



Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

# Dokumentácia k riadeniu projektu

Autori: Jakub Mačina, Jozef Staňo, Matej Leško, Jozef Gáborík  
Predmet: Tímový projekt  
Akademický rok: 2016/2017  
Dátum odovzdania: 18. 11. 2015

## Obsah

1	Úvod .....	5
2	Tím.....	6
2.1	Členovia tímu .....	6
2.1.1	Jozef Gáborík .....	6
2.1.2	Matej Leško .....	6
2.1.3	Jakub Mačina .....	6
2.1.4	Jozef Staňo.....	6
2.2	Roly členov tímu .....	7
2.2.1	Manažérske roly .....	7
2.2.2	Krátkodobé roly.....	7
3	Používané metodiky .....	8
3.1	Manažment kvality, podpory vývoja a integrácie .....	9
3.1.1	Metodika verziovania aplikácie .....	9
3.1.2	Metodika organizácie zdrojových kódov.....	10
3.1.3	Metodika písania unit testov.....	13
3.1.4	Metodika prehliadky kódu .....	15
3.2	Manažment rizík .....	16
3.3	Manažment plánovania a rozsahu .....	19
3.3.1	Spôsob plánovania a riadenia rozsahu.....	19
3.3.2	Plán.....	20
3.4	Manažment komunikácie a ľudských zdrojov .....	22
3.4.1	Metodika komunikácia v tíme .....	22
3.5	Manažment tvorby dokumentácie .....	25
3.5.1	Metodiky pre tvorbu dokumentácie .....	25
4	Aplikácie manažmentov .....	26
4.1	Manažment kvality, podpory vývoja a integrácie .....	27
4.2	Manažment plánovania a rozsahu .....	27
4.3	Manažment komunikácie a ľudských zdrojov .....	27
4.4	Manažment tvorby dokumentácie .....	28
5	Sumarizácie šprintov .....	29
5.1	Šprint 1.....	30
5.2	Šprint 2.....	37



5.3	Šprint 3.....	43
6	Globálna retrospektíva .....	45
6.1	Po zimnom semestri .....	45
7	Úvod .....	47
8	Ciele pre zimný semester .....	48
9	Šprint 1 - Piskor .....	48
9.1	Presunutie funkcionality z prototypu .....	48
9.2	Spustenie aplikácie pri štarte PC .....	49
9.2.1	Analýza .....	49
9.2.2	Návrh .....	49
9.2.3	Implementácia a testovanie .....	49
10	Šprint 2 - Myška.....	49
10.1	Automaticky vyber data providera podľa pripojených kamier .....	49
10.1.1	Analýza .....	49
10.1.2	Návrh .....	49
10.1.3	Implementácia.....	49
10.2	Taskbar notifikácia .....	50
10.2.1	Analýza .....	50
10.2.2	Návrh .....	51
10.2.3	Implementácia a testovanie .....	51
10.3	Popup notifikácia .....	51
10.3.1	Analýza .....	51
10.3.2	Návrh .....	51
10.3.3	Implementácia a testovanie .....	52
10.4	Podpora kamery Intel F200 .....	52
10.4.1	Analýza .....	52
10.4.2	Návrh .....	53
10.4.3	Implementácia a testovanie .....	53
10.5	Implementácia webcam data source nezávislou od runtime .....	53
10.5.1	Analýza .....	53
10.5.2	Návrh .....	53
10.5.3	Implementácia.....	54
11	Šprint 3 - Škrečok.....	55



11.1	Plánovanie notifikácií.....	55
11.1.1	Analýza .....	55
11.1.2	Návrh .....	55
11.1.3	Implementácia a testovanie .....	55
11.2	Logujeme používanie aplikácie .....	56
11.2.1	Úloha .....	56
11.2.2	Analýza .....	56
11.2.3	Návrh .....	56
11.2.4	Implementácia a testovanie .....	56
11.3	Spúšťanie a riadenie strážcov .....	57
11.3.1	Analýza .....	57
11.3.2	Návrh .....	57
11.3.3	Implementácia.....	57
12	Celkový pohľad na systém.....	59
12.1	Použité knižnice a frameworky.....	60
12.2	Dátový model.....	61
12.3	Moduly.....	62
12.3.1	Modul pre štatistiky .....	62
12.3.2	Modul notifikácií .....	63
12.3.3	Modul pre analýzu a vyhodnocovanie kvality sedenia .....	64
12.3.4	Modul pre získavanie dát z kamier.....	65
12.3.5	Modul pre kalibráciu .....	67



# 1 Úvod

Tento dokument obsahuje dokumentáciu k riadeniu projektu Spine Hero v rámci predmetu Tímový projekt na Fakulte informatiky a informačných technológií na Slovenskej technickej univerzite v Bratislave. Projekt Spine Hero je aplikácia, ktorá monitoruje zdravie používateľa pri práci s počítačom.

Táto dokumentácia opisuje fungovanie tímu z pohľadu manažmentu, procesy a metodiky používané v tíme. V dokumentácii sa ďalej diskutuje, ako sa nám darí jednotlivé metodiky dodržiavať a akým spôsobom ich zlepšujeme a vyvíjame.

Táto dokumentácia je určená predovšetkým pre členov tímu, aby nám pomohla ujasniť a zjednodušiť prácu a zvýšiť našu tímovú efektivitu.

**Webové sídlo projektu:** <http://labss2.fiit.stuba.sk/TeamProject/2015/team21is-si/>

## 2 Tím

### 2.1 Členovia tímu

#### 2.1.1 Jozef Gáborík

Zaujíma sa o grafický dizajn, UX, ale aj o architektúru softvéru. Práve tieto veci využíva aj v rámci tímu pri tvorbe aplikácie Spine Hero. Má na starosti grafické rozhranie aplikácie, tvorbu grafických materiálov a dizajn web stránky. Pri všetkom chce, aby veci spolu ladili a tvorili kompaktný celok.

S tvorbou webových stránok začal už na základnej škole a odvtedy pokračuje vo vzdelávaní sa v tejto sfére. Vďaka tomu nadobudol už skúsenosti v jazykoch Python, PHP, Ruby a iných. Má rád, keď vidí čistý kód, ktorý do seba zapadá a má jasnú štruktúru. Je fanúškom kvalitnej literatúry, vyzná sa najmä vo fantasy knihách.

#### 2.1.2 Matej Leško

Matej je známy svojou záľubou v technológiách od spoločnosti Microsoft, ktorá sa naplno rozvinula na súťaži Imagine Cup. Má dobré základy v platforme .NET a jeho obľúbeným programovacím jazykom je C#. Má snahu všetko robiť precízne a problémom sa venuje zaryto až dovtedy, pokým ich nepokorí. V projekte sa venoval najmä komunikácii s externými hardverovými zariadeniami a detekciou kvality držania tela z hĺbkovej mapy.

Je preňho typická dobrá nálada, ktorú prenáša aj na ostatných kolegov v tíme. Aktívne sa venuje športu, ktorý mu poskytuje oddych po práci na projekte.

#### 2.1.3 Jakub Mačina


V projekte sa najviac zaoberal rozpoznávaním obrazu pomocou strojového učenia. Jakub navrhol niekoľko prototypov riešenia použitím neurónových sietí. Vedomosti v tejto oblasti nadobudol najmä z kníh a online kurzov od Geoffrey Hinton a Andrew Ng. Jakub má taktiež skúsenosti s tvorbou webových aplikácií v Ruby on Rails a desktopových aplikácií v Java a C#. Ma rád aj dátovú analýzu, na ktorú používa programovacie jazyky R a Python.

Jakub je usilovný, vždy dobre naladený a je otvorený novým nápadom. Má takisto rád premýšľanie o smerovaní projektu po technickej aj po biznis stránke. Vo voľnom čase rád relaxuje, číta a spoznáva čokoľvek nové. Najradšej však športuje, a futbal, ktorý hrá už od detstva, ho naučil byť správnym tímovým hráčom.

#### 2.1.4 Jozef Staňo

Ak práve nepracuje na projekte, tak ho určite nájdete rozoberať, preinštalovávať alebo inak vylepšovať svoj počítač. Medzi jeho ďalšie aktivity patrí bicyklovanie, cestovanie a varenie. Jozef si vie vynikajúco zmanažovať svoj čas, takže do tímu priniesol presnosť. Taktiež má rád ak sa práca robí efektívne, uznáva "lean" princípy a dokáže ostatných členov tímu udržať vždy pri hlavnej podstate veci. Na problémy sa vie pozrieť aj z iného pohľadu a preto dokáže vyriešiť akúkoľvek výzvu, ktorá stojí pred ním.

Jozef sa v projekte zaoberal rozpoznávaním tváre používateľa z webkamery a následným spracovaním získaných črt na určenie správnosti držania tela. Má skúsenosti so spracovaním



obrazu pomocou knižnice OpenCV v programovacích jazykoch C# a Java. Jozef sa nebojí ani takých pojmov ako obfuskácia kódu, pretože sa zaujíma aj o bezpečnosť informačných systémov.

## 2.2 Roly členov tímu

### 2.2.1 Manažérske roly

Jozef Gáborík

- manažér dokumentácie, plánovania a rozsahu
- dizajnér

Matej Leško

- manažér kvality a podpory vývoja
- hlavný architekt

Jakub Mačina

- manažér dokumentácie, komunikácie a ľudských zdrojov
- vedúci tímu

Jozef Staňo

- manažér rizík, nákladov
- zástupca vedúceho tímu

### 2.2.2 Krátkodobé roly

**Scrum master**

Jakub Mačina

**Zapisovateľ stretnutí**

Jozef Gáborík, Jakub Mačina

**Overovateľ zápisov so stretnutia**

Jakub Mačina, Jozef Gáborík

**Webová stránka na TP**

Jakub Mačina



## 3 Používané metodiky

Zoznam metodík, ktorými sa študenti riadili v rámci tímu.



## 3.1 Manažment kvality, podpory vývoja a integrácie

Táto časť sa zameriava na proces vývoja aplikácie, ktorý pozostáva z týchto činností:

1. Vytvorenie novej vetvy.
2. Implementácia požadovanej funkcionality, opravy chyby alebo refaktorizácie.
3. Napísanie testov, resp. regresné testovanie.
4. Vytvorenie pull requestu.
5. Prehliadka kódu.
6. Integrácia kódu.

K týmto jednotlivým procesom boli vytvorené metodiky, ktoré sú rozpísané v nasledujúcich častiach v logickom poradí ako nasledujú.

### Roly a zodpovednosti

Rola	Zodpovednosť
Manažér kvality, podpory vývoja a integrácie	Správa a konfigurácia repositárov. Hlavný kontrolór kvality kódu. Hlavný integrátor.
Kontrolór kvality kódu	Kontrola dodržania code conventions. Kontrola správnej funkcionality kódu. Schválenie alebo zamietnutie pull requestu, resp. napísanie nedostatkov autorovi kódu alebo ich rýchla oprava.
Programátor (Autor kódu)	Vytvorenie novej vetvy. Napísanie zdrojového kódu. Pokrytie zdrojových kódov testami. Zapísanie zmien na vzdialený server. Vytvorenie pull requestu a pridanie členov na kontrolu kódu.

### 3.1.1 Metodika verziovania aplikácie

Táto metodika je určená pre všetkých členov tímu, ktorí sa podieľajú na implementácii aplikácie. Je zaradená v rámci manažmentu podpory vývoja. Úlohou tejto metodiky je zabezpečiť aby sa verziovanie aplikácie robilo jednotne, aby bola dodržaná štruktúra vetvenia a pomenovania jednotlivých vetiev.

#### 3.1.1.1 Pomenovanie vetiev

V aplikácii sú tieto hlavné vetvy:

- master – v tejto vetve je aktuálna stála verzia aplikácie,
- development – vývoj momentálnej verzie aplikácie.

Ostatné vetvy musia spĺňať nasledovnú konvenciu pomenovania:



[Typ]/[Identifikátor]-[Číslo úlohy]-[Krátky popis]

[Typ] – Predstavuje typ vetvy a môže nadobúdať hodnotu:

- *feature* – pridanie novej funkcionality
- *bug* – oprava chyby

[Identifikátor] – jedinečný identifikátor pre projekt – „SH“.

[Číslo úlohy] – také ako v JIRA.

[Krátky popis] – taký ako v JIRA.

### 3.1.1.2 Vytvorenie novej vetvy

Novú vetvu je možné vytvoriť viacerými spôsobmi. Preferované je však vytváranie vetvy v nástroji JIRA, ktorej postup je:

- V nástroji JIRA si otvoríme úlohu pre ktorú chceme vytvoriť novú vetvu.
- Klikneme na možnosť *CreateBranch*.
- Vyberieme vetvu z ktorej sa bude čerpať.
- Ubezpečíme sa, že názov vetvy spĺňa vyššie spomenuté podmienky
- Potvrdíme vytvorenie novej vetvy.

## 3.1.2 Metodika organizácie zdrojových kódov

Hlavným architektonickým štýlom aplikácie je MVVM (Model-View-ViewModel) od čoho sa odráža aj jej základná štruktúra. Súbory je potrebné kvôli prehľadnosti organizovať do priečinkov.

### 3.1.2.1 Hlavná štruktúra projektu:

- Model
- Views
- ViewModels
- ModulA
- ModulB
- ...
- ModulN
- Utils
- ClassDiagrams
- Resources
- Vendor

V rámci tejto štruktúry môže existovať hierarchia priečinkov na základe logickej alebo funkcionálnej zviazanosti daných zdrojových súborov, ktoré sa v nich nachádzajú. Hierarchia v rámci *Views* a *ViewModels* je rovnaká.

## Namespace

Štruktúra priečinkov odráža namespace jednotlivých súborov v nich. Napr. trieda *ShellViewModel* sa nachádza v priečinku *ViewModels*, teda namespace tejto triedy je *[BaseNamespace].ViewModels* pričom *[BaseNamespace]* je základný namespace, ktorý je rovnaký pre všetky súbory. Jeho hodnota je momentálne *SpineHero*.

## Súbory a typy

Súbory môžu obsahovať viacero tried / rozhraní alebo enumov. Treba však dodržiavať to aby jednotlivé časti súboru mali medzi sebou pevnú závislosť boli krátke a jednoduché, je vhodné aby boli tieto časti boli v rámci jedného súboru, prípadne vytváranie nových súborov je zbytočné a znížila by sa tým prehľadnosť. Napríklad:

```
using SpineHero.DataSources;

namespace SpineHero.PostureMonitoring.Watchers
{
    public interface IWatcher
    {
        ResultMessage AnalyzeImages(ImageWrapper images);
    }

    public interface IColorImageWatcher : IWatcher { }

    public interface IDepthImageWatcher : IWatcher { }
}
```

Používanie CaliburnMicro si vyžaduje určité konvencie v nazývaní tried pre *View* a *ViewModel* a ich namespace. Preto sa vyžaduje aby triedy pre *View* s názvom *ClassName* mali príponu *View* a nachádzali sa v namespace *[BaseNamespace].Views.[Category]* a triedy pre *ViewModel* mali príponu *ViewModel* a nachádzali sa v namespace *[BaseNamespace].ViewModels.[Category]*. *Category* predstavuje vlastné členenie v rámci *View/ViewModel*. Napríklad pre *ClassName* = "Settings" a *Category* = "MainMenuItems" prislúcha *View* s celým názvom:

*[BaseNamespace].Views.MainMenuItems.SettingsView*

a *ViewModel*:

*[BaseNamespace].ViewModels.MainMenuItems.SettingsViewModel.*

### 3.1.2.2 Štýl pomenovania

V nasledujúcej tabuľke sa nachádza ako sa majú pomenovávať jednotlivé elementy zdrojových kódov:



Typ	Pomenovanie	Poznámka
File and types	PascalCase	Vid'. vyššie.
Namespace	PascalCase	Vid'. vyššie.
Interfaces	IPascalCase	Všetky rozhrania začínajú s "I".
Methods, Properties, Events	PascalCase	Výstižne pomenovávať podľa toho čo vykonávajú.
Local variables	camelCase	Namiesto typu je možné použiť <i>var</i> v prípade, že z pravej strany je jasné o aký typ premennej sa jedná.
Parameters	camelCase	Použiť skrátenejší názov v prípade, že je jasné na čo sa daný parameter používa, resp. to vyplýva z typu parametru.
Private fields	camelCase	
Public fields	PascalCase	Vyhnuť sa ich používaniu. Namiesto nich používať Properties. Neplatí pre public static field.
Static readonly fields	ALL_UPPER	Konštanty vypočítavané za behu programu (Např. načítané z nastavení).
Constants	ALL_UPPER	
Enum members	PascalCase	

### Organizácia typov

Organizácia jednotlivých tried by mala byť nasledovná:

- Fields
- Constructors
- Destructors
- Delegates
- Events

- Enums
- Interfaces
- Properties
- Indexers
- Methods
- Structs
- Classes

Túto štruktúru je možné porušiť v prípade zhluknutia určitej funkcionality (napr. použitie návrhových vzorov Observer, Disposable, a iné) do regiónu:

```
#region IDisposable Support
private bool disposed = false;

protected virtual void Dispose(bool disposing)
{
    /* Some code not important for this example. */
    disposed = true;
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
#endregion IDisposable Support
```

### 3.1.2.3 Nástroje pre zlepšenie kvality písania zdrojových kódov

Pre zlepšenie kvality a podporu vývoja aplikácie používame dva nástroje. Prvým je *ReSharper* od JetBrains. Tento nástroj má veľké množstvo funkcionalít. Pričom z kvalitatívneho sú najdôležitejšie ukazovanie na potencionálne chyby, možnosť zjednodušenia napísanej funkcionality, chyby v pomenovaní. Druhým nástrojom je *CodeMaid*, ktorý umožňuje jednoduchý prehľad triedy a jej reorganizáciu a taktiež ku jednotlivým metódam prepočítava ich zložitosť.

