8. Patterns 6: Model-View-ViewModel (MVVM)

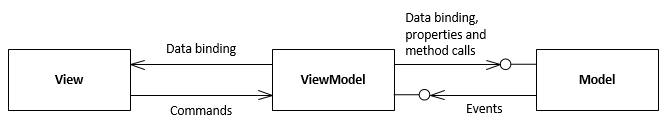
# Redegør for, hvad et software design pattern er.

Et software design pattern er en general genbrugelig løsning til problemer der tit opstår i en given kontekst i software design. Det er ikke et færdigt design der kan laves direkte til kilde kode. Det er en beskrivelse eller skabelon for hvordan et problem kan løses i mange forskellige situationer. Det er formaliserede bedste praksisser som en programmør kan bruge til at løse problemer med.

# Redegør for Model-View-ViewModel mønstret og dets variationer.

MVVM er en specialicering af MVP, det er Microsofts model der bruges sammen med WPF frameworket.

Opbygningen er således:



Opbygningen i MVVM er næsten den samme som i MVP:

* **View**

Kan skrives i XAML og dermed eliminere kode behind, det gør UI designere i stand til at lave View’et mens programmørene kan håndtere GUI logik i ViewModel.

Kender sin viewmodel gennem DataContext.

Er alt hvad der er displayed i GUI. Kan repræsentere hele viduet eller en user control eller datatemplate.

* **ViewModel**

Står for at udstille modellens atributter til View’et, dog kan view’et også hente disse direkte i modellen. View kan også databinde til viewmodel så bliver det automatisk opdateret når der sker ændringer. Sålænge INotifyPropertiesChanged er implementeret.

View kender til viewmodel men ViewModel kender IKKE view.

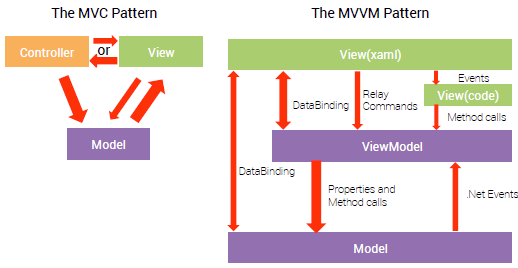
Den udstiller properties og commands.

* **Model**

Er de domæne klasser der er blevet lavet ud fra kravspecifikation eller use-cases

Det er her forretningslogikken bor. Her kan INotifyPropertiesChanged implementeres så view kan opdatere.

Her ses MVVM i et mere visuelt format:



Her kan det også ses at databinding til properties er dejligt men det gælder ikke for user input!

Derfor har vi Relay Commands de hjælper os med at nå ViewModel fra View når det kommer til user input. Da Relay Commands groft sagt er properties der er knyttet til metoder i vores ViewModel.

## Connecting ViewModel and View

Der er to måder at gøre det på der er ”View First” og ”ViewModel First”

**View First** Creating a ViewModel In Code Behind

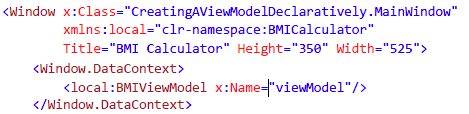
Fordele:

* Der kan sendes parametre med i ViewModels constructor.
* Der kan laves logik der vælger hvilket view der skal oprettes.

Ulemper:

* Kode i code behind er svært at teste og designeren kan ikke gives dummy data i design mode.

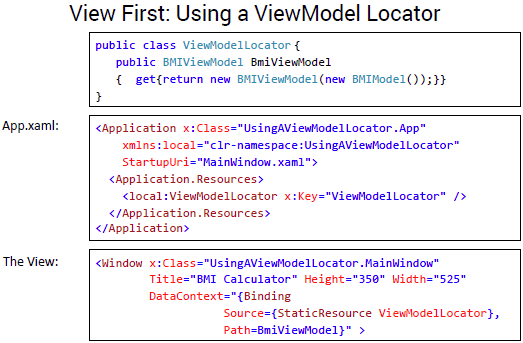
**View First** Creating a ViewModel declaratively

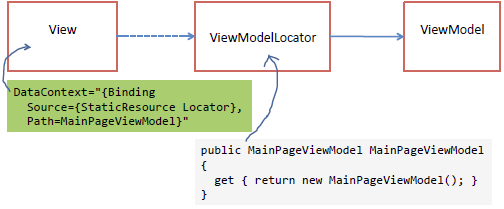
Fordele:

* Der kan bruges dummy data i design mode
* Eventhandlers kan referer viewmodel gennem det navn den har i XAML

Ulemper:

* Ingen parametre i viewmodel constructor



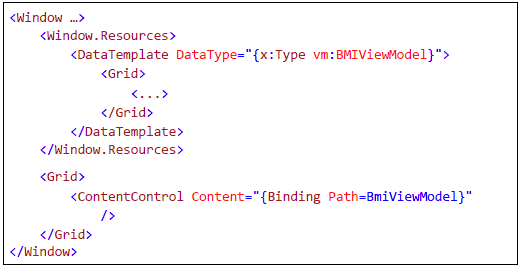


**Fordele:**

* Der kan sendes parametre med i konstructor til viewmodel
* Der kan laves logik der bestemmer hvilken viewmodel og model der skal bruges
* Ingen code behind
* Dummy data

**Ulemper:**

* Det kan være svært at åbne et nyt view fra viewmodel og sætte den nye viewmodel til at refere data i forretnings logikken.

**Viewmodel First: Data template as view**

**Fordele:**

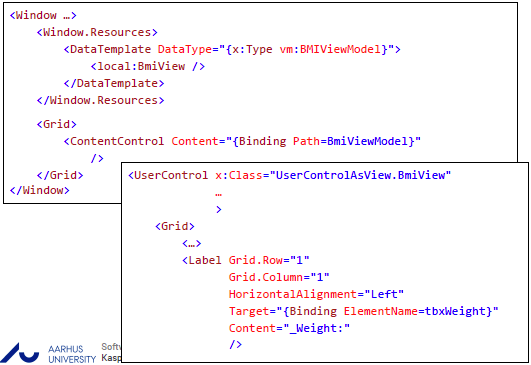
* Når der ændres viewmodel vil kontrollen automatisk forbinde til det samhørende view.
* Der kan sendes parametre med i viewmodels constructor
* Der kan laves logik der vælger view model.
* Ingen codebehind i data templates!

**Ulemper:**

* Ingen dummy data og visual studio har ikke support for design af data templates!

**Viewmodel First: UserControl as view**

**Samme fordele og ulemper som ovenfor. Her er blot mulighed for at lave code behind.**

VS har support for at designe UserControls.

Her ses den rigtige måde at bruge MVVM på!

