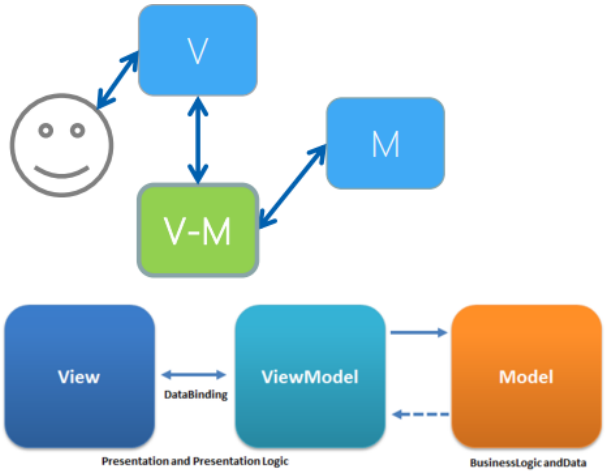
IUR

1. Techniky pro efektivní implementaci uživatelského rozhraní.

Implementace MVVM modelu pomocí návrhových vzorů (např. observer, event-delegate, publish-subscribe).

Reprezentuje stav aplikace a chování prezentace nezávisle na ovládání GUI. Rozděleno na Model, View a ViewModel.

Model - ukládá data
ViewModel - obsahuje data ve formátu pro View
View - prezentuje data uživateli, přímo "nabindovaný" na data ve ViewModelu



Observer vzor

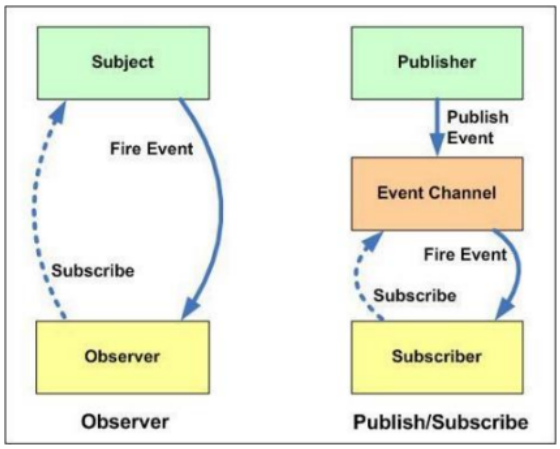
- jeden ku mnoha vazba
- observeři sledují subject a jsou upozorněni, když nastane změna

Event-delegate vzor

- mechanizmus na implementaci observer vzoru v C#
- seznam observerů - delegates

Publish-Subscribe vzor

- odesílatelé zpráv (publishers) posílají zprávy do eventového kanálu
- event kanál upozorní všechny svoje observery (Subsribers)



Řešení vztahu View-ViewModel pomocí návrhového vzoru “Data Binding”.

Technika jak propojit data/informace ze dvou zdrojů. Implementuje se pomocí:

Binding

- Syntax v XAML {Binding NameOfProperty}
- V kódu pro XAML potřeba nastavit Data context (data context je v podstatě ViewModel pro daný View)

INotifyPropertyChanged()

- Na pozadí se provede to, že UI elementy poslouchají zda se nezmění data ve ViewModelu

Commands (Příkazy)

- Objekt, který unifikuje akční chování UI
- Definuje 2 funkce, **Execute()** (vykoná příkaz) a **CanExecute()** (zavolá se před execute a otestuje podle definovaných podmínek zda je možné příkaz vykonat)

Přizpůsobení (customization) UI komponent pomocí šablon (DataTemplate, ControlTemplate), stylů a triggerů.

Přizpůsobení je způsob, jak upravit vzhled komponent UI. Používá se, aby aplikace mohla vypadat podle požadavků designéra a lépe se uživateli používala. Nastavují se jednotlivé vlastnosti jako barva, ohraničení, změna při kliknutí atd.

Style

- Mechanizmus pro rozdělení definice vzhledu a definice layoutu UI elementů
- Definuje hodnoty jednotlivých vlastností
- Kolekce setterů, které ^

Triggers

- Kolekce setterů jako u stylu
- Použijí se pokud platí nějaká podmínka/y
- Když už podmínka neplatí změnu jsou vráceny
- Typy podmínek
  - Property - reagují na změnu nějaké vlastnosti,
  - Data - reagují na změnu Bindingů
  - Event - když proběhne nějaký event jako kliknutí atd. většinou spustí animaci jako reakci na event

Data Template

- Definuje vizuální strom komponenty na základě dat
- Například Item Template u ListBox, kde se specifikuje, jak bude vypadat každá položka v listu

Control Template

- Umožnují změnit vizuální strom (vzhled, reakce na eventy a vlastnosti, strukturu celé komponenty)

Skins - kolekce stylů pro všechny komponenty, rychle vyměnitelné
Themes - imo vlastně skin, který se mění na základě systémových nastavení jako dark mode atd.

