set  
    {  
        text = value;  
        PropertyChanged(this, new PropertyChangedEventArgs("Text"));  
    }  
}
```

INotifyPropertyChanged – Single method

- Dit ziet er voor alle properties zeer gelijkaardig uit

- Leidt gemakkelijk tot fouten
- Plaats het in 1 methode

- Let op CallerMemberName

- Is de naam van de calling functie

```
private string text;
2 references
public string Text
{
    get => text;
    set
    {
        text = value;
        OnPropertyChanged();
    }
}

0 references
private void OnPropertyChanged([CallerMemberName] string propertyName = null)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

ViewModel testen

- ▣ Het viewmodel heeft geen referenties naar window items
 - ▬ Eenvoudig te testen via de reeds beschikbare properties
 - ▬ We kunnen alles doen wat de GUI doet
- ▣ Schrijf tests voor de volgende zaken:
 - ▬ Wanneer een itemSelected is
 - ▣ Worden de text en author property ingesteld
 - ▣ Wordt het PropertyChangedEvent aangeroepen voor deze properties
 - ▬ Wordt het CreateQuoteCommand opgeroepen
 - ▣ De lijst van quotes is aangepast

ViewModelTesten – ItemSelected - Properties

[Test]

0 references

```
public void SelectedQuote_ItemSelected_AuthorAndTextPropertiesAreChanged()
{
    // Arrange
    QuoteViewModel sut = new QuoteViewModel();
    Quote quote = new Quote("text", "author");

    // Act
    sut.SelectedQuote = quote;

    // Assert
    Assert.AreEqual("text", sut.Text);
    Assert.AreEqual("author", sut.Author);
}
```

ViewModelTesten – ItemSelected - Events

■ Voeg eventhandler toe door middel van lambda functie

- ▬ 2 parameters
 - Sender
 - Eventargs
- ▬ Haal de gewijzigde property op
- ▬ Assert kijkt of de juiste gewijzigd is

```
[Test]
0 references
public void Text_Changed_PropertyChangedEventsIsFired()
{
    // Arrange
    QuoteViewModel sut = new QuoteViewModel();

    string property = "";
    sut.PropertyChanged += (sender, eventargs) =>
    {
        property = eventargs.PropertyName;
    };

    // Act
    sut.Text = "A Quote";

    // Assert
    Assert.AreEqual("Text", property);
}
```

ViewModel testen - Commands

- Voer het Commando uit
- Roep Execute op
 - ▬ Parameter niet gebruikt dus mag null zijn
- Assert
 - ▬ Controleer de individuele parameters (1-3)
 - ▬ Override Equals en vergelijk de objecten (4-5)

```
[Test]
0 references
public void CreateQuoteCommand_WithAuthorAndTextFilled_AddNewItemToQuotes()
{
    // Arrange
    QuoteViewModel sut = new QuoteViewModel();
    sut.Author = "Author";
    sut.Text = "Text";

    // Act
    sut.CreateQuoteCommand.Execute(null);

    // Assert
    Assert.AreEqual(1, sut.Quotes.Count);
    Assert.AreEqual("Text", sut.Quotes[0].Text);
    Assert.AreEqual("Author", sut.Quotes[0].Author);
    Assert.AreEqual(new Quote("Text", "Author"), sut.Quotes[0]);
    Assert.Contains(new Quote("Text", "Author"), sut.Quotes);
}
```

```
0 references
public Quote()
{
}

0 references
public override bool Equals(object? obj)
{
    if (obj is Quote)
    {
        Quote quote = (Quote)obj;
        return quote.Author == Author && quote.Text == Text;
    }

    return false;
}
```



Coördinator

Coördinator patroon

▣ Design patroon

- Algemeen gebruikte structuur om veel voorkomend probleem op te lossen
- In deze cursus: vereenvoudigde versie van het coordinator patroon

▣ Doel: Helpen om het volgende uit te voeren

- Tonen views wanneer nodig
- Juiste dependencies instellen
- ViewModel koppelen aan View klassen

Coördinator voorbeeld

■ Implementeer een interface

- Eenvoudiger om te testen

■ De meeste methodes doen het volgende

1. Maak Window/View
2. Maak Viewmodel
 - Coordinator als argument
3. Stel datacontext van view in
4. Toon view

```
public interface ICoordinator
{
    2 references
    void ShowMainWindow();
    2 references
    void ShowDialog(string message);
    2 references
    void ShowAddQuoteWindow();
}

2 references
public class Coordinator : ICoordinator
{
    2 references
    public void ShowDialog(string message)
    {
        MessageBox.Show(message);
    }

    2 references
    public void ShowMainWindow()
    {
        MainWindow mainWindow = new MainWindow();
        QuoteViewModel quoteViewModel = new QuoteViewModel(this);
        mainWindow.DataContext = quoteViewModel;
        mainWindow.Show();
    }
}
```

