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: 2015/2016

Dátum odovzdania: 14. 12. 2015

Obsah

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 dokumentácií 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

Tím tvoria štyria členovia. Vedúcim tímu je Ing. Jakub Šimko, PhD.

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 fanúšikom 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 súťaží 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 hardvérovými 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 Jave 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 manažovať 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 čŕt 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.

2.2Roly 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

2.2.3 Podiel práce

Každý člen tímu pracoval podľa svojej manažérskej pozície na častiach dokumentácie k daným oblastiam manažmentu.

3 Používané metodiky

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** – Implementačná činnosť je inicializovaná vytvorením novej vetvy na základe definovaných úloh v nástroji JIRA. K tejto činnosti prislúcha Metodika verziovania aplikácie.
2. **Implementácia** – Predstavuje proces samotnej implementácie riešenia zadanej úlohy, či už sa jedná o pridanie novej funkcionality, opravy chyby alebo refaktorizácie časti kódu. Pred začatím implementácie by mal byť programátor oboznámený s Metodika organizácie zdrojových kódov.
3. **Testovanie** – Riešenie úlohy by malo byť pokryté z čo najväčšej časti testami aby sa zabezpečilo, že pri budúcej úprave sa nepoškodia časti, ktoré fungovali doteraz správne (regresné testovanie). Ako písať jednotkové testy je opísané v Metodika písania unit testov.
4. **Úprava dokumentácie –** Dôvod písania dokumentácie k riešeniu je jednoduchý. Umožňuje neblokujúcim spôsob inému programátorovi pochopiť spôsob implementácie časti kódu ktorej sa nevenoval, resp. si už nepamätá detaily. K postupu písania dokumentácie je napísaná Metodiky pre tvorbu dokumentácie.
5. **Vytvorenie pull requestu –** Touto činnosťou programátor formálne ukončuje implementačnú činnosť na danej úlohe. V Metodika prehliadky kódu je uvedený odporúčaný spôsob a je aj stanovená forma ako vytvárať pull requesty.
6. **Prehliadka kódu** – Je veľmi dôležitou činnosťou, ktorá umožňuje odhalenie prípadných nedostatkov riešenia implementácie danej úlohy. Pre tento proces bola vytvorená Metodika prehliadky kódu.
7. **Integrácia kódu** – Ak zdrojový kód a prislúchajúce testy prešli prehliadkou kódu je možné vytvorený pull request spojiť s vetvou do ktorej sa pull request vytváral. V väčšine prípadov by sa malo jednať o hlavnú vývojársku vetvu. Viac k spájaniu vetiev je spomenuté v Metodika prehliadky kódu.

K týmto jednotlivým procesom boli vytvorené metodiky, ktoré sú rozpísané v nasledujúcich častiach. Tieto metodiky sú určené pre všetkých členov tímu, ktorí sa podieľajú na

implementácii aplikácie. V tabuľka 2 sa nachádza zoznam rol a úlohy za, ktoré sú jednotlivé roly zodpovedné.

Tabuľka 1Slovník pojmov

Pojem	Vysvetlenie
Zdrojový kód	Kolekcia súborov zrozumiteľných človeku napísaných v programovacom jazyku, ktoré sa kompilujú do binárnej formy pre danú počítačovú architektúru.
Lokálny repozitár	Lokálna kópia histórie zmien uložená na počítači.
Vzdialený repozitár	História zmien uložená na centrálnom serveri.
Zápis zmien (angl. commit)	Zápis množiny vykonaných zmien do lokálneho repozitára. Táto činnosť vytvorí novú revíziu (angl. commit).
Nahratie zmien (angl. push)	Zápis lokálnych revízií do vzdialeného repozitára.
Stiahnutie pracovnej kópie (angl. checkout)	Zo vzdialeného repozitára sa stiahne pracovná verzia do lokálneho repozitára.
Žiadosť o natiahnutie (angl. pull request)	Vezme sa vetva do ktorej chceme spájať a vetva, ktorú chceme pripojiť a vytvorí sa zhrnutie všetkých zmien, ktoré je potrebné n.
Spojenie vetiev (angl. merge)	Integrácia zmien jednej vetvy do druhej.

Tabuľka 2Roly a zodpovednosti

Rola	Zodpovednosť
Manažér kvality, podpory vývoja a integrácie	Správa a konfigurácia repozitárov. Hlavný kontrolór kvality kódu. Hlavný integrátor.
Kontrolór kvality kódu	Kontrola dodržiavania konvencií pri písaní zdrojového kódu. Kontrola správnej funkcionality kódu. Schválenie alebo zamietnutie pull requestu. Napísanie nedostatkov autorovi kódu alebo ich rýchla oprava.
Integrátor	Organizácia vetiev a ich spájanie.
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 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 Použité nástroje

Ako nástroj pre verziovanie aplikácie sa používa verziovací systém git[1]. Každý člen tímu, ktorí sa podieľa na vývoji aplikácie musí byť oboznámený s týmto nástrojom. Táto metodika však neobsahuje to ako pracovať s verziovacím nástrojom vzhľadom na to, že na oficiálnej stránke git-u je rozsiahla dokumentácia pokrývajúca túto časť. Táto metodika obsahuje iba konvencie pri používaní verziovacieho systému. Výber klientského nástroja na prácu s git-om nie je pevne daný a jeho výber je na zvážení každého člena tímu. Odporúčaným klientským nástrojom je SourceTree[2]. Taktiež je možné použiť na prácu git konzolu v prípade potreby. Ako vzdialený repozitár sa používa služba Bitbucket[3].

3.1.1.2 Pomenovanie vetiev

V aplikácii sú tieto hlavné vetvy:

- master – v tejto vetve je aktuálna stála verzia aplikácie,
- development – hlavná vývojárska vetva aplikácie.

Ostatné vetvy musia spĺňať nasledovnú konvenciu pomenovania podobnú k tomu čo sa generuje v JIRA:

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

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

[Číslo úlohy] – také aké bolo vygenerované JIRA.

[Krátky popis] – taký ako v JIRA, ak je príliš dlhý tak porozmýšľať nad výstižnejším názvom úlohy a premenovať ju.

[1] GIT: https://git-scm.com/

[2] SourceTree: https://www.sourcetreeapp.com/

[3] Bitbucket: https://bitbucket.org/

3.1.1.3 Vytvorenie novej vetvy

Každá implementačná úloha z JIRA by mala mať vlastnú vetvu. Každá nová vetva, ktorá sa vytvára pre samostatné úlohu by mala čerpať z *development* vetvy. Podúlohy by mali čerpať z úlohy, ktorá je ich otcom.

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 konvencie.
- Potvrdíme vytvorenie novej vetvy.

3.1.1.4 Zápis zmien

Štýl pomenovania jednotlivých revízií je nasledovný:

[Typ]: [Commit message]

[Typ] – typ revízie. Môže nadobúdať hodnoty *ADD* a *FIX*.

[Commit message] – maximálne 50 znakový, výstižný opis zmeny.

Je dôležité „commitovať“ pravidelne po malých častiach aby bolo jednoduchšie identifikovať pôvod vzniku problémov.

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 nachádali sa v namespace *[BaseNamespace].Views.[Category]* a triedy pre *ViewModel* mali príponu *ViewModel* a nachádali 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:

Tabuľka 3 Štýl pomenovania typov

Typ	Pomenovanie	Poznámka
File and types	PascalCase	Viď. vyššie.
Namespace	PascalCase	Viď. 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 *var* v prípade, že z pravej strany je jasné o aký typ premennej sa jedná.
Parameters	camelCase	Použiť skrátený názov v prípade, že je jasné na čo sa daný parameter používa, resp. to vyplýva z typu parametru.
Private fields	camelCase	
Public fields	PascalCase	Vyhnúť sa ich používaniu. Namiesto nich používať Properties. Neplatí pre public static field.
Static readonly fields	ALL_UPPER	Konštanty vypočítavané za behu programu (Napr. načítané z nastavení).
Constants	ALL_UPPER	
Enum members	PascalCase	

Oranizácia 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 funkcionalita 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

Tabuľka 4 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 funkcionalitu, preto v niektorých prípadoch bude stále treba použiť *Assert* (napr. *Assert.Throw)*.

```
[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));
    }
}
```

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 kvalitou a funkcionalitou požiadavkám. Prehliadky kódu umožňujú ostatným členom tímu 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 pridelenie 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.