## 3.1.3 Metodika písania unit testov

Písanie testov je veľmi dôležitá činnosť a pomocou nej vieme zabezpečiť v prípade úpravy kódu aby sa zachovala pôvodná funkcionality kódu, resp. aby sme dosiahli požadovaný výsledok. Na testovanie je použitý testovací framework NUnit. Je vytvorený samostatný projekt pre testy, pričom štruktúra organizácie testov sa má mapovať na organizáciu v hlavnom projekte.

## 3.1.3.1 Štruktúra testov

```

[TestFixture]
internal class TestFixtureExample : AssertionHelper
{
    [TestFixtureSetUp]
    public void Init()
    { /* Called once before all tests. */ }

    [TestFixtureTearDown]
    public void Cleanup()
    { /* Called once after all tests. */ }

    [SetUp]
    public void InitBeforeTest()
    { /* Called before each method. */ }

    [TearDown]
    public void CleanupAfterTest()
    { /* Called after each method. */ }

    [Test]
    public void TestExample()
    {
        var pi = Math.PI;
        Expect(pi, EqualTo(Math.PI));
    }

    [Test]
    public void TestThrow()
    {
        int d = 0;
        Assert.Throws<DivideByZeroException>(() => d = 42 / d);
    }

    [TestCase(2, 5, 10)]
    public void TestCaseExample(int a, int b, int sum)
    {
        /* Test with parameters. */
        Expect(a * b, EqualTo(sum));
    }
}

```

Opis použitých atribútov (anotácií) triedy a testovacích metód:

Atribút	Význam
TestFixture	Označuje triedu v ktorej sú testovacie metódy
TestFixtureSetUp	Inicializácia testovacej triedy. Vykoná sa raz pred všetkými testami v triede.



TestFixtureTearDown	Uvoľnenie zdrojov. Vykoná sa raz po skončení všetkých testov.
SetUp	Inicializácia pre testovaciu metódu. Vykoná sa pred každou testovacou metódou.
TearDown	Uvoľnenie zdrojov. Vykoná sa po každej testovacej metóde.
Test	Testovacia metóda bez parametrov.
TestCase	Testovacia metóda s parametrami.

### Spôsoby pre overovanie správnych výsledkov

Je možné použiť dve rôzne spôsoby pre overovanie výsledkov. Základným spôsobom je použitie statickej metódy *Assert.That*. Druhý a kratší spôsob je použitie metódy *Expect*. V tomto prípade musí testovacia trieda dediť od triedy *AssertionHelper*. Preferovaný je tento spôsob overovania výsledkov. Nepodporuje však zatiaľ všetku funkcionálnosť, preto v niektorých prípadoch bude stále treba použiť *Assert* (napr. *Assert.Throw*).

## 3.1.4 Metodika prehliadky kódu

Každý programátor má iné skúsenosti a iný štýl programovania. Prehliadky kódu zabezpečujú aby sa do vývojárskej vetvy dostal len kód, ktorý zodpovedá kvalite a funkcionálnosťou požiadavkám. Prehliadky kódu umožňujú ostatným používateľom sa oboznámiť s kódom časti aplikácie, ktorú naprogramoval niekto iný a nájsť prípadné chyby a nedostatky, ktoré je potom potrebné aj diskutovať s autorom kódu.

### 3.1.4.1 Vytvorenie pull requestu a pridanie členov na kontrolu kódu

Vytvorenie pull requestu zahŕňa tieto činnosti:

- Výber vetvy z ktorej chceme vytvoriť pull request v nástroji Bitbucket.
- Kliknutie na *Create pull request*.
- Výber vetvy do ktorej sa majú zmeny zapísať (vo väčšine prípadov development).
- Pomenovanie pull requestu je zhodné s pomenovaním vetvy.
- Pridanie členov na kontrolu kódu
- Potvrdenie vytvorenia pull requestu

Prehliadku kódu môže dobrovoľne spraviť akýkoľvek člen tímu. Autor kódu môže taktiež priradiť kohokoľvek na prehliadku kódu, pričom daná osoba nie je povinná vykonať túto prehliadku, okrem prípadu ak zmeny v kóde zasahujú do kódu, ktorého je daný kontrolór autorom. Hlavný kontrolór kvality by mal skontrolovať každý pull request.

### 3.1.4.2 Prehliadka kódu

Postup:

- Pochopenie úlohy, ktorá bola implementovaná
- Stiahnutie vetvy do lokálneho repozitára
- Spustenie testov
- Otestovanie funkčnosti aplikácie a vykonaných zmien



- Kontrola vykonaných zmien v aplikácii a ich pochopenie
- Kontrola testov
- Priebežné písanie zistených nedostatkov
- Zamietnutie / Schválenie pull requestu.

Princípy prehliadky kódu:

- Ak bolo v kóde vykonaných veľa zmien rozložiť si kontrolu na viac častí (200-400 riadkov zmien na jedno posedenie max.).
- Kontrole kódu venovať dostatočný čas.
- Snažiť sa pochopiť kód - každý má iný štýl programovania.
- Ak niečomu nerozumiete komunikujte s autorom kódu.
- Pri nesúhlase s implementáciou je potrebný aj kompromis.
- Všetky nedostatky okomentujte v Bitbuckete.
- Overte si, že nedostatky boli opravené.
- Pull request zamietnite ak je kód vo veľmi zlej kvalite a je potrebné ho úplne prerobiť alebo napísať odznovu.
- Pull request schváľte ak prípadné nedostatky boli opravené alebo vydiskutované.

### 3.1.4.3 Integrácia zmien

Ak pull request prešiel prehliadkou kódu integrátor môže kód spojiť s vetvou do ktorej sa pull request vytváral (vo väčšine prípadov development). V prípade, že nastali konflikty pri spájaní vetiev integrátor musí tieto konflikty vyriešiť. Pri tomto procese sa môže vyžadovať aj komunikácia s autormi kódov v ktorých nastali pri spájaní konflikty.

Integrátor môže danú vetvu z ktorej sa vytváral pull request zatvoriť v prípade, že práca na danej vetve je ukončená.

## 3.2 Manažment rizík

Zabezpečuje identifikáciu, evidovanie a vyhodnocovanie rizík spojených s projektom. Manažment rizík je kontinuálna činnosť jej cieľom je znižovať straty spôsobené nepriaznivými okolnosťami prípadne navrhnúť postup ktorým sa danej okolnosti vyhneme.

Ku každému identifikovanému riziku určujeme:

- čas v ktorom môže riziko prepuknúť
- spúšťač (udalosť ktorá zapríčiní prepuknutie rizika)
- dopad na projekt
- pravdepodobnosť s ktorou toto riziko nastane.

Po identifikovaní atribútov navrhujeme kroky ktoré znížia riziko prepuknutia ale aj riešenie ak riziko prepukne.

Identifikovali sme tieto riziká:





ID	1
Názov	Nepostačujúca presnosť detekcie pomocou hĺbkovej kamery
Spúšťač	
Čas	Testovanie s používateľom
Dopad	Zlyhanie celého projektu – jeho principiálna zmena
Pravdepodobnosť	Stredná
Opatrenia:	
Preventívne opatrenia	Začať testovať s používateľom v čo najkratšom čase
Riešenie po prepuknutí	Zmena koncepcie aplikácie
ID	2
Názov	Nedodanie hĺbkových kamier do začiatku testovania
Spúšťač	Neskoré objednanie
Čas	Začiatok projektu
Dopad	Odloženie testovania s používateľmi. Ohrozenie progresu celého projektu
Pravdepodobnosť	Stredná
Opatrenia:	
Preventívne opatrenia	-
Riešenie po prepuknutí	Objednanie kamier pomocou zrýchlenej dodávky
ID	3
Názov	Výpadok člena tímu
Spúšťač	Nepredvídateľná skutočnosť
Čas	Počas celého projektu
Dopad	Zdržanie oproti plánovanému harmonogramu
Pravdepodobnosť	Nízka
Opatrenia	
Preventívne opatrenia	Zdieľanie vedomostí. Každú časť aplikácie by malo poznať viacero členov tímu
Riešenie po prepuknutí	Vyššie nasadenie ostatných členov tímu. Úprava harmonogramu.
ID	4
Názov	Implementácia nevhodnej funkcionality
Spúšťač	Nesprávny odhad trhu a potrieb zákazníka
Čas	Počas celého projektu
Dopad	Zdržanie oproti plánovanému harmonogramu
Pravdepodobnosť	Stredná



Opatrenia	
Preventívne opatrenia	Často validovať pridávanú funkcionality. Komunikovať zo zákazníkom a zbierať spätnú väzbu.
Riešenie po prepuknutí	
ID	5
Názov	Neskoré začatie implementačných úloh
Spúšťač	Nedostatok času, odkladanie povinností
Čas	Počas celého projektu
Dopad	Nesplnená úloha v šprinte, zdržanie projektu
Pravdepodobnosť	Stredná
Opatrenia	
Preventívne opatrenia	Prípraviť analýzu riešenia úloh na začiatku šprintu. Zabezpečiť lepší odhad koľko času je naozaj potrebného na splnenie úlohy podľa definition of done.
Riešenie po prepuknutí	Prenos úlohy do ďalšieho šprintu.
ID	6
Názov	Nedostatočná komunikácia pri implementácii
Spúšťač	Nedodržanie metodiky
Čas	Počas celého projektu
Dopad	Pridanie zbytočnej práce
Pravdepodobnosť	Nízka
Opatrenia	
Preventívne opatrenia	Pravidelne komitovať riešenie, častejšie stand-up stretnutia
Riešenie po prepuknutí	Prehodnotenie naplánovania úloh, dohovorenie zúčastneným stranám

## 3.3 Manažment plánovania a rozsahu

Správne plánovanie je jednou z kľúčových vecí, na ktorých tkvie úspech projektu. Plán dáva tímu hranice a umožňuje lepšie sledovať progres a stíhanie vytýčených cieľov. Manažér plánovania a rozsahu má práve vytvárať takéto plány a dozerať na jeho plnenie.

V našom tíme sú do plánovania zapájaní aj ostatní členovia tímu. Plán nie je považovaný za dogmu, ktorá sa nesmie nikdy meniť. Je to skôr niečo čo určuje smer a rozsah, ale je možné ho flexibilne meniť podľa aktuálnych požiadaviek a okolností.

Pri plánovaní sa dbá na to, aby boli najskôr dokončené tie najdôležitejšie veci. Pri rozsahu sa určuje množstvo práce, ktoré sme ako tím schopní vykonať. Podobne sa dbá aj na stanovanie závislostí medzi úlohami. Snažíme sa totižto vyhnúť prípadu, keď je jeden člen tímu blokován iným členom. V tomto všetkom nám pomáhajú zvolené metodológie a nástroje.

### 3.3.1 Spôsob plánovania a riadenia rozsahu

V našom projekte využívame SCRUM. K tomu nám slúži nástroj JIRA, respektíve jeho rozšírenie JIRA Agile.

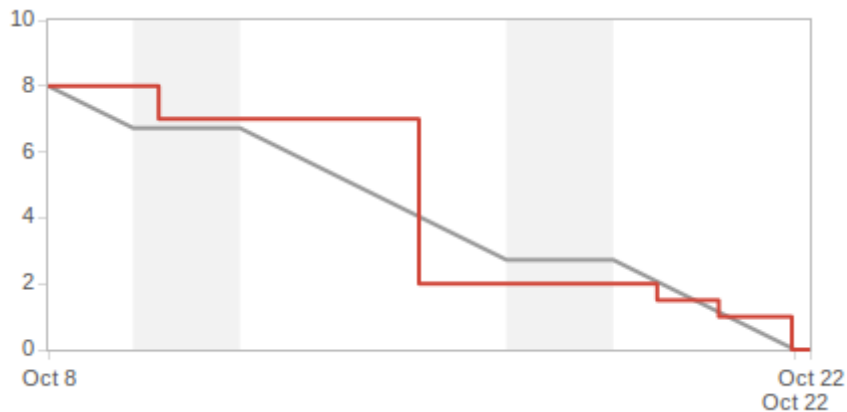
#### 3.3.1.1 SCRUM

V našom tímovom projekte využívame, rovnako ako aj ostatné tímy, agilnú metodológiu SCRUM. Využívame jeho upravenú verziu. Oficiálny spôsob použitia SCRUM-u totižto vyžaduje denné "stand-up" mítingy. To však nie je možné aplikovať v našej situácii ako študentov s rôznymi rozvrhmi a ďalšími povinnosťami. Z toho dôvodu sa stand-up míting koná raz týždeň na stretnutí tímu. Samotný šprint trvá dva týždne.

Plánovanie v SCRUM-e funguje na niekoľkých základných princípoch. Veci, ktoré je treba spraviť, sú zapísané v *product backlog*. Delia sa na *user story*, *task* a *bug*. *Product backlog* sa udržiava zoradený podľa dôležitosti. Na vrchných položkách sa začína pracovať najskôr. Všetko v *product backlog*-u je ohodnotené pomocou *story points*. Proces pridelovania týchto bodov je popísaný nižšie. Pre každý šprint tím spoločne naplánuje, ktoré položky z *product backlog* budú doň aktuálne zaradené.

Pri tvorení šprintu dbá manažér plánovania a rozsahu na to, aby bolo vybrané dostatočné a zároveň splniteľné množstvo *story points*. Rovnako dáva pozor, aby nebol nejaký člen tímu nadmerne zaťažovaný, keďže niektoré úlohy si zo svojej podstaty vyžadujú konkrétneho člena tímu. Vhodné množstvo *story points* vyberá na základe *velocity* tímu. *Velocity* definuje počet bodov *story points*, ktoré je schopný tím vyriešiť za jeden šprint. Určuje sa na základe výsledkov tímu z predchádzajúcich šprintov.

Počas šprintu sa sleduje progres na základe *Burndown chart*. Ten hovorí, koľko *story points* v danom momente v rámci šprintu ostáva vyriešiť.



Obrázok 1 Príklad Burndown chart

Položka z *product backlog* je ohodnotená ako dokončená, keď spĺňa *Definiton of Done*. Náš tím sa dohodol na tejto definícii: Kód pre danú úlohu musí to byť už vo vetve master. Prejde schvaľovacím procesom – overenie na viac ako 1 zariadení a sú k tomu napísané testy.

### 3.3.1.2 Planning poker

Na určenie *story points*, pre každú položku z *product backlog* sa využíva planning poker. Princíp je nasledovný. Najskôr sa prečíta a opíše práve hodnotená položka. Následne každý člen tímu vyloží zo svojho balíčka kariet kartu s číslom, na koľko story points by danú úlohu ohodnotil. Ak nenastane zhoda, členovia tímu opisujú prečo zvolili danú kartu. Svoje rozhodnutie zdôvodňujú najmä členovia s najväčším a najmenším ohodnotením.

Karty v balíčku sú ohodnotené číslom. Väčšinou sa jedná o hodnoty na základe Fibonacciho postupnosti. Náš tím používa karty s číslami 1, 2, 3, 5, 8, 13, 21, 40, 100, ?. Symbol ? znamená, že človek aktuálne nevie ohodnotiť danú položku. Indikuje to, že nebola dobre popísaná alebo vysvetlená. Ak je niektorá z položiek ohodnotená ako 8 a viac, tak je to pre nás signál, že je veľká a mala by sa rozbiť na menšie úlohy.

## 3.3.2 Plán

Vyvíjaná aplikácia má ambíciu byť vydaná pre verejnosť a následne sa predávať. Plán jej vývoja nie je zostavený z presných dátumov, ale skôr z fáz (respektíve míľnikov), ktoré chce postupne dosiahnuť. Každá fáza obsahuje zoznam minimum vlastností, ktoré musí aplikácia v danej fáze poskytovať, aby bolo možné označiť míľnik za dosiahnutý.

### 3.3.2.1 Fáza 1 - Privátna beta

**Cieľ:** Po dosiahnutí tohto míľnika sa aplikácia dostane do stavu privátnej bety. To znamená, že aplikácia bude poskytnutá vybraným ľuďom na testovanie.

#### Požadované vlastnosti:

- Prechod aplikácie na framework Caliburn.Micro. Minimálne nasledovné časti:
  - Modul notifikácií
  - GUI (grafické rozhranie)
  - Získavanie dát z webkamery a hĺbkových kamier.
- Zakúpenie hĺbkových kamier Intel F200 (aspoň 5 ks).



- Vytvoriť modul na komunikáciu s kamerami Intel F200.
- Logovanie do súboru.
- Nájdenie dobrovoľníkov na testovanie.
- Nainštalovanie aplikácie a poskytnutie potrebného hardvéru na testovanie pre testerov.

### 3.3.2.2 Fáza 2 - Otvorená beta

**Cieľ:** Aplikácia sa poskytne širokej verejnosti na testovanie.

**Požadované vlastnosti:**

- Získanie spätnej väzby z privátnej bety. Na jej základe následne odstrániť nedostatky.
- Umožnenie posielania spätnej väzby na server.
- Posielanie logov na server.
- Umožnenie stiahnutia aplikácie z webovej stránky aplikácie.
- Monitorovanie spôsobu používania aplikácie používateľom.
- Rozšírenie modulu notifikácií o monitorovanie používateľovej aktivity na počítači.
- Vytvorený inštalačný balíček.
- Vytvorená úvodná obrazovka.
- Vytvorená obrazovka a potrebné materiály na vysvetlenie správneho sedenia.

### 3.3.2.3 Fáza 3 - Vydanie verzie 1.0

**Cieľ:** Aplikácia je oficiálne vydaná. Je možné ju stiahnuť a zakúpiť cez web.

**Požadované vlastnosti:**

- Založenie firmy.
- Web umožňuje zakúpenie aplikácie.
- Vytvorená oficiálna dokumentácia.
- Aplikácia obsahuje tvorbu tréningov.
- Vytvorený modul štatistiky.

### 3.3.2.4 Nasledujúce fázy

- Vytvorenie cross-platform verzie.
- Prepojenie s health systémami.
- Monitovavanie používateľa cez odmeny.
- Umožnenie zdieľania štatistík a odznakov na sociálnych sieťach.

## 3.4 Manažment komunikácie a ľudských zdrojov

Komunikácia je veľmi dôležitou súčasťou práce v tíme. Často sa však berie ako samozrejmosť, no nie každý člen tímu vždy chápe dané informácie rovnako. Zlepšenie komunikácie vedie k zvýšeniu efektivity každého člena tímu.

Manažér komunikácie riadi komunikáciu v tíme tak, aby boli informačné potreby všetkých zúčastnených uspokojené. Jeho úlohou je tiež iniciovať, viesť a smerovať komunikáciu.

Úlohou manažéra ľudských zdrojov v našom tíme je motivovať ostatných členov tímu, zabezpečiť súdržnosť a efektívne vykonávanie úloh v tíme. Navyše, jeho úlohou je starať sa o členov tímu aby sa mohli sústrediť a vykonávať svoju prácu.

### 3.4.1 Metodika komunikácia v tíme

Cieľom tejto metodiky je opísať a zdefinovať efektívne prístupy pre komunikáciu v tíme. Metodika opisuje aké komunikačné kanály použiť pri jednotlivých typoch komunikácie, zodpovednosti členov tímu a definuje postupy pre vytvorenie a zdieľanie informácií o projekte.

#### 3.4.1.1 Komunikačné nástroje

V tíme používame tieto nástroje formálne komunikačné nástroje: email, wiki, Jira, Bitbucket a zápisnice. Nástroje neformálnej komunikácie sú: Slack, telefón, Google Hangouts a osobné stretnutia.

Telefonická komunikácia je vhodná najmä v súrnych prípadoch, ak je na odozvu obmedzené nízke množstvo času ale aj pri akýchkoľvek typoch problémov, ktoré je možné vyriešiť s jedným zodpovedným členom tímu. Osobné stretnutie zainteresovaných členov majú vždy prioritu a sú vhodné vo všetkých prípadoch.

Použitie komunikačných nástrojov pri vybraných situáciách (nástroje sú zoradené podľa priority):

- Formálna komunikácia s celým tímom - mail
- Manažovanie tímu - mail, Slack
- Reportovanie činnosti tímu - mail, Jira
- Oznámenie nového obsahu na wiki - mail, Slack
- Komunikácia k úlohám - Jira, Slack (#tp16)
- Pomoc pri probléme - Jira, Slack, Google Hangouts
- Súrny problém - telefón, mail
- Žiadosť o revíziu kódu - Bitbucket
- Nastavenie prostredia, nástrojov - wiki
- Zápis so stretnutí - zápisnica

#### 3.4.1.1.1 Mailová komunikácia

Na mailovú komunikáciu v tíme používame Google skupinu [imaginecup2015@googlegroups.com](mailto:imaginecup2015@googlegroups.com) a na túto mailovú schránku máme aj nastavený aj alias [info@spinehero.com](mailto:info@spinehero.com). Mailovú komunikáciu sa často používa aj na komunikáciu s používateľmi, novinármi a inými externými osobami.



### Vhodné pre typ komunikácie

- Formálna komunikácia s celým tímom
- Reportovanie činnosti tímu
- Oznámenie pridania nového obsahu na wiki

### Zodpovednosti

V prípade, ak prišiel email ktorý je smerovaný na náš tím, je úlohou manažéra komunikácie odpovedať v mene tímu na tento mail do 48 hodín. Jeho povinnosťou je dať do kópie skupinový email, aby mali všetci aktuálne informácie o dianí v tíme. Pri komunikácií s externými osobami musí odosielateľ pripojiť na koniec správy podpis v tvare:

Meno

Priezvisko

Spine

Hero

[www.spinehero.com](http://www.spinehero.com)

### Obsah správy

V predmete správy by mal byť jasne uvedený obsah správy, z ktorého je jasný cieľ tejto správy. Obsahom správy by mala byť práve jedna vec, problém, oznam. Pri urgentných správach pre všetkých členov tímu zvážiť aj telefonický kontakt na členov tímu bez mobilného internetového pripojenia.

#### 3.4.1.1.2 Chat komunikácia - Slack

Na rýchlu a neformálnu komunikáciu používame nástroj určený pre tímovú komunikáciu [Slack](#).

### Vhodné pre typ komunikácie

upresnenie podrobností ohľadom úloh

komunikácia s jednotlivými členmi tímu

off-topic komunikácia

### Používané kanály

*app-bitbucket* - kanál, ktorý je zintegrovaný s repozitárom kódu a notifikuje o akýchkoľvek zmenách v repozitári. Slúži ako prehľad o aktivitách v rámci repozitári kódu. Vhodný aj na akékoľvek otázky a doplnenia k aktivitám v repozitári.

*business* - komunikácia ohľadom tímového biznisu, PR, smerovania tímu, konkurencie a súťaží.

*random* - off-topic komunikácia.

*general* - všeobecný chat, vhodný ak žiadny iný kanál nevyhovuje

*design* - kanál, ktorý sa zameriava na diskutovanie o dizajnu aplikácie a prezentácie tímu na verejnosti.

*tp16* - všetko komunikácia, ktorá súvisí s predmetom tímový projekt. Slúži najmä na komunikáciu k úlohám v rámci šprintu.



*top-secret* - komunikácia členov tímu bez vedúceho. Tento kanál používať len v nutných prípadoch.

### **Odporúčania**

Pre rýchlejšiu tímovú komunikáciu je odporúčané nainštalovať si mobilnú aplikáciu Slack alebo si nechať neustále otvorenú webovú aplikáciu ako kartu v prehliadači.

Je vhodné používať anotácie @meno, ak je nutné, aby odpovedal najmä daný člen tímu. Prípadne použiť anotáciu @channel na vyjadrenie názoru všetkých členov tímu.

#### **3.4.1.1.3 Bitbucket repozitár**

Vhodný pre žiadosť o revíziu kódu.

*Postup vytvorenia žiadosti o revíziu kódu*

1. Vytvoriť pull request
2. Do poľa reviewers pridáme daných členov tímu

#### **3.4.1.1.4 Jira**

Služi najmä na aktualizáciu stavu úloh, aby nenastala situácia, že danú úlohu riešia viacerí bez toho aby o tom vedeli. Taktiež slúži aj na komunikáciu k úlohám v rámci šprintu a backlogu.

*Komunikácia k úlohám*

1. otvoriť príslušnú úlohu
2. pridať komentár k úlohe aj s označením zodpovedného človeka na odpovedanie
3. v prípade, že je potrebná akcia iného člena tímu, je nutné vytvoriť novú pod-úlohu a prideliť ju zodpovednej osobe



## 3.5 Manažment tvorby dokumentácie

Manažér tvorby dokumentácie je zodpovedný za jednotnosť dokumentácie a zápisov so stretnutí.

### 3.5.1 Metodiky pre tvorbu dokumentácie

Nižšie sú popísané metodiky pre tvorbu jednotlivých častí dokumentácie. Pre každú časť bola vytvorená šablóna dokumentu.

#### Zápisnice

Zápisnice sa vytvárajú priebežne v rámci každého stretnutia . Vždy je jeden človek zodpovedný za zapisovanie zápisnice. Každú zápisnicu musí aspoň jeden ďalší člen tímu overiť.

Štruktúra hlavičky zápisnice je nasledovná:

- Dátum vytvorenie zápisnice vo formát DD.MM.YYYY
- Miestnosť konania stretnutia
- Vedúci stretnutia (zápis osôb vo formáte Meno Priezvisko)
- Zapisovateľ stretnutia
- Prítomní členovia

Ďalej sa v niekoľkých úrovniach odrážok zapisuje priebeh stretnutia. Ak vzniknú počas riešenia nové úlohy pre tím, vyčlenia sa samostatne.

#### Zápis retrospektívy

Štruktúra retrospektívy je nasledovná:

- Trvanie šprintu vo formáte DD.MM.YYYY - DD.MM.YYYY
- Zapisovateľ stretnutia
- Zoznam user stories v tabuľke s názvom, počet story points, stav user story a prípadnými poznámkami
- Časť s čím sme spokojný
- Časť s návrhmi čo zlepšiť

#### Export úloh z JIRA

Postup exportu úloh z JIRA issue tracker:

- V záložke Agile vyberieme položku Reports
- Vyberieme Time Tracking Report
- Vo formulári sub-task inclusion vyberieme Including all sub-tasks a klikneme na next
- Vyberieme excel view



## 4 Aplikácie manažmentov

Nasledujúca časť opisuje aplikáciu manažmentov, ktorých metodiky sme opísali vyššie v dokumente. Zameriavame sa najmä na hodnotenie jednotlivých metodík nakoľko fungovali v rámci nášho projektu a nášho tímu.

Rovnako sú opísané aj naše zmeny v metodikách a príčiny, ktoré nás k nim viedli.

## 4.1 Manažment kvality, podpory vývoja a integrácie

Na projekte pracuje viacero ľudí a preto je veľmi dôležité aby každý člen tímu dodržiaval určité pravidlá, konvencie aby sa zaručila, že kvalita vyprodukovaného zdrojového kódu bola na čo najvyššej úrovni a najmä konzistentná.

Vzhľadom na to, že na projekte už pracujeme dlhší čas, tak sme mali zaužívané praktiky, ktoré nám v tom čase aj postačovali. Avšak vzhľadom na to, že tieto kvalitatívne veci neboli pevne dané a robilo sa skôr metódou „On to robí takto, spravím to aj ja tak.“ nastávali prípady, kedy sa pomenovania a štruktúra organizácie nezhodovala s predstavami manažéra kvality. Po dokončení jednotlivých metodík a ich preštudovaní a pochopení členmi tímu budú prípadné ďalšie excesy neprípustné.

## 4.2 Manažment plánovania a rozsahu

Plánovanie je kritické pre každý projekt. Takisto aj rozsah úloh pre daný šprint je niečo, čo častokrát tvoríme s ružovými okuliarmi na očiach. V našom tíme sa prejavili obidva problémy.

Pri plánovaní sme od začiatku pracovali SCRUM spôsobom a teda sme ihneď začali využívať planning poker. Okrem zlého odhadu koľko story points si môžeme vziať do jedného šprintu, čo je typický problém každého nového tímu, sme taktiež urobili chybu v ohodnocovaní user stories. Pracovali sme totižto aj s user stories s veľkým počtom bodov. V prvom šprinte sa dokonca vyskytla user story s 13 story points. Takéto hodnotenie svedčí o nedostatočnom rozbití user story na menšie celky. Ak toto čítaš, vyslúžil si si odmenu v podobe piva. Daj nám o tom nejakým spôsobom vedieť a my ťa pozývame na pivo. Aj to malo za následok nestihnutie všetkých úloh pre daný šprint. Tím sa preto rozhodol takéto veľké úlohy vždy rozbiť na menšie.

Dôležité je taktiež určenie správneho množstva úloh pre daný šprint. Úlohy nemajú priradeného konkrétneho riešiteľa, avšak vyberajú sa vždy tak, aby nebol nejaký člen tímu preťažený, keďže niektoré úlohy sú zo svojej podstaty určené práve pre konkrétneho člena tímu.

## 4.3 Manažment komunikácie a ľudských zdrojov

Tímové nástroje sme mali už pred začiatkom projektu vybrané a zaužívané. Na neformálnu komunikáciu sa nám najviac osvedčil Slack, ktorý sme aj najčastejšie používali, a na formálnu komunikáciu sa ukázala vhodná mailová komunikácia.

Počas prvých šprintov sme v retrospektíve identifikovali komunikáciu v tíme ako nedostatočnú, pretože sme málo komunikovali o riešení projektu. Pri vyskytnutí problému sme taktiež málo hľadali pomoc u kolegov tímu. Ďalším problémom bolo aj to, že sme pravidelne neaktualizovali stav úloh v Jire. Pretože komunikácia je pre správne fungovanie tímu dôležitá, v ďalších šprintoch sme jej venovali väčší dôraz.

Rozhodli sme sa pre častejšie stretnutia mimo oficiálnych naplánovaných časov, pretože osobné stretnutia pokladáme za najefektívnejšiu formu spoločnej práce. Taktiež sme sa dohodli na používaní anotácie @ na označenie konkrétnych ľudí pre vyjadrenie sa k téme na slacku. Navrhli sme, aby každý člen tímu používal mobilnú verziu Slacku a bol tak dostupný čo



najčastejšie online. Repozitár s kódom sme tiež prepojili so Slackom aby mal každý člen tímu aktuálny prehľad o nových zmenách.

## 4.4 Manažment tvorby dokumentácie

Tvorba dokumentácia bola pre tím náročná. Chybou bolo, že písanie dokumentácie nebolo od začiatku zaraďované ako časť práce na user story. Takisto „Definition of Done“ v sebe neobsahuje klauzulu, ktorá by vyžadovala napísanie dokumentácie k naimplementovanej story.

Keďže ale písanie je jednou z podmienok pri riešení nášho projektu, táto práca sa presúvala na neskôr a následne nahromadila na posledný možný termín.

## 5 Sumarizácie šprintov

Stretnutia pred začatím šprintov:

### Zápis so stretnutia #1

<i>Dátum:</i>	24.09.2015	
<i>Miestnosť:</i>	softvérové	štúdio
<i>Vedúci stretnutia:</i>	Jakub	Šimko
<i>Zapisovateľ:</i>	Jakub	Mačina
<i>Prítomní:</i>	Jakub	Šimko
	Jozef	Gáborík
	Matej	Leško
	Jozef	Staňo
	Jakub Mačina	

#### Priebeh stretnutia

- Diskusia o SCRUM-e
  - Dohoda na definition of done
  - Pýtame sa 5x prečo keď hľadáme benefit v user story
- Diskusia o smerovaní projektu v ZS
  - Aké sú naše priorit

#### Nové úlohy

- Objednať kamery
- Vybrať systém na management projektu, ktorý bude podporovať definované typy user stories, úlohy, fidelity, odhad v story pointoch, vytváranie šprintov
- Naplniť backlog

### Zápis so stretnutia #2

<i>Dátum:</i>	01.10.2015	
<i>Miestnosť:</i>	softvérové	štúdio
<i>Vedúci stretnutia:</i>	Jakub	Šimko
<i>Zapisovateľ:</i>	Jakub	Mačina
<i>Prítomní:</i>	Jakub	Šimko
	Jozef	Gáborík
	Matej	Leško
	Jozef	Staňo
	Jakub Mačina	

#### Priebeh stretnutia

- Dohoda o používaní systému JIRA
  - Používame školskú verziu
- Dohodli sme na tom, čo pre nás znamená 1 story point – je to pre nás referenčné úloha pridanie spustenie aplikácie pri štarte PC
- Ohodnocovali sme user stories pokrovými kartami
- Prioritizovali sme backlog



## Nové úlohy

- Dokončenie ohodnocovanie úloh v backlogu a zoradenia podľa priorít

## 5.1 Šprint 1

### Zápis so stretnutia #3

<i>Dátum:</i>	8.10.2015	
<i>Miestnosť:</i>	softvérové	štúdio
<i>Vedúci stretnutia:</i>	Jakub	Šimko
<i>Zapisovateľ:</i>	Jakub	Mačina
<i>Prítomní:</i>	Jakub	Šimko
	Jozef	Gáborík
	Matej	Leško
	Jozef	Staňo
	Jakub Mačina	

#### Priebeh stretnutia

- Plánovanie šprintu a ohodnocovanie úloh

#### Nové úlohy



## Zápis so stretnutia #4

<b>Dátum:</b>	15.10.2015	
<b>Miestnosť:</b>	softvérové	štúdio
<b>Vedúci stretnutia:</b>	Jakub	Šimko
<b>Zapisovateľ:</b>	Jakub	Mačina
<b>Prítomní:</b>	Jakub	Šimko
	Jozef	Gáborík
	Matej	Leško
	Jozef	Staňo
	Jakub Mačina	

### Priebeh stretnutia

- StartupAwards – dohodli sme sa, že treba sa sústrediť na vývoj a validáciu nápadu a preto účasť na súťaži posúvame to o rok.
- Intel – chceme získať hromadnej zľavu pri nákupe väčšieho počtu kamier.
- Dokumentácia k retrospektívne – nebudeme tam písať zoznam úloh, ktoré sme vyriešili. To sa robí v sprint review.
- JIRA – dohodli sme sa, že budeme písať ako subtasky tie bugy, ktoré sa objavia počas práca na konkrétnej user story
- Treba zlepšiť aktualizáciu úloh na kanban tabuly.

### Nové úlohy

- Záložka na stránke o tíme
- Bezpečnosť deploy skriptu pre tp stránku

## Export úloh

FIIT STU	Key	Status	Priority	Summary
Issue Type	SPINEHERO-3	Open	Minor	Používateľské účty
Story	SPINEHERO-3	Open	Minor	Používateľské účty
Task	SPINEHERO-4	Resolved	Major	Rozhovor s Denníkom N
-> Sub-task	SPINEHERO-5	Resolved	Major	-> Odpísať reportérovi na mail
-> Sub-task	SPINEHERO-6	Resolved	Major	-> Stretnutie s reportérom
Task	SPINEHERO-7	Open	Major	Vytvorenie formulára s otázkami na firmy ohľadom sedenia
Epic	SPINEHERO-8	Open	Minor	Štatistiky - kvalita sedenia
Task	SPINEHERO-9	Open	Major	Zistenie podrobností o forme firmy s.r.o. vs startup vec
Improvement	SPINEHERO-10	Open	Major	Prepojenie z db



## SUMARIZÁCIE ŠPRINTOV

Task	SPINEHERO-11	Open	Major	Prihlásenie do Startup Awards
-> Sub-task	SPINEHERO-27	Open	Minor	-> Vybrať kategóriu súťaže
Task	SPINEHERO-12	In Progress	Major	Objednať Intel Realsense kamery
Story	SPINEHERO-13	In Progress	Major	Spustenie pri štarte pc
Task	SPINEHERO-14	Open	Major	Komunikácia s Intelom - hromadná zľava na kamery
New Feature	SPINEHERO-15	Open	Minor	Redizajn aplikácie
Task	SPINEHERO-16	Open	Major	Testovanie app - osobne s používateľov
Task	SPINEHERO-17	Open	Major	Testovanie app - používateľa samostatne
Story	SPINEHERO-18	Open	Minor	Gamifikácia
Improvement	SPINEHERO-19	Open	Major	Notifikácie
Task	SPINEHERO-20	Open	Major	Práva - Pribalenie knižnice z Intel Realsense k aplikácii
Task	SPINEHERO-21	Resolved	Critical	Stránka na tp fiit
-> Sub-task	SPINEHERO-51	Resolved	Major	-> Inštalácia servera
-> Sub-task	SPINEHERO-52	Resolved	Major	-> Vytvorenie web stránky
-> Sub-task	SPINEHERO-53	Closed	Major	-> Automatic deployment
-> Sub-task	SPINEHERO-54	Resolved	Major	-> Vytvorenie šablón na dokumenty
New Feature	SPINEHERO-22	Open	Minor	Branding
Story	SPINEHERO-23	Open	Minor	Štatistiky - Čas strávený sedením pri pc
Bug	SPINEHERO-24	Open	Major	Aplikácia padne pri výbere data providera
Story	SPINEHERO-25	Open	Minor	Štatistiky - Punch card notifikácií
Story	SPINEHERO-26	Open	Minor	Štatistiky - progres, maximá
Improvement	SPINEHERO-28	Open	Minor	Kliknutím na ikonku v tray sa spustí aplikácia
Epic	SPINEHERO-29	Open	Blocker	Získanie väčšej presnosti detekcie a zisťovanie či používateľ sedí pri pc.





Improvement	SPINEHERO-30	Open	Blocker	Naportovanie funkcionality z prototypu
-> Sub-task	SPINEHERO-36	Open	Major	-> Použitie aplikačného frameworku.
-> Sub-task	SPINEHERO-37	Open	Major	-> Spúšťanie a riadenie strážcov
-> Sub-task	SPINEHERO-48	In Progress	Major	-> Pridanie strážcov.
-> Sub-task	SPINEHERO-49	In Progress	Major	-> Pridanie okien

## Zápis so stretnutia #5

*Dátum:* 22.10.2015

*Miestnosť:* softvérové

*Vedúci stretnutia:* Jakub

*Zapisovateľ:* Jozef

*Prítomní:* Jakub

Jozef

Matej

Jozef

Jakub Mačina

štúdio

Mačina

Gáborík

Šimko

Gáborík

Leško

Staňo

### Priebeh stretnutia

- Rozsiahli šprint review a retrospektíva šprintu.
- Dohodli sme si Definition of Done
- Dohodli sme sa, že chceme robiť cross-platform aplikáciu, ale betu si dávame iba na Windows.
- Dohodli sme sa na vlastnostiach aplikácie v open beta:
  - Detekcia sedenia
  - Notifikácie (bez monitorovania používateľovej aktivity)
  - Dashboard
  - Settingom - kalibráciu, vyber datasource, ktoré notifikácie a po akom čase
  - Obrazovka, ktorá vysvetlí ako správne sedieť.
  - Logovanie chýb a čo sa deje v aplikácii
  - Poslanie spätnej väzby
- Rozprávali sme sa o komunikácii s Intelom ohľadom zľavy na kamery. Dohodli sme sa, že sa necháme nasmerovať na Sales oddelenie. Povieme, že máme záujem o desiatky kamier. Spýtať sa, kedy bude finálna verzia kamery a SDK.

### Nové úlohy

- Novú úlohy sú zaznamenané v šprint review.



## Export úloh

FIIT STU Issue Type	Key	Status	Priority	Summary
Task	SPINEHERO-50	Open	Major	Nainstalovať aplikáciu Ivanovi Stéfunkovi
Story	SPINEHERO-47	Open	Major	Urobiť prieskum ako by ľudia odosieli feedback o nesprávnom sedení
Story	SPINEHERO-46	Open	Major	Možnosť zareportovať nesprávne meranie
New Feature	SPINEHERO-45	Open	Major	Silent report of crash
Epic	SPINEHERO-44	Open	Major	proste píšeme testy
Story	SPINEHERO-43	Open	Major	Odosielanie logov
Story	SPINEHERO-42	Open	Major	Používateľ pošle všeobecný feedback
Story	SPINEHERO-41	Open	Major	Logujeme používanie aplikácie
Epic	SPINEHERO-40	Open	Major	Synchronizácia údajov na server aby používateľ mohol sa prihlásiť a vidieť a započítavať svoje údaje na rôznych zariadeniach
New Feature	SPINEHERO-39	Open	Critical	Nofikovanie používateľa o padnutí app
Bug	SPINEHERO-38	Open	Major	Aplikácia niekedy padá pri vizualizácia spracovávaných dát
Story	SPINEHERO-35	Open	Major	Nastavenie vlastného avatara
Story	SPINEHERO-34	Open	Minor	Kontrola aktualizácií
Story	SPINEHERO-33	Open	Major	Detekcia držania tela používateľa
Story	SPINEHERO-32	Open	Minor	Určovanie či používateľ sedí za počítačom



## SUMARIZÁCIE ŠPRINTOV

Epic	SPINEHERO-31	Open	Critical	Monitorovanie kvality sedenia pomocou kamery
Improvement	SPINEHERO-30	Open	Blocker	Naportovanie funkcionality z prototypu
-> Sub-task	SPINEHERO-58	In Progress	Major	-> Portnutie Head Watchera
-> Sub-task	SPINEHERO-57	In Progress	Major	-> Intel F200 data source
-> Sub-task	SPINEHERO-56	Resolved	Major	-> Pridanie okien - Calibration
-> Sub-task	SPINEHERO-49	Resolved	Major	-> Pridanie okien - Dashboard
-> Sub-task	SPINEHERO-48	Resolved	Major	-> Pridanie strážcov - Depth Watcher
-> Sub-task	SPINEHERO-37	Open	Major	-> Spúšťanie a riadenie strážcov
-> Sub-task	SPINEHERO-36	Resolved	Major	-> Použitie aplikačného frameworku.
Epic	SPINEHERO-29	Open	Blocker	Získanie väčšej presnosti detekcie a zisťovanie či používateľ sedí pri pc.
Improvement	SPINEHERO-28	Open	Minor	Kliknutím na ikonku v tray sa spustí aplikácia
Story	SPINEHERO-26	Open	Minor	Štatistiky - progres, maximá
Story	SPINEHERO-25	Open	Minor	Štatistiky - Punch card notifikácií
Bug	SPINEHERO-24	Open	Major	Aplikácia padne pri výbere data providera
Story	SPINEHERO-23	Open	Minor	Štatistiky - Čas strávený sedením pri pc
New Feature	SPINEHERO-22	Open	Minor	Branding
Task	SPINEHERO-21	Resolved	Critical	Stránka na tp fiit
-> Sub-task	SPINEHERO-55	Resolved	Major	-> Bezpečnosť deploy skriptu
-> Sub-task	SPINEHERO-54	Resolved	Major	-> Vytvorenie templatov na dokumenty
-> Sub-task	SPINEHERO-53	Closed	Major	-> Automatic deployment

# Retrospektíva šprintu #1 - Piskor

Trvanie šprintu: 22.10.2015

Zapisovateľ: Jozef Gáborík

## S čím sme spokojný

- Jozef Staňo: Komunikácia. Spoločné stretnutie s programovaním.
- Jozef Gáborík: -
- Matej: Spoločné stretnutie s programovaním.
- Jakub Šimko: Zvládli sme drill v Scrum. Nebola vec, ktorú by sme vyslovene nevedeli zvládnuť.
- Jakub Mačina: Spoločné stretnutie s programovaním.

## Návrhy čo zlepšiť

- Mali sme veľkú úlohu (13b), ktorú sme z väčšej časti spravili, ale nebola úplne Done. Návrh riešenia: Rozbiť úlohu na menšie časti.
- Nemáme dohodnutú Definiton of Done. Návrh riešenia: Musí to byť na mastri. Prejde to schvaľovacím procesom – overenie na viac ako 1 zariadení a sú k tomu napísané testy.
- Zlá komunikácia. Málo sme diskutovali to, čo sme robili. Návrh riešenia: Viac komunikovať na Slacku. Používať @ ak chcem niečo od konkrétneho človeka. Častejšie robiť SCRUM míting. Prepojiť Slack s bitbucketom.
- Nebola určená časová následnosť – niektoré veci nás blokovali. Nebolo veľa vecí v mastri a teda sme nevedeli, na čo sa pýtať. Návrh riešenia: Pouvažovať, čo nás blokuje.
- Stojace pull requesty. Nemali sme Code Review úlohy. Návrh: Urobiť vetvu v mastri „development“. Zapísať to do JIRA. Navrhnuť dopredu, kto spraví code review.
- Nie je určený človek, ktorý je zodpovedný za architektúru. Riešenie: Matej je architekt.
- Nerobili sme testovanie – nemali sme na to úlohy. Riešenie: Robiť testovanie.
- Nepresúvali sme úlohy v JIRA. Riešenie: Presúvať úlohy v JIRA.
- Robili sme až 2. týždeň. Návrh: Čím skôr robiť analýzu. Nahádzať teda do JIRA všetky tasky.

## Sprint review:

Jozef Gáborík – nestihol šprint. Dôvody sú popísané pri úlohe „Notifikácie“.

Jozef Staňo: Objednal kamery. Ešte neprišli. Robil automatický štart aplikácie. Problém bol s integráciou, keďže časť na Settings nebola dokončená. (Matej už spravil Settings). Jozef stál, lebo nemal čo robiť. Jozefovi nejde nová kamera od Intelu – lebo nepodporuje Win 7. Jozef sa snažil naportovať staršieho headwatchera – funguje to, ale nie je to zintegrované.

Nový Intel Realsense – zistili sme, že nefunguje s Windows7.

Matej – veľká úloha nie je dokončená. Sú spravené okná, nie sú dokončený watchere. Ostatné veci sú z Matejov úlohy spravené.

Jakub – robil webstránku (skript na automatic deployment). Komunikoval s Intelom. Robil DataSource na novú Intel webkameru. Užho to funguje, nie je to zintegrované.

## User stories:



Stránka na tp fiit – Done. Dohodnuté na 5 story point. Nerátal, že to bude treba inštalovať server. Hodnotil by to viacerými bodmi.

Rozhovor s Denníkom N – Done.

Naportovanie funkcionality z prototypu – Nedokončené. Problémy s technológiou spôsobili stratu času (nefungovala kamera na Windows 7). Súhlasil s hodnotou 13b. Navrhuje rozdeliť na menšie úlohy.

Príhlásenie do Startup Awards – Closed. Rozhodli sme sa neísť do súťaže.

Notifikácie – Nedokončené. Bol som blokový, musel som čakať na veci od Mateja. Učiť sa Caliburn Micro. Robiť testovanie. Neskoro som na tom začal robiť.

Spustenie pri štarte počítača – Done.

Objednať Intel Realsense kamery – Done.

Komunikácia s Intelom - hromadná zľava na kamery – Done.

## 5.2 Šprint 2

### Zápis so stretnutia #6

*Dátum:* 29.10.2015

*Miestnosť:* softvérové

*Vedúci stretnutia:* Jakub

*Zapisovateľ:* Jozef

*Prítomní:* Jakub

Jozef

Matej

Jozef

Jakub Mačina

štúdio

Mačina

Gáborík

Šimko

Gáborík

Leško

Staňo

#### Priebeh stretnutia

- Spravený stand-up, kde sme si povedali kto na čom pracoval, čo ho blokovalo a čo sa chystá spraviť.
  - Matej – Vytvoril modul pre štatistiky.
  - Jozef G. – Vytvoril notifikáciu v notifikačnej oblasti.
  - Jozef S. – Vytváral testy pre HeadWatcher. Dataprovider na kameru.
  - Jakub – Vytvoril Intel F200 datasource.
- Diskutovali sme možnosť použiť CI (continuous integration). Predbežne sme rozmýšľali nad využitím Team City.
- Usporiadali a ohodnotili sme úlohy v product backlog.

#### Nové úlohy

- Prezrieť možnosti na CI.
- Z začať písať dokumentáciu.
- Prieskum povinností na Tímový projekt. Prieskum ako spisovať metodiky.

## Export úloh

FIIT STU				
Issue Type	Key	Status	Priority	Summary
Bug	SPINEHERO-89	Open	Minor	Vynútenie vytvorenia kalibrácie pred spustením monitorovania
Bug	SPINEHERO-88	Open	Major	Veľká spotreba RAM pri kalibrácii
Task	SPINEHERO-84	Open	Major	Zlepsit tricka
Story	SPINEHERO-80	Open	Major	Logovanie ak padne aplikacia
Story	SPINEHERO-79	Open	Major	Logovanie nasich vnutornych eventov aplikacie
Story	SPINEHERO-78	Open	Major	Vyber logovacieho frameworku
Improvement	SPINEHERO-77	Open	Major	Nastavenie continuous integration
Improvement	SPINEHERO-72	Resolved	Major	Modul pre statistiky
Improvement	SPINEHERO-71	Open	Major	Pokrytie testami spracovania hlbkoych a rgb dat
Improvement	SPINEHERO-70	Open	Major	Upozornenie na odpojenie kamery pocas behu aplikacie
Improvement	SPINEHERO-69	Resolved	Major	Implementacia webcam data source nezavislou od runtime
-> Sub-task	SPINEHERO-85	Resolved	Major	-> Code review
-> Sub-task	SPINEHERO-76	Closed	Major	-> Vytvorenie testov
-> Sub-task	SPINEHERO-75	Resolved	Major	-> Implementacia data source
-> Sub-task	SPINEHERO-74	Closed	Major	-> Implementacia listu kamier
-> Sub-task	SPINEHERO-73	Resolved	Major	-> Najdenie vhodnej libky
Story	SPINEHERO-68	Open	Minor	Obrazovka ako sediet spravne
Improvement	SPINEHERO-67	Open	Major	Testovanie a pokrytie testami depth watchera
Story	SPINEHERO-66	Open	Major	Stmavenie obrazovky



Story	SPINEHERO-65	Resolved	Major	Popup notifikacia
Story	SPINEHERO-64	Resolved	Major	Taskbar notifikacia
Story	SPINEHERO-63	In Progress	Major	Planovanie notifikacii
Improvement	SPINEHERO-61	Resolved	Major	Testovanie a pokrytie testami head watchera
-> Sub-task	SPINEHERO-87	Resolved	Major	-> Review kodu
-> Sub-task	SPINEHERO-86	Resolved	Major	-> Nevyhladavat hlavu pri kazdom spracovani obrazka
-> Sub-task	SPINEHERO-83	Closed	Major	-> Hladanie hlavy na mensom useku
-> Sub-task	SPINEHERO-82	Closed	Major	-> Najdenie inej vhodnejšie kniznice
-> Sub-task	SPINEHERO-81	Resolved	Major	-> Pridanie testov
Story	SPINEHERO-60	Resolved	Major	Automaticky vyber data providera podla pripojenych kamier
Improvement	SPINEHERO-59	In Progress	Major	Spúšťanie a riadenie strážcov
Task	SPINEHERO-50	Open	Major	Nainstalovat aplikaciju Ivanovi Stefunkovi
Story	SPINEHERO-47	Open	Major	Urobiť prieskum ako by ľudia odosieli feedback o nesprávnom sedení
Story	SPINEHERO-46	Open	Major	Možnosť zareportovať nesprávne meranie
New Feature	SPINEHERO-45	Open	Major	Silent report of crash

## Zápis so stretnutia #7

Dátum: 05.11.2015

Miestnosť: softvérové

Vedúci stretnutia: Jakub

Zapisovateľ: Jozef

Prítomní: Jakub

Jozef

Matej

Jozef

Jakub Mačina

štúdio

Mačina

Gáborík

Šimko

Gáborík

Leško

Staňo

## Priebeh stretnutia

- Jakub prezentoval čo všetko má byť obsahom dokumentáciu.
- Dokumentácia k inžinierskemu dielu – dohodli sme sa:
  - Pri šprintoch budeme opisovať User Story – analýza, návrh, dokumentácia. Nebudeme opisovať všetky úlohy.
  - Nakoniec sa popíšu aj moduly systému – analýza, návrh, dokumentácia.
- Dokumentácia k riadeniu projektu:
  - Rozdelili sme si roly v projekte.
    - Jakub Mačina – Manažér dokumentácie, Manažér komunikácie a ľudských vzťahov, Vedúci tímu
    - Jozef Staňo – Manažér rizík, Manažér nákladov, Zástupca vedúceho tímu
    - Jozef Gáborík – Manažér dokumentácie, Manažér plánovania a rozsahu, Dizajnér
    - Matej Leško – Manažér kvality, Hlavný architekt
- Dohodli sme sa, že raz za semester vymeníme scrum mastra.
- Naplánovanie ďalšieho šprintu.

## Nové úlohy

- Návrh: Písať odpracované časy na úlohách do JIRA.

## Export úloh

FIIT STU				
Issue Type	Key	Status	Priority	Summary
Bug	SPINEHERO-89	Open	Minor	Vynútenie vytvorenia kalibrácie pred spustením monitorovania
Bug	SPINEHERO-88	Open	Major	Veľká spotreba RAM pri kalibrácii
Task	SPINEHERO-84	Open	Major	Zlepsit tricka
Story	SPINEHERO-80	Open	Major	Logovanie ak padne aplikacia
Story	SPINEHERO-79	Open	Major	Logovanie nasich vnutornych eventov aplikacie
Story	SPINEHERO-78	Open	Major	Vyber logovacieho frameworku
Improvement	SPINEHERO-77	Open	Major	Nastavenie continuous integration
Improvement	SPINEHERO-72	Resolved	Major	Modul pre statistiky
Improvement	SPINEHERO-71	Open	Major	Pokrytie testami spracovania hlbkoych a rgb dat



Improvement	SPINEHERO-70	Open	Major	Upozornenie na odpojenie kamery počas behu aplikacie
Improvement	SPINEHERO-69	Resolved	Major	Implementacia webcam data source nezavislou od runtime
-> Sub-task	SPINEHERO-85	Resolved	Major	-> Code review
-> Sub-task	SPINEHERO-76	Closed	Major	-> Vytvorenie testov
-> Sub-task	SPINEHERO-75	Resolved	Major	-> Implementacia data source
-> Sub-task	SPINEHERO-74	Closed	Major	-> Implementacia listu kamier
-> Sub-task	SPINEHERO-73	Resolved	Major	-> Najdenie vhodnej libky
Story	SPINEHERO-68	Open	Minor	Obrazovka ako sediet spravne
Improvement	SPINEHERO-67	Open	Major	Testovanie a pokrytie testami depth watchera
Story	SPINEHERO-66	Open	Major	Stmavenie obrazovky
Story	SPINEHERO-65	Resolved	Major	Popup notifikacia
Story	SPINEHERO-64	Resolved	Major	Taskbar notifikacia
Story	SPINEHERO-63	In Progress	Major	Planovanie notifikacii
Improvement	SPINEHERO-61	Resolved	Major	Testovanie a pokrytie testami head watchera
-> Sub-task	SPINEHERO-87	Resolved	Major	-> Review kodu
-> Sub-task	SPINEHERO-86	Resolved	Major	-> Nevyhladavat hlavu pri kazdom spracovani obrazka
-> Sub-task	SPINEHERO-83	Closed	Major	-> Hladanie hlavy na mensom useku
-> Sub-task	SPINEHERO-82	Closed	Major	-> Najdenie inej vhodnej knihnice
-> Sub-task	SPINEHERO-81	Resolved	Major	-> Pridanie testov
Story	SPINEHERO-60	Resolved	Major	Automaticky vyber data providera podla pripojenych kamier
Improvement	SPINEHERO-59	In Progress	Major	Spustanie a riadenie strážcov
Task	SPINEHERO-50	Open	Major	Nainstalovat aplikaciju Ivanovi Stefunkovi
Story	SPINEHERO-47	Open	Major	Urobiť prieskum ako by ľudia

				odosielali feedback o nesprávnom sedení
Story	SPINEHERO-46	Open	Major	Možnosť zareportovať nesprávne meranie
New Feature	SPINEHERO-45	Open	Major	Silent report of crash

## Retrospektíva šprintu #2 - Myška

Trvanie šprintu: 05.11.2015

Zapisovateľ: Jozef Gáborík

### S čím sme spokojný

- Jozef Staňo: Spravili sme viac „issues“
- Matej Leško: Stretli sme sa 2x za šprint, nemali sme veľké issues
- Jakub Mačina: Spravili sme viac „issues“. Bol označený ako reviewer v pull-request.
- Jozef Gáborík: Viacej sme sa venovali sprintu, snažili sme sa lepšie si rozložiť čas.

### Návrhy čo zlepšiť

- Nesprávna úloha naplánovaná do šprintu.
- Nesprávne rozložený čas na riešenie úloh. Všetko sme chceli spraviť na poslednú chvíľu.
- Píšeme málo testov. Sme v tom neskúsení.
- Lepšie plánovať veci, ktoré nás blokujú.

### Sprint review

**Jozef Gáborík:** Stihol naprogramovať notifikácie. Nestihol som dokončiť všetky úlohy. Zlý manažment času.

**Matej Leško:** Vytvoril štatistický modul, automatické spúšťanie „datasource“. Nepokryl testami automatický výber dataprovidera. Dôvodom bola neskúsenosť pri písaní testov.

**Jakub Mačina:** Urobil testovanie kamery Intel F200. Študoval dokumentáciu a ako vytvoriť jej obsah.

**Jozef Staňo:** Stihol všetky úlohy. Programovanie watchera urobil na poslednú chvíľu a tým blokoval Mateja.

### User stories

Automaticky vyber data providera podľa pripojených kamier – Done.

Testovanie a pokrytie testami head watchera – Done.

Testovanie Intel f200 datasource – Done.

Taskbar notifikácia – Done.

Popup notifikácia – Done.

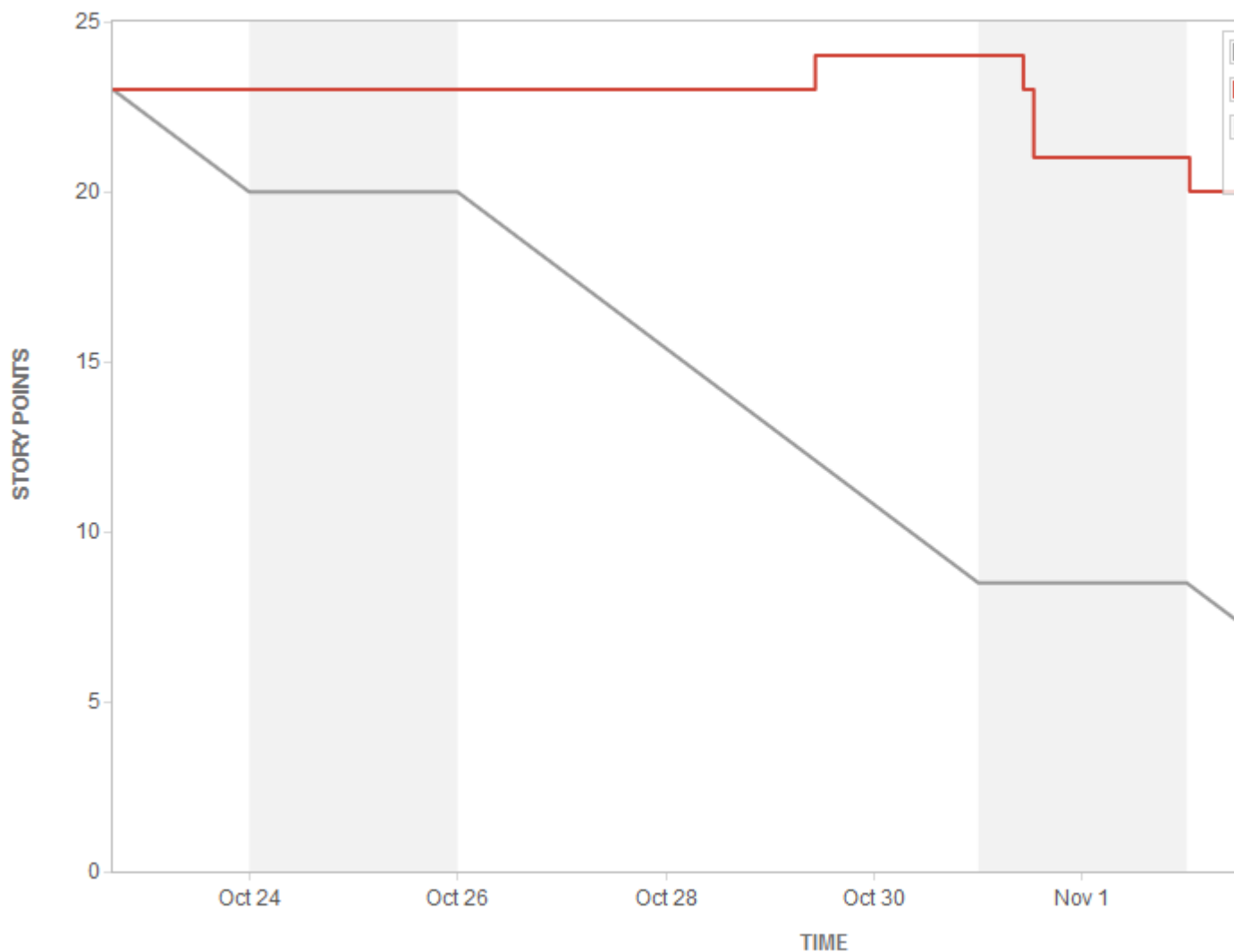
Implementácia webcam data source nezávislou od runtime – Done.

Modul pre statistiky – Done.



Spúšťanie a riadenie strážcov – Not completed. Chýbalo pokrytie kódu testami.

Planovanie notifikácii – Not completed. Nestihnuté naprogramovanie zobrazenia notifikácii a testy.



## 5.3 Šprint 3

### Zápis so stretnutia #8

*Dátum:* 12.11.2015

*Miestnosť:* softvérové

*Vedúci stretnutia:* Jakub

*Zapisovateľ:* Jozef

*Prítomní:* Jakub

Jozef

Matej

Jozef

Jakub Mačina

štúdio

Mačina

Gáborík

Šimko

Gáborík

Leško

Staňo



### **Priebeh stretnutia**

- Týždenný stand-up míting:
- Matej: Dokončil spúšťania strážcov. Plánuje písať testy na depth watchera + refaktor + refaktor vecí nad ním.
- Jakub: Výber logovacieho frameworku. Vybral NLog a PostSharp (na logovanie cez aspekty). Vytvoril na to vlastnú dokumentáciu a ukážku použitia. Plánuje pokryť aplikáciu týmto spôsobom logovania. Problém: Nevie čo všetko má logovať.
- Jozef Staňo: Riešil memory leak v aplikácii. Riešil to cez vlastné volanie Garbage Collector-u. Je to takmer dokončené. Ide robiť: vynútenie kalibrácie pred spustením monitorovania.
- Jozef Gáborík: Riešil a ďalej ide pracovať na notifikáciách.

### **Nové úlohy**

- Spísať dokumentáciu podľa dohodnutej formy.



## 6 Globálna retrospektíva

S každým ďalším šprintom sa zlepšujeme v oblasti odhadovania úloh, plánovania šprintu a plnenia úloh. Sme si však vedomí, že stále musíme pracovať na komunikácii v rámci tímu.

### 6.1 Po zimnom semestri



Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

# Dokumentácia k inžinierskemu dielu



## 7 Úvod

Naším cieľom je vytvoriť aplikáciu, ktorá monitoruje zdravie používateľa pri práci s počítačom. Monitorovanie používateľa prebieha len pomocou kamery a deteguje sa kvalita držania tela používateľa a frekvencia prestávok. V prípade nesprávneho držania tela alebo príliš dlhého nepretržitého sedenia aplikácia upozorní používateľa. Cieľom aplikácie je zlepšiť štýl sedenia a návyky používateľa pri práci s počítačom a tým aj zlepšiť jeho celkové zdravie.

Na aplikácii sme pracovali už v rámci bakalárskeho štúdia a podaním prihlášky na vlastnú tému v rámci predmetu Tímový projekt sme preukázali záujem ďalej rozvíjať túto myšlienku.

Dokument opisuje vývoj projektu v rámci predmetu Tímový projekt.

## 8 Ciele pre zimný semester

Naším cieľom v zimnom semestri je implementovať hlavnú funkčnosť aplikácie, ktorú následne chceme otestovať a zvalidovať s prvotnými používateľmi. Preto plánujeme testovanie použiteľnosti a aj testovanie s niekoľkými používateľmi, ktorí budú po stanovenú dobu aktívne používať aplikáciu.

Medzi hlavnú funkčnosť aplikácie patrí:

- vyhodnocovanie kvality držania tela
- notifikácie
- obrazovka s aktuálnou kvalitou sedenia
- obrazovka vysvetľujúca ako správne sedieť
- obrazovka s nastaveniami
- podpora farebných kamier
- podpora hĺbkových kamier Intel F200 a Intel Sens3D

Na validáciu základnej funkčnosti aplikácie a analýzu jej používania plánujeme navyše implementovať možnosť nahlásenia chyby, odosielanie spätnej väzby a anonymné logovanie používania aplikácie.

V rámci práce počas zimného semestra je naším cieľom poskytnúť na stiahnutie aplikáciu širšej verejnosti, a to najmä desiatkam ľudí ktorý prejavili záujem o aplikáciu na našej webstránke. Cieľom je získanie spätnej väzby ohľadom presnosti detekcie a rušivosti notifikácií. Od tejto fázy očakávame, že nám podarí pochopiť očakávania a požiadavky našich budúcich zákazníkov. Tejto fáze prikladáme veľkú prioritu pretože bude určovať ďalšie smerovanie vývoja aplikácie.

## 9 Šprint 1 - Piskor

### 9.1 Presunutie funkčnosti z prototypu

Prototyp, ktorý sme vyvinuli v rámci bakalárskej práce a súťaže Imagine Cup sme sa rozhodli zahodiť. Dôvodom zahodenia prototypu bola zmena časti architektúry, presnejšie riadenia behu aplikácie a jej jednotlivých modulov. Nový projekt je taktiež napísaný v jazyku C# spolu s GUI knižnicou WPF (Windows Presentation Foundation) pod frameworkom .NET verzie 4.6. Nasledovná funkčnosť bola presunutá do nového projektu z prototypu:

- Views
- *HeadWatcher* a *DepthWatcher*.
- *DataSources*

Základným architektonickým vzorom ostal MVVM (Model-View-ViewModel). Na obohatenie funkčnosti MVVM sa použil framework Caliburn.Micro, ktorý poskytuje možnosť použitia publish/subscribe systému *IEventAggregator*, ktorý sa použije na komunikáciu jednotlivých modulov aplikácie. Caliburn.Micro taktiež obsahuje aj vlastný IOC (Inversion of control) kontajner. Pomocou tohto frameworku sa zjednoduší aj práca s Views a ViewModelmi.



## 9.2 Spustenie aplikácie pri štarte PC

### 9.2.1 Analýza

Implementácia prinesie používateľovi jednoducho pridať a odobrať aplikáciu zo zoznamu spúšťaných aplikácií pri štarte operačného systému. Pri analýze možností ako túto funkcionality implementovať sme identifikovali dve možnosti. Prvou bolo pridať odkaz na aplikáciu do priečinka Startup. Druhou možnosťou je pridať do registrov cestu k aplikácií.

### 9.2.2 Návrh

Rozhodli sme sa pre implementáciu pomocou pridania hodnoty do registra. Hodnota bude vždy aktualizovaná podľa stavu registrov. Manipulácia s hodnotou a zisťovanie aktuálneho stavu v registri je podstatne jednoduchšia a efektívnejšia ako pridávanie a mazanie odkazu na aplikáciu.

### 9.2.3 Implementácia a testovanie

Implementáciu zabezpečuje statická trieda *AutoStartup*. V tejto triede sa nachádza property *AutoStart*. Pri prístupe k hodnote sa zistí či existuje príslušná hodnota v registroch. Pri nastavení property *AutoStart* sa hodnota v príslušnom registri pridá alebo vymaže. Na túto property bol na mapované zaškrtnuté políčko v časti nastavenia.

## 10 Šprint 2 - Myška

### 10.1 Automaticky vyber data providera podľa pripojených kamier

#### 10.1.1 Analýza

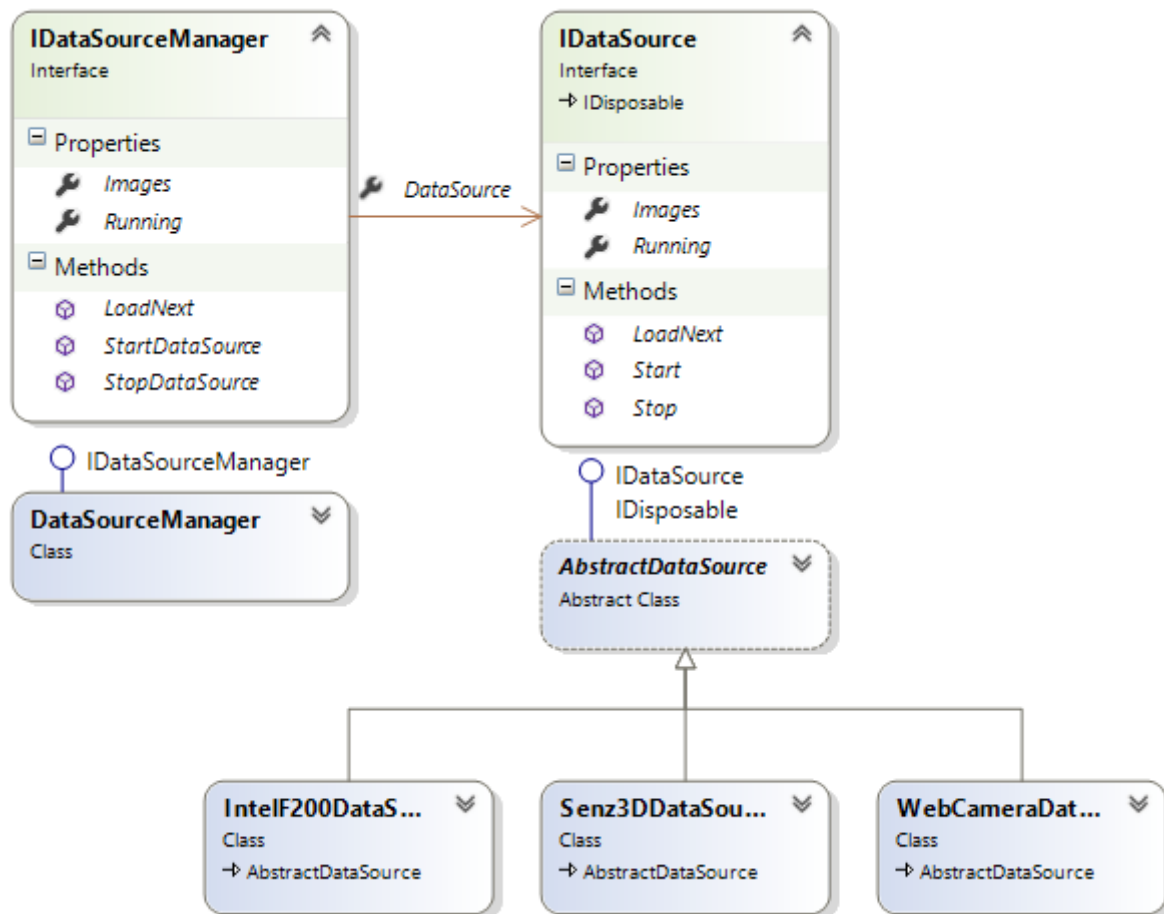
Táto funkcionality slúži na to aby sme používateľa odbremenili od činnosti výberu kamery, ktorá je použitá ako zdroj dát. Používateľ si bude vyberať len to či chce použiť hĺbkovú kameru ak je nejaká pripojená alebo nie.

#### 10.1.2 Návrh

V aplikácii sa pracuje s *DataSource* na viacerých miestach. Namiesto priameho prístupu k *DataSource* sa bude pristupovať ku triede, ktorá bude mať na starosti jeho správu a riadenie. Na základe toho ako si používateľ nastaví či chce používať hĺbkovú kameru alebo nie sa bude spúšťať *DataSource*. V prípade, že nechce použiť hĺbkovú kameru, tak sa automaticky spustí *DataSource* pre webovú kameru. V druhom prípade sa pokúsi spustiť *DataSource* pre hĺbkové kamery. Pokračuje sa pokým sa nenájde zariadenie, ktoré je možné spustiť. Ak nie je možné spustiť žiadny *DataSource* pre hĺbkové kamery tak sa spustí *DataSource* pre webovú kameru.

#### 10.1.3 Implementácia

Na implementáciu manažéra pre *DataSource* bol použitý návrhový vzor *Bridge*.



Obrázok 2 Diagram tried pre automatický výber data providera

K jednotlivým metódam rozhrania *IDataSourceManager* sa pristupuje z viacerých vláken, preto aby sa zabezpečila thread-safety, metódy triedy *DataSourceManager* používajú spoločný lock.

V nastaveniach bolo pridané zaškrávané políčko, ktoré určuje či sa má používať hĺbková kamera ak je k dispozícii.

## 10.2 Taskbar notifikácia

### 10.2.1 Analýza

Pri analýze sa dbalo najmä možnosti vytvorenia ikony v notifikačnej oblasti v operačnom systéme Windows. Vylúčili sme možnosť vytvoriť si vlastné riešenie, keďže existujú už vytvorené knižnice, ktoré poskytujú tvorbu ikoniek v notifikačnej oblasti.

Aplikácia používa na tvorbu GUI knižnicu WPF (Windows Presentation Foundation). Avšak ona neposkytuje automaticky takýto Control (prvok GUI). Inou možnosťou je využiť staršiu knižnicu na tvorbu GUI od Microsoftu - Windows Forms. Tá poskytuje takýto prvok, avšak znamenalo

by to pridanie ďalšej závislosti do aplikácie. Existuje aj open source knižnica WPF NotifyIcon<sup>1</sup>, ktorá je vytvorená v rámci WPF a poskytuje možnosť ovládania ikony cez WPF. Ďalšie alternatívy využívali volanie funkcií mimo jazyka C#.

Bola vybraná knižnica WPF NotifyIcon, keďže poskytuje vhodné prepojenie s WPF.

## 10.2.2 Návrh

Cieľom ikony je vizuálne sprostredkovať používateľovi informáciu, v akom stave sa nachádza, teda nakoľko správne sedí. Ikona má niekoľko stavov. Ak nie je spustené monitorovanie, alebo je vypnutá notifikácia pomocou ikony v notifikačnej oblasti, ikona má ako obrázok logo aplikácie a prislúchajúci text.

Ak je spustené monitorovanie a je povolená notifikácia pomocou ikony v notifikačnej oblasti, ikona mení svoj obrázok podľa aktuálnej pozície sedenia. Ak je správna, je zelená. Akonáhle sa sedenie zhorší, zmení sa na oranžovú a po istej hranici nesprávneho sedenia je červená. Tak môže mať používateľ okamžite informáciu nakoľko sedí správne.

## 10.2.3 Implementácia a testovanie

Na implementáciu bol použitý vzor MVVM (Model, View, View-model), ktorý je bežný v rámci WPF aplikácii.

- View - GUI prvok ikony je umiestnený v rámci ShellView spolu s ďalšími GUI prvkami pre celú aplikáciu.
- View-model - NotificationAreaIconViewModel obsahuje potrebné *properties*, na ktoré je "nabindované" View. Jedná sa najmä o obrázok pre ikonu a tooltip text.
- Model - NotificationAreaNotification obsahuje zdrojové *properties*, ktoré využíva view-model. Obsahuje logiku výberu ikony a textu podľa aktuálnej polohy používateľa. Zachytáva a spracováva udalosť (event) evaluácie používateľovho sedenia.

Testovanie bolo prevedené pomocou unit testov a manuálne. V rámci testovania sa dbalo na to, či po vytvorení novej evaluácie sa správne zmení ikona a jej tooltip.

## 10.3 Popup notifikácia

### 10.3.1 Analýza

Ďalší spôsob upozorňovania v rámci aplikácie je popup notifikácia. Cieľom je, aby dala jasne najavo, že používateľ sedí nesprávne, pritom má však vyzerať čo najpríjemnejšie. Pri analýze boli skúmané notifikácie v rámci operačných systémov Windows a Ubuntu, za vzor však bola vybraná notifikácia od antivírusu Nod32 z dielne firmy Eset.

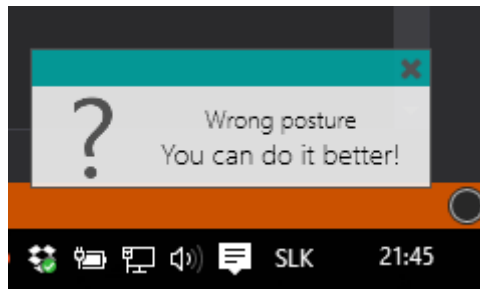
### 10.3.2 Návrh

Notifikácia sa zobrazuje ako okno v pravom dolnom rohu obrazovky. Je z časti priesvitné, aby sa dalo rozoznať čo sa nachádza pod ním. Zobrazuje aktuálnu polohu používateľa a text s upozornením. Je možné ju vypnúť tlačidlom (krížik v pravom hornom rohu). Po istom čase sa

---

<sup>1</sup> <https://bitbucket.org/hardcodet/notifyicon-wpf>

notifikácia vypne aj sama. Pred vypnutím sa najskôr na okno aplikuje “fade-out” efekt, čiže okno sa postupne zvyšuje priehľadnosť až sa stane úplne priehľadným.



Obrázok 3 Pop-up notifikácia

### 10.3.3 Implementácia a testovanie

Pri implementácii bol opäť využitý vzor MVVM (ako pri taskbar notifikácii vyššie). Popup notifikácia sa nezobrazuje sama. Jej zobrazenie má na starosti modul na plánovanie notifikácií. To znamená, že sama o sebe nezachytáva udalosť novej evaluácie. Spracováva iba udalosť zmeny polohy. Vtedy mení obrázok znázorňujúci používateľovu polohu. Na to sa zameralo aj unit testovanie. Ukážka časti testu:

```
[Test]
public void CurrentRepresentationOfPostureIsChanging()
{
    Expect (popupNotification.CurrentPostureImagePath,
    EqualTo (ResourceHelper.GetResourcePath (@"Resources\Images\StickBoy\Unknown.png")));

    eventAggregator.PublishOnCurrentThread (new Evaluation (0, Posture.Wrong));

    Expect (popupNotification.CurrentPostureImagePath,
    EqualTo (ResourceHelper.GetResourcePath (@"Resources\Images\StickBoy\Wrong.png")));
}
```

## 10.4 Podpora kamery Intel F200

### 10.4.1 Analýza

Úlohou bolo pridať podporu pre kameru [Intel RealSense F200](#), aby bolo možné aplikáciu používať aj s touto kamerou. Na vývoj aplikácií pre túto kameru je potrebné Intel Realsense SDK. Po preštudovaní dokumentácie sme zistili, že SDK bolo značne pozmenené oproti verzii pre podporu kamery Intel Senz3D, ktorá je už implementovaná.

Hĺbkové snímky poskytované kamerou sú v dvoch rôznych formátoch, v surovom formáte v rozsahu ako bezrozmerné číslo unsigned short integer (0-65535) alebo už spracované údaje v milimetroch.

## 10.4.2 Návrh

Navrhli sme zaradiť získavanie dát z tejto kamery do už existujúcej hierarchie tried v rámci projektu. Budeme získavať jednotlivo farebné a hĺbkové snímky z kamery. Použijeme hĺbkové dáta poskytované kamerou, ktoré sú v milimetroch, pretože už máme existujúce metódy na spracovanie takýchto dát.

## 10.4.3 Implementácia a testovanie

Implementovali sme triedu *IntelF200DataSource*, ktorá dedí od abstraktnej triedy *AbstractDataSource*. Implementovali sme potrebné metódy pre zapnutie (*Start*), vypnutie (*Stop*), načítanie aktuálneho snímku z kamery (*LoadNext*) a metódu pre deštrukciu triedy a jej zdrojov. Získavané snímky po spracovaní a zkonvertované do štandardného formátu *OpenCVSharp.Mat*, ktorý ukladáme do inštancie triedy *ImageWrapper* prístupnej na analyzovanie týchto dát. Následne je nutné uvoľniť získané snímky z pamäte volaním metódy *Dispose*, pretože sa jedná o nemanážovanú pamäť.

Súčasťou tejto úlohy bolo napísanie unit testov pre existujúcu statickú metódu *ImageUtils.TrimDepth*, ktorá sa používa pri orezávaní dát z hĺbkového obrázka. Metóda vracia normalizovanú hĺbku len ak je v určenom rozsahu, inak vracia 0. Danú metódu sme otestovali na viacerých štandardných a krajných testovacích vstupoch.

## 10.5 Implementácia webcam data source nezávislou od runtime

### 10.5.1 Analýza

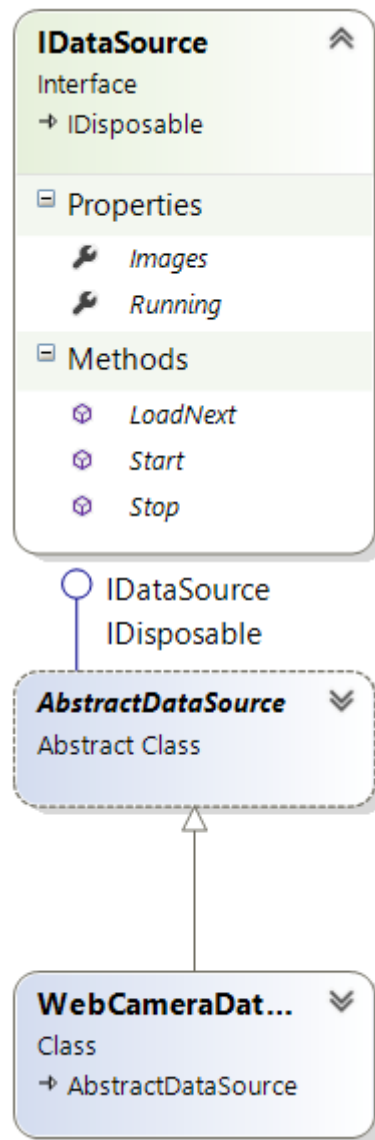
V doterajšej implementácii využívame intel SDK Runtime Distributable na získavanie obrazu z web kamery. Ten však je príliš veľký (558MB) a je nutné ho dodatočne inštalovať. Cieľom je implementovať získavanie údajov za pomoci menších a bežne dostupných knižníc.

Pre použitie sme identifikovali viacero knižníc: *DirectShowLib-2005*, *OpenCvSharp*, *EasyWebCam*.

Na spracovanie obrazu v aplikácii používame *OpenCvSharp*. Preto sme sa ju rozhodli použiť a nepridávať ďalšiu do aplikácie.

### 10.5.2 Návrh

Funkcionalitu budeme implementovať v triede *WebCameraDataSource*.



Obrázok 4 WebCameraDataSource

### 10.5.3 Implementácia

Pre správne fungovanie bolo potrebné implementovať metódy *Start()*, *Stop()* a *LoadNext()*.

Pri implementácii metódy *LoadNext()* sme narazili na problém. V metóde je od verzie 3.0.0 tejto knižnice na v nej nachádza chyba ktorá spôsobuje nefunkčnosť priameho čítania frame z kamery ktorú implementuje metóda *QueryFrame()*.

Získať obrázok z kamery bolo možné za použitie natívnej metódy.

```
NativeMethods.videoio_VideoCapture_operatorRightShift_Mat(capture.CvPtr,
colorImage.CvPtr);
```

# 11 Šprint 3 - Škrečok

## 11.1 Plánovanie notifikácií

### 11.1.1 Analýza

Aplikácia vyhodnocuje stav používateľa. Ak používateľ sedí istú dobu nesprávne, upozorní ho na to.

Pri module na plánovanie notifikácií je potrebné zohľadniť, že udalosti môžu prichádzať z rôznych vlákien, resp. ďalšia udalosť nastane skôr, ako modul stihne vyhodnotiť predchádzajúcu. Modul musí notifikácie zobrazovať správne podľa toho, či používateľ naozaj sedí nesprávne a či sedí dostatočne dlho nesprávne. Ak sedí používateľ nesprávne, zobrazí mu notifikáciu. To však iba vtedy, ak už nie je zobrazená alebo naplánovaná iná notifikácia. Naopak, ak je zobrazená alebo naplánovaná notifikácia a používateľ sa posadí správne, modul skryje zobrazené notifikácie a zruší naplánované notifikácie.

Modul reaguje na zmenu nastáví v aplikácii. Modul postupne vyberá notifikácie podľa toho, ako boli nastavené ich časové limity v nastaveniach.

### 11.1.2 Návrh

Aby bolo umožnené postupné spracovanie udalostí aj z viacerých vlákien, využil sa princíp z paralelného programovania známy ako "producer-consumer".

Modul na plánovanie notifikácií sa skladá z týchto hlavných častí:

- EventPreprocessor - spracováva rôzne druhy udalostí. Má úlohu producenta (z *producer-consumer*).
- NotificationPlanner - Konzumuje udalosti produkované EventPreprocessorom. Obsahuje logiku výberu správnej notifikácie, zároveň aj logiku ohľadom skrytia notifikácií.
- NotificationManager - Má na starosti samotné zobrazenie a skrytie notifikácií. Uchováva naplánované notifikácie. Má informáciu, ktorá notifikácia je aktuálne zobrazená.

### 11.1.3 Implementácia a testovanie

Implementácia *producer-consumer* je zabezpečená pomocou `BlockingCollection` a `ConcurrentQueue`, ktoré poskytuje .NET framework. Ukážka:

```
public void Handle(PostureMonitoringStatusChange message)
{
    if (message.IsMonitoring)
    {
        eventQueue = new BlockingCollection<Evaluation>(new
        ConcurrentQueue<Evaluation>());
        Task.Run(() => notificationPlanner.Consume(eventQueue));
    }
    ...
}
```



Rozsiahle testovanie prebehlo pomocou unit testovania.

## 11.2 Logujeme používanie aplikácie

### 11.2.1 Úloha

Vybrať logovací rámec, vytvoriť konfiguračný súbor logovania a logovať interakciu používateľa s aplikáciou. Cieľom logovania pre vývojárov je uľahčiť hľadanie chýb v aplikácií a lepšie pochopiť štruktúru aplikácie. Navyiac, logy vygenerované pri používaní aplikácie chceme použiť na analýzu používania aplikácie používateľmi.

### 11.2.2 Analýza

Požiadavky na logovanie:

- jednoduchosť použitia
- “nezašpinenie” kódu
- možnosť nastaviť formát výstupu, úroveň logovania a miesto logovania
- spolupráca s Caliburn Micro
- podpora Eventov a WPF
- možnosť odosielania na server

Vyhovujúce logovacie frameworky

- log4net
- nlog

### 11.2.3 Návrh

Rozhodli sme sa použiť logovací framework [nlog](#), pretože je aktívne vývíjaný, jednoduchý na používanie a podporovaný analytickými nástrojmi (napr. [Application Insights](#)) a frameworkom CM.

Pri logovaní sa väčšinou loguje na začiatku a konci vykonávania funkcie. Tento kód pre logovanie je duplicitný a zhoršuje čitateľnosť kódu. Žiadne dostupné logovacie frameworky neriešia tento problém. Preto sme sa rozhodli využiť princípy aspektovo-orientovaného programovania (AOP) pre riešenie tohto problému. Platforma .NET nepodporuje AOP a jeho implementáciu je možné realizovať obalením do Proxy triedy, ktorá smeruje všetky volania cez aspekty, alebo doplnením aspektov do vygenerovaného IL (angl. Intermediate Language) binárneho kódu. Rozhodli sme sa použiť framework [Postsharp](#) prístup doplnenie aspektov do IL kódu, pretože dopĺňa aspekty až po builde a umožňuje vytvoriť anotácie pre použitie aspektov.

### 11.2.4 Implementácia a testovanie

Implementovali sme triedu *NLogLogger*, ktorá zabezpečuje logovanie. Táto trieda realizuje rozhrania *ILog*, ktoré používa framework CM v rámci svojho logovania. Realizáciou tohto rozhrania sme zjednotili logovanie aplikácie a logovanie v rámci CM. Toto rozhranie poskytuje metódy *Info*, *Warn*, *Error*.

Pre účely nášho testovania sme implementovali rozhranie *ILogger*, ktoré dedí od rozhrania CM *ILog*. Toto rozhranie pridáva metódu *Debug*, ktorá je vhodná pre vývojárske účely.



Ukážka ako si vypýtať logger pre triedu DataSourceManager a ukážka ako logovať.

```
private readonly ILogger log = NLogLogger.GetLogger<DataSourceManager>();
log.Debug("Start data source choice");
```

Súbor *NLog.config* je jednotné miesto pre nastavenie úrovne logovania pre jednotlivé triedy, formátovanie výstupu a miesta, kam logovať.

Implementovali sme aj anotáciu *LogMethodCall*, ktorá dedí od Postsharp triedy *OnMethodBoundaryAspect*. Umožní nám prístup k metódam:

- OnEntry - spustí sa pred vykonaním tela metódy
- OnSucess - spustí sa po úspešnom vykonaní metódy (bez vyhodenej výnimky)
- OnExit - spustí sa vždy po vykonaní metódy
- OnException - spustí sa po vykonaní metódy, ak táto metóda vyhodí výnimku

Priklad kodu alebo [OnMethodBoundaryAspect](#)

```
public override void OnSuccess(MethodExecutionArgs args)
{
    log.Info($"{message} {className}.{methodName}() sucessfully returns {args.ReturnValue}");
}
```

Použitie anotácie pre logovanie

```
[LogMethodCall]
public bool TryStartDataSource(IDataSource dataSource)
{
    ...
}
```

## 11.3 Spúšťanie a riadenie strážcov

### 11.3.1 Analýza

V aplikácii sa momentálne používajú dvaja strážcovia. Jeden analyzuje hĺbkový obraz a druhý analyzuje iba pozíciu hlavy vo farebnom obraze. Je potrebné rozumne spraviť ich spúšťanie vzhľadom na to, že na analýzu si vyžadujú rôzny procesorový čas. Keďže sa analýza bude vykonávať vo vlastných vláknach je potrebné aby volajúca trieda mala aj možnosť počkať na výsledky od všetkých strážcov. Taktiež je potrebné spraviť spájanie týchto výsledkov do celku, ktorý sa následne ďalej spracováva v aplikácii.

### 11.3.2 Návrh

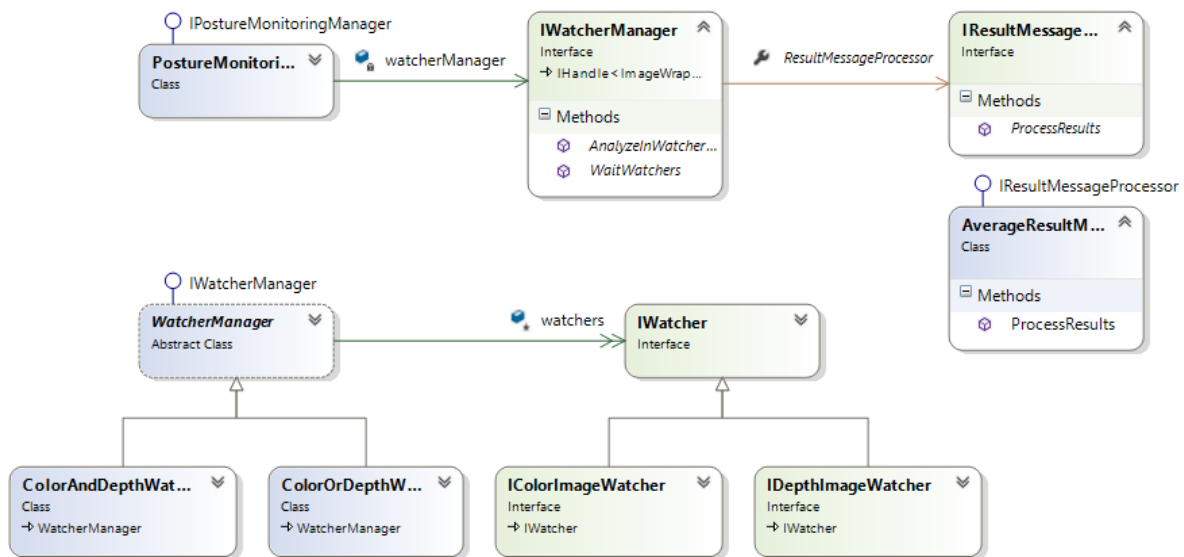
Analýza jednotlivých strážcov sa bude vykonávať paralelne vo vlastnom vlákne. Analýza v strážcoch sa spustí na základe toho či sú k dispozícii hĺbkový a farebný obraz. Na výpočet celkového sedenia používateľa sa výsledky z jednotlivých strážcov spriemerujú.

### 11.3.3 Implementácia

Implementované triedy a rozhrania a ich účel:

- **IWatcherManager** - rozhranie pre riadenie strážcov.

- *AnalyzeInWatchers()* - spustí analýzu v strážcoch.
- *WaitWatchers()* - volajúci počká kým sa dokončí analýza vo všetkých strážcoch.
- **WatcherManager** - abstraktná trieda obsahujúca zoznam strážcov a vykonávaných úloh.
- **ColorAndDepthWatcherManager** - spustí sa analýza hĺbkového a farebného obrazu.
- **ColorOrDepthWatcherManager** - spustí sa analýza hĺbkového alebo farebného obrazu.
- **IResultMessageProcessor** - rozhranie pre rôzne metódy spracovania výsledkov strážcov.
- **AverageResultMessageProcessor** - ako celkový výsledok sa berie priemer výsledkov strážcov.
- **IColorImageWatcher** - strážca realizujúci toto rozhranie vyžaduje farebný obraz.
- **IDepthImageWatcher** - strážca realizujúci toto rozhranie vyžaduje hĺbkový obraz.



Obrázok 5 Diagram tried pre riadenie a spúšťanie strážcov

## 12 Celkový pohľad na systém

Za prvé tri šprinty sa nám podarilo zlepšiť architektúru a stabilitu aplikácie. Existujúca funkcionálnosť ako GUI, notifikačný systém a analýza kvality držania tela bola zachovaná a prispôbena na novú architektúru. Zdrojový kód jednotlivých modulov sa nám podarilo pokryť unit testami. Bola pridaná podpora najnovšej verzie Intel kamery a logovanie.

Stanovili sme si cieľ, že po troch šprintoch chceme, aby boli implementované najdôležitejšie časti aplikácie a aplikácia bola použiteľná na testovanie. Cieľ sa nám podarilo naplniť a projekt sa posúvame do fázy testovania aplikácie vybranými používateľmi, ktorí budú aplikáciu dlhodobejšie používať.

Kompletná funkcionálnosť aplikácie po troch šprintoch:

- Získavanie dát z kamery - aplikácia podporuje všetky farebné kamery a vybrané hĺbkové kamery (Intel Realsense Sens3d, Intel Realsense F200) a automaticky zvolí najlepšiu dostupnú kameru.
- Vyhodnocovanie kvality držania tela - implementované sú dve metódy vyhodnocovania kvality držania tela z jednotlivých snímok. Tieto metódy vyhodnocujú aktuálny typ sedenia (predklonenie, zaklonenie, správna poloha) a odhadujú kvalitu držania tela hodnotami v rozmedzí 0-100.
- Kalibrácia - aplikácia umožňuje nastaviť kalibračnú snímku
- Notifikácie - aplikácia upozorní používateľa na nesprávne sedenie popup a taskbar notifikáciou.
- Nastavenia - aplikácia umožňuje nastaviť dobu po uplynutí ktorej sa zobrazí príslušná notifikácia a či sa má používať aj hĺbková kamera.

## 12.1 Použité knižnice a frameworky

- Postsharp - framework, ktorý sa používa na podporu AOP.
- NLog - knižnica na logovanie.
- Caliburn.Micro – framework pre návrh a tvorbu aplikácií. Podporuje veľké množstvo MV\* vzorov.
- Hardcodet.NotifyIcon.Wpf – implementácia task bar icony pre WPF platformy.
- OpenCvSharp3-AnyCPU - cross platform OpenCV wrapper pre .NET Framework.
- OxyPlot – knižnica pre vytváranie grafov pre .NET.
- Newtonsoft – JSON framework pre prácu s json súbormi.
- DirectShowLib-2005 - knižnica určená pre prácu s webovou kamerou.



## 12.2 Dátový model

Zatiaľ nemáme databázu. Plánujeme však ukladať jednotlivé vyhodnocovania kvality držania tela, ktoré plánujeme zobrazovať zoskupené podľa časových období na grafoch.

## 12.3 Moduly

### 12.3.1 Modul pre štatistiky

#### Analýza

Úlohou tohto modulu má byť uchovávanie a spracovanie údajov o tom ako používateľ sedí za počítačom a prípadnú ich úpravu na iný tvar.

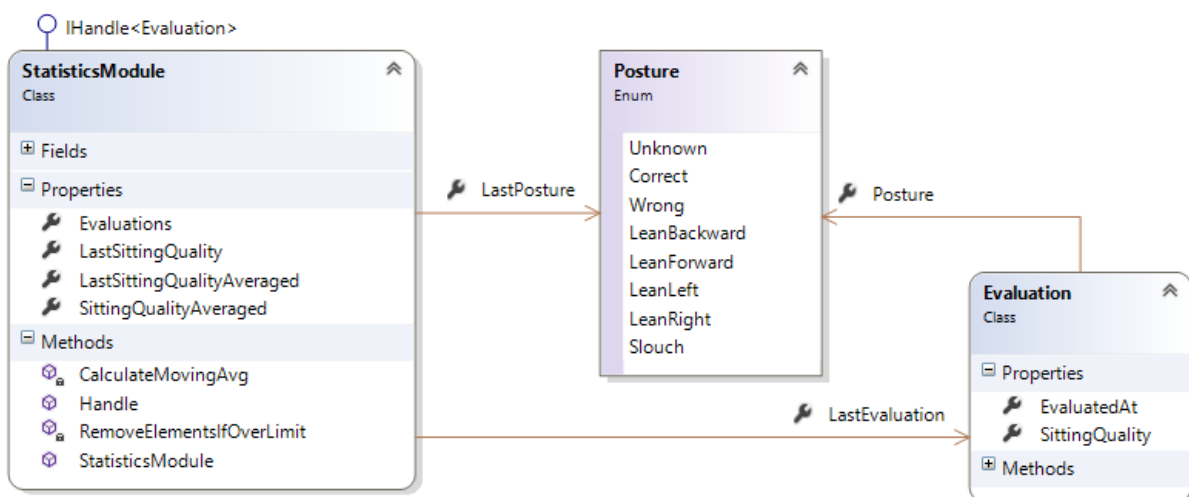
#### Návrh

Modul sa prihlási na príjem všetkých potrebných udalostí, ktoré aj spracuje. Uchovávať sa bude zoznam evaluácií sedenia a aj jeho upravená verzia. Dôvod úpravy je, že hodnota kvality sedenia jednotlivých evaluácií môžu skákať. Preto sa bude robiť z týchto evaluácií kľzavý priemer, ktorý sa tiež bude uchovávať. Na to aby sa zbytočne nepridávali evaluácie donekonečna sa budú tieto dva zoznamy priebežne mazat'.

#### Implementácia

Bola implementovaná trieda *StatisticsModule*, vid' obrázok nižšie, ktorá má nasledovné public properties:

- **Evaluations** - zoznam prijatých evaluácií sedenia. Evaluácie obsahujú kvalitu sedenia a polohu tela používateľa.
- **LastEvaluation** - posledná prijatá evaluácia sedenia (Posledný prvok z listu Evaluations).
- **LastPosture** - posledná pozícia toho ako používateľ sedel.
- **LastSittingQuality** - posledná vyhodnotená kvalita sedenia používateľa
- **LastSittingQualityAveraged** - posledná spriemerovaná hodnota kvalít sedenia používateľa (Posledný prvok z listu *SittingQualityAveraged*).
- **SittingQualityAveraged** - zoznam spriemerovaných hodnôt kvality sedenia používateľa.



Obrázok 6 Diagram tried modulu pre štatistiky

Zoznamy *Evaluations* a *SittingQualityAveraged* sa postupne mažú. Momentálne je maximálna kapacita týchto zoznamov 200 prvkov a po jej prekročení sa vymaže 100 najstarších hodnôt. Kľzavý priemer sa počíta z piatich posledných hodnôt kvality sedenia. Tieto nastavenia je možné zmeniť v *Const.Settings*. Kľzavý priemer sa počíta sa nasledovným spôsobom:

```
private int CalculateMovingAvg()
{
    var cnt = Evaluations.Count;
    if (cnt > EVALUATION_MOVING_AVG_COUNT)
        cnt = EVALUATION_MOVING_AVG_COUNT;
    var sum = 0;
    for (int i = 1; i <= cnt; i++)
    {
        sum += Evaluations[Evaluations.Count - i].SittingQuality;
    }
    sum /= cnt;
    return sum;
}
```

## 12.3.2 Modul notifikácií

### 12.3.2.1 Analýza

Aplikácia vyhodnocuje stav používateľa. Ak používateľ sedí istú dobu nesprávne, upozorní ho na to. Za upozorňovanie používateľa je zodpovedný modul notifikácií.

Pri module na plánovanie notifikácií je potrebné zohľadniť, že udalosti (*events*) môžu prichádzať z rôznych vlákien, resp. ďalšia udalosť nastane skôr, ako modul stihne vyhodnotiť predchádzajúcu. Modul musí notifikácie zobrazovať správne podľa toho, či používateľ naozaj sedí nesprávne a či sedí dostatočne dlho nesprávne. Ak sedí používateľ nesprávne, zobrazí mu notifikáciu. To však iba vtedy, ak už nie je zobrazená alebo naplánovaná iná notifikácia. Naopak, ak je zobrazená alebo naplánovaná notifikácia a používateľ sa posadí správne, modul skryje zobrazené notifikácie a zruší naplánované notifikácie.

Modul reaguje na zmenu nastaví v aplikácii. Modul postupne vyberá notifikácie podľa toho, ako boli nastavené ich časové limity v nastaveniach.

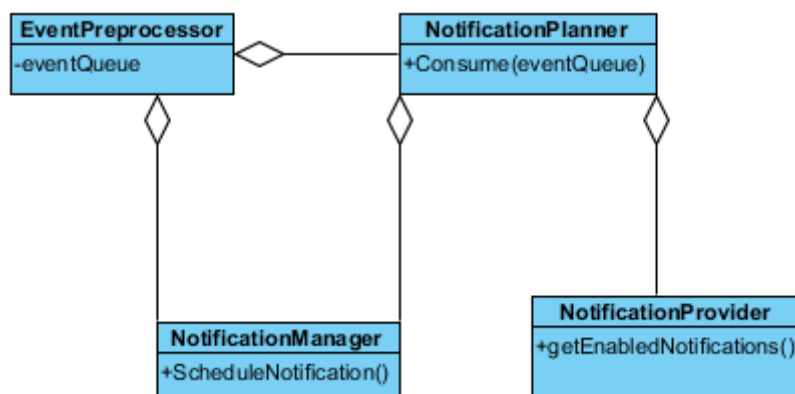
### 12.3.2.2 Návrh

Aby bolo umožnené postupné spracovanie udalostí aj z viacerých vlákien, využil sa princíp z paralelného programovania známy ako “producer-consumer”.

Modul na plánovanie notifikácií sa skladá z týchto hlavných častí:

- EventPreprocessor - spracováva rôzne druhy udalostí. Či už je to zapnutie (vypnutie) monitorovania, zmenu nastavení, ale hlavne novú evaluáciu používateľovho sedenia. Má úlohu producenta (z *producer-consumer*). Zachytené udalosti evaluácie ukladá do radu (queue) - “produkuje” ich pre NotificationPlanner.
- NotificationPlanner - Konzumuje udalosti produkované EventPreprocessorom. Obsahuje logiku výberu správnej notifikácie, zároveň aj logiku ohľadom skrytia notifikácií.

- NotificationManager - Má na starosti samotné zobrazenie a skrytie notifikácii. Uchováva naplánované notifikácie. Má informáciu, ktorá notifikácia je aktuálne zobrazená.



### 12.3.2.3 Implementácia a testovanie

Implementácia *producer-consumer* je zabezpečená pomocou `BlockingCollection` a `ConcurrentQueue`, ktoré poskytuje .NET framework.

Rozsiahle testovanie prebehlo pomocou unit testovania. Testoval sa výber správnej notifikácie, správneho časového limitu pre notifikácie a ďalšie.

## 12.3.3 Modul pre analýzu a vyhodnocovanie kvality sedenia

### 12.3.3.1 Analýza

Na analýzu a vyhodnocovanie kvality sedenia je možné použiť dva prístupy. Je možné k tomuto problému pristúpiť ako spracovanie video záznamu v reálnom čase alebo spracovanie jednotlivých snímok v pravidelných časových intervaloch.

K dispozícii bude farebný a hĺbkový obraz, ktorý sa bude analyzovať v strážcoch.

### 12.3.3.2 Návrh

Proces monitorovania obsahuje nasledujúce činnosti:

1. Spustenie monitorovania
2. Získanie dát na analýzu.
3. Analýza týchto dát v strážcoch.
4. Spojenie výsledkov strážcov do jedného celku.
5. Zverejnenie evaluácie kvality sedenia a držania tela používateľa.
6. Pokračuje sa krokom 2. pokiaľ sa neukončí monitorovanie.

### 12.3.3.3 Implementácia

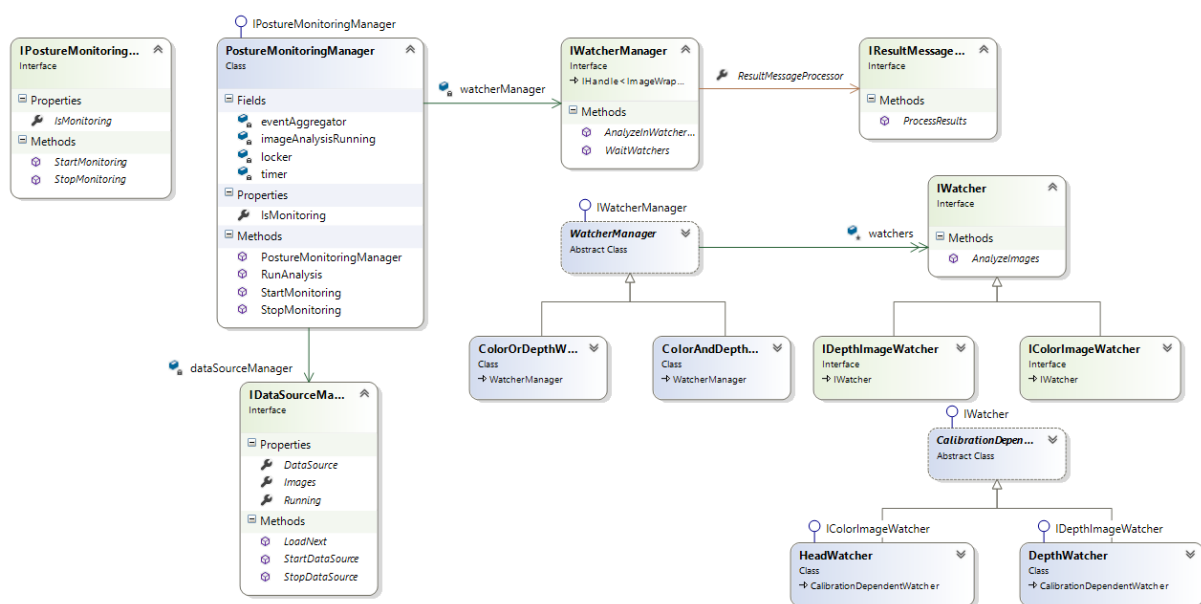
Hlavnou riadiacou jednotkou tohto modulu je *PostureMonitoringManager*. Táto trieda realizuje rozhranie *IPostureMonitoringManager*, ktorej úlohou je poskytovanie možnosti pre zapínanie a vypínanie analýzy toho ako používateľ sedí za počítačom. Táto trieda má tieto dve hlavné závislosti:



- **IWatcherManager** – má na starosti spúšťanie a riadenie strážcov a vrátenie výslednej evaluácie. Viac informácií o princípe fungovania riadenia strážcov a výpočtu celkovej evaluácie sedenia je v kapitole **Chyba! Nenašiel sa žiaden zdroj odkazov..**
- **IDataSourceManager** – umožňuje zapnúť / vypnúť *IDataSource* a používa sa na získavanie dát pomocou metódy *LoadNext()*. Ďalšie informácie sú v kapitole 10.4 a 12.3.4

V aplikácii sa používajú na analýzu obrázkov dvaja strážcovia:

- **HeadWatcher** – určuje kvalitu sedenia a držania tela na základe pozície hlavy vo farebnom obrázku.
- **DepthWatcher** – určuje kvalitu sedenia a držania tela na základe analýzy a spracovania hĺbkovej mapy.



Obrázok 7 Diagram tried modulu pre analýzu a vyhodnocovanie kvality sedenia

## 12.3.4 Modul pre získavanie dát z kamier

### 12.3.4.1 Analýza

Naším cieľom je podpora všetkých farebných a hĺbkových kamier. Chceme aby pre podporu novej kamery boli potrebné žiadne alebo minimálne zmeny v zdrojovom kóde. Zistili sme však, že na trhu existuje veľa hĺbkových kamier a mnoho z nich má svoj vlastný spôsob integrácie s aplikáciami cez vlastné knižnice.

Pretože logika analýzy a vyhodnocovania kvality sedenia je navrhnutá tak, aby sa spracovávali jednotlivé snímky, je nutné poskytnúť rozhranie pre prístup k aktuálnej snímke z kamery.

### 12.3.4.2 Návrh

Pre zabezpečenie modularity, lepšej testovateľnosti a abstrahovania od používanej kamery navrhujeme použiť rozhranie. Toto rozhranie bude poskytovať metódy na spustenie/vypnutie

kamery a poskytnutie aktuálneho snímku z kamery. Jednotlivé implementácie získavania dát z kamier budú realizovať toto rozhranie.

Navrhujeme ďalej vytvoriť triedu pre správu pripojených kamier, automatické spúšťanie vhodnej kamery a poskytovanie aktuálnych snímkov z kamery.

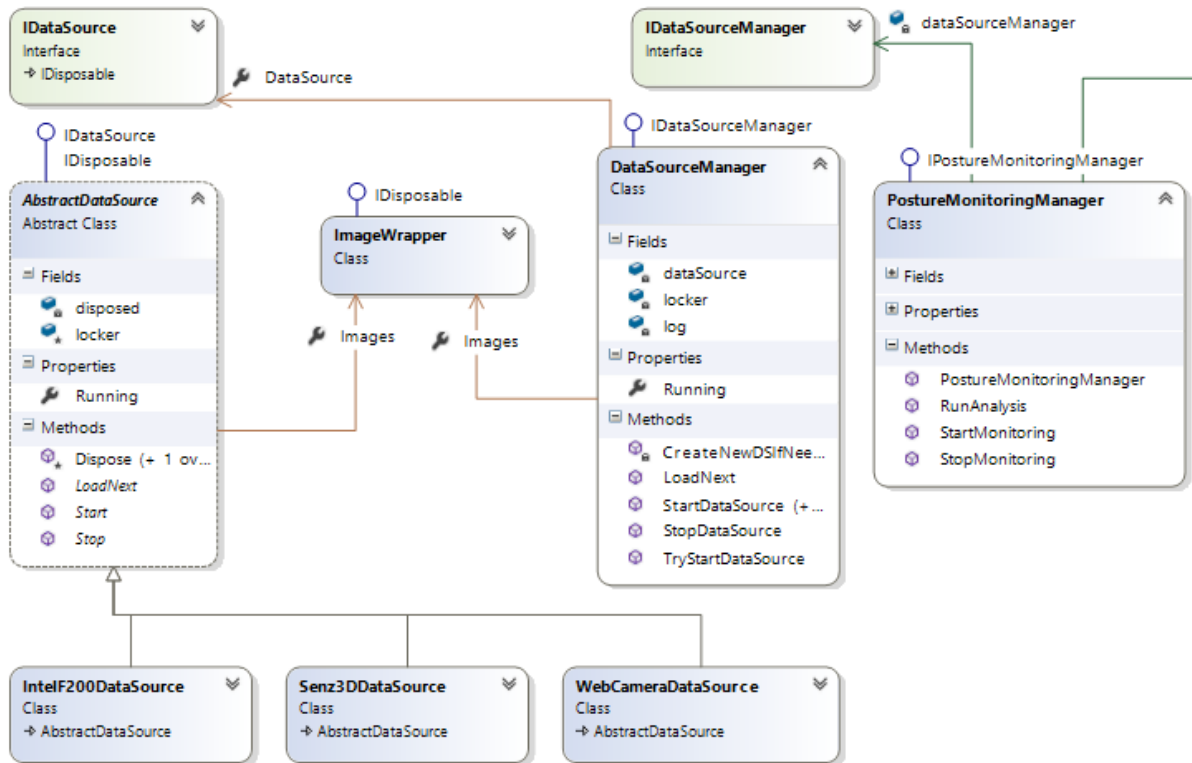
#### 12.3.4.3 Implementácia a testovanie

Implementovali sme rozhranie *IDataSouce* pre získavanie dát z kamier. Abstraktná trieda *AbstractDataSource* pridáva metódy na prácu s nemanážovaným kódom. Trieda *ImageWrapper* poskytuje pomocné triedy na spracovanie snímku a konverziu snímkov do jednotného formátu *OpenCvSharp.Mat*.

Načítanie aktuálneho snímku z *DataSource* (aktuálne používanej kamery) v triede *DataSourceManager*.

```
public ImageWrapper LoadNext()
{
    lock (locker)
    {
        if (!Running || !DataSource.LoadNext()) return null;
        return DataSource.Images;
    }
}
```

Diagram tried pre modul získavania dát z kamier. Zobrazené aj napojenie na triedu *PostureMonitoringManager*, ktorá riadi proces získavania snímkov a následnú analýzu a vyhodnocovanie snímkov.



Obrázok 8 Diagram tried modulu pre získavanie dát z kamier

## 12.3.5 Modul pre kalibráciu

### 12.3.5.1 Analýza

Modul má zabezpečiť kalibráciu aplikácie. Zobrazíť používateľovi ako sedí ten sa následne upraví do správnej polohy a vytvorí kalibráciu, ktorú budú watcher-e používať na vyhodnotenie aktuálnej polohy. Modul ďalej zabezpečuje uloženie kalibrácie na disk a jej načítanie pri spustení alebo zmene data providera.

### 12.3.5.2 Návrh

Vytvoríť dialógové okno v ktorom sa zobrazí obraz z kamery. AK používateľ zvolí kalibráciu obrázok skontrolujem tak že v ňom nájdem hlavu. Ak na obrázku nie je hlava kalibráciu označím za neúspešnú. Obraz z kamery získavam z *DataSouceManager* čím zabezpečujeme nezávislosť od dátového zdroja.

### 12.3.5.3 Implementácia a testovanie

Implementovali sme *CalibrationManager*. Trieda zabezpečuje uloženie a načítanie kalibrácie z pevného disku. Kalibráciu distribuujeme pomocou eventu.

Používateľovi zobrazíme dialógové okno kde je zobrazený farebný obraz, ktorý je získaný z triedy *DataSourceManeger*. Ak používateľ potvrdí nakalibrovanie aplikácie skontrolujeme či na obrázku vieme nájsť hlavu. Ak nie ponúkneme možnosť kalibráciu zopakovať. Ak sme hlavu našli je potrebné kalibráciu potvrdiť. Po potvrdení je kalibrácia eventom odoslaná ostatným častiam aplikácie a *CalibrationManager* uloží získanú kalibráciu na pevný disk.