Applicatie starten met coordinator

- ▣ Niet meer starten vanaf MainWindow maar vanaf coordinator
- ▣ Maak ergens de coordinator aan en open het eerste scherm
 - Dit kan je doen in app.xaml en app.xaml.cs

```
<Application x:Class="Quotes.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Quotes"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```



```
<Application x:Class="Quotes.MVVMC.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Quotes.MVVMC"
    >
    <Application.Resources>
    </Application.Resources>
</Application>
```

```
public partial class App : Application
{

    public App()
    {
        Coordinator coordinator = new Coordinator();
        coordinator.ShowMainWindow();
    }
}
```

Coordinator gebruiken in viewmodel – Tonen MessageBox

- Geen view-code in ViewModel
 - Dus ook geen MessageBox
- Gebruik hiervoor de coordinator
 - MessageBox wordt hier pas aangemaakt

```
}  
2 references  
public class Coordinator : ICoordinator  
{  
    2 references  
    public void ShowDialog(string message)  
    {  
        MessageBox.Show(message);  
    }  
}
```

```
1 reference  
private void CreateQuote()  
{  
    if(string.IsNullOrEmpty(Text) && string.IsNullOrEmpty(Author))  
    {  
        coordinator.ShowDialog("Vul alle gegevens in");  
    } else  
    {  
        Quote quote = new Quote(Text, Author);  
        quoteRepository.AddQuote(quote);  
        LoadQuotes();  
    }  
}
```

Coordinator – Andere views tonen

- Ook bij het tonen van andere views
 - Gebruik de coordinator!

```
1 reference
private void AddNewQuote()
{
    coordinator.ShowAddQuoteWindow();
    LoadQuotes();
}
```

```
public class Coordinator : ICoordinator
{
    2 references
    public void ShowDialog(string message)
    {
        MessageBox.Show(message);
    }

    2 references
    public void ShowMainWindow()
    {
        MainWindow mainWindow = new MainWindow();
        QuoteViewModel quoteViewModel = new QuoteViewModel(this);

        mainWindow.DataContext = quoteViewModel;

        mainWindow.Show();
    }

    2 references
    public void ShowAddQuoteWindow()
    {
        AddQuoteWindow addQuoteWindow = new AddQuoteWindow();
        AddQuoteViewModel viewModel = new AddQuoteViewModel(this, addQuoteWindow);
        addQuoteWindow.DataContext = viewModel;

        addQuoteWindow.ShowDialog();
    }
}
```

Views sluiten vanuit het viewmodel

- Op dit moment moet een viewmodel kennis hebben van een view om het te sluiten
 - ▬ Dit is niet gewenst
 - ▬ Gebruik hiervoor een interface met 1 methode: Close()

```
public interface IClosable
{
    void Close();
}
```

Views sluiten vanuit het viewmodel

- ▣ Het view moet nu deze interface implementeren
- ▣ De code moet echter niet geïmplementeerd worden want ze bestaat reeds

```
4 references
public partial class AddQuoteWindow : Window, IClosable
{
    1 reference
    public AddQuoteWindow()
    {
        InitializeComponent();
    }
}
```

Views sluiten vanuit het viewmodel

■ Om het te gebruiken

- Coordinator geeft het window door als IClosable aan het viewmodel
- De IClosable dependency moet dus ook in de constructor geïnjecteerd worden

```
1 reference
public AddQuoteViewModel(ICoordinator coordinator, IClosable view): this(coordinator, view, new QuoteRepository())
{
}
```

```
1 reference
public AddQuoteViewModel(ICoordinator coordinator, IClosable view, IQuoteRepository quoteRepository)
{
    this.coordinator = coordinator;
    this.quoteRepository = quoteRepository;
    CreateQuoteCommand = new RelayCommand(CreateQuote);
    this.view = view;
}
```

```
2 references
public void ShowAddQuoteWindow()
{
    AddQuoteWindow addQuoteWindow = new AddQuoteWindow();
    AddQuoteViewModel viewModel = new AddQuoteViewModel(this, addQuoteWindow);
    addQuoteWindow.DataContext = viewModel;

    addQuoteWindow.ShowDialog();
}
```


Conclusie

- ▣ Veel nieuwe klassen
 - ViewModels
 - Coordinator
 - RelayCommand
- ▣ Al deze klassen maken het eenvoudiger om beter te kunnen testen
- ▣ Voor deze les alles in de code-behind, nu is de code
 - Loosely coupled
 - Single responsibilities