Vytváření tzv. User Control a Custom Control.

User Control

- Kombinuje několik existujících komponent do skupiny
- XAML + code behind
- Nelze používat styly/templaty

CUSTOMIZATION | USER CONTROL


```
public partial class FileInputBox : UserControl {
   public FileInputBox() {
     InitializeComponent();
     theTextBox.TextChanged += new TextChangedEventHandler(OnTextChanged);
   }

   private void theButton_Click(object sender, RoutedEventArgs e) {...}
   private void OnTextChanged(object sender, TextChangedEventArgs e) {...}

   public string FileName {get set} //presenting the content
   public event RoutedEventHandler FileNameChanged {...}

   public static readonly DependencyProperty FileNameProperty =
DependencyProperty.Register("FileName", typeof(string), typeof(FileInputBox));
        // to allow default setting in XAML code

   public static readonly RoutedEvent FileNameChangedEvent =
EventManager.RegisterRoutedEvent("FileNameChanged", RoutingStrategy.Bubble,
typeof(RoutedEventHandler), typeof(FileInputBox));
        // to allow default setting in XAML code
}
```

Custom Control

- Rozšiřuje existující komponentu a přidává nové funkcionality
- Code file + styl v Themes/Generic.xaml
- Lze používat styly/templaty

CUSTOMIZATION | CUSTOM CONT.

- Tips for custom control creation
  - Start with behavior rather then visual appearance
  - Think about similarities in order to select base class, e.g. select **Button** rather then **CustomControl**

Validace uživatelského vstupu pomocí validačních pravidel a interfaců (Data source exception, data error interface), případně vlastní validační třídou.

Validační pravidla

- Mechanizmus zabudovaný do Bindingu v WPF (Windows Presentation Foundation)
- **Data Source Exception** - voláno při změně dat, lze použít jako konvertor dat

- **Data Error Interface** - zdroj dat implementuje interface IDataErrorInfo, použito pro asynchronní validaci on vlákně v pozadí, ovládá moment validace
- **Custom Validation class** - Vytvoří se třída dědící od ValidationRule a overriduje se funkce ValidationResult

Prezentace chyb pomocí šablon a triggerů.

Vytvoří se ControlTemplate (šablona) a nastaví se u validace jako ErrorTemplate.

ERROR | CONTROL TEMPLATE

```
<TextBox
Text="{Binding UserName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged,
    ValidatesOnNotifyDataErrors=True}"
    Validation.ErrorTemplate="{StaticResource TextBoxErrorTemplate}">
</ TextBox>


<ControlTemplate x:Key="TextBoxErrorTemplate">
    <StackPanel>
        <AdornedElementPlaceholder Name="MyAdorner" />
        <ItemsControl ItemsSource="{Binding}">
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding ErrorContent}"
                               Foreground="Red" />
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </StackPanel>
</ControlTemplate>
```

User name:

Username cannot be empty.
Username must be at least 6 characters long.

Prezentovat chyby jde i pomocí stylů a triggerů. Vytvoří se trigger na property Validation.HasError. Poté pomocí setterů jde nastavit například tooltip.

2. Příprava uživatelského rozhraní pro testování s/bez uživatele.

Postupy pro důsledné oddělení jednotlivých částí software podle vzoru MVC/MVP/MVVM.

MVC

- Model - Data a logika
- View - Prezentování dat uživateli
- Controller - Process user input
- v reálu není jasná separace V-C, View často mívá logiku Modelu, uživatel interaguje jak s View, tak Controller

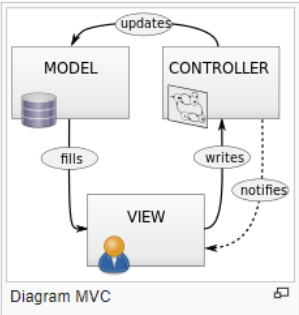
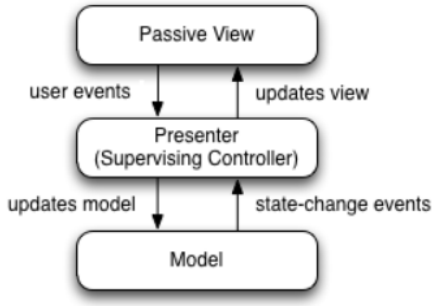


Diagram MVC

MVP

- Input přes View
- Presenter dohlíží

- Možnost i kde V a M nejsou propojené (Passive View)



Vytváření UI test skriptu (testovací kód, nahrávání interakce, GUI ripping).

Testovací kód

- Vytvoří se kód, který vyzkouší sérii interakcí uživatele

UI TEST SCRIPT | WRITING CODE

- Typically executable code

```
public void testTemp script() throws Exception {
        selenium.open("/BrewBizWeb/");
        selenium.click("link=Start The BrewBiz Example");
        selenium.waitForPageToLoad("30000");
        selenium.type("name=id", "bert");
        selenium.type("name=Password", "biz");
        selenium.click("name=dologin");
        selenium.waitForPageToLoad("30000");
    }
```

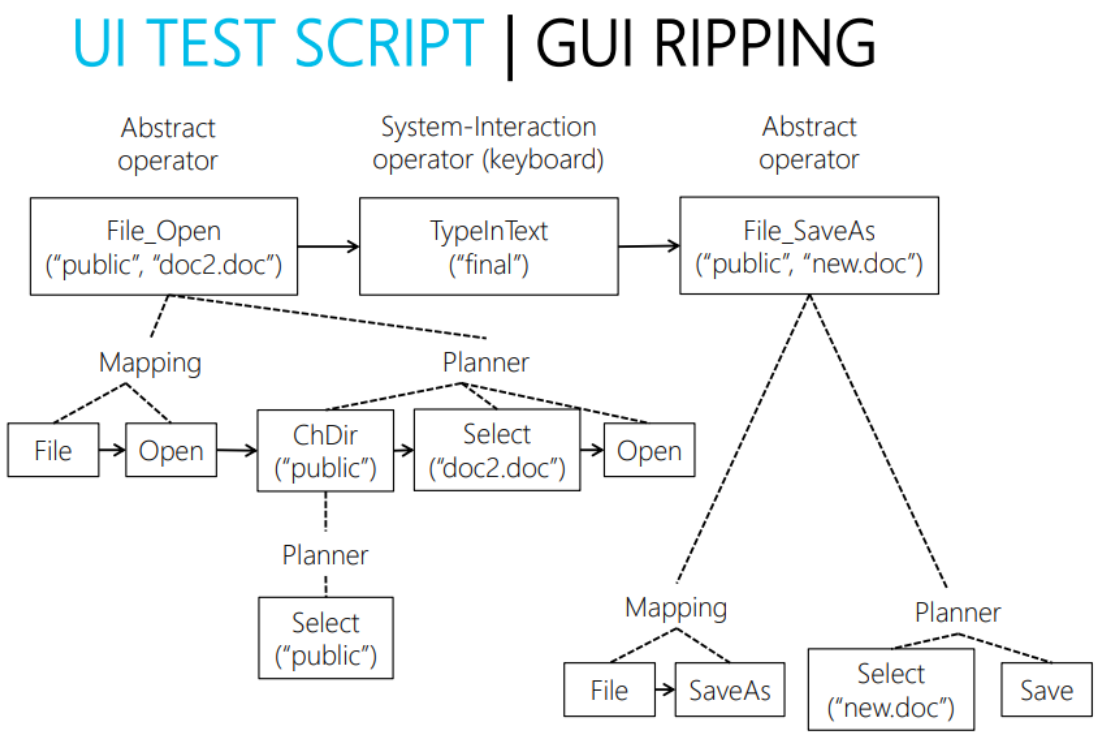
- Additional metadata may be included
- Human readibility is advantage

Nahrávání interakce

- Nahrávají se kroky uživatele, jak interaguje s UI.

GUI Ripping

- Je vytvořen hierarchický model celého UI aplikace se seznamem operací a předpokladů pro dané operace. Designér testu vybere scénář pro generaci test case. Nástroj vygeneruje testovací case pro daný scénář.



Příprava software pro uživatelské testování - data mocking, podpora metody Wizard-of-Oz, sběr dat z chování software a interakce s uživatelem

Typ dat získaný z testování

- Basic - Audio video nahrávky, zápisky, dotazníky
- Advanced - záznamy aplikace (Logs), eye tracking, sensory (poloha, akcelerometr), low level interakce

Testování s uživatelem

- Testy lo-fi prototypů
- laboratorní testy - kontrolované podmínky
- nebo testy v terénu - hi-fidelity prototyp, některé testy jinak nejdou

Data mocking

- Místo wireframe dat nebo reálných dat použijeme data, která vypadají reálně, aby se mohla aplikace otestovat jako kdyby data byla reálná.

Wizard-of-Oz

- Způsob jak testovat systémy, které ještě neexistují. Daný systém je simulovaný reálným člověkem. Speech interaction s robotem.

Testování bez uživatele

- Odhaluje jen závažné problémy
- Velká pravděpodobnost zaujatosti
- Velmi rychlá metoda a lze dělat na velmi brzkém modelu aplikace

Heuristická evaluace

- Identifikuje problémy s použitelností
- Na jakékoliv úrovni aplikace (lo-fi, hi-fi...)
- Používají se designové heuristiky

- Dělají ji experti

DESIGN HEURISTICS BY J. NIELSEN

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

Kognitivní průchod

- Simulace průchodu aplikace uživatelem
- V každém kroku se ptáme na několik kontrolních otázek
  - Questions for each step:
    - Q1 = "Will the correct action(s) be evident to the users?"
    - Q2 = "Will the users connect the label of an action with their goals?"
    - Q3 = "Will the user receive a sensible feedback?"