3.2Manažment 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 navrhneme 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 člennou 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 funkcionalitu. 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	Pripraviť analýzu riešenia úloh na začiatku šprintu. Zabezpečí lepši 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 implementácií
Spúšťač	Nedodržanie metodiky
Čas	Počas celého projektu
Dopad	Pridanie zbytočnej práce
Pravdepodobnosť	Nízka
Opatrenia	
Preventívne opatrenia	Pravidelne nahrávať zmeny do repozitáru projektu, častejšie stand-up stretnutia
Riešenie po prepuknutí	Prehodnotenie naplánovania úloh, dohovorenie zúčastneným stranám

3.3Manažment 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 takýto plán 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 blokovaný 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

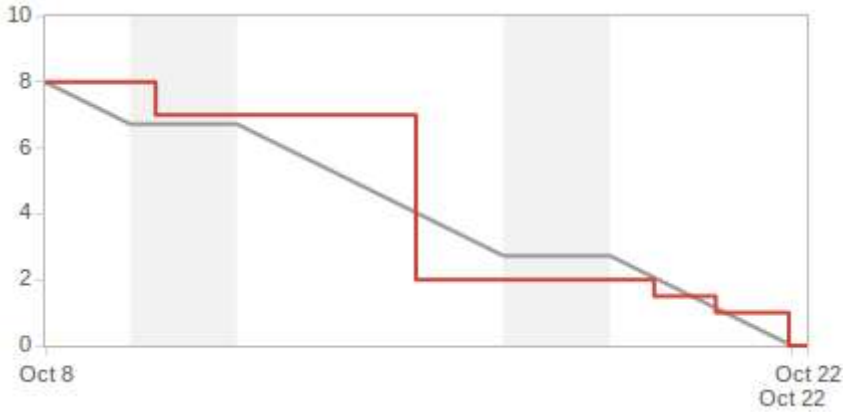
V našom tímovom projekte využívame, rovnako ako aj ostatné tímy, agilnú metodológiu SCRUM. Využívame jeho upravenú verziu. Oficiálny spôsob použitia SCRUM-u totižto vyžaduje denné “stand-up” mítingy. To však nie je možné aplikovať v našej situácii ako študentov s rôznymi rozvrhmi a ďalšími povinnosťami. 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 prideľovania 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 zaťažený, 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

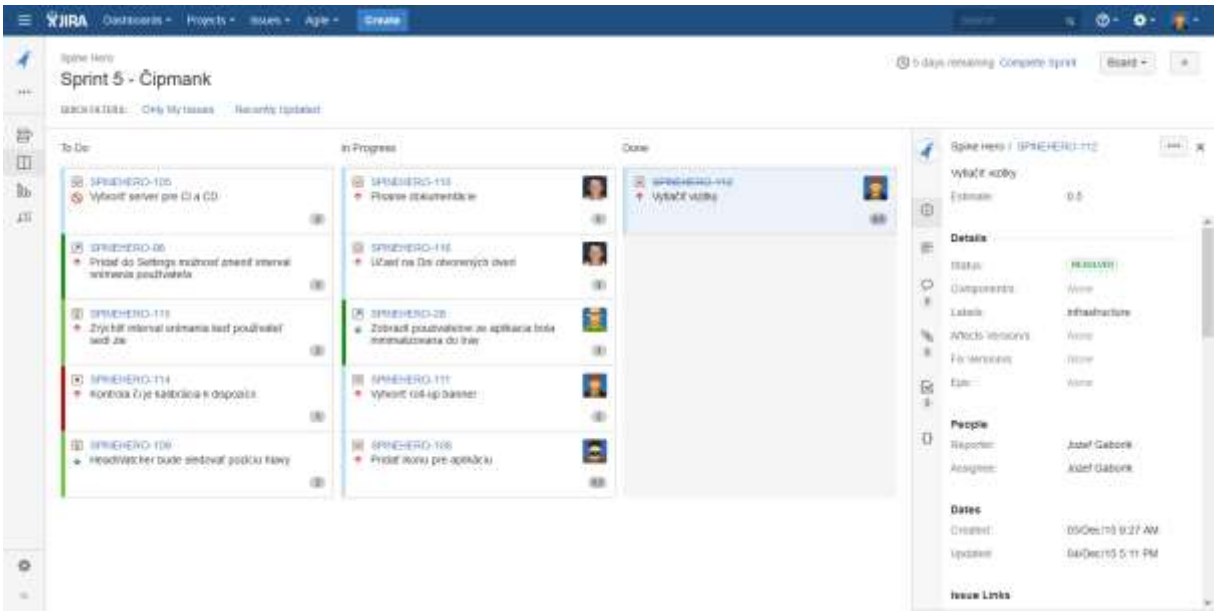
Na určenie *story points*, pre každú položku z *product backlog* sa využíva planning poker. Princíp je nasledovný. Najskôr sa prečíta a opíše práve hodnotená položka. Následne každý člen tímu vyloží zo svojho balíčka kariet kartu s číslom, na koľko story points by danú úlohu ohodnotil. Ak nenastane zhoda,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

3.3.1.3 Použité nástroje

Celé plánovanie prebieha v nástroji JIRA od spoločnosti Atlassian spolu s jej rozšírením JIRA Agile. Základom je zobrazenie product backlog-u a aktuálneho šprintu. Pri zobrazení aktuálneho šprintu je možné vidieť úlohy v troch stĺpcoch (To Do, In Progress a Done).

Vďaka tomu vieme presne zistiť, ktoré úlohy sú už zadelené, ktoré dokončené a na ktorých je treba ešte len začať pracovať.



Obrázok 2 Zobrazenie aktuálneho šprintu v nástroji JIRA

Položky v product backlogu sú označované štítkami podľa toho, ktorej oblasti práce sa dotýkajú (development, business, infrastructure). Položky sú navyše rozdelené podľa druhu na user story, task, improvement a new feature. V rámci tímu máme snahu čo najviac využívať user story, avšak niektoré veci už priamo zo svojej podstaty nie je možné takto napísať.

Každá položka obsahuje minimálne názov a popis. Pri veľmi jednoduchých úlohách sa môže popis vynechať. Už z názvu musí byť jasné čo sa skrýva pod danou položkou. Popis slúži na zodpovedanie otázky prečo je potrebné riešiť danú položku. Obsahuje aj podrobnejší popis toho čo treba vykonať, aby bolo jasné, kedy je daná úloha dokončená.

3.4Manažment 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. Naviac, 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 zadefinovať efektívne prístupy pre komunikáciu v tíme. Metodika opisuje aké komunikačné kanály použité 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, #app-bitbucket)
- 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 a na túto mailovú schránku máme aj nastavený aj alias info@spinehero.com. Mailová 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

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
- všeobecná diskusia

- hodnotenie, hlasovanie
- 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

3.5Manažment 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 čim 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

Dokumentácia

Povinnosťou každého člena tímu je dokumentovať používateľský príbeh, ktorý vyvíjal. Táto časť dokumentácie sa zapisuje do Dokumentácie k inžinierskemu dielu do časti Šprint X - názov šprintu. Dokumentácia k používateľskému príbehu pozostáva z týchto bodov:

- Analýza - analýza problému, v ktorom je sú diskutované možnosti riešenia.
- Návrh - návrh riešenia so zdôvodnením prečo sme sa rozhodli pre toto riešenie.
- Implementácia - Krátky popis implementácie. Môže obsahovať aj časti zdrojových kódov.
- Testovanie - Opis ako prebehlo testovanie.

Pre písanie dokumentácie platia nasledovné všeobecné pravidlá:

- Používať pripravenú šablónu a prednastavené štýly.
- Minimalizovať používanie anglických výrazov, okrem prípadov ak je tento výraz všeobecne používaný v tíme. Odporúčané je použiť slovenský výraz a anglický dať do zátvorky vo formáte (angl.).

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 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. Aj to malo za následok nestihnutie všetkých úloh pre daný šprint. Tím sa preto rozhodol takéto veľké úlohy vždy rozbíjať 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 nedostatočne hľadali pomoc u kolegov tímu. Ďalším problémom bolo aj to, že sme pravidelne neaktulizovali 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.

Scrum master sa s pribúdajúcimi skúsenosťami postupne zlepšoval vo vedení tímových stretnutí, motivovaní a organizovaní tímu.

4.4Manažment 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.

Na začiatku sme zápisy so stretnutia vôbec neoverovali, či obsahujú dostatočné informácie a majú správnu štruktúru. Postupne sme zaviedli proces overovanie zápisov zo stretnutia aspoň jedným ďalším členom. Tento proces sme ešte vylepšili tak, že zápis so stretnutia sa zobrazuje pomocou projektoru na plochu tak, aby každý ihneď videl, čo sa zapisuje.

5 Sumarizácie šprintov

Stretnutia pred začatím šprintov:

Zápis zo stretnutia #1

Dátum:	24.09.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Šimko
Zapisovateľ:	Jakub Mačina
Prítomní:	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 zo stretnutia #2

Dátum:	01.10.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Šimko
Zapisovateľ:	Jakub Mačina
Prítomní:	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 zo stretnutia #3

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

Priebeh stretnutia

- Plánovanie šprintu a ohodnocovanie úloh

Zápis zo stretnutia #4

Dátum:	15.10.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Šimko
Zapisovateľ:	Jakub Mačina
Prítomní:	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

Issue Type	Key	Status	Priority	Summary
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
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 - používatelia 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 templatov 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	-> Spúštanie a riadenie strážcov
-> Sub-task	SPINEHERO-48	In Progress	Major	-> Pridanie strážcov.
-> Sub-task	SPINEHERO-49	In Progress	Major	-> Pridanie okien

Zápis zo stretnutia #5

Dátum:	22.10.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

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

Issue Type	Key	Status	Priority	Summary
Task	SPINEHERO-50	Open	Major	Nainstalovat aplikaciu 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
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

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 Gábork: -
- Matej Leško: Spoločné stretnutie s programovaním.
- Jakub Šimko: Zvládli sme drill v Scrume. 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. Navrhnúť 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 Leško– veľká úloha nie je dokončená. Sú spravené okná, nie sú dokončený watchere. Ostatné veci sú z Matejov úlohy spravené.

Jakub Mačina– robil webstránku (skript na automatic deployment). Komunikoval s Intelom. Robil DataSource na novú Intel webkameru. Uňho 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.
Prihlásenie do Startup Awards	Closed. Rozhodli sme sa neísť do súťaže
Notifikácie	Nedokončené. Bol som blokovaný, 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

5.2Šprint 2

Zápis zo stretnutia #6

Dátum:	29.10.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Na začiatku stand-up, kde sme si povedali kto na čom pracoval, čo ho blokovalo a čo sa chystá spraviť.
    - Matej Leško – Vytvoril modul pre štatistiky.
    - Jozef Gáborík – Vytvoril notifikáciu v notifikačnej oblasti.
    - Jozef Staňo – Vytváral testy pre HeadWatcher. Dataprovider na kameru.
    - Jakub Mačina – 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.
- 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 hlbkovych 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 vhodnejsiej 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	Spúštanie a riadenie strážcov
Task	SPINEHERO-50	Open	Major	Nainstalovat aplikaciu 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

Zápis zo stretnutia #7

Dátum:	05.11.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

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 hlbkovych 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 vhodnejsiej 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	Spúštanie a riadenie strážcov
Task	SPINEHERO-50	Open	Major	Nainstalovat aplikaciu 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 úlôh. 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.

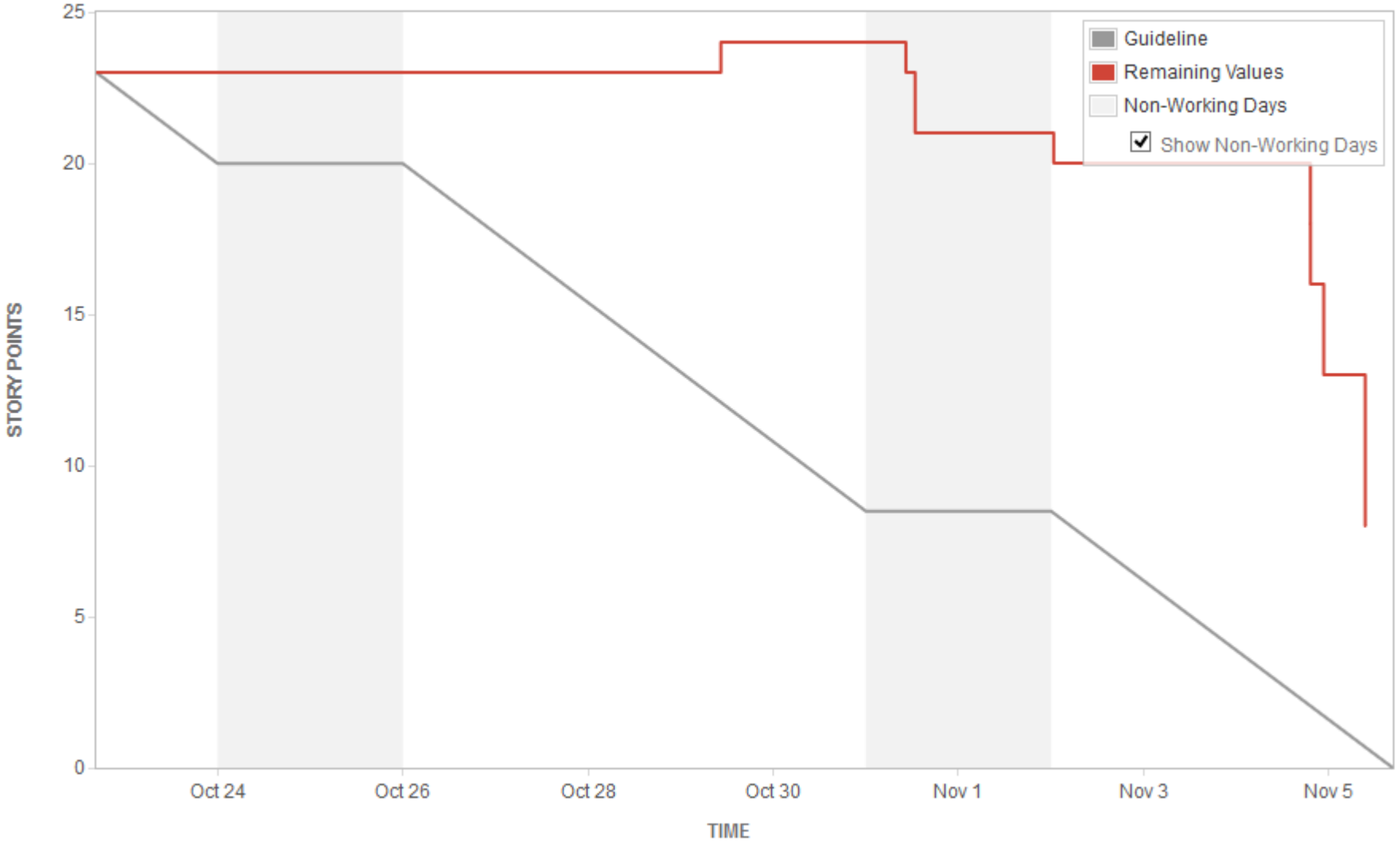
Jakub Mačina: Urobil testovanie kamery Intel F200. Študoval dokumentáciu a ako vytvoriť jej obsah.

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

User stories

Testovanie Intel f200 datasource	Done
Taskbar notifikacia	Done
Popup notifikacia	Done
Implementacia webcam data source nezavislou od runtime	Done
Modul pre statistiky	Done
Spúštanie a riadenie strážcov	Not completed. Chýbalo pokrytie kódu testami.
Planovanie notifikacii	Not completed. Nestihnuté naprogramovanie zobrazenia notifikácii a testy.

Burndown Chart



5.3Šprint 3

Zápis zo stretnutia #8

Dátum:	12.11.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Týždenný stand-up míting:
- Matej Leško: Dokončil spúšťania strážcov. Plánuje písať testy na depth watchera + refaktor + refaktor vecí nad ním.
- Jakub Mačina: 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.

Zápis zo stretnutia #9

Dátum:	19.11.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Jakub Š. navrhol použitie statickej analýzy kódu, napr. Sonar.
- Zaslanie spísanej dokumentácie na e-mailový účet vedúceho.
- Predvedenie výsledku šprintu vedúcemu.
- Pozretie podielu pokrytia kódu testami. Naplánovanie nového šprintu.
- Retrospektíva.
- Plánovanie nového šprintu.

Nové úlohy

- Nájsť nástroj, ktorý lepšie vyhodnotí pokrytie kódu testami.

Retrospektíva šprintu #3 - Škrečok

Trvanie šprintu: 19.11.2015
Zapisovateľ: Jozef Gáborík

S čím sme spokojný

- Jozef Staňo: páčilo sa mu robiť dokumentáciu hneď vtedy, keď sa implementovalo
- Jozef Gáborík: páčilo sa mu, že sme písali testy
- Matej Leško: -
- Jakub Mačina: Veľa sme vyriešili. Páčilo sa mu spoločné stretnutie.

Návrhy čo zlepšiť

- Jozef Staňo: Dokumentácia sa nepísala postupne. Následne sa stratili niektoré dôležité veci (linky na dôležité stránky).
- Zlepšenie k dokumentácie:
    - Inžinierske dielo – Opisuje softvér. Neopisuje šprinty. Má tu byť čo najmenej stôp ako došlo k vytvoreniu softvéru.
        - Funkcionálna dekompozícia vs štrukturálna dekompozícia. Môžeme sa rozhodnúť. Jakub Š. navrhuje funkcionálnu dekompozíciu. Funkcionalitu zoskupovať podľa vecnej podoby, nie podľa času (teda nie podľa šprintov).
        - Štruktuálny pohľad sa nemôže vynechať. Na začiatku opísať architektúru. Opísať moduly – ako v princípe funguje, čo obsahuje.
    - Dokumentácia k riadeniu – Neopisuje softvér. Šprinty opisuje len z pohľadu „overview“. Môžu tam byť názvy user stories a názvy taskov. Uvádzajú sa charakteristiky taskov z pohľadu manažmentu. Napr. diskusia prečo sa nestihol daný šprint.
- Jozef Gáborík: Nepáčilo sa mu písanie dokumentácie. Je toho veľa. Riešenie: Metodiky sa píšu iba raz a iba sa aktualizujú.
- Matej Leško: Technické problémy s Wordom. Nedalo sa tomu predísť.
- Jakub Mačina: Nie je vyladená komunikácia. Málo píšeme na Slacku. Dal pull-request, ale až neskoro mu ho skontrolovali. To ho mohlo zdržať pri napr. následnom opravovaní chýb. Riešenie: Písanie správ do všeobecných kanálov nie do súkromných správ.
- Napísať metodiku ako písať dokumentáciu.
- Pridanie napísania dokumentácie (inžiniersko dielo) k definition of done.

Sprint review

Matej Leško – Mal 2 úlohy. Dokončil 1 úlohu (napísal testy). Dokončil aj druhú úlohu.

Jakub Mačina – Dokončil výber logovacieho frameworku. Logovanie funguje s anotáciou. Zároveň pokryl tieto veci testami.

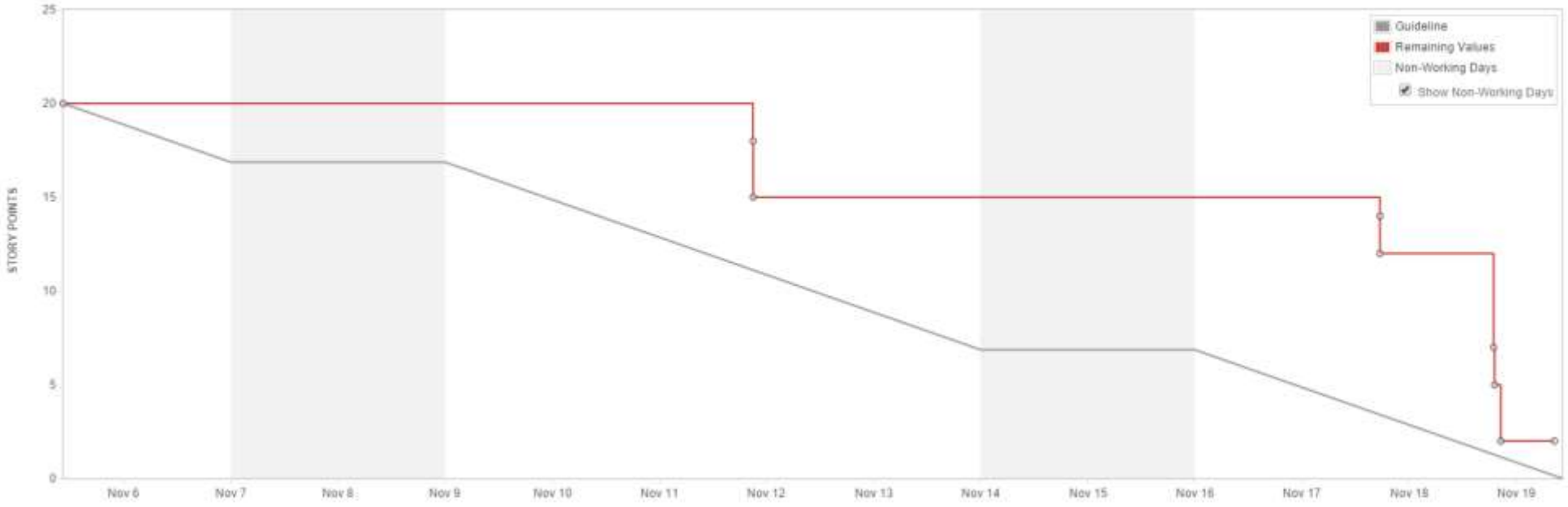
Jozef Staňo – Dokončil vynútenie kalibrácie pri spustení kalibrácie. Správu pamäte pri kalibrácii.

Jozef Gáborík – Dokončil plánovanie notifikácií. Nestihol notifikáciu stmavením obrazovky.

User stories

Logujeme používanie aplikácie	Done
Spúštanie a riadenie strážcov	Done
Planovanie notifikacii	Done
Testovanie a pokrytie testami depth watchera	Done
Vyber logovacieho frameworku	Done
Veľká spotreba RAM pri kalibrácii	Done
Vynútenie vytvorenia kalibrácie pred spustením monitorovania	Done
Zobrazit pouzivatelovi ze aplikacia bola minimalizovana do tray	Not completed. Nikomu nebola pridelená úloha.
Stmavenie obrazovky	Not completed. Málo času kvôli písaniu dokumentácie.

Burndown Chart



5.4Šprint 4

Zápis zo stretnutia #10

Dátum:	26.11.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Týždenný standup:
  - Jakub Mačina: Vytvoril možnosť všeobecného feedbacku. Správa príde na mail, ktorý na to vytvoril. Spravil analýzu odosielania logov na server. Plánuje sa pozrieť na správne sedenie a vytvorenie podstránky.
  - Jozef Gáborík: Vytvoril build-y. Opravil skoré zobrazenie notifikácie. Ďalej ide pracovať na Dim notifikácií a obrazovke o správnom sedení.
  - Jozef Staňo: Robil na balónovom okienku, keď sa aplikácia zminimalizuje. Mal s tým problém, lebo na Windows 10 to nefunguje riešenie od MS. Musí vytvoriť vlastné okno.
  - Matej Leško: Opravoval chyby. Neskorá reakcia tlačidla štart na kliknutie. Dlhá doba analýzy, kvôli logu. Spustenie štart po pauznutí spôsobuje pád aplikácie. Ďalej chce pracovať na debug obrazovke.
- Nasledovalo predvedenie aplikácie.
- Jakub Mačina oboznámil tím s jeho analýzou odosielania logov. Dohodli sme sa, že na logy zatiaľ budeme používať Application Insights, jeho free verziu.
- Rozdelili sme si približne na akých úlohách budeme pracovať ďalší týždeň.
- Jakub Šimko oboznámil tím s jeho výsledkami po používaní aplikácie.
  - Kamera ho nezobrala celého.
  - Bolo to veľmi prísne. Keď sa posunul na stoličke 10cm dozadu v rovnakej polohe, ukazovalo mu to, že sedí zle. Dohodli sme sa, že Matej upraví DepthWatcher.
  - Nedetegovalo ho to, keď pohol vedome s panvou. Keď to urobil
  - Nepáčil sa mu kĺzavý priemer na grage v Dashboard. Dohodli sme sa, že ho odstránime.
  - Notifikácia cez popup bola veľmi rušivá.
- Problém nesprávnej detekcie, keď si sadnem správne, ale do inej polohy ako pri kalibrácii. Návrh: ukladať si viacero kalibrácií.
- Návrh: Pri open beta zobrať fotky a vylepšiť celkový model. Aby sme vylepšili robustnosť vzhľadom na prostredie. Stačilo by iba vytiahnuť nejaké features.

Nové úlohy

- Porozmýšľať ako vytvoriť iné stavy ako len dobre alebo zle. Keď ešte nesedím veľmi zle, tak ho chvíľu nechám nech si oddýchne. Až potom ho upozorním.

Zápis zo stretnutia #11

Dátum:	03.12.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Sprint review.
- Jakub Mačina oboznámil všetkých s výsledkom analýzy CI.
- Diskusia o spolupráci s EUBA.
- Plánovanie ďalšieho šprintu.

Retrospektíva šprintu #4 – Morské prasiatko

Trvanie šprintu: 03.12.2015
Zapisovateľ: Jozef Gáborík

S čím sme spokojní

- Správne rozdelenie úloh (Jakub pomáhal s angličtinou ako najlepší angličtinár).
- Dobrá komunikácia (osobne, Slack).
- Dobre sme dekomponovali úlohy.

Návrhy čo zlepšiť

- Nedávať si úlohy za 0 story points. Riešenie: Minimálna hodnota 0.5 story point.
- Nedodržiavanie metodík – iný spôsob logovania (Jozef Gáborík).

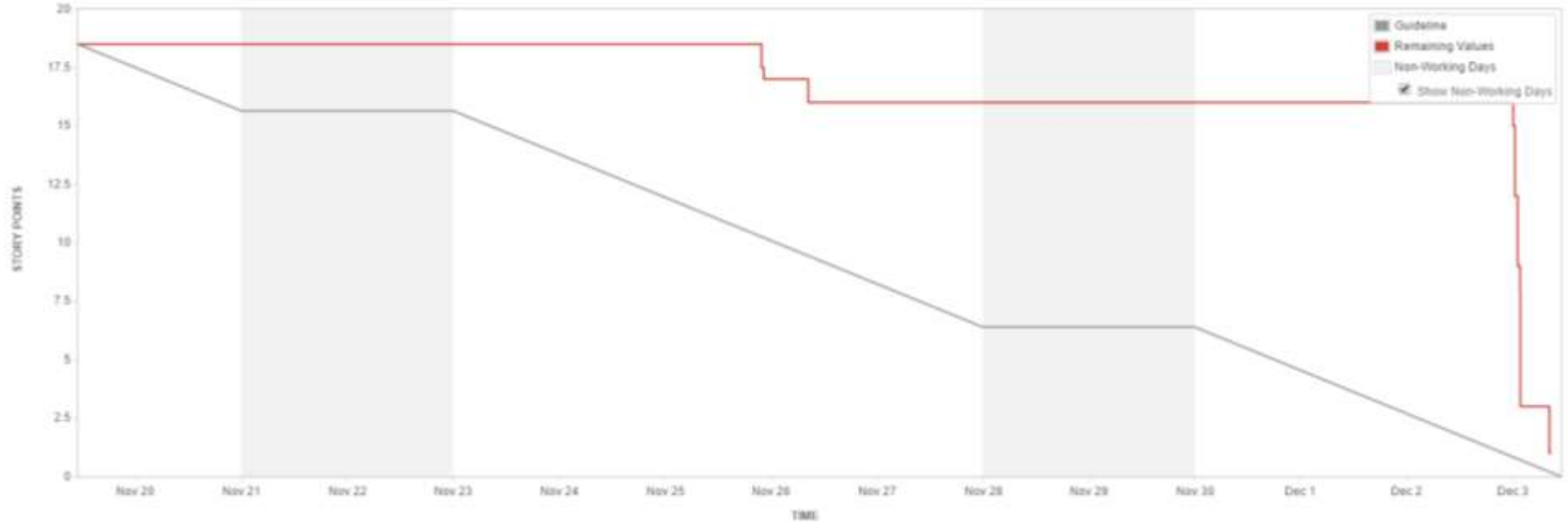
Sprint review

- Jozef Gáborík: Pracoval na Dim notifikácii, opravil časovanie notifikácií, pracoval na obrazovke o správnom sedení. Následne by chcel pracovať na vylepšení dim notifikácie.
- Jozef Staňo: Spravil report zlého sedenia. Pracoval na okne, ktoré upozorní na minimalizovanie do notifikačnej oblasti (nedokončil).
- Matej Leško: Logovanie unhandled exception. Pracoval na debug screen.
- Jakub Mačina: Web stránka o správnom sedení. Spravil taktiež analýzu CI (continuous integration).

User stories

Zobrazit pouzivatelovi ze aplikacia bola minimalizovana do tray	Nedokončené
Analýza CI	Done
Používateľ pošle všeobecný feedback	Done
Možnosť zareportovať nesprávne meranie	Done
Stmavenie obrazovky	Done
Obrazovka ako sediet spravne	Done
Logovanie ak padne aplikacia	
Aplikácia padá ak je dlho pozastavené monitorovanie	Done
Tlačidlo na zastavenie monitorovania reaguje oneskorene – Done.	Not completed. Málo času kvôli písaniu dokumentácie.
Debug screen	Done
Použitie SittingQualityAveraged	Done
Popup sa zobrazuje okamžite nie po časovom limite	Done
Analýza odosielania logov na server	Done
Nasadiť Jakubovi Š. Aplikáciu	Done
Po zastavení monitorovania sa Dashboard nevráti do pôvodného stavu	Done

Burndown Chart



5.5Šprint 5

Zápis zo stretnutia #12

Dátum:	10.12.2015
Miestnosť:	softvérové štúdio
Vedúci stretnutia:	Jakub Mačina
Zapisovateľ:	Jozef Gáborík
Prítomní:	Jakub Šimko
	Jozef Gáborík
	Matej Leško
	Jozef Staňo
	Jakub Mačina

Priebeh stretnutia

- Stand-up míting:
  - Jakub Mačina: Práca na dokumentácii, vrátane tvorby diagramu tried. Spisovanie úloh do Trella. Analýza Azure.
  - Jozef Staňo: Práca na dokumentácii.
  - Jozef Gáborík: Práca na dokumentácii. Tlačenie vizitiek a roll-upu.
  - Matej Leško: Práca na dokumentácii. Pridanie ikony aplikácie.
- Spísanie objavených chýb.
- Architektúra musí obsahovať:
  - Definovať moduly a opísať čo v nich je.
  - Ak 2 moduly komunikujú, musí tam byť napísané API. Teda kto poskytuje aké rozhranie (metódy). Iný spôsob je očíslovať vzťahy a pod diagramom napísať rozhranie, ktoré vyjadruje daný vzťah.
- Dokumentácia
  - Aplikácie manažmentov – diskusia čo sme robili, či to bolo dobré alebo nie. Čo z toho vzišlo.
  - Plán – urobiť ako príloha. Prílohy: backlog, výpis úloh (export úloh).
  - Globálna retrospektíva – hlavným zdrojom sú predchádzajúce retrospektívy.
- Objednanie servera na BizPark.
- TODO: Aktulizovať Definiton of Done.
  - „Úloha je dokončená, keď kód má napísané testy, prejde cez code review, je k nemu napísaná dokumentácia a je vo vetve *development*.“

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

Pre zlepšenie práce na projekte bolo pre nás dôležité si dohodnúť pravidlá, kedy je daná úloha dokončená. K tomu nás povzbudzoval aj samotný SCRUM. Z toho dôvodu sme si dohodli Definition of Done. Tá sa vyvíjala, ako sme sa postupne dohodli na písaní testov a postupnom spisovaní dokumentácie.

Pri plánovaní sme spočiatku pridávali do šprintu aj úlohy, ktoré boli veľké (mali veľa story points). Výsledkom toho bolo, že sme ich častokrát nestihli dokončiť v danom šprinte. Aktuálne je snaha v rámci tímu rozdeľovať všetky úlohy, ktoré majú viac ako 5 story points.

Komunikačné kanály boli už v našom tíme na začiatku nastavené a používané. No počas prvých šprintov sme identifikovali, že počas riešenia úloh málo navzájom komunikujeme. Tento problém sa nám podarilo zmierniť tým, že si každý člen tímu nainštaloval aplikáciu Slack aby sa zrýchlila komunikácia. Naviac sme začali využívať aj anotácie pre označenie ľudí, ktorí sa majú vyjadriť. Taktiež sme si vždy naplánovali aj ďalšie aspoň jedno stretnutie mimo oficiálnych stretnutí v škole, aby sme mohli osobne riešiť úlohy.

Ďalším problémom v úvodných šprintoch bolo nedostatočné využívanie nástroja JIRA na aktualizáciu stavu úloh. Po identifikácií tohto problému, začali sme viac dbať dôraz na aktualizáciu úloh a úlohou scrum mastera bolo dohliadať na správne používanie nástroja JIRA.

Počas tohto obdobia sme tiež odstránili predošlý proces overovanie zápisov so stretnutí. Zápis so stretnutia sa premieta a každý člen tak ihneď vidí zapísaný text, na ktorý môže reagovať.

Náš tím sa zlepšil v testovaní aplikácie. V prvom šprinte sme nemali napísaný žiaden test. Pridali sme do definition of done písanie testov a postupne sme sa zlepšili v kvalite testov a pokrytí zdrojového kódu testami.

Pri analýze a výbere úloh sme na začiatku semestra nebrali ohľad na procesy, ktoré na seba nadväzovali. Čo spôsobovalo, že danú úlohu nebolo možné začať riešiť keďže úloha, ktorá ju blokovala ešte nebola vyriešená. Preto sme pri plánovaní začali s identifikáciou úloh, ktoré

blokujú iné úlohy v danom šprinte. Tieto úlohy majú v danom šprinte zvýšenú prioritu a sú prednostne riešené.

Počas viacerých šprintov sa v backlogu objavili úlohy ktoré sme ohodnotili na nula story points. Avšak pri viacerých takýchto nulových úlohách v jednom šprinte to značne ovplyvnilo výkonnosť tímu. Keďže na splnenie všetkých aj krátkych úloh je potrebné vynaloženie určitého úsilia rozhodli sme sa aj malým úlohám priradiť pol story points.

7 Príloha - 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.

7.1 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.

7.2 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.

7.3Fáza 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.

7.4Nasledujúce fázy

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

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dokumentácia k inžinierskemu dielu

Autori: Jakub Mačina, Jozef Staňo, Matej Leško, Jozef Gáborík

Predmet: Tímový projekt

Akademický rok: 2015/2016

Dátum odovzdania: 14. 12. 2015

8 Úvod

Našim 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.

9 Ciele pre zimný semester

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

Medzi hlavnú funkcionalitu 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 Senz3D

Na validáciu základnej funkcionality aplikácie a analýzu jej používania plánujeme naviac 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.

10 Celkový pohľad na systém

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

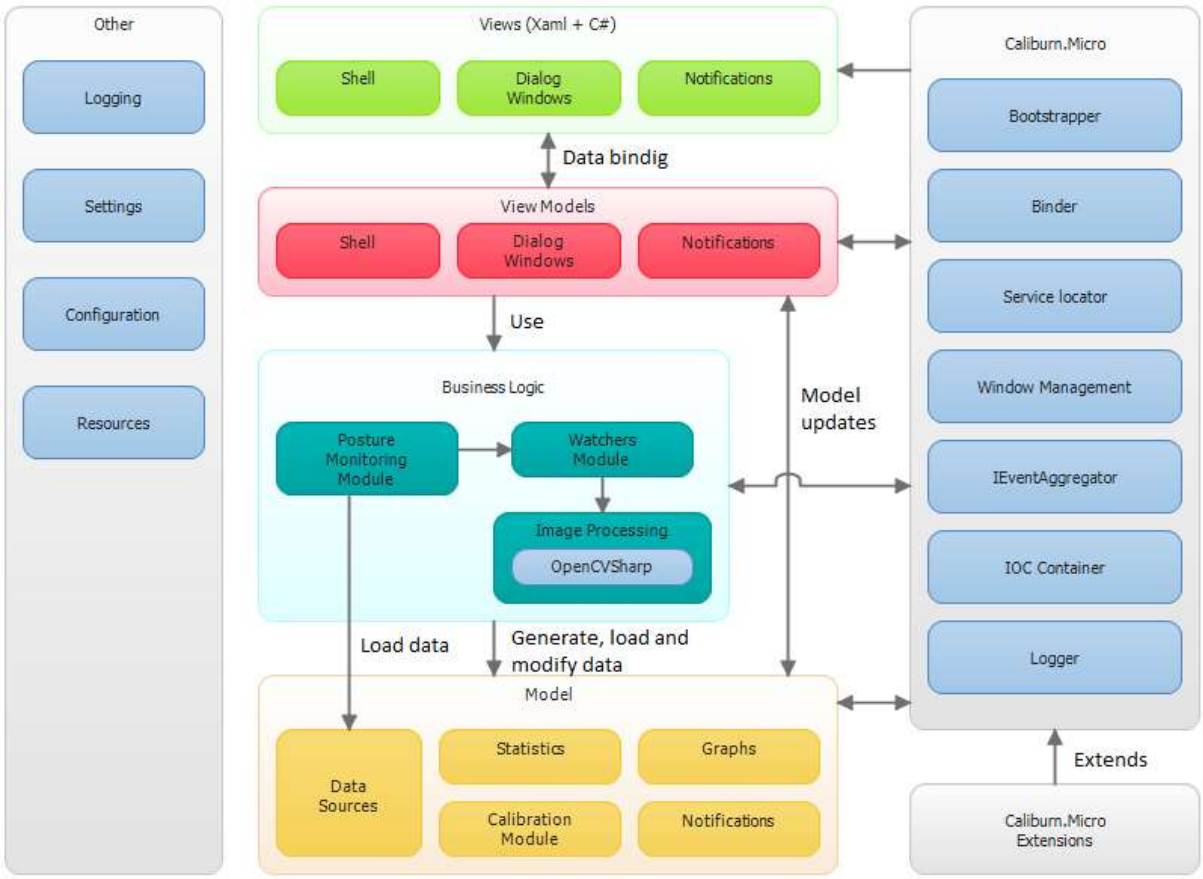
Implementovali sme najdôležitejšie časti aplikácie a projekt sa posúva do fázy testovania aplikácie vybranými používateľmi, ktorý budú aplikáciu dlhodobejšie používať.

Kompletná funkcionalita aplikácie:

- Získavanie dát z kamery - aplikácia podporuje všetky farebné kamery a vybrané hĺbkové kamery (Intel Realsense Senz3d, Intel Realsense F200) a automaticky zvolí najlepšiu dostupnú kameru.
- Vyhodnocovanie kvality držania tela - implementované sú dve metódy vyhodnocovania kvality držania tela z jednotlivých snímkou. 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. Implementované je tiež upozornenie stmavením obrazovky.
- Nastavenia - aplikácia umožňuje nastaviť dobu po uplynutí ktorej sa zobrazí príslušná notifikácia a či sa má používať aj hĺbková kamera.
- Odoslanie všeobecnej spätnej väzby - aplikácia umožňuje používateľovi odoslať spätnú väzbu na jej používanie.
- Reportovanie nesprávneho merania - v prípade, ak používateľ nie je spokojný s aktuálnym meraním kvality sedenia, môže stlačením tlačidla zareportovať autorom tento problém a pomôcť im tak vylepšiť detekciu kvality sedenia.

10.1 Architektúra

Spine hero je klasická desktopová aplikácia a hlavným architektonickým štýlom aplikácie je Model-View-ViewModel (MVVM). Schéma architektúry je zobrazená na Obrázok 3 Schéma architektúry.



Obrázok 3 Schéma architektúry

10.1.1 Views

Predstavujú grafické používateľské rozhranie. Sú úzko previazané s ViewModelmy pomoucou „bindovania“ dát, pričom ich vzťah je zväčša 1 ku 1 alebo N ku 1 v prípade, že pre jeden ViewModel máme viacero Views. Views môžu obsahovať code behind, ale ten je určený len pre veci, ktoré súvisia priamo s daným view (napríklad pre okná bez okraju je potrebné nastaviť to, aby bolo možné presúvať toto okno pri podržaní ľavého tlačidla myši.)

10.1.2 ViewModels

Obsahujú dáta a logiku vykonávania príkazov vykonaných nad používateľským rozhraním a notifikujú prislúchajúci view o zmene dát. Prepojenie View s ViewModelom a jednotlivé bindingy dát a funkcií zabezpečuje kontrolná jednotka, ktorou je v našej aplikácii

Caliburn.Micro. Triedy ViewModelov by mali rozširovať, ale nemusia, niektorú zo základných tried Caliburn.Micro určených pre ViewModely. *Screen* predstavuje základný prvok, ktorý v zásade neobsahuje ďalšie prvky. *Conductor* je prvok, ktorý slúži na združovanie ďalších *Conductors* a *Screens* a umožňuje viacero stratégií manažovania životného cyklu týchto potomkov (Jeden/Všetky prvky aktívne).

10.1.3 Caliburn.Micro[4]

Je aplikačný rámec v ktorom beží aplikácia. Zjednodušuje prácu s MVVM a zabezpečuje:

1. **Beh a inicializáciu aplikácie** – Rozšírenie triedy *BootstrapperBase* je prvým krokom. V tejto triede sa inicializuje IOC kontajner a iné komponenty potrebné pre štart aplikácie. Taktiež sa nastavuje funkcionalita pri štarte a ukončení aplikácie a spôsob narábania s neošetrenými výnimkami.
2. **Prepájanie View s ViewModelom -** Sú dva spôsoby riadenia prepájania:
    a. ViewModel to View – K dostupnému ViewModelu, ktorý je inicializovaný sa nájde na základe konvencií prislúchajúci View.
    b. View to ViewModel – Nápodobne ako predchádzajúci spôsob len opačným smerom.

    V našej aplikácii sa používa prvý prístup.
3. **Data binding** – Na základe konvencií sa prepájajú dáta a funkcie ViewModelu s dátami a udalosťami z View.
4. **Riadenie zobrazovania okien** – Na zobrazovanie okien sa používa *IWindowManager*.
5. **Komunikáciu modulov** – *IEventAggregator* je publish/subscribe systém a slúži na medzi-modulovú komunikáciu. Namiesto priameho naviazania modulov sú moduly závislé na *IEventAggregator*. Pre príjem publikovanej udalosti je potrebné sa prihlásiť sa na odber a trieda musí taktiež realizovať rozhranie *IHandle*<*T*>, pričom *T* predstavuje typ správy, ktorý sa má prijať a spracovať.
6. **Inverzia kontroly (angl. Inversion of control, skr. IOC)** – Caliburn.Micro obsahuje kontajner pre vkladanie závislostí (angl. Dependency Injection) *SimpleContainer*, ktorého funkcionalita postačuje pre účely projektu a preto nebol nahradený externým IOC kontajnerom. Tento kontajner zabezpečuje registráciu a vkladanie závislostí pre triedy a tým zjednodušuje manažovanie závislostí aplikácie.

[4] Caliburn.Micro: http://caliburnmicro.com/

7. **Logovanie** – Caliburn.Micro umožňuje prepojiť jeho logovací systém na logovací systém, ktorý sa používa v aplikácii a tým sú dostupné záznamy činností, ktoré vykonáva tento aplikačný rámec.
8. **Rozšírenia Caliburn.Micro** – Z dostupného API Calibrurn.Micro je jednoduché rozšíriť jeho funkcionalitu. Momentálne jediným rozšírením je pridanie podpory na parsovanie a prepájanie klávesových skratiek a gest z View na ViewModel, vzhľadom na to, že základná funkcionalita Caliburn.Micro toto neobsahuje.

10.1.4 Biznis logika

Predstavuje hlavnú funkcionalitu aplikácie, ktorá sa riadi z jednotlivých ViewModelov a obsahuje nasledovné moduly:

1. **Modul pre analýzu kvality sedenia** – Tento modul komunikuje priamo s modulmi:
    a. Modulom pre získavanie dát - Využíva funkcie rozhrania *IDataSourceManager Start/Stop* pre spustenie, resp. zastavenie prislúchajúceho zdroju dát a funkciu *LoadNext*, ktorá vráti ďalší načítaný obrázok zo zdroju dát (Webová a/alebo hĺbková kamera).
    b. Modul pre vyhodnocovanie kvality sedenia používateľa – Používa sa funkcia *AnalyzeInWatchers*, ktorá spustí analýzu u strážcov, rozhrania *IWatcherManager*.
    c. Spracovanie obrazu.
2. **Modul pre vyhodnocovanie kvality sedenia používateľa** – Využíva funkcionality zo spracovania obrazu na analýzu a vyhodnotenie kvality sedenia používateľa zo získaných dát. Jeho súčasťou sú jednotlivé algoritmy pre analýzu farebného, resp. hĺbkového obrazu.
3. **Spracovanie obrazu** – Na spracovanie obrazu sa používa priamo knižnica *OpenCVSharp*[5] alebo rozšírená časť spojenej, často používanej funkcionality prostredníctvom návrhového vzoru fasáda.

10.1.5 Model

Predstavuje vrstvu prístupu k dátam a samotnú reprezentáciu dát. Vo väčšine prípadov model prijíma zmeny dát nepriamo prostredníctvom udalostí prijatých z *IEventAggregator*. Priamy prístup sa využíva iba pri načítavaní stavu a dát.

1. **Zdroj dát** – Zdrojom dát sú webová kamery a/alebo hĺbková kamera. Vzhľadom na to, že rôzne druhy hĺbkových kamier majú rôznu funkcionalitu z ich dostupných SDK bolo

[5] OpenCVSharp: https://github.com/shimat/opencvsharp

potrebné spraviť spoločné prepojenie funkcionalít pomocou proxy. Poskytuje služby pre zapnutie a vypnutie zdroju dát a načítanie obrázkov z tohto zdroja.

2. **Štatistický modul** – Predstavuje repozitár histórie dát kvality sedenia používateľa v rôznych formách. Používa sa vo ViewModeloch, ktoré zobrazujú tieto dáta.
3. **Dáta grafov** – Obsahuje dáta potrebné pre správne zobrazenie grafov. Je doplnkom štatistického modulu v prípade, že je potrebné reprezentovať dáta v inej forme. Tak ako pri štatistickom module sa používa vo ViewModeloch.
4. **Modul kalibrácie** – Slúži na načítavanie a ukladanie kalibrácie. Táto činnosť sa vykonáva nepriamo prostredníctvom udalostí. Priamy prístup je len ku aktuálnej kalibrácii.
5. **Modul notifikácií** – Tento modul má na starosti riadenie a správu notifikácií. Komunikačnou časťou tohto modulu je *EventProcessor* a ako už z názvu vyplýva tento modul komunikuje len prostredníctvom udalostí z *IEventAggregator* a notifikácie zobrazuje prostredníctvom *IWindowManager*.

10.1.6 Iné zdroje

1. **Logovanie** - V celej aplikácii sa používa logovanie. Momentálne sa v aplikácii používa logovací rámec NLog. Zmena rámca neovplyvní časti kódu v ktorom sa používa logovanie vzhľadom na použitie vlastného adaptéra pre logovanie, ktorý už obsahuje logiku použitia príslušného logovacieho rámca.
2. **Nastavenia a konfigurácia** – Aplikácia využíva základné spôsoby pre správu nastavení a konfigurácií dostupných v aplikačnom rámci .NET.
3. **Zdroje –** Obrázky, ikony a iné zdroje dát potrebné pre beh aplikácie. Načítavajú sa priamo prostredníctvom funkcií dostupných z .NET rámca alebo sa používa statická trieda *ResourceHelper*.

10.2 Použité knižnice a frameworky

- Postsharp - framework, ktorý sa používa na podporu AOP.
- NLog - knižnica na logovanie.
- Caliburn.Micro – framework pre návrh a tvorbu aplikácii. 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.

10.3 Dátový model

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

10.4 Moduly

10.4.1 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.

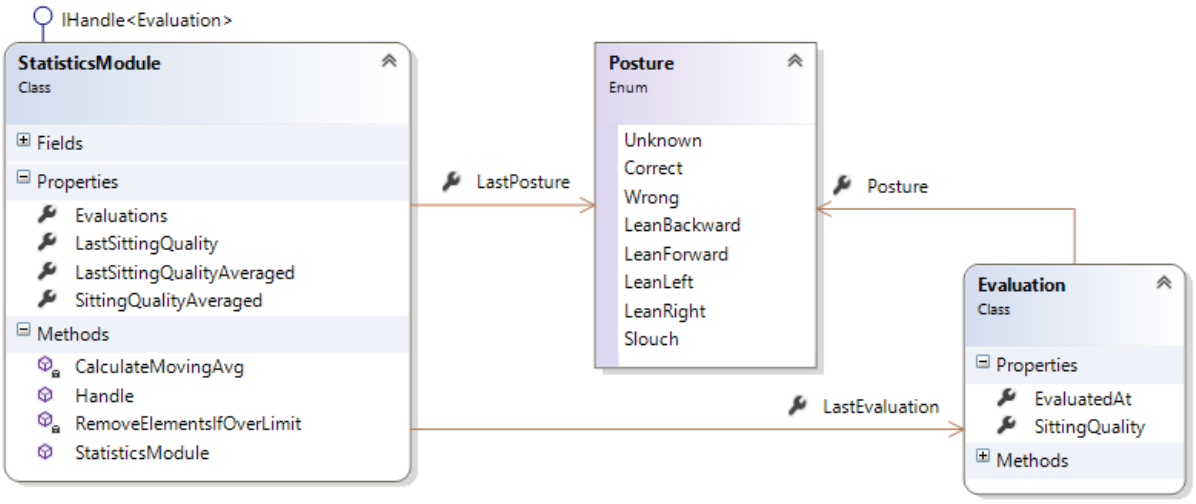
Návrh

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 kĺzavý priemer, ktorý sa tiež bude uchovávať. Na to aby sa zbytočne nepridávali evaluácie donekonečna sa budú tieto dva zoznamy priebežne mazať.

Implementácia

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

- **Evaluations** - zoznam prijatých evaluácií sedenia. Evaluácie obshujú 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 4 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. Kĺzavý priemer sa počíta z piatich posledných hodnôt kvality sedenia. Tieto nastavenia je možné zmeniť v *Const.Settings*. Kĺzavý 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;
```

10.4.2 Modul notifikácií

10.4.2.1 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 notifikácii 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.

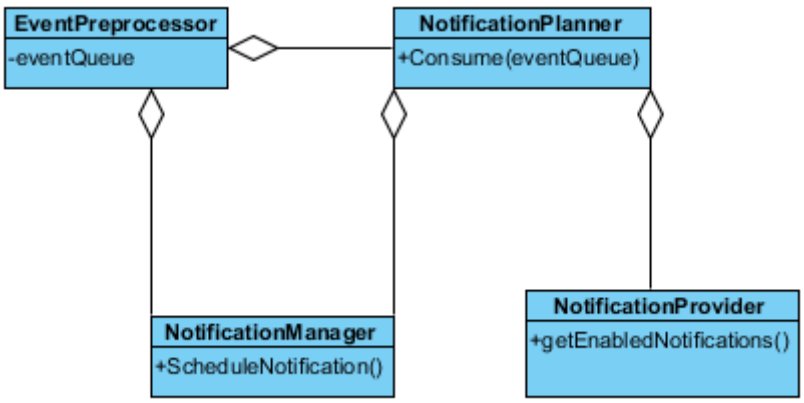
Modul reaguje na zmenu nastavení v aplikácii. Modul postupne vyberá notifikácie podľa toho, ako boli nastavené ich časové limity v nastaveniach.

10.4.2.2 Návrh

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

Modul na plánovanie notifikácii 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 EventPrepocessorom. Obsahuje logiku výberu správnej notifikácie, zároveň aj logiku ohľadom skrytia notifikácii.
- NotificationManager - Má na starosti samotné zobrazenie a skrytie notifikácii. Uchováva naplánované notifikácie. Má informáciu, ktorá notifikácia je aktuálne zobrazená.



10.4.2.3 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.

10.4.3 Modul pre analýzu a vyhodnocovanie kvality sedenia

10.4.3.1 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 snímkov v pravidelných časových intervaloch.

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

10.4.3.2 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. pokým sa neukončí monitorovanie.

10.4.3.3 Implementácia

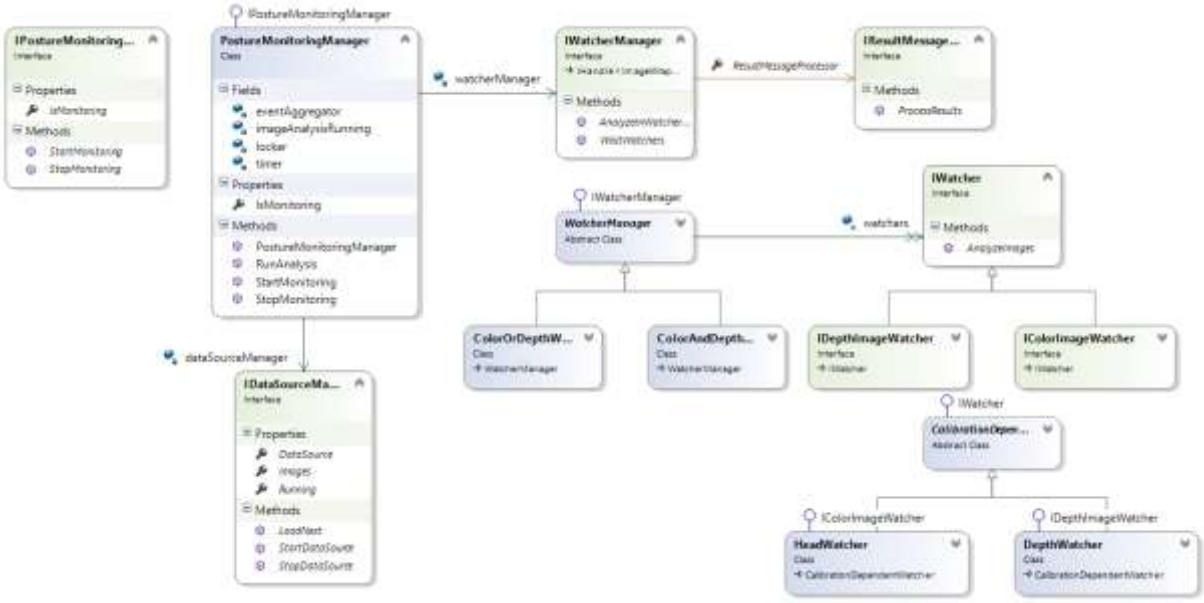
Hlavnou riadiacou jednotkou tohto modulu je *PostureMonitoringManager*. Táto trieda realizuje rozhranie *IPostureMonitoringManager*, ktorej úlohou je poskytovanie možnosti pre zapínanie a vypínanie analýzy toho ako používateľ sedí za počítačom. Táto trieda má tieto dve hlavné závislosti:

- **IWatcherManager** – má na starosti spúšťanie a riadenie strážcov a vrátenie výslednej evaluácie. Viac informácii 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 pomocu metódy *LoadNext()*. Ďalšie informácie sú v kapitole 11.2.4 a **Chyba! Nenašiel sa žiaden zdroj odkazov.**

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 5 Diagram tried modulu pre analýzu a vyhodnocovanie kvality sedenia

10.4.4 Modul pre získavanie dát z kamier

10.4.4.1 Analýza

Našim 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.

10.4.4.2 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.

10.4.4.3 Implementácia a testovanie

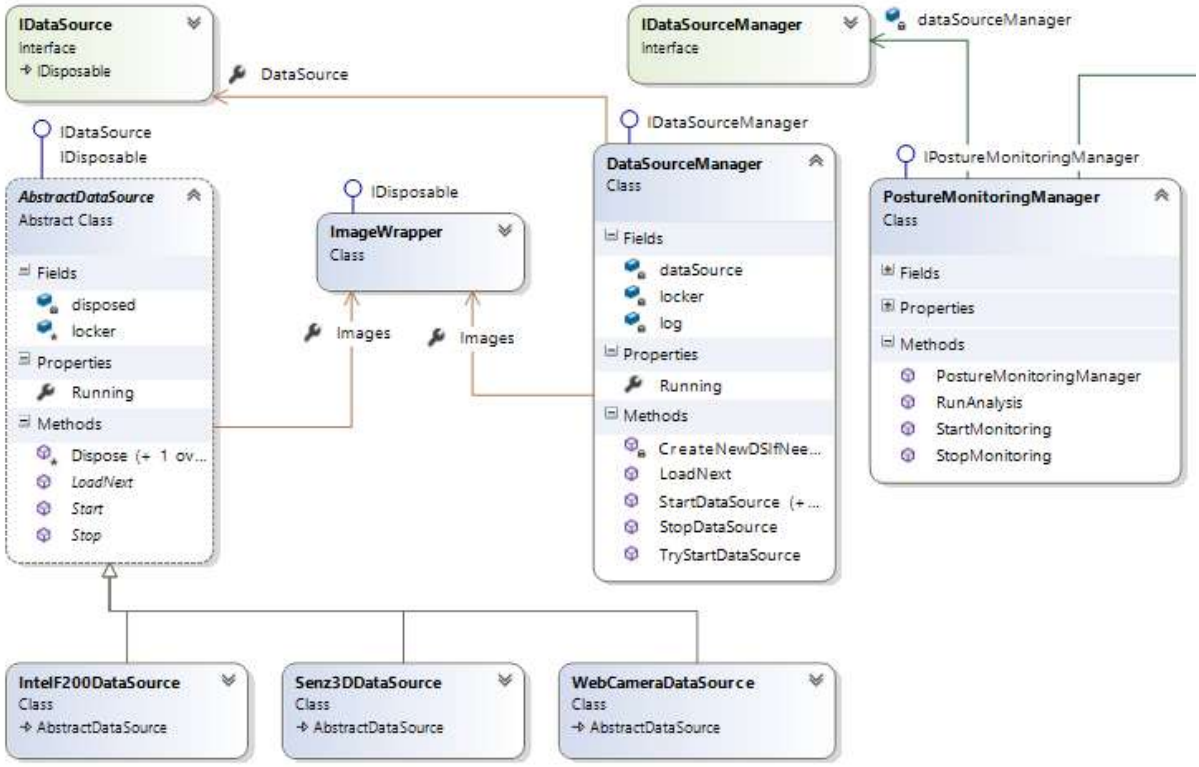
Implementovali sme rozhranie *IDataSouce* pre získavanie dát z kamier. Abstraktná trieda *AbstractDataSource* pridáva metódy na prácu s nemanažovaným kódom. Trieda *ImageWrapper* poskytuje pomocné triedy na spracovanie snímkou a konverziu snímkov do jednotného formátu *OpenCvSharp.Mat*.

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

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

DataSourceManager.

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 6 Diagram tried modulu pre získavanie dát z kamier

10.4.5 Modul pre kalibráciu

10.4.5.1 Analýza

Modul má zabezpečiť kalibráciu aplikácie. Zobraziť 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.

10.4.5.2 Návrh

Vytvoriť 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 získavame z *DataSouceManager* čím zabezpečujeme nezávislosť od dátového zdroja.

10.4.5.3 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 DataSourceManager. 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.

11 Implementované user stories počas šprintov

11.1 Šprint 1 - Piskor

11.1.1 Presunutie funkcionality 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 funkcionalita bola presunutá do nového projektu z prototypu:

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

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

11.1.2 Spustenie aplikácie pri štarte PC

11.1.2.1 Analýza

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

11.1.2.2 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.

11.1.2.3 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 zaškrtávacie políčko v časti nastavenia.

11.2 Šprint 2 - Myška

11.2.1 Automaticky vyber data providera podľa pripojených kamier

11.2.1.1 Analýza

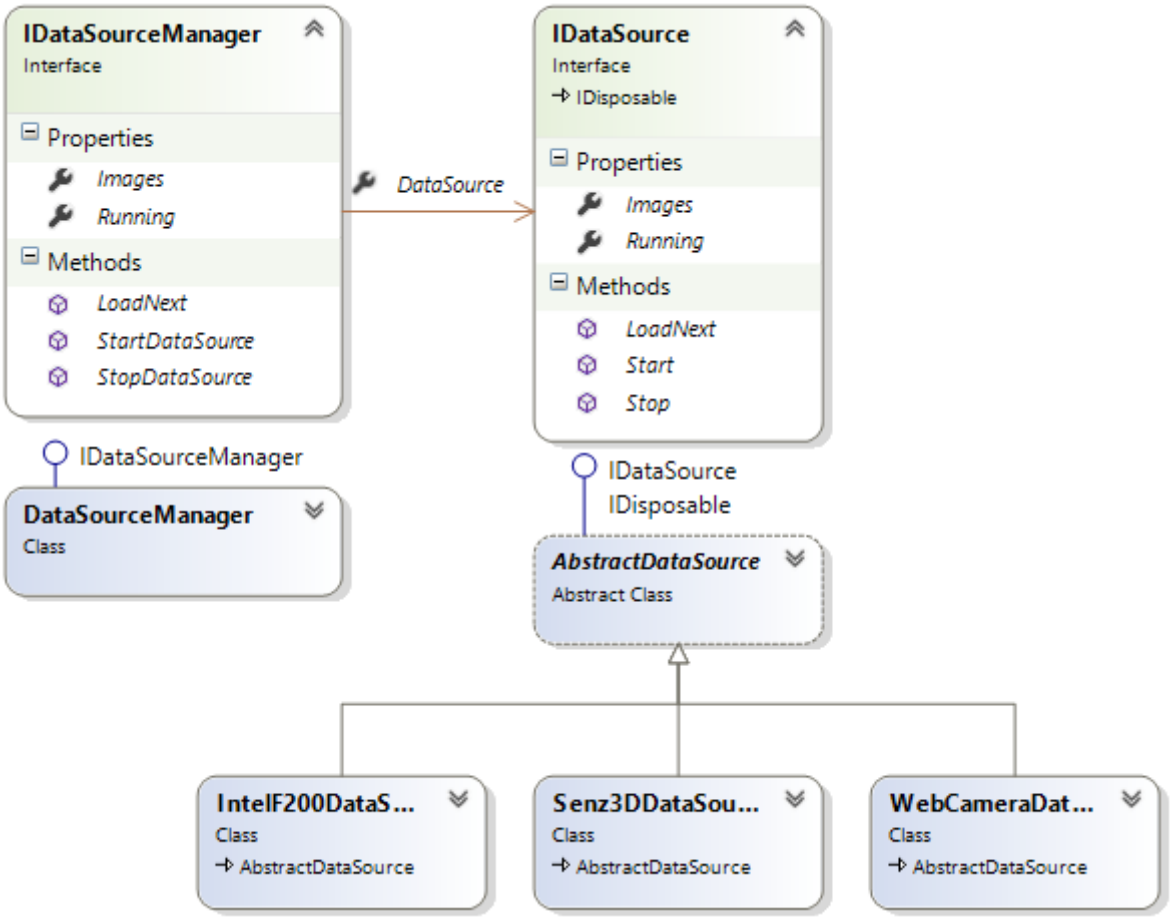
Táto funkcionalita slúži na to aby sme používateľa odbremenili od činnosti výberu kamery, ktorá je použitá ako zdroj dát. Používateľ si bude vyberať len to či chce použiť hĺbkovú kameru ak je nejaká pripojená alebo nie.

11.2.1.2 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.

11.2.1.3 Implementácia

Na implementáciu manažéra pre DataSource bol použitý návrhový vzor Bridge.



Obrázok 7 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 zaškrtávacie políčko, ktoré určuje či sa má používať hĺbková kamera ak je k dispozícