Univerzita Pardubice

Dopravní fakulta Jana Pernera

Aplikace pro sledování činností uživatelů

Bc. Lukáš Rajm

Diplomová práce

2015

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou, nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 28. 8. 2015

Bc. Lukáš Rajm

**Poděkování**

Rád bych poděkoval panu doc. Ing. Karlu Greinerovi, Ph.D. za pomoc s volbou vhodného tématu diplomové práce. Dále za jeho cenné rady, čas a připomínky k práci. Mé další poděkování patří těm, co ve mne věřili a pomáhali mi při studiích.

**ANOTACE**

Předmětem této práce je vývoj aplikace složené z klientského programu a aplikačního serveru. Klientský program sleduje aktivity uživatele (připojení USB disku, spuštění aplikace apod.) definované aplikačním serverem a odesílá je na aplikační server. Aplikační server tyto aktivity zaznamenává do databáze nebo souboru a zobrazuje je na obrazovce. Aplikace může případně sloužit také k omezení některých činností uživatele.

**KLÍČOVÁ SLOVA**

Monitorování, C#, Sqlite, Unifikovaný proces, Windows Communication Foundation, Windows Presentation Foudation, Entity Framework

**TITLE**

Application for monitoring the activities of users

**ANNOTATION**

The subject of this work is develop an application consisting of a client program and the application server. The client program will monitor user activity (USB drive, lunch applications, etc.), Defined application server and send them to the application server. The application server for these activities recorded in the database orfile and display sit on the screen. Applications may possibly also be used to restrict in user actions.

**KEYWORDS**

Monitoring, C#, Sqlite , UnifiedProcess, Windows Communication Foundation, Windows Presentation Foudation, Entity Framework

# Obsah

[Úvod 13](#_Toc435556282)

[1 Vývoj a projektovaní softwaru 14](#_Toc435556283)

[1.1 UML 14](#_Toc435556284)

[1.1.1 Historie UML 14](#_Toc435556285)

[1.1.2 Proč „UML“ 14](#_Toc435556286)

[1.1.3 Struktura jazyka UML 15](#_Toc435556287)

[1.1.4 Stavební bloky 15](#_Toc435556288)

[1.1.5 Předměty 15](#_Toc435556289)

[1.1.6 Relace 16](#_Toc435556290)

[1.1.7 Diagramy 16](#_Toc435556291)

[1.1.8 Obecná mechanika jazyka UML 19](#_Toc435556292)

[1.1.9 Architektura 20](#_Toc435556293)

[1.2 Metodiky vývoje softwaru 21](#_Toc435556294)

[1.2.1 Tradiční metodiky 22](#_Toc435556295)

[1.2.2 Agilní metodiky 24](#_Toc435556296)

[2 Technologie 26](#_Toc435556297)

[2.1 .Net Framework 26](#_Toc435556298)

[2.1.1 Architektura .NET Framework 27](#_Toc435556299)

[2.2 WPF 28](#_Toc435556300)

[2.2.1 Jazyk XAML 28](#_Toc435556301)

[2.3 WCF 30](#_Toc435556302)

[2.4 Entity Framework 31](#_Toc435556303)

[2.4.1 O/RM 32](#_Toc435556304)

[2.4.2 Architektura 33](#_Toc435556305)

[2.4.3 Návrhový vzor MVVM 34](#_Toc435556306)

[2.5 SQLite database 35](#_Toc435556307)

[2.6 Jazyk C# 36](#_Toc435556308)

[3 Realizace práce 37](#_Toc435556309)

[3.1 Krok první „Zahájení“ 37](#_Toc435556310)

[3.2 Zadání práce 37](#_Toc435556311)

[3.3 Požadavky 38](#_Toc435556312)

[3.4 Návrh architektury systému 38](#_Toc435556313)

[3.5 Krok dvě „Rozpracování“ 40](#_Toc435556314)

[3.5.1 Případy užití 40](#_Toc435556315)

[3.5.2 Analytické třídy 43](#_Toc435556316)

[3.5.3 Sekvenční Diagram 44](#_Toc435556317)

[3.5.4 Datový model 45](#_Toc435556318)

[3.6 Krok třetí „Konstrukce“ 46](#_Toc435556319)

[3.6.1 Implementace datového modelu 46](#_Toc435556320)

[3.6.2 Implementace aplikace 48](#_Toc435556321)

[3.7 Krok čtvrtý „Zavedení“ 60](#_Toc435556322)

[3.7.1 Testování 60](#_Toc435556323)

[4 Závěr 62](#_Toc435556324)

[5 Použitá Literatura 63](#_Toc435556325)

[6 Seznam Příloh 64](#_Toc435556326)

# Seznam ilustrací

[Obr. 1 Struktura jazyka UML[1] 15](#_Toc436202635)

[Obr. 2 Zobrazení balíčku a informace[1] 16](#_Toc436202636)

[Obr. 3 Třináct různých diagramu v UML[1] 17](#_Toc436202637)

[Obr. 4 Diagram tříd 18](#_Toc436202638)

[Obr. 5 Diagram případu užití 18](#_Toc436202639)

[Obr. 6 Sekvenční diagram 19](#_Toc436202640)

[Obr. 7 Diagram s ornamenty 20](#_Toc436202641)

[Obr. 8 Architektura 4+1[1] 21](#_Toc436202642)

[Obr. 9 Schéma životního cyklu Vodopádu[8] 22](#_Toc436202643)

[Obr. 10 Spirálový model[8] 23](#_Toc436202644)

[Obr. 11 Diagram fází projektu[1] 24](#_Toc436202645)

[Obr. 12 Objem práce v jednotlivých fázích metodiky UP[1] 24](#_Toc436202646)

[Obr. 13 Logo .Net Framework[7] 26](#_Toc436202647)

[Obr. 14 Knihovnyverzí .NET Framework[7] 26](#_Toc436202648)

[Obr. 15 Architektura .NET Framework[3] 27](#_Toc436202649)

[Obr. 16 XAML vs C# code[14] 29](#_Toc436202650)

[Obr. 17 Vazba mezi cílem a zdrojem[14] 30](#_Toc436202651)

[Obr. 18 Komunikace mezi klientem a službou[3] 30](#_Toc436202652)

[Obr. 19 Schéma tvorby nebo přístupu k databázi[11] 32](#_Toc436202653)

[Obr. 20 Schéma ORM[11] 32](#_Toc436202654)

[Obr. 21 Architektura EF[11] 33](#_Toc436202655)

[Obr. 22 Princip MVVM[13] 34](#_Toc436202656)

[Obr. 23 Schéma komunikace MVVM[13] 35](#_Toc436202657)

[Obr. 24 Diagram Funkčních a nefunkčních požadavků 38](#_Toc436202658)

[Obr. 25 Diagram architektury v ArchiMate 39](#_Toc436202659)

[Obr. 26 Případ užití mezi klientem a APS 41](#_Toc436202660)

[Obr. 27 Scénář případu užití 42](#_Toc436202661)

[Obr. 28 Matice pokrytí požadavků případy užití 42](#_Toc436202662)

[Obr. 29 Diagram analytické třídy 43](#_Toc436202663)

[Obr. 30 Sekvenční diagram UC06 45](#_Toc436202664)

[Obr. 31 Datový model 45](#_Toc436202665)

[Obr. 32 SQL syntaxe pro tvorbu tabulek 47](#_Toc436202666)

[Obr. 33 Schéma vytvořených tabulek 48](#_Toc436202667)

[Obr. 34 Schéma vygenerované EF 50](#_Toc436202668)

[Obr. 35 Stavy klientské aplikace 51](#_Toc436202669)

[Obr. 36 Úvodní obrazovka Big Brother Viewer 53](#_Toc436202670)

[Obr. 37 Položka File z menu 54](#_Toc436202671)

[Obr. 38 Dialogové okno pro tvorbu události. 54](#_Toc436202672)

[Obr. 39 Zobrazení sledovaných uživatelů. 57](#_Toc436202673)

[Obr. 40 Modální okna seznamu upozorňovacích názvu 58](#_Toc436202674)

[Obr. 41 Obrazovka historických data 59](#_Toc436202675)

[Obr. 42 Schéma domácí sítě 60](#_Toc436202676)

# Seznam tabulek

[Tab. 1 Typy relací v UML[1] 16](#_Toc436203435)

# Seznam zkratek

|  |  |
| --- | --- |
| ACID | Je zkratka pro 4 požadavky transakce, které musí splňovat:  Atomicity – nerozložitelnost, tzn. transakce musí proběhnout buď celá, nebo vůbec. A to i v případě totálního pádu serveru.  Consistency – konzistence, celá transakce se musí navenek jevit jako jediná operace.  Isolation – izolace, všechny změny provedené před potvrzením transakce musí být izolovány od zbytku systému.  Durability – stálost, po potvrzení transakce musí být databáze opět v konzistentním stavu a to natrvalo, aby se při selhání dala nějakým způsobem obnovit. |
| APS | Aplikační server |
| EA | Enterprise Architect – jedná se o modelovací nástroj pro zobrazování UML návrhu a byznys analýzy. Dobře se v něm dokumentují projekty, kde jeden diagram může říct víc, než celý dokument. |
| EF | Entity Framework – je objektově/relační mapování (ORM). |
| GUI | Grafické uživatelské rozhraní, které umožňuje ovládání grafických prvků |
| IL | Microsoft Intermediate Language – MSIL neboli zkráceně IL. Jedná se o strojový jazyk, do kterého platforma .Net převádí veškerý kompilovatelný kód. |
| LINQ | Language Integrated Query – integrovaný jazyk pro dotazování nad jakýmikoliv daty |
| MVVM | Model-View-ViewModel – je softwarová architektura navržená přímo pro technologii WPF. |
| PC | Počítač |
| OOP | Objektově orientované programovaní |
| ORM | Nástroj pro ukládání doménových objektů, které mají relaci do databáze |
| OS | Operační systém |
| UC | Use case – neboli také případ užití. Charakterizuje se určitou činností v systému za pomoci účastníka. |
| UML | Unified Modeling Language – unifikovaný modelovací jazyk pro specifikaci, vizualizaci, konstrukci a dokumentaci vývoje softwarových systémů |
| UP | Unified Process – metodika pro vývoj softwaru |
| WCF | Windows Communications Foundation – je komunikační technologie, která kombinuje všechny předešlé komunikační technologie .Net do jediné. |
| WPF | Windows Presentation Foundation – knihovna sloužící k vytváření uživatelského rozhraní aplikací, pomocí deklarativního programování jazykem XAML |
| XAML | Extensible Application Markup Language – je založený na XML jazyku, vektorově grafický značkovací jazyk sloužící k definování prezentační vrstvy grafického prostředí |

# Úvod

V dnešní době se skoro všude používají počítače. Může jít o usnadnění business procesů ve firmách nebo u zkoušek na vysokých školách. Daná instituce či firma chce, aby uživatelé na PC v pracovní době dělali jen svoji práci a netrávili čas např. hraním her, chatováním na sociálních sítích nebo opisováním z učebních dokumentů v elektronické podobě. Kontrola těchto uživatelů je velice nákladná. Čas je v dnešní době drahocenný! Není tedy reálné každému zaměstnanci či studentovi postavit za záda osobu, která by jej při práci hlídala.

Cílem této diplomové práce je vyvinout aplikaci složenou z klientského programu a aplikačního serveru, kde klientský program zachytává informace o spuštěných programech a přídavných zařízeních. Tyto informace se mají odesílat na APS, který je následně bude ukládat do databáze. APS bude jedním z počítačů v síti, který spustí komunikační službu mezi klientem a jím. Dále bude aplikace umět zobrazovat informace o uživatelích z databáze v reálném čase. Navíc umožní zobrazovat historické události o uživatelských aktivitách.

# Vývoj a projektovaní softwaru

Abychom se dobře seznámili s metodikami vývoje softwaru, musíme se nejprve dozvědět o fungování jazyka UML.

## UML

UML je jazyk pro specifikaci, vizualizaci, konstrukci a dokumentaci vývoje SW systému. V žádném případě se nejedná o programovací jazyk, neboť má syntaxi v grafické podobě a sémantiku, kde je výrazům přiřazen jistý význam. V odvětví analýzy a návrhu se stal standardem a proto by se v něm programátoři měli orientovat. Jedná se o užitečný nástroj k usnadnění návrhu a vývoje informačního systému.

### Historie UML

Při velké expanzi používání OOP se hledal způsob, jak se dívat na informační systémy. V 90. letech se firmě Rational Software podařilo pomocí několika metodik vytvořit standard UML. Po čase se ukázalo, že velké korporace mají zájem na tom, aby existoval nadále udržovatelný standard. Důvod byl jednoznačný – usnadnit komunikaci. Aby dokumentaci různých projektů rozuměli všichni a nemuseli se učit nový jazyk. Díky tomu výrazně klesly náklady na školení zaměstnanců. Následně vzniklo mezinárodní konsorcium s názvem OMG (Object Management Group), které na specifikace UML dohlíží. Instituce figurující v této společnosti, jako např. Oracle, Microsoft apod., společně vytvářejí a zdokonalují jazyk UML.

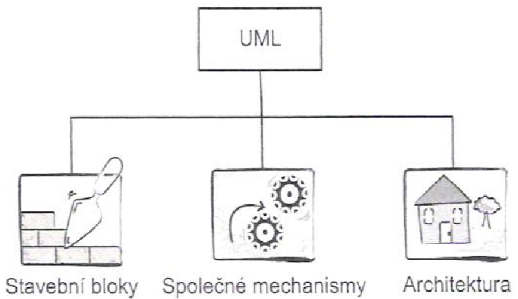
### Proč „UML“

V dnešní době roste komplexnost informačních systémů a doba, kdy celý systém naprogramoval jediný člověk, je pryč. Při tvorbě nových systémů je nutné nejdříve udělat návrh daného systému a pak jej předat týmu programátorů. Týmy musí mezi sebou dobře komunikovat a reagovat na změny ze strany zákazníka. UML pomáhá se s těmito problémy vypořádat. V etapě analýzy se komunikuje se zákazníkem – **co** se vlastně bude programovat. Dále v etapě návrhu se řeší – **jak** se to bude programovat.

### Struktura jazyka UML

Struktura obsahuje tyto části:

* **stavební bloky** – základní prvky modelu, relace a diagramy
* **společné mechanizmy** – způsoby, jimiž se v UML dosahuje společných cílů
* **architektura** – pohled na architekturu navrhovaného systému



Obr. 1 Struktura jazyka UML[1]

### Stavební bloky

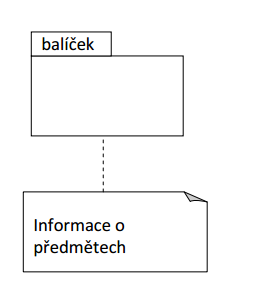
Dělí se na tři části:

* **předměty** – samostatné prvky modelu
* **vztahy** – poukazují na to, jaké jsou souvislosti mezi dvěma a více předměty
* **diagramy** – pohled na architekturu navrhovaného systému

### Předměty

Předměty jsou podstatnými jmény v modelu UML a dělí se následovně:

* **strukturní abstrakce** – jde o podstatná jména modelu UML jako třídy, rozhraní, spolupráce, případy užití, aktivní třída, komponenta a uzel
* **chování** – zahrnuje slovesa, např. interakce a stav
* **seskupení** – používané balíčky ke svázání významově souvisejících prvků
* **poznámky** – anotace, jež lze připojit k modelu pro zachycení důležitých informací



Obr. 2 Zobrazení balíčku a informace[1]

### Relace

Relace umožňují zobrazit vztah mezi jednotlivými předměty. Upřesňují jejich podstatu. Důležitou součástí je porozumění sémantice různých typů relace.

|  |  |  |
| --- | --- | --- |
| Typ relace | Syntaxe UML  zdroj cíl | Popis |
| závislost |  | Změna v určitém předmětu ovlivňuje význam závislého předmětu. |
| asociace |  | Popis množiny spojeny objektem |
| agregace |  | Cílový prvek součástí zdrojového prvku |
| kompozice |  | Silnější forma agregace |
| ochranná doba |  | Zdrojový prvek obsahuje cílový prvek |
| zobecnění |  | Jeden prvek je specializací jiného prvku a lze jej nahradit obecnějším prvkem |
| realizace |  | Asociace mezi klasifikátory, kde jeden klasifikátor určuje dohodu, jejíž uskutečnění zaručuje druhý klasif. |

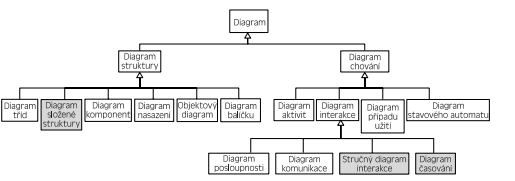
Tab. Typy relací v UML[1]

### Diagramy

V nástrojích založených na jazyku UML jsou vytvořeny předměty a relace nově vznikajícího modelu. Model se tak stává repositářem a popisuje požadované chování navrhovaného systému.

Diagramy jsou pohled na model, ale zároveň nejsou modelem. Rozdíl je v tom, že z diagramu se předměty dají odstranit, ale v modelu zůstávají.

Celkem existuje 13 základních typů diagramů v UML:



Obr. 3 Třináct různých diagramu v UML[1]

Diagramy rozdělujeme na dvě základní skupiny:

* **diagramy struktury** – určují strukturu systému, z jakých objektů jsou složeny
* **diagramy chování** – určují chování systému, tedy jak funguje

V diagramech chování je samostatná skupina:

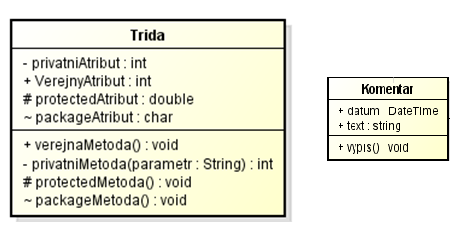
* **Diagramy interakce –** poukazují na interakci mezi jednotlivými částmi systému.

Každý diagram má svůj účel. Diagramy bychom měli dělat proto, že je potřebujeme a mají pro nás nějakou přidanou hodnotu. Jestliže nás někdo nutí vytvářet diagramy jen proto, aby existovaly, obvykle je to pouze ztráta času. Ukážeme si pár základních typů diagramů.

#### Diagram tříd

Jedná se o implementační diagram pro programátora, který by za pomoci tohoto diagramu neměl řešit žádné zásadní otázky ani problémy při programování. Návrh vytvářejí zkušení analytici nebo programátoři. Pokud je diagram tříd kvalitně zpracovaný a je zvolena vhodná míra detailu, může jej využít i méně zkušený a pro firmu méně finančně nákladný zaměstnanec. Finální podoba diagramu tříd již ve značné míře upouští od abstrakce, a je závislá na konkrétní implementační platformě. Musí obsahovat:

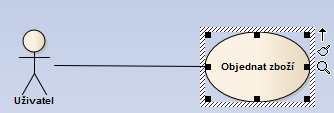
* identifikátor bez diakritiky
* atributy s přesně definovanými datovými typy pro zvolený implementační jazyk
* metody definované názvem buď s parametry



Obr. 4 Diagram tříd

#### Diagram případů užití

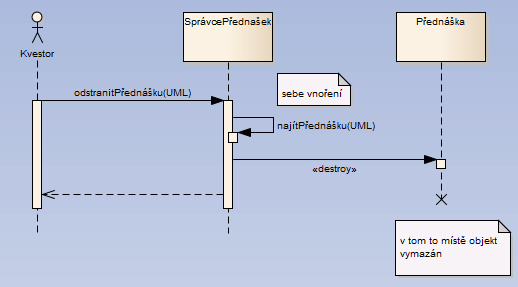
Zobrazuje chování systému z pohledu uživatele. Účelem tohoto diagramu je zachytit funkcionality systému. Diagram vypovídá o tom, co má daný systém umět, ale neudává, jak to bude dělat. Proto se při návrhu systému dělají jako první diagramy případů užití.



Obr. 5 Diagram případu užití

#### Sekvenční diagram

Vyjadřuje vzájemnou interakci a působení mezi více objekty. Podstatou sekvenčního diagramu jsou časové osy, které definují návaznosti na jednotlivé reakce. Na obrázku je na první pozici zobrazen aktér Kvestor, který odesílá zprávu odstranitPřednášku(„UML“) objektu SpravaPřednášek. Cílový objekt v sobě volá operaci najítPřednášku(„UML“). Pokud přednáška existuje, tak ji smaže a zde končí časová osa objektu Přednáška.



Obr. 6 Sekvenční diagram

### Obecná mechanika jazyka UML

Jazyk UML obsahuje čtyři společné mechanismy k modelování objektu, jež jsou používány v různých kontextech celého jazyka UML.

#### Specifikace

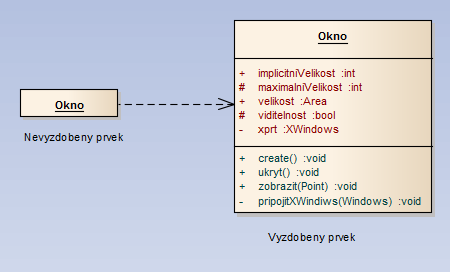
Má dva rozměry:

* **grafický** – znázornění pomocí diagramů
* **textový** – sémantický popis prvků

Souhrn specifikací utváří jádro modelu a dává mu smysl.

#### Ornamenty

Většina prvků je vyjádřena jednoduchým symbolem. Můžeme je tak třeba obohatit ornamentem, je-li potřeba zobrazovat více informací na diagramu. Ornamenty by se měly používat jen tehdy, pokud je to nezbytné – v případě, že zvyšují čitelnost a srozumitelnost diagramu.



Obr. 7 Diagram s ornamenty

#### Podskupiny

Popisují různá chápaní světa, v UML je dělíme následovně:

* **klasifikátor a instance** – klasifikátor je abstraktním vyjádřením typu předmětu, např. bankovní účet. Instance vyjadřuje abstraktní představy, např. můj bankovní účet nebo váš bankovní účet,
* **rozhraní a implementace** – rozhraní je tlačítko na obrazovce, jež na pozadí spouští složitý mechanizmus, který byl naimplementován.

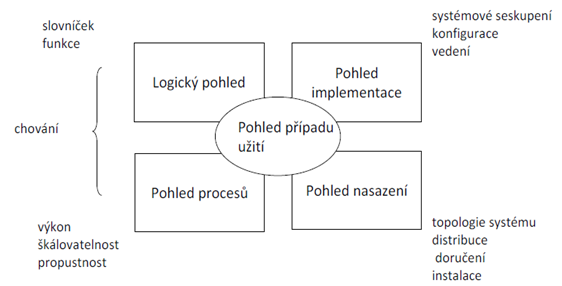
#### Mechanizmy rozšiřitelnosti

Mechanizmy rozšiřitelnosti slouží pro uspokojení všech potřeb uživatelů univerzálního modelovacího jazyka. Nejvíce se uplatňují tři jednoduché mechanizmy:

* **omezení** – rozšiřují sémantiku prvku, umožňují tak přidávat nová pravidla,
* **stereotypy** – definují nový prvek v modelu, jenž je založen na existujícím prvku – definuje se sémantika prvku,
* **označené hodnoty** – rozšiřují specifikace prvku, tím se přidá informace.

### Architektura

Zachycuje strategické aspekty vyšší struktury systému. Lze se na ni dívat několika způsoby. Nejčastější pohled je tzv. „4+1“.



Obr. 8 Architektura 4+1[1]

#### Logický pohled

Zdůrazňuje jak objekty, tak třídy. Tvoří základ systému implementující jeho chování.

#### Pohled procesů

Modeluje spustitelná vlákna a procesy. Je to procesová varianta logického pohledu, který obsahuje stejné artefakty.

#### Pohled implementace

Modeluje fyzické použití souborů a komponentů, které tvoří hotový kód systému. Znázorňuje závislosti mezi jednotlivými komponenty a spravuje konfiguraci množin vytvořených z těchto komponentů.

#### Pohled nasazení

Modeluje fyzické nasazení komponentů na výpočetním stroji.

#### Pohled případů užití

Případy užití jsou výchozím bodem pro všechny další pohledy. Jsou zde zachyceny základní požadavky na příslušný systém.

## Metodiky vývoje softwaru

Metodika vývoje softwaru se dotýká otázek „Proč?“, „Kdo?“, „Kdy?“, „Co?“. Jedná se o souhrn postupů vedoucích k dodání funkčního softwaru. V dnešní době se metodiky vývoje dělí na **Tradiční** a **Agilní**. Tyto metodiky se zavádějí a používají, abychom dosáhli ekonomické tvorby softwaru.

### Tradiční metodiky

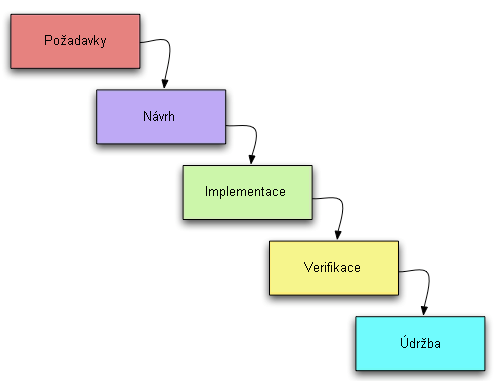
Problémem tradičních metodik byla jejich složitost a malá flexibilita. Nestačily reagovat na nově přicházející požadavky projektu. Příliš se zaměřovaly na striktní dokumentace a nejrůznější revize a schvalovaní. Ani postupné vylepšování těchto metodik nepomohlo vyřešit zmíněné problémy.

Pro ukázku jsem vybral 3 metodiky – Vodopád, Spirála a UP.

#### Vodopádový model (Waterfall)

Tento model je považován za úplně první ucelenou metodiku vývoje softwaru. Základním rysem jsou sekvenčně seřazené fáze bez iterací. Každá fáze musí projít procesem schvalování, aby mohla pokročit dále.

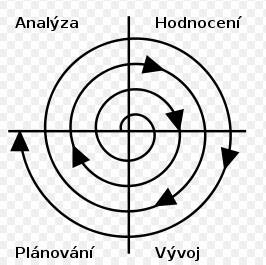
Posloupnost fází je zobrazena na obrázku.



Obr. 9 Schéma životního cyklu Vodopádu[8]

#### Spirálový model

Vodopádový model postupem času přestal vyhovovat, a proto se softwaroví inženýři snažili vymyslet nový přístup, který by jeho nedostatky eliminoval. Spirálový model vychází z vodopádu, ale přináší s sebou dvě nové vlastnosti – iterativní přístup a podrobnou analýzu rizik. Základními fázemi modelu jsou analýza, hodnocení, vývoj a plánování na další iteraci. Nevýhodou tohoto modelu je, že výsledný produkt je předán až po dokončení posledního cyklu. Tuto vlastnost zdědil po svém předchůdci. V každém cyklu je sice vytvořen prototyp, ale ten se může týkat pouze malých částí a nemusí fungovat v ostrém provozu.



Obr. 10 Spirálový model[8]

#### Unified Proccess (UP)

Smyslem metodiky je rozložit celý projekt na menší mini projekty. Každý ze zmiňovaných mini projektů je považován za iteraci. A každá iterace obsahuje prvky softwarového projektu:

* Plánování
* Analýza a návrh
* Tvorba
* Interní nebo externí uvedení

Všechny iterace mají takzvanou základní linii, která se skládá z části finální verze systému. Tyto linie jsou postupně na sebe vrstveny, dokud není dosaženo konečné podoby vytvářeného systému.

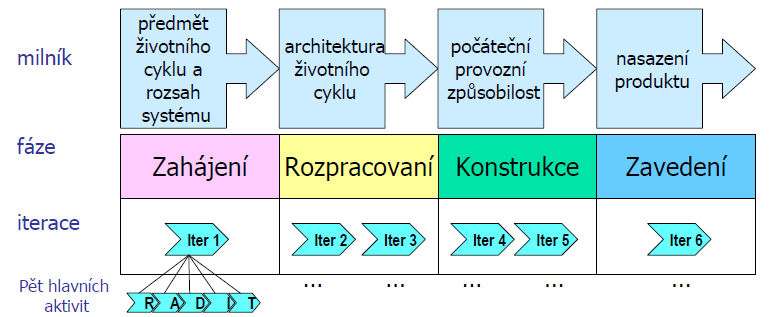
V každé iteraci je pět základních postupů, jež říkají, co je třeba udělat, a jakým způsobem toho dosáhnout.

* **Požadavky** – udávají, co by měl systém dělat
* **Analýza** – úprava požadavků a jejich strukturování
* **Návrh** – realizace požadavků v architektuře systému
* **Implementace** – tvorba softwaru
* **Testování** – ověření, zda implementace funguje podle očekávání

Životní cyklus projektu se rozděluje do čtyř fází:

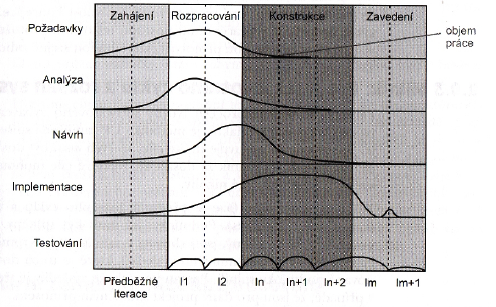
* **zahájení** – plánovaní
* **rozpracování** – architektura
* **konstrukce** – počátek provozu schopností
* **zavedení** – nasazení do uživatelského prostředí.

Všechny fáze končí nominovanými milníky (indikátor pokroku v projektu). Přesný počet iterací ve fázích určuje velikost projektu. Podle zkušeností by neměla trvat víc jak dva až tři měsíce.



Obr. 11 Diagram fází projektu[1]

Jak projekt přechází z jedné fáze do druhé, mění se objem vykonávané práce. Nejvíce času je věnováno požadavkům a analýze. Ve fázi rozpracování se zaměřujeme na sběr požadavků, analýzu a částečně na návrh. Fáze konstrukce se zaměřuje na návrh a implementaci. Poslední fáze zdůrazňuje testování a implementaci.



Obr. 12 Objem práce v jednotlivých fázích metodiky UP[1]

### Agilní metodiky

Agilní metodiky vývoje softwaru se zaměřují na týmovou spolupráci, otevřenou komunikaci, zapojení zákazníka a připravenost na změnu.

Všechny agilní metodiky sledují následující principy:

* **Iterativní a inkrementální vývoj s krátkými iteracemi**  
  Vývoj je rozdělen do fází, takže celková funkcionalita je dodávána po částech. Zákazník má možnost sledovat stav vývoje a vyjádřit se tak ke změnám. Na konci má jistotu, že nedostane to, co neočekával.
* **Komunikace mezi zákazníkem a vývojovým týmem**  
  Zákazník je přímo součástí vývojového týmu, má tak možnost okamžitě reagovat na výsledky vývoje daného systému. Podílí se na testování a dává přímou zpětnou vazbu pro vývojáře.
* **Průběžné automatizované testování**Z důvodů využívání iterací se aplikace mění velice rychle, proto je nutné průběžně ověřovat její funkčnost. Testy by měly být automatizované. Při každé změně by měla být sada testů, aby nedošlo k porušení již vytvořených částí systému.

#### Extrémní programování

Jedná se zřejmě o nejznámější zástupce agilních metodik. XP se hodí především pro malé týmy s nepříliš rozsáhlým projektem. Používají se zde postupy, které se dostávají až do extrému. Například, jestliže se osvědčují revize kódu, budeme neustále revidovat (myšlenka párového programování). Pokud se vyplácí testování, budeme nepřetržitě testovat jak my, tak i zákazník (testování jednotek, funkcionality, akceptační testy). Osvědčuje-li se návrh, stane se součástí naší každodenní činnosti (refaktorizace), atd.

#### Scrum

Principem této metodiky je iterativní vývoj definovaný na 3-8 fází tzv. sprintu, kde každá z nich trvá přibližně měsíc. Dále se zde zavádějí pravidelné schůzky vývojového týmu, tzv. *scummeetings.* Na schůzkách jednotliví členové sdělují svou aktivitu od poslední schůzky a probírají, co je potřeba v další fázi řešit. Každý sprint je ukončen funkční demo-aplikací, která je předána zákazníkovi. Ten následně poskytne zpětnou vazbu.

# Technologie

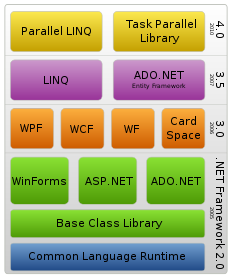
Volba technologií je nedílnou součástí vývoje softwaru, a alespoň stručný teoretický základ tak nelze v této práci vynechat. Technologie umožňují vývojářům tvorbu daného softwaru na zakázku. Vyšší programovací jazyky dnes pomáhají programátorům s rychlejším a komfortnějším vývojem. Technologie použité v této práci jsou produktem firmy Microsoft, a běží na platformě .Net Framework. Ke zvolení těchto technologií mě vedly zkušenosti s frameworkem a rozsáhlé používání OS Windows.

## .Net Framework



Obr. 13 Logo .Net Framework[7]

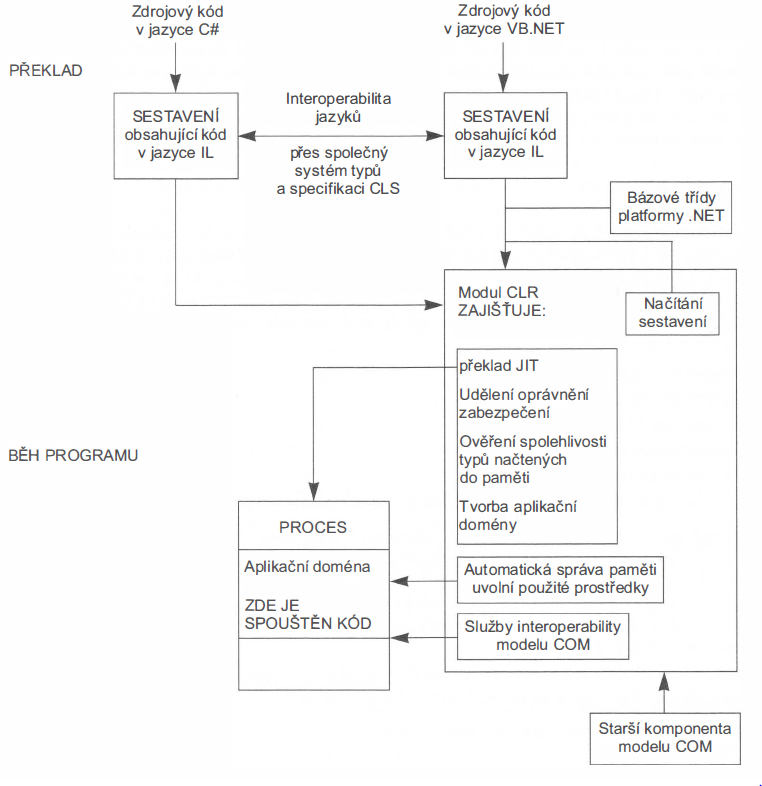
„Platforma **.NET“** již dnes sahá do všech softwarových řešení Microsoftu. .NET tvoří soubor technologií v softwarových produktech, který je dostupný jak pro desktop, tak i pro Web. Velkou výhodou je jazyková nezávislost. Proto existuje celá řada podporovaných jazyků, kde neznámější jsou C#, VisualBasic .NET, J#, F# a spravované C++. Díky podpoře více jazyků mohou mezi sebou sdílet knihovny, které předpřipravují řadu struktur a komponentů pro práci např. s konzolí, uživatelským GUI, databázemi apod.



Obr. 14 Knihovnyverzí .NET Framework[7]

### Architektura .NET Framework

Tuto architekturu si popíšeme jen ve stručnosti, zaměříme se pouze na důležité části. Obr. 15 Architektura .NET Framework zobrazuje kostru zmíněné architektury.



Obr. 15 Architektura .NET Framework [3]

#### CLR

Jedná se o nejdůležitější část běhového prostředí pro spuštění řízeného kódu, včetně alokace a řízení vláken. Před spuštěním libovolného kódu musí proběhnout překlad ve dvou krocích:

* překlad zdrojového kódu do jazyka IL
* překlad jazyka IL do kódu specifického pro cílovou platformu pomocí modulu CLR

#### Jazyk IL

Je založen na myšlence nízko úrovňového jazyka s jednoduchou syntaxí (která pracuje s čísly, nikoliv s textem). Lze tak rychleji přeložit nativní strojový kód. Syntaxe přináší výhody nezávislosti na platformě a spolupráci mezi jazyky.

#### CTS

Zabezpečuje správu mezi řízeným kódem a běhovým prostředím. Systém pak určuje základní datové typy a jejich hierarchii, kterými disponuje jazyk IL.

#### CLS

Specifikace jazyka zajišťuje společně s CTS spolupráci jazyků. CLS musí zahrnovat sadu minimálních standardů, které dodržují překladače platformy .NET. Autoři s oblibou překladače omezují, aby mohli dobře podporovat množinu funkcí jazyka IL a systém CTS. Překladač tak podporuje vše, co je definováno ve specifikaci CLS.

#### Automatická správa paměti

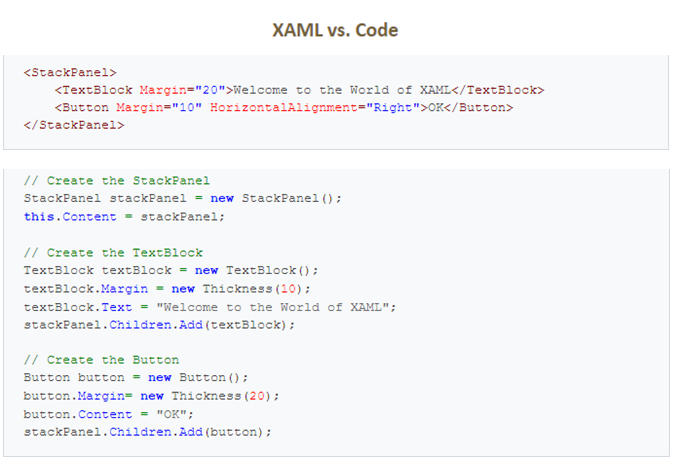
Hlavním úkolem je získávat zpět paměť, kterou požadovaly spuštěné aplikace. Běhové prostředí dnes používá automatickou správu paměti, takzvaný *garbagecollector*. Jedná se o program, jenž slouží k čištění paměti. Přidělená paměť je na haldě, a v časových intervalech zjišťuje, jestli není halda zaplněna, a případně volá čištění paměti. Program pak prochází proměnné na haldě a kontroluje jejich reference. Všechny objekty, které na sebe nemají referenci, jsou automaticky odebrány, protože se nepočítá s tím, že by se měly nadále používat.

## WPF

Jedná se o rozšířené rozhraní .NET Framework od verze 3.0. Knihovna slouží k vytváření uživatelského rozhraní na klientských aplikacích. V porovnání s Windows Form, který používá nativní prvky systému Windows založených na pixelech, WPF využívá rozhraní DirectX vektorovou grafiku. WPF má jedno veliké plus, a to, že umožňuje rozdělení práce pro vývojáře a návrháře. WPF je založeno na jazyku XAML.

### Jazyk XAML

XAML je zápis definovaný hierarchickou strukturou uživatelského rozhraní. Vizuální prezentace je tvořena značkami (*tagy*), které odpovídají dané třídě a atributu. Většina elementů v sobě může obsahovat další elementy, které jsou instancemi dalších tříd. Jazyk XAML zkracuje zápis o proti .NET programovacím jazykům.



Obr. 16 XAML vs C# code[14]

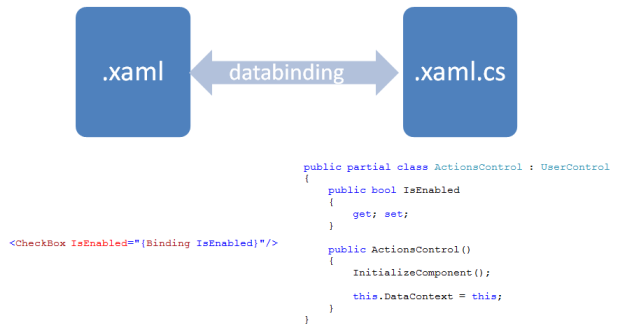
Silnou stránkou XAMLu jsou předdefinované základní animační prvky. Dále umí využívat šablony a styly. V XAMLu můžeme také tvořit vlastní komponenty.

#### Kód na pozadí

Jazyk XAML definuje především vzhled aplikace. Do kódu na pozadí zasahujeme pouze tehdy, kdy potřebujeme vykonávat nějakou složitou logiku, např. reagovat na spuštěné události.

#### Data-Binding

Ve WPF se jedná o techniku propojení XAML elementů a dat. Změna provedená u jednoho z prvků se projevuje i na druhém. Dále je nutné specifikovat výběr propojení jednosměrné nebo obousměrné. Pak se tu mohou provádět validace a konvertovat data, např. z čísla na barvu atd. Tato technika nabízí rozsáhlé možnosti.

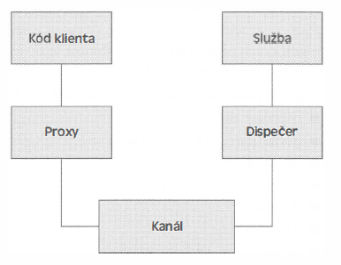


Obr. 17 Vazba mezi cílem a zdrojem[14]

## WCF

Příchod .NET Framework v. 3.0 s sebou přinesl novou komunikační technologii, která kombinuje všechny staré technologie v jeden model: Web service, Remoting, Message queuning a Enterprice service. WCF tímto snižuje odbornou znalost na technologie pro programátory.

Princip WCF je jednoduchý – jde o zasílání zpráv klienta službě a jejich následný příjem. Komunikace mezi procesy či systémy probíhá po internetu nebo síti.



Obr. 18 Komunikace mezi klientem a službou[3]

Z vyplývá, žeklient nejprve volá metodu proxy serveru, který je vydefinován službou. Samotné volání metody probíhá komunikačním kanálem. Kanál se skládá ze serverové a klientské časti, které spolu komunikují pomocí síťových protokolů. Zpráva poslaná prostřednictvím kanálu je odchycena dispečerem, který ji předá službě na zpracování.

Základním stavebním prvkem je **služba**, jež je tvořena jedním nebo více endpointy, které slouží k odesílání SOAP zpráv. Služba publikuje metadata, jež jsou tvořena těmito částmi:

* třída služby
* hostovací prostředí
* jeden nebo více endpointů

**Endpoint** je místo sloužící k příjímání a odesílání zpráv. Je tvořen třemi částmi ABC:

* **adress –** slouží k identifikaci, kam budou zprávy zasílány. Jde o místo, kde služba běží
* **binding –** používá se pro komunikaci se službou. Určuje, jaký protokol se použije, výběr bezpečnosti, transakce, session atd.
* **contract –** specifikuje rozhraní služby, metody a další

**SOAP** jsou zprávy zasílané mezi klientem a serverem, jsou definovány XML schématem. Jedná se o požadavek a odpověď pomocí komunikačního kanálu. Typy komunikace:

* **oneway** – klient odešle zprávu a neočekává odpověď
* **request**-**response** – klient odešle požadavek a čeká na odpověď
* **duplex** – obousměrná komunikace mezi klientem a službou probíhá asynchronně

**Metadata** slouží k popisu služby. Tento popis udává všechny důležité údaje k nakonfigurování klienta. Díky tomu klient ví, na jakém protokolu běží služba.

**Hostovací prostředí** je místo, kde služba poběží např. IIS, Windows proces nebo bude součástí nějaké aplikace.

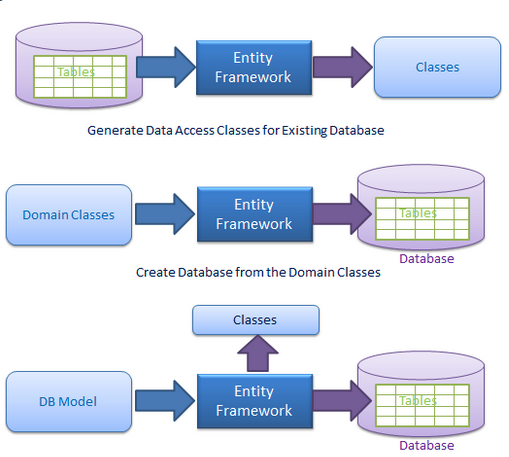
**Kanál** je prostředí pro přenášení zpráv. Prostředí vzniká pouze v tom okamžiku, kdy klient zasílá nějakou zprávu.

**Proxy** je prostředník mezi klientem a serverem, používá se na straně klienta. Pokud klient chce komunikovat se serverem, musí vždy vytvořit instanci proxy třídy. Metody publikované službou jsou pak obsaženy v proxy třídě.

## Entity Framework

Microsoft ADO.NET Entity Framework je objektově/relační mapování (ORM). Framework umožňuje vývojářům pracovat s relačními daty jako s objekty konkrétní domény. Vývojáři tak mohou používat dotazy LINQ pro získávání a manipulaci s daty. EF rozšiřuje ADO.NET a umožňuje tak automatizovaný přístup k datům v databázi.

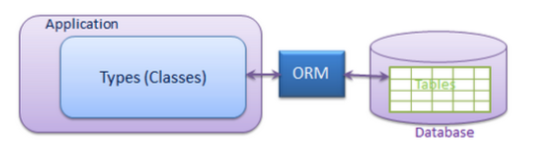
EF využívá tři scénáře. První scénář mapuje již existující databáze k aplikaci a vytváří tak doménové třídy. Druhý scénář vytváří databázi pomocí doménových tříd. A třetí vytváří doménové třídy a databázi na základě schématu databáze vytvořeného v designovém prostředí.



Obr. 19 Schéma tvorby nebo přístupu k databázi[11]

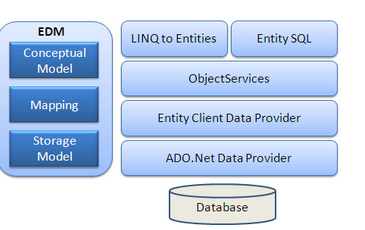
### O/RM

Jedná se o nástroj pro ukládání doménových objektů, které mají relaci do databáze bez složitého programování. Doménové objekty se mapují na objekty databáze (tabulky, pohledy a procedury). ORM tak umožňuje oddělit databázový návrh od doménových tříd. Tím se aplikace stává udržitelná a rozšiřitelná. Také automatizuje operace jako CREATE, READ, UPDATE a DELETE, bez nutnosti psát ručně kód.



Obr. 20 Schéma ORM[11]

### Architektura



Obr. 21 Architektura EF[11]

#### Entity data model

EDM se skládá ze tří částí: Koncepční model, Model uložiště a Mapování.

#### Koncepční model

Obsahuje modelové třídy a jejich vztahy, nezávislé na návrhu databáze.

#### Model úložiště

Je návrh databáze, který obsahuje tabulky, pohledy, procedury atd.

#### Mapovaní

Skládá se z informací o mapování konceptuálního modelu na model úložiště.

#### LINQ to Entities

Jde o dotazovací jazyk používaný na modely objektů. Vrací entity, které jsou definovány v konceptuálním modelu.

#### Entity SQL

Je rovněž dotazovací jazyk, ale odlišný od LINQ to Entites.

#### Object Service

Představuje hlavní bod pro přístup k datům z databáze.

#### Entity client data provider

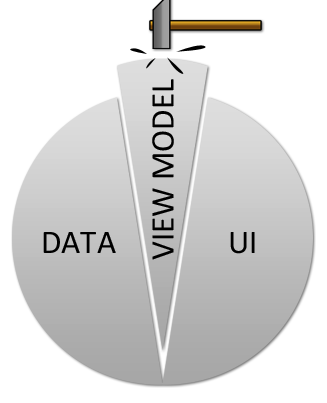
Hlavním úkolem této vrstvy je převést L2E nebo Entity SQL na dotaz SQL, kterému rozumí databáze. Řeší komunikaci s ADO.NET data providerem, který odesílá nebo načítá data z databáze.

#### ADO.NET data provider

Tato vrstva komunikuje s databází pomocí standartního ADO.NET.

### Návrhový vzor MVVM

MVVM je jedním z návrhových vzorů pro aplikace WPF. Umožňuje oddělení uživatelského rozhrání od logiky aplikace. Kód se tak stává velice přehledným a další implementace je z pohledu programátora intuitivnější. Hlavní myšlenka je velice prostá – vytvořit třídu, která drží stav aplikace ViewModel. Ta se dotazuje na uživatelské GUI, kde jsou vykreslovány ovládací prvky. Pokud uživatel naopak provede nějakou změnu v GUI – přenese se změna do ViewModelu.

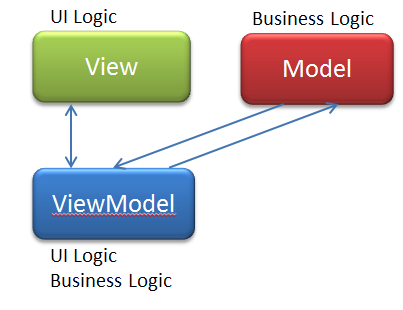


Obr. 22 Princip MVVM[13]

View model je nejdůležitější třída, poskytuje totiž data GUI (View). Poskytovaná data jsou ve strukturách vyvolávajících události při změně. To znamená, pokud dojde ke změně na View, automaticky se projeví změna ve ViewModelu. Využívají se dva základní kameny kolekce ObservalCollection<T> – přidání a odebraní prvku. Dále rozhraní INotifyPropertyChanged popisující změnu vlastností ve ViewModelu.

**Vrstvy MVVM**:

* Model – tvoří jej doménové třídy, připojení do databáze, mapování tabulek databáze (*viz.* Entity Framework) a reference na komunikační služby.
* ViewModel – spojuje model s view, a drží tak stav aplikace. Prvky jsou propojeny pomocí bindingu, díky němuž čerpají svůj obsah. Provádí se tu filtrování dat podle stavu aplikace.
* View – je prezentační vrstva napsaná v XAML.



Obr. 23 Schéma komunikace MVVM[13]

## SQLite database

SQLite není plnohodnotnou databází typu Oracle, MS SQL Server apod. Našla si ale svoje místo ve světě mobilních technologií i některých desktopových aplikací. SQLite je knihovna soběstačná bez serveru, s nulovou konfigurací a s transakčním SQL enginem. Databáze SQLite je k dispozici zdarma. Lze ji využít jak pro komerční, tak pro soukromé účely. Implementuje všechny základní "povinnosti" standardní databáze ACID: Atomicita, konzistence, izolovanost operací a trvanlivost dat. Manipulace s daty se provádí pomocí jazyka SQL. Od uživatelů (programátorů) je tedy vyžadována vyšší znalost syntaxe SQL. Velkou výhodou SQLite je podpora EF.

## Jazyk C#

Díky specifikaci CLS v technologii WPF může být použit kód jakéhokoliv jazyka z .NET frameworku. Volbou pro tuto práci je jazyk C#. Jedná se o objektově orientovaný jazyk. Nejvíce je inspirován jazykem C++ a Visual Basic.

Mezi kladné vlastnosti patří:

* dokumentační komentáře XML
* podpora vlastností a událostí
* využití LINQ
* dostupnost rozhraní Windows API
* zapouzdřené metody s názvem *„delegáti“*, které povolují typovou bezpečnost
* možnost využití ukazatelů a přímý přístup do paměti
* automatická správa paměti

# Realizace práce

Při realizaci aplikace bylo vyžadováno použití jedné z metodik pro vývoj softwaru. Byla zvolena metodika UP, jež je založena na frameworku pro iterativní vývoj softwaru s využitím UML. Vývoj se dělí na čtyři fáze: zahájení, rozpracovaní, konstrukce a zavedení.

V každé z těchto etap je pět hlavních aktivit:

* Požadavky
* Analýza
* Návrh
* Implementace
* Testování

## Krok první „Zahájení“

V tomto kroku se snažíme nashromáždit co nejvíce požadavků na systém a pokládat si při tom otázku „Čeho chceme dosáhnout?“, nikoliv „Jak toho chceme dosáhnout?“.

Pro sběr požadavků se využívá několik technik: revize dokumentů, rozhovory, dotazníky, pozorovaní atd. Podle daných požadavků by měl být vytvořen návrh architektury systému.

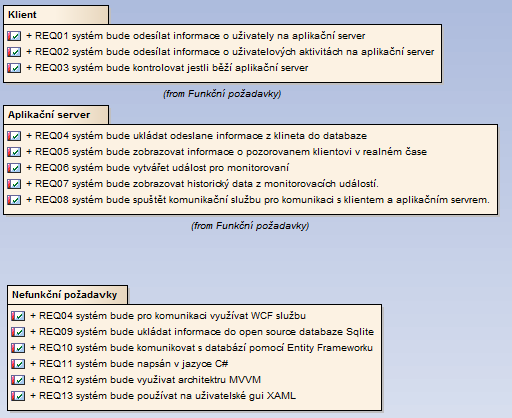
## Zadání práce

Cílem práce je vyvinout aplikaci složenou z klientského programu a aplikačního serveru. Klientský program bude sledovat aktivity uživatele (připojení USB disku, spuštění aplikace apod.) definované aplikačním serverem a odesílat je na něj. Aplikační server tyto aktivity zaznamená do databáze nebo souboru a zobrazí je na obrazovce.

## Požadavky

Při definování požadavků bylo nutné pečlivě prostudovat zadání. Dalším krokem byl rozhovor se zadavatelem. Požadavky byly rozděleny do dvou skupin podle konvencí metodiky UP. Jedná se o dělení na funkční a nefunkční. Funkční požadavky představují, jaké funkcionality má systém nabízet. Nefunkční požadavky kladou důraz na omezení, vzhled a provedení.

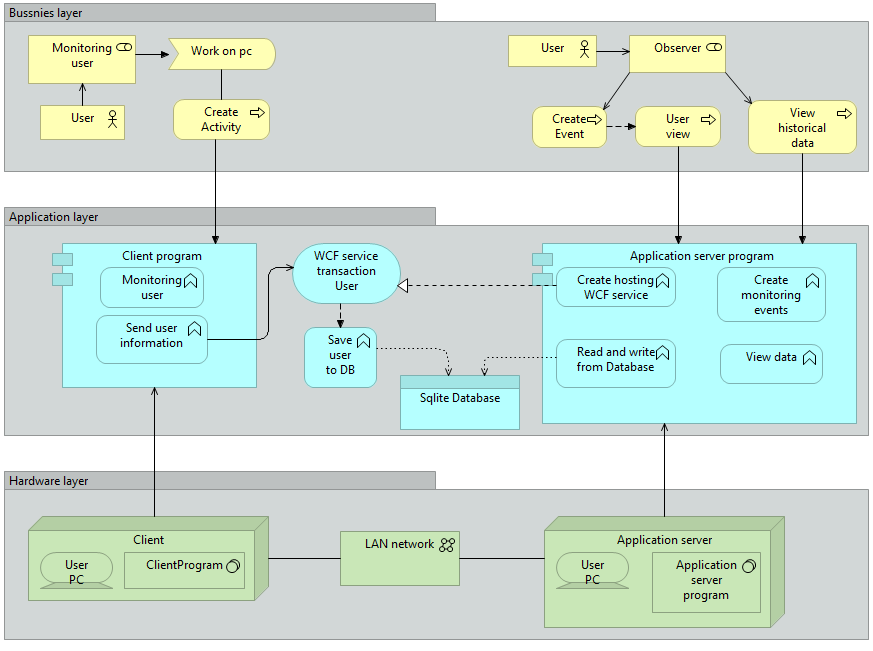
Požadavky pro tuto práci definované a zaznamené v EA:



Obr. 24 Diagram Funkčních a nefunkčních požadavků

## Návrh architektury systému

Základní návrh byl realizován v ArchiMate modelovacím nástroji. Jedná se o zdarma dostupný open-source nástroj pro tvorbu architektonických modelů mezi jednotlivými vrstvami. V tomto případě zachycuje tři vrstvy: business, aplikační a hardwarovou. Na obrázku níže vidíme, jak jednotlivé vrstvy mezi sebou komunikují.



Obr. 25 Diagram architektury v ArchiMate

#### Business vrstva

Popisuje uživatele, kterému byla přiřazena role monitorování uživatel. Při své práci na PC vytváří aktivity pomocí zapnutých aplikací. Druhý uživatel je v roli pozorovatele, který vytváří proces: vytvoř monitorovací událost připojených uživatelů a zobraz jejich informace. Dále může nahlížet do historie monitorovaných událostí a procházet si revizi.

#### Aplikační vrstva

Aplikační vrstva zobrazuje klientský program s jeho funkcemi, které reprezentují monitorovaní uživatelových aktivit a odesílání pomocí komunikační služby WCF. Následně jsou informace o uživatelích ukládány do databáze. Aplikační program spouští komunikační službu pro klienty, aby mohli komunikovat s databází. Další funkcionalitou je vytváření monitorovaných událostí, které se ukládají do databáze a vytvářejí tak relaci mezi monitorovanými uživateli a monitorovací událostí. V  neposlední řadě zobrazuje informace o aktuálně vytvořené monitorovací události s informacemi o sledovaných uživatelích nebo si můžeme zobrazit historické události se sledovanými uživateli a jejich aktivitami v minulosti.

#### Hardwarová vrstva

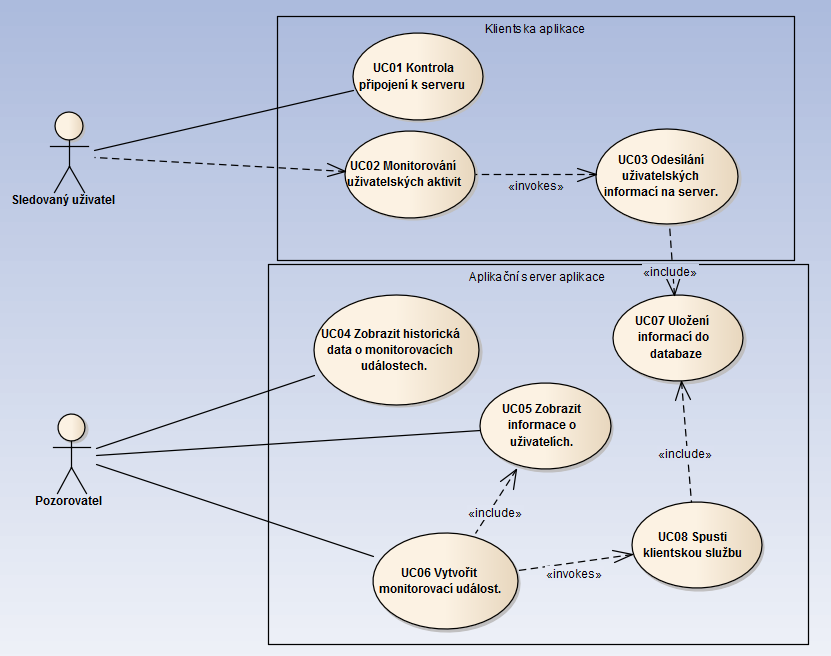
Reprezentuje fyzická zařízení spojená sítí LAN. Na klientských PC poběží monitorovací program, který bude pomocí sítě posílat zprávy na Aplikační server. APS bude reprezentován jedním z PC v LANu, kde bude zobrazovat nashromážděné výsledky.

## Krok dvě „Rozpracování“

V této fázi byly schváleny požadavky a návrh architektury komunikace mezi jednotlivými vrstvami. Následujícím úkolem je vypracování případů užití a popis jejich chování. Dále tvorba analytických tříd a datového modelu.

### Případy užití

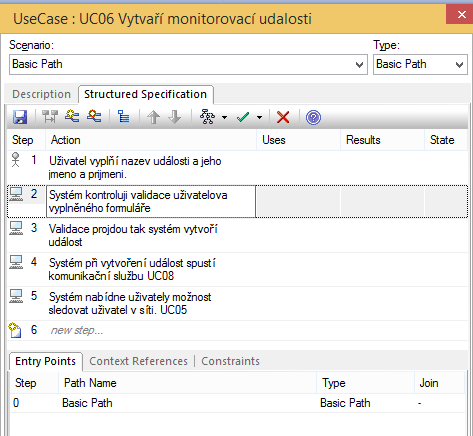
Diagramy případů užití slouží k zobrazení aktérů, kteří komunikují s informačním systémem. Komunikace mezi uživatelem a systémem poukazuje na funkcionalitu, která se dostává určitému aktérovi. V našem případě se vyskytují dva základní aktéři: **Sledovaný uživatel** a **Pozorovatel**. **Sledovaný uživatel** využívá funkce klientské aplikace, kde zjišťuje, zda je stále připojen k serveru. Při vytváření aktivit na PC dochází k jeho monitorování a odesílání informací na server. Naproti tomu **Pozorovatel** používá aplikaci na aplikačním serveru. **Pozorovatel** má k dispozici funkcionalitu pro vytváření událostí sledování uživatelů. Ti jsou sledováni od vzniku události až po její skončení. Je tedy zaznamenáván celý její životní cyklus. Následně si **Pozorovatel** může nechat zobrazit historické záznamy všech událostí.



Obr. 26 Případ užití mezi klientem a APS

Na obr. 26 můžeme vidět relace, resp. asociace include a invokes. V tomto případě asociace poukazuje na UC, které může aktér využívat. Invokes vyvolává další funkčnost a include rozšiřuje funkčnosti vyvolávajícího UC.

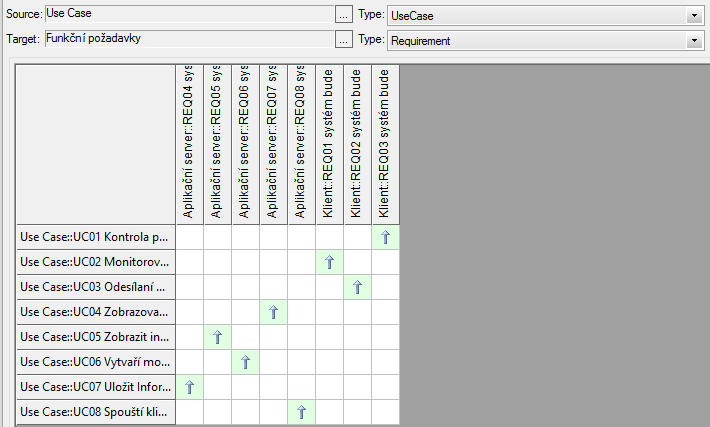
K identifikaci případů užití se používá označení UCXX (X představuje číslice) a název, který by měl vypovídat o chování v systému. V programu EA je možné zachytit úplné chování UC se vstupními/výstupními podmínkami pomocí scénářů.



Obr. 27 Scénář případu užití

Scénáře jsou skvělou pomůckou pro ujasnění chování uživatele a systému v případě, že nevíme, jak si název UC vysvětlit. Scénáře samozřejmě nezachycují podrobný výklad chovaní, ale spíše podstatu UC.

Každý funkční požadavek musí být pokryt případem užití. Pro tyto účely nabízí EA užitečnou funkci – matici požadavků a případů užití. Pomocí šipek tak lze sledovat, jaké požadavky jsou pokryty případem užití.



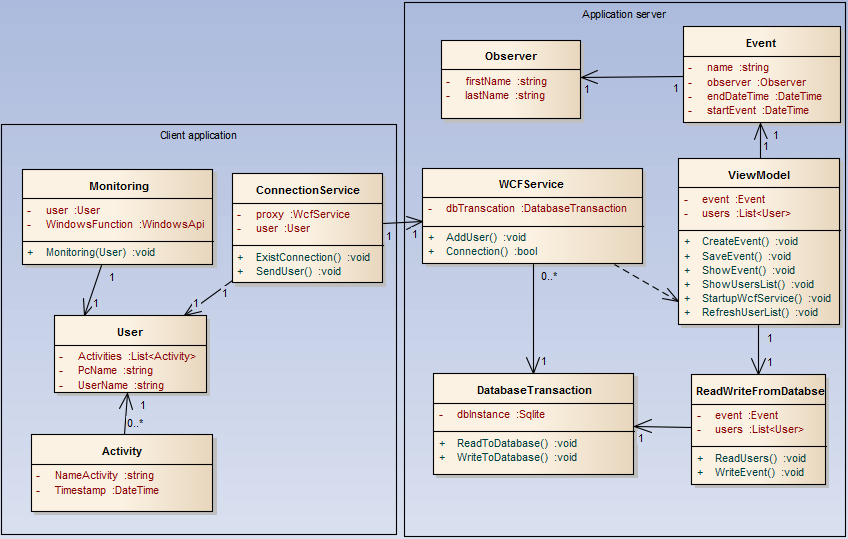
Obr. 28 Matice pokrytí požadavků případy užití

### Analytické třídy

Myšlenkou analytické třídy je zachytit podstatu abstrakce obchodního modelu. Analytické třídy neslouží jako návrh k implementaci, jedná se spíše o předvoj pro návrh tříd. Později se třídy zpravidla ještě rozdělí do více menších objektů s přesnou implementací. Anatomie analytické třídy by měla zobrazovat:

* název – ten je povinný
* atribut – datové typy nejsou povinné
* operaci – jen obecně vyjádřit odpovědnost
* typ viditelnosti – nespecifikuje se
* stereotyp – pouze pro vylepšení modelu
* označené hodnoty – mohou být

Na je zachycen obchodní model systému a komunikace mezi klientskou aplikací a aplikací na APS.



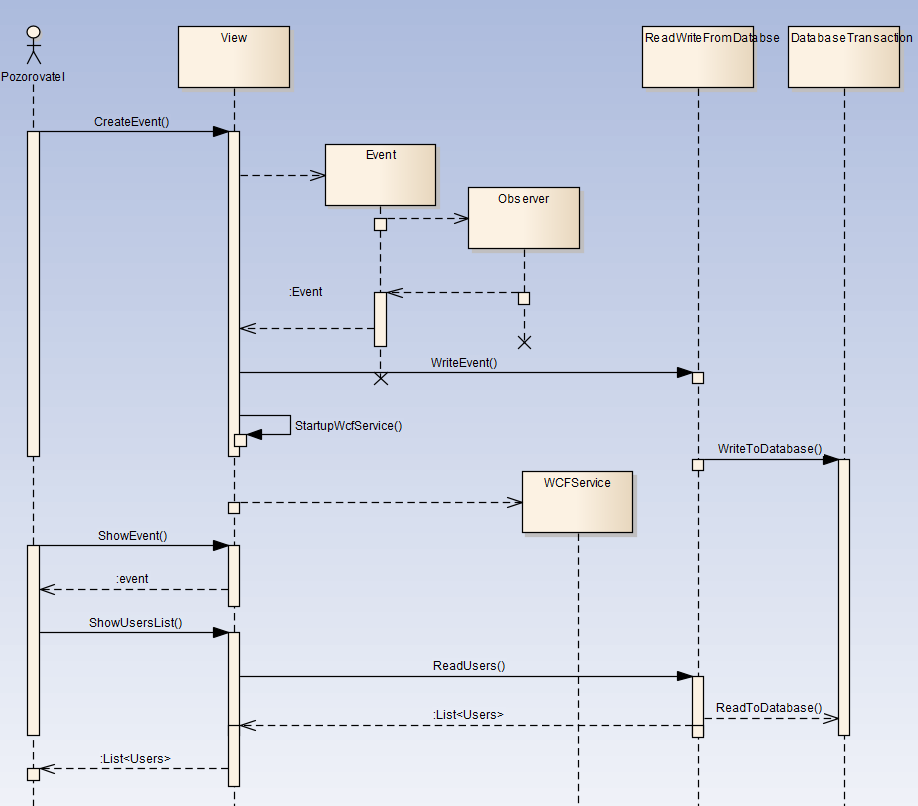
Obr. 29 Diagram analytické třídy

**Klientská aplikace** bude obsahovat informace uživatele s jeho aktivitami, dále monitorovací třídu, která si bude zjišťovat pomocí Windows API funkci informace o otevíraných aplikacích uživatele. A v neposlední řadě třída připojení ke službě, jenž bude monitorovat připojení ke službě a odesílat informace na APS. Třída **User** ponese v sobě atributy o názvu uživatele, názvu PC a dále tu bude seznam aktivit. **Activity** v sobě ponesou název aktivity a časové razítko, kdy daná aktivita začala.

**Aplikační server** obsahuje třídu **WCFService**, která poskytuje metody pro komunikaci s klientskou stranou. Tato třída posílá informace o uživatelích do třídy **DatabaseTransaction**, kde se budou informace ukládat do SQLite databáze nebo se naopak načítat pro **ViewModel** třídu, která si bude držet stav aplikace. Bude mít přehled o vytvořené monitorovací události, také osobu co dohlíží na monitorovací událost a seznam sledovaných uživatelů s jejich aktivitami. **Event** je třída, jež si bude uchovávat informace názvu události, třídu o dohlížející osobě a kdy začala a skončila daná událost. **Observer** obsahuje jen jméno a příjmení

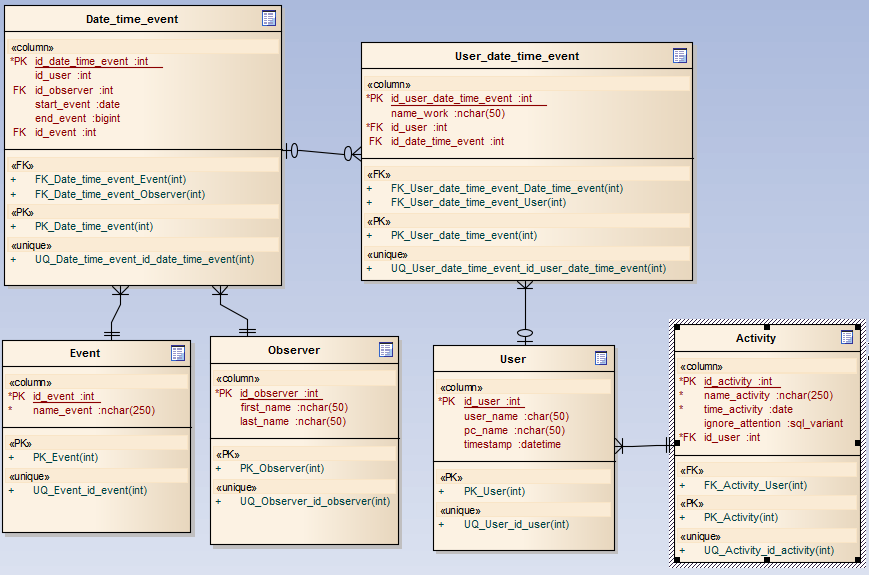
### Sekvenční Diagram

Ve chvíli, kdy máme definované případy užití a model analytických tříd, můžeme si podle scénáře vytvořit sekvenční diagram, který zachycuje časové osy objektů a komunikací mezi nimi. Sekvenční diagram si znázorněme na UC06.



Obr. 30 Sekvenční diagram UC06

### Datový model



Obr. 31 Datový model

Datový model je jedním z hlavních bodů této aplikace. Byl vytvořen na základě analytického modelu tříd a normálních forem. Pomocí normálních forem byla eliminována redundance dat, což ušetřilo čas strávený nacházením potenciálních chyb návrhu.

**Popis jednotlivých tabulek v datové modelu:**

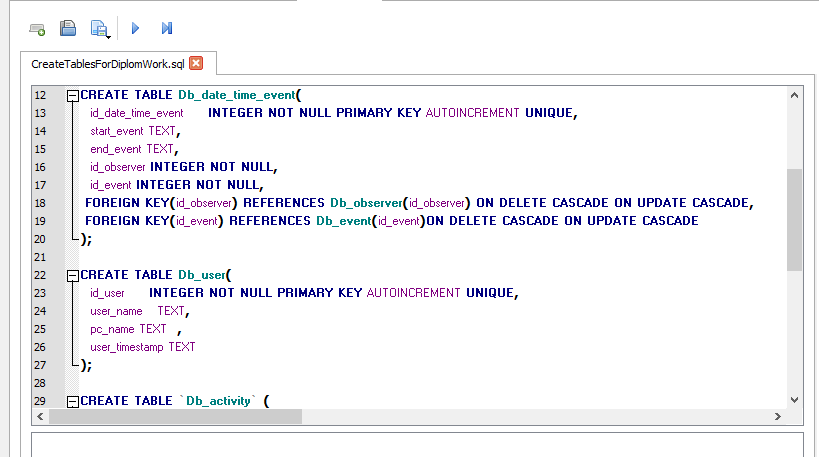
* Event – zajišťuje pouze název události.
* Observer – má v sobě pouze jméno a příjmení o pozorovateli.
* Date\_time\_event – určuje, kdy monitorovací událost začala a skončila. Má v sobě odkaz na Event a Observer, jenž se na tom podílel.
* User\_date\_time\_event – říká, kolik monitorovaných uživatelů je k date\_time\_event připojeno a na čem měli uživatelé pracovat pomocí atributu name\_work.
* User – nese v sobě základní informace, jako je název uživatele v počítači, název počítače a časové razítko. Časové razítko se využívá jen pro poslední odeslání informací na server.
* Acitivity– popisuje název aktivity, kterou uživatel vykonával s časovým razítkem. Atribut ignore\_attention využívá pouze pozorovatel k tomu, aby mohl ignorovat varování o nevyžádané aplikaci uživatele.

## Krok třetí „Konstrukce“

Fáze konstrukce zahrnuje implementaci navržených modelů do vývojových nástrojů pro tvorbu softwaru. Databáze byla vytvořena v nástroji SqliteBrowser. Jedná se o freewarový nástroj pro vytváření SQLite databází. Tento nástroj umožňuje základní práci s databázemi. Konkrétně jde například o tvorbu SQL dotazů do DB, prohlížení dat v GUI, ukládání důležitých SQL dotazů do souborů či reverzování databáze. Dalším nástrojem vývoje bude Visual studio 2013 Ultimate Edition s ReSharprem. Visual studio 2013 je podle mého názoru výborným vývojovým prostředím pro tvorbu softwaru na platformě .NETframework. ReSharper je nástroj pro Visual studio 2013 sloužící pro refaktorizaci kódu a jeho snadnou udržovatelnost. Ve Visual studio 2013 budeme používat tyto technochnologie:

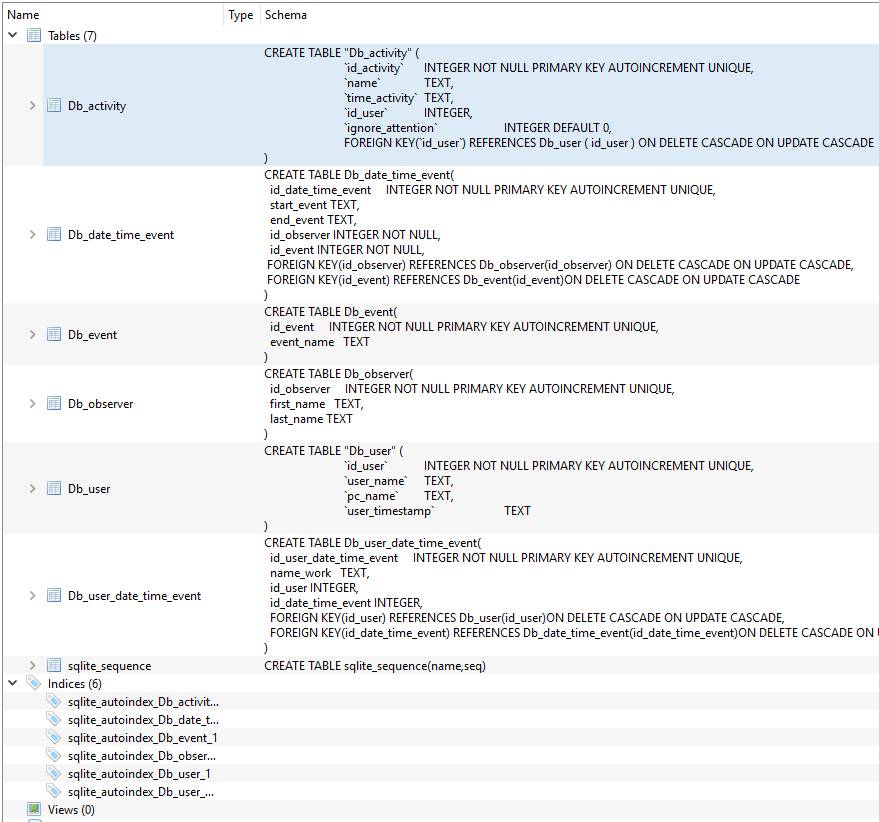
* Programovací jazyk C#
* WPF
* WCF
* Entity Framework

### Implementace datového modelu

Návrh datového modelu byl původně vytvořen v aplikaci EA. Bohužel, EA nepodporuje přímý import datového modelu do SqliteBroswseru. Umožňuje sice generování skriptů ve formátu sqlfile, ale SqliteBrowser si pro změnu neporadí s DB CONSTRAIN. Model tak posloužil jako podrobný vzor pro „ruční“ tvorbu tabulek v jazyce SQL. 

Obr. 32 SQL syntaxe pro tvorbu tabulek

zachycuje připojení prefixu Db\_ všem tabulkám. Důvod bude vysvětlen v části Entity Framework.



Obr. 33 Schéma vytvořených tabulek

Jak můžeme na schématu vytvořených tabulek vidět, některé atributy jako user\_timestamp nepodporují datový typ čas. Řešení je tedy následující – ukládat a číst časové datové typy jako text, který je následně nutné transformovat v technologii, kterou využíváme pro zápis a čtení.

### Implementace aplikace

Aplikace je rozdělena na dva samostatné moduly: klientský s názvem Big Brother a serverový Big Brother Viewer. Hlavním cílem práce bylo vyvinout aplikaci, která bude monitorovat aktivity uživatelů. Aktivity se následně mají odesílat na aplikační server k nahlížení pozorovateli, který vytvořil událost pro sledování uživatelů. Příkladem události je např. zkouška z předmětu C#, kdy je žádoucí, aby byli studenti (uživatelé) pod neustálým dohledem. Nyní se podíváme na konkrétní implementaci s přímou ukázkou částí kódu.

#### WCF Služba

WCF služba se vytváří jako samostatná knihovna v projektu. V ní je definováno rozhraní ILibrary a třída Library, jež implementuje toto rozhraní.

namespace WcfServiceLibrary

{

[ServiceContract]

public interface ILibrary

{

[OperationContract]

void AddUser(User user);

[OperationContract]

bool IsAlive();

}

}

V rozhraní jsou vydefinovány metody, které bude služba nabízet. Pomocí atributu [ServiceContract] bude služba poskytovat metody tohoto rozhraní. Atribut [OperationContract] specifikuje, že metoda bude použita pomocí služby.

namespace WcfServiceLibrary

{

[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]

public class Library : ILibrary

{

public void AddUser(User user)

{

DBTransaction dbTransaction = DBTransaction.ReturnDatabaseInstance();

dbTransaction.AddUser(user);

}

public bool IsAlive()

{

return true;

}

}

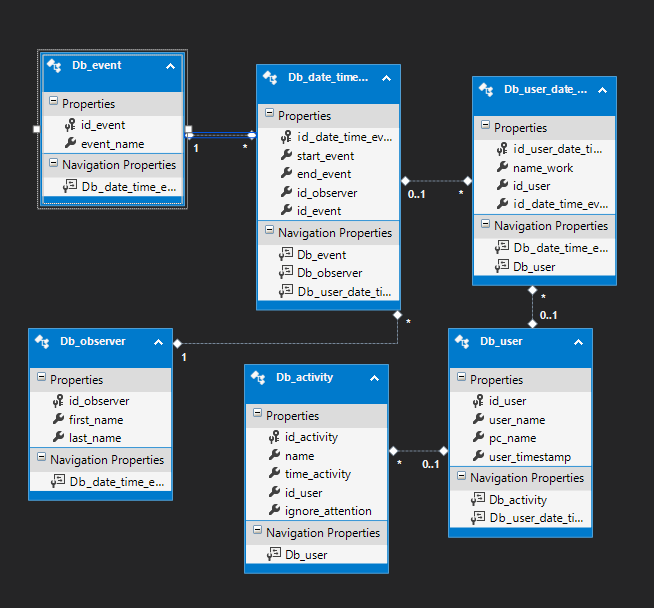
}

Metody v této třídě už definují logiku toho, co se bude dělat. Metoda IsAlive slouží ke kontrole, zda služba na klientské straně běží. AddUser metoda získává instanci ke třídě, která má v sobě zabudovanou funkcionality pro operace s databází. V našem případě se jedná o uložení parametru typu User do databáze. Atribut [ServiceBehavior] je tu pro vytváření jednotlivých instancí této třiídy a slouží klientům, kteří začnou službu využívat.

Služba WCF je hostována v programu Big Brother Viewer, a také bude spouštěna na PC, kde tato aplikace poběží. Zde se služba spustí. Vzhledem k tomu, že program Big Brother má referenci na tuto službu, může k ní bez problémů přistupovat.

#### Entity Framework

Vytvořil jsem si vlastní knihovnu SqliteDatabase.dll. Knihovna se také nachází v programu Big Brother Viewer spolu s databázovým souborem. Jejím obsahem jsou operace s daty pomocí EF. V tomto případě jsem EF mapoval na již existující soubor databáze BigBrotherDB.sqlite. Visual studio si vytvořilo schéma databáze s relacemi, a dále vytvořilo třídy reprezentující tabulky v databázi i s jejich atributy, které jsou ve třídě reprezentovány jako vlastnosti.



Obr. 34 Schéma vygenerované EF

Databázovým tabulkám byl přidán prefix Db\_. Při použití stejných názvů tabulek, jako u doménové třídy, v aplikaci Big Brother Viewer docházelo ke kolizím. Z tohoto důvodu byl použit pomocný prefix. Všechny vygenerované třídy se nacházejí pod třídou BigBrotherEntites, která je odvozena od třídy DbContext. DbContext je velice důležitý, jedná se o most mezi vygenerovanými třídami a databází.

public Db\_user GetUserFromDb(int userId)

{

using (var context = new BigBrotherEntities())

{

return context.Db\_user.SingleOrDefault(u => u.id\_user == userId);

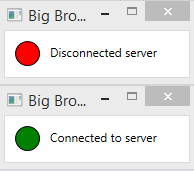
}

}

Metoda Db\_user vytvoří instanci BigBrotherEntities a z ní si načte kolekci Db\_user. Následně s pomocí LINQ vyhodnotí výraz hledaného Id uživatele a vrátí instanci Db\_user. Jinak řečeno, BigBrotherEntities vytvoří spojení s databází a LINQ se převede na SQL syntaxi a vrátí jeden záznam, který odpovídá výrazu hledaného Id. Ze záznamu udělá instanci třídy Db\_user, se kterou můžeme pracovat v C#. V knihovně je řada metod pro práci s daty z databáze, ale jsou tak implementačně rozsáhlé, že se jimi nebudeme podrobně zabývat. Myslím, že tento malý příklad je dostačující a pro vysvětlení fungovaní EF stačí. V EF se dotazy nemusí psát jen pomocí LINQ. Další možností je napsat dotaz SQL pomocí klasického stringu – v této práci jsem však tuto možnost nevyužil.

#### Klientská aplikace Big Brother

Klientská aplikace má za úkol monitorovat uživatelovy aktivity na PC. Konkrétně sleduje, kolik času strávil na jednotlivých aktivitách. V určitých časových intervalech následně odesílá informace na server. Při spuštění aplikace se zobrazí malé okno, které informuje uživatele o tom, zda je připojen na server nebo ne.



Obr. 35 Stavy klientské aplikace

Na pozadí probíhá získávání informací o uživatelových aktivitách pomocí funkcí Windows API. Funkce se defaultně volají každou sekundu.

class WindowsApiFunction

{

//HWND WINAPI GetForegroundWindow (void);

[DllImport("user32.dll")]

public static extern IntPtr GetForegroundWindow();

// int WINAPI GetWindowText (\_\_in HWND hWnd,\_\_out LPTSTR lpString, \_\_in int nMaxCount);

[DllImport("user32.dll")]

public static extern int GetWindowText(IntPtr hWnd, [Out]StringBuilder lpString, int nMaxCount);

}

Obě tyto funkce se nacházejí v knihovně *user32.dll.* Pro naimportování této knihovny se používá atribut [DllImport]. Funkce GetForegroundWindow slouží ke zjišťování aktivních oken, funkce GetWindowText vrací název aktivního okna. Pomocí těchto funkcí je každou vteřinu kontrolováno, jakou aplikaci má uživatel otevřenou. Pokud se názvy aplikací liší, dojde k vytvoření nové instance Activity s názvem aplikace. Následně se přidá záznam do kolekce uživatelových aktivit s aktuálním časovým razítkem. Časový rozestup mezi posledním razítkem a právě vytvořenou aktivitou udává čas, který uživatel strávil na předchozí aktivitě.

Informace o uživateli, jako název uživatele a název PC, jsou získávány z knihovny WindowsIdentity, která obsahuje metodu GetCurrent, jež vrací text s názvem uživatele a jeho PC. Tento text pak pomocí parsingu rozděluji dle potřeby. Dalším požadavkem na aplikaci bylo zaznamenávání přípojených USB zařízení. Pro tuto potřebu existuje knihovna DriveInfo, která v sobě obsahuje kolekci přípojných zařízení k PC. V této kolekci je potřeba USB pomocí výčtu DriveType.Removable najít.

Připojení k WCF službě funguje tak, že aplikace má vygenerovanou referenci na službu, jenž vytváří třídu pro komunikaci se službou LibraryClient. Třída má několik konstruktorů – s parametrem i bez parametru. Bezparametrický konstruktor si bere informace o **Endpointu** služby a **Bindingu** z konfiguračního souboru App.config. V aplikaci jsem použil konstruktor bez parametru. Nastavuje se tu adresa z mého vlastního konfiguračního souboru s adresou **Endpointu** a časovým intervalem pro posílání informací na server. **Binding** je napevno nastaven v kódu třídou NetTcpBinding. Je nutné mít nastavený port a IP adresu počítače, na kterém aplikace Big Brother Viewer běží. Řetězec může mít např. následující podobu:

<?xml version="1.0" encoding="utf-8" ?>

<ConnectionServer>

<Address>net.tcp://192.168.0.102</Address> --IP adresa hostované služby

<Port>8080</Port> -- port

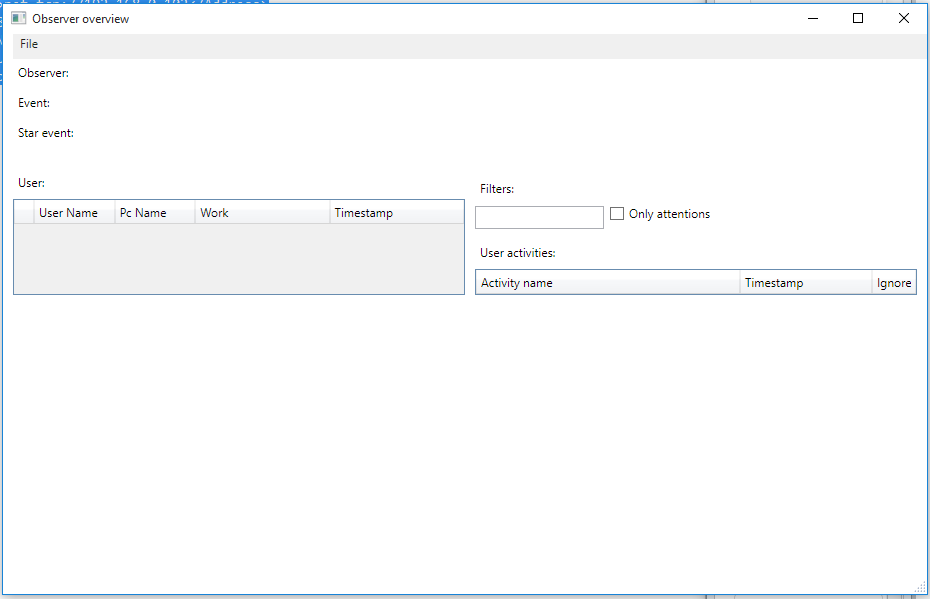
<ServiceName>Monitoring</ServiceName> --název služby, která poskytuje metody

<SendingUserInformationTimeIntervalInSeconds>60</SendingUserInformationTimeIntervalInSeconds> -- časový interval odesílání informací o uživateli na APS v sekundách

</ConnectionServer>

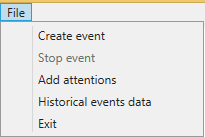
#### Aplikace Big Brother Viewer na APS

Tato aplikace má za úkol sledovat aktivity uživatelů připojených v síti ve vytvořené události Pozorovatele. Vytvořením události se spustí hosting pro komunikační službu WCF. Po skončení události se služba ukončí. Jak ukazuje následující obrázek, po zapnutí aplikace jsou ovládací prvky na obrazovce úplně prázdné.



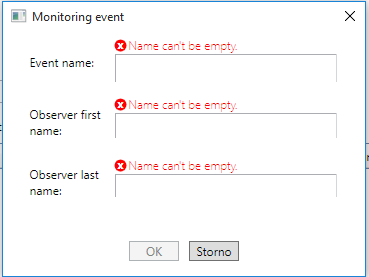
Obr. 36 Úvodní obrazovka Big Brother Viewer

Pro vytvoření události je nutné v horní části obrazovky aplikace otevřít položku **File.**  Nyní máme na výběr z několika možností.



Obr. 37 Položka File z menu

Volbou **Create event** se zobrazí dialogové okno. V okně je nutné vyplnit jméno a příjmení pozorovatele a název události. Jednotlivá textová pole jsou v XAMLu opatřeny nutnými validacemi. Buňky nesmí být prázdné a musí obsahovat minimální délku znaku. V opačném případě dialogové okno nedovolí vytvoření události.



Obr. 38 Dialogové okno pro tvorbu události.

Implementace šablony pro validační hlášky u textové pole v XAMLu.

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<ControlTemplate x:Key="ValidationErrorTemplate">

<DockPanel>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

<Grid Width="12" Height="12">

<Ellipse Width="12" Height="12"

Fill="Red" HorizontalAlignment="Center"

VerticalAlignment="Center"

></Ellipse>

<TextBlock Foreground="White" FontWeight="Heavy"

FontSize="8" HorizontalAlignment="Center"

VerticalAlignment="Center" TextAlignment="Center"

ToolTip="{Binding ElementName=ErrorAdorner,

Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"

>X</TextBlock>

</Grid>

<TextBlock Foreground="Red" FontWeight="12" Margin="2,0,5,0"

Text="{Binding ElementName=ErrorAdorner,

Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"

></TextBlock>

</StackPanel>

<AdornedElementPlaceholder

x:Name="ErrorAdorner" ></AdornedElementPlaceholder>

</DockPanel>

</ControlTemplate>

</ResourceDictionary>¨

Následující úryvek kódu zachycuje implementaci TextBoxu ve View. Obsahuje mj. šablonu pro validační hlášky a třídu NameValidator, jež má za úkol kontrolovat vstupy do TextBoxu.

<TextBox Name="ObserverLastName" Grid.Row="2" Grid.Column="1" Margin="5,22,-27,15" Height="24" VerticalAlignment="Center" Validation.ErrorTemplate="{StaticResource ValidationErrorTemplate}" KeyboardNavigation.TabIndex="2">

<TextBox.Text>

<Binding Path="LastNameObserver" Mode="TwoWay"

UpdateSourceTrigger="PropertyChanged">

<Binding.ValidationRules>

<viewModel:NameValidator ValidatesOnTargetUpdated="True">

</viewModel:NameValidator>

</Binding.ValidationRules>

</Binding>

</TextBox.Text>

</TextBox>

Třída NameValidator je napsaná v jazyce C# a dědí třídu ValidationRule. Ve třídě jsou implementovány velice jednoduché, ale účelné validace.

public class NameValidator : ValidationRule

{

public override ValidationResult Validate(object value, CultureInfo cultureInfo)

{

if (value == null)

return new ValidationResult(false, "Name can't be empty.");

if (value.ToString().Length < 3)

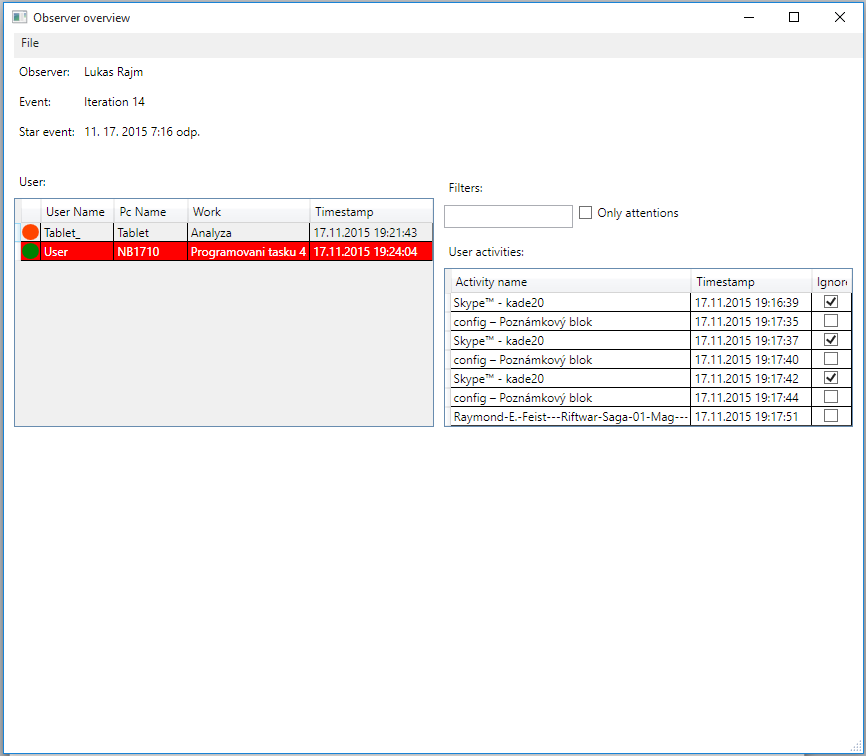
return new ValidationResult(false, "Name must be more than 3 characters long.");

return ValidationResult.ValidResult;

}

}

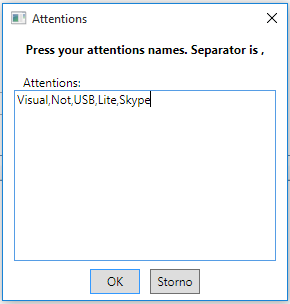
Po vytvoření události dochází každou minutu k načítání uživatelů s jejich aktivitami z databáze. Jde o uživatele, kteří jsou na službu připojeni. Tito se budou na obrazovku vypisovat. Hlavní obrazovka zobrazuje, jaký pozorovatel založil událost, jak se událost jmenuje a kdy začala. Uživatelé a jejich aktivity jsou tu zobrazováni pomocí komponenty DataGridu. Tabulka uživatelů ukazuje stav připojení, což je reprezentováno zeleným kolečkem. Dále je tu název uživatele na PC a název uživatelova PC. Sloupec work, který vyplňuje pozorovatel, je tu proto, aby se vědělo, na čem daný uživatel pracuje (např. *„Implementace Tasku 153“)*. V poslední řadě je tu časové razítko, které zobrazuje, kdy naposledy přišli informace o uživateli na server. Při kliknutí na některého uživatele se zobrazí seznam jeho aktivit, které vytvořil. Dále je tu název aktivity, informace o tom, kdy byla spuštěna a jestli na daném řádku chceme ignorovat upozornění na nevyžádanou aktivitu. Upozornění na nevyžádanou aktivitu se projeví začervenáním řádku v aktivitě. Tímto způsobem je na uživatele poukazováno.



Obr. 39 Zobrazení sledovaných uživatelů.

Filters nad User activities umožňují filtrovat seznam aktivit, které hledáme. Do textového pole napíšeme část textu. Seznam se nám zkrátí jen na aktivity, které tento text v názvu obsahují. Při zadávání je nutné zohledňovat velká a malá písmena. Další funkcionalitou je CheckBoxOnlyAttentions – ten v seznamu zobrazí pouze nevyžádané aktivy.

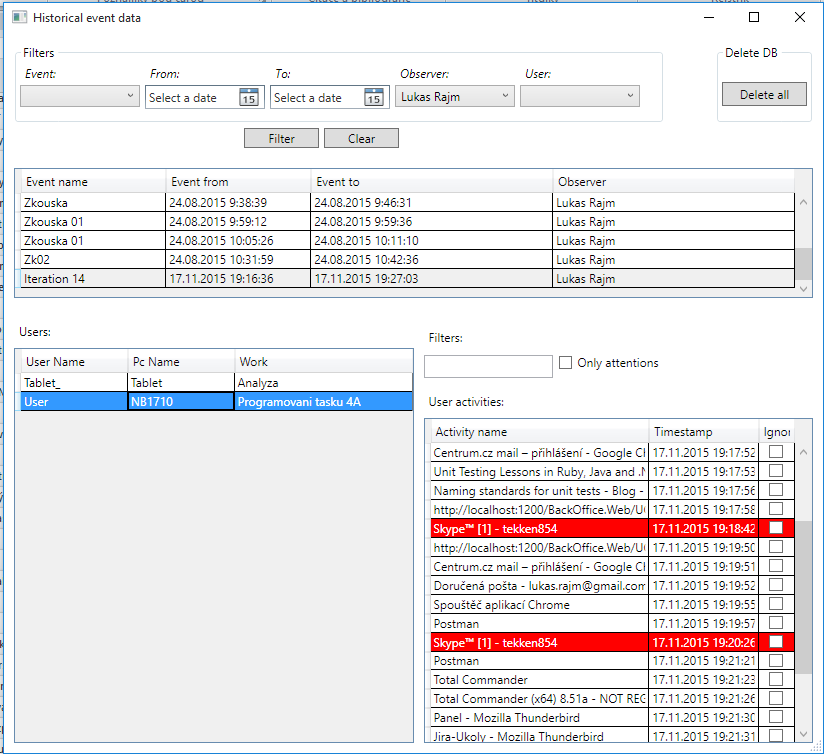
Nevyžádané aktivity lze definovat v menu **File->Addattentions.** Tato volba otevře modální okno se skupinou názvů aktivit. Jde o aktivity, na které chce být pozorovatel upozorňován. Jednotlivé názvy "hlídaných" aktivit se v dialogovém okně odděluje čárkou.



Obr. 40 Modální okna seznamu upozorňovacích názvu

Pro ukončení sledovací události ve **File->Stop event** se ukončí a vypne hostovaní komunikační služby a přidělí se jí čas ukončení.

Pokud chceme prohlížet historii sledovaných událostí, musíme se přepnout do nového okna přes **File->Historical events data**



Obr. 41 Obrazovka historických data

Obr. 41 zachycuje sekci filtru, kde si můžeme vyfiltrovat události, které obsahují uživatele s aktivitami. V tomto případě jsem nechal vyhledávat pozorovatele a uživatele. Zobrazí se tak pouze události, které obsahují jméno filtrovaného pozorovatele a filtrovaného uživatele. V tabulce událostí se zobrazí název události, pozorovatel a časová razítka začátku a konce událostí. Po kliknutí na konkrétní událost se zobrazí seznam uživatelů. Při kliknutí na řádek uživatele se ukáže seznam aktivit.

#### Shrnutí

Obě dvě aplikace jsou vytvořeny návrhovým vzorem MVVM. Business logika je tedy na úrovni modelu, View-model si udržuje stav aplikace. Pomocí události ovládá vstup a výstup na View.

## Krok čtvrtý „Zavedení“

Ve fázi zavedení se očekává, že implementace obou aplikací je kompletní a aplikace jsou spustitelné. Prioritou je testovat monitorování uživatelských aktivit na klientských počítačích a odesílat údaje na aplikační server. Pozorovatel potom může aktivity uživatelů sledovat.

### Testování

Po dokončení implementace aplikací proběhlo jejich testování. Testování bylo uskutečněno na vlastní domácí síti. APS byl stolní počítač a běžela na něm aplikace Big Brother Viewer. Dále byly využity dva notebooky a tablet s Windows 8. Na těchto zařízeních byla spuštěna aplikace Big Brother. Po vytvoření sledovací události se všechna zařízení dala do stavu **Connected to server**. Následně se pomocí komunikační služby na stolní počítač odesílali informace o uživatelských aktivitách. Odesílání probíhalo v předem daných intervalech. Na cílovém stolním počítači se data ukládala do databáze. Poté byla načtena aplikací Big Brother Viewer a zobrazena pozorovateli. I po ukončení sledovací události lze historická data nalézt v databázi.



Big BroteherViewer

Big BroteherViewer

Big Brother

Big Brother

Big Brother

Big Brother

Big Brother

Big Brother

Router

Router

LAN

LAN

Obr. 42 Schéma domácí sítě

Testování lze označit za úspěšné. Aplikace splnily svůj účel – sledovaly aktivity uživatelů na jejich zařízeních a odesílaly informace na Aplikační server, kde byly pozorovateli poskytnuty k nahlížení.

# Závěr

První část práce je věnována teoretickému základu metodik vývoje softwaru a jazyka UML. Tento blok je sice relativně rozsáhlý, nicméně je nezbytný k objasnění toho, jak v dnešní době probíhá vývoj softwaru ve firmách. Z textu jasně plyne, že k vývoji nestačí jedna jediná osoba, která jen sepíše kód a software ihned publikuje. Celý proces je z pravidla mnohem složitější. Správně zvolené metodika vývoje softwaru šetří čas, peníze, lidské zdroje a zlepšuje komunikaci mezi týmy. O její důležitosti tedy není pochyb.

Druhá část práce je věnována technologiím. Konkrétně jde o technologie pro návrh a implementaci vyvíjeného softwaru. Některé technologie, které kapitola popisuji, jsou posledních pár let velkým hitem mezi .NET vývojáři. Důvodem je usnadnění práce při vývoji a vyšší flexibilita. Většina z představovaných technologií není obsahem učiva VŠ, bylo tedy nutné vynaložit značné úsilí při samostudiu.

Praktická část práce popisuje samotnou realizaci vývoje aplikace. Při vývoji byla použita metodika UP. Text postupně dokumentuje všechny čtyři základní fáze této metodiky. První fází je analýza, jež zahrnuje vytvoření požadavků na software a vytvoření business architektury. Ve fázi rozpracování je pomocí diagramů případů užití zachycena funkcionalita a chovaní systému. Dále je zde vytvořen datový model pro ukládání dat do databáze. Následuje fáze implementace, která popisuje vytvoření databáze na základě datového modelu v nástroji SqliteBrowers. Pro implementaci samotné aplikace jsou využity jazyky C# a XAML. V závěrečné fázi je popsáno zavedení aplikace a testování. Aplikace je nasazena v domácí síti, kde jsou připojena čtyři zařízení. Tři zařízení jsou připojena na aplikační server, kam odesílají informace o uživatelích. Čtvrté zařízení je v roli pozorovatele a sleduje činnosti jednotlivých uživatelů. Dle mého názoru aplikace nabízí veškerou požadovanou funkcionalitu. Testování funkčnosti dopadlo dobře.

Závěrem bych rád uvedl, že aplikace je plně funkční a nic nebrání jejímu případnému nasazení na větší síť. A tak ji lze využít např. při testování studentů na stolních PC.

# Použitá Literatura

1. ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.
2. BISHOP, J. C#: návrhové vzory. Vyd. 1. Brno: Zoner Press, 2010, 328 s. Encyklopedie Zoner Press. ISBN 978-80-7413-076-2.
3. NAGEL, Christian. C# 2008: programujeme profesionálně. Vyd. 1. Brno: Computer Press, 2009, 2 sv. (1126, 772 s.). Programujeme profesionálně. ISBN 978-80-251-2401-7.
4. LÖWY, Juval. Programming WCF services. 2nd ed. Beijing: O'Reilly, c2009, xxxi, 750 p.:. ISBN 978-0-596-52130-1.
5. PETZOLD, Charles. Mistrovství ve Windows Presentation Foundation: [aplikace = kód + markup]. Vyd. 1. Brno: Computer Press, 2008, 928 s. Mistrovství. ISBN 978-80-251-2141-2.
6. SQLite web page [online]. [cit. 2015-08-25]. Dostupné z: <https://www.sqlite.org/about.html>
7. Jazyk C# [online]. [cit. 2015-08-25]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/z1zx9t92.aspx
8. Metodika vývoje softwaru [online]. [cit. 2015-08-25]. Dostupné z: <https://cs.wikipedia.org/wiki/Metodika_v%C3%BDvoje_softwaru>
9. Klasické a agilní metodiky vývoje software [online]. [cit. 2015-08-25]. Dostupné z: <http://blog.nny.cz/740763-klasicke-a-agilni-metodiky-vyvoje-software.php>
10. WCF tutorial [online]. [cit. 2015-08-25]. Dostupné z: <http://www.wcftutorial.net>
11. Entity Framework tutorial [online]. [cit. 2015-08-25]. Dostupné z: <http://www.entityframeworktutorial.net/>
12. SQLite: Databáze pro váš web [online]. [cit. 2015-08-25]. Dostupné z: <http://www.zdrojak.cz/clanky/sqlite-databaze-pro-vas-web/>
13. Mvvm: model-view-viewmodel [online]. [cit. 2015-08-25]. Dostupné z: <http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel>
14. WPF Tutorial.net [online]. [cit. 2015-08-25]. Dostupné z: http://www.wpftutorial.net/

# Seznam Příloh

Příloha A – CD-ROM obsahující:

* Projekt s aplikacemi Big Brother a BigBortherViewer s databází
* Návrhy a diagramy projektu v EA
* Návrh byznys modelu v ArchiMate
* Dokument diplomové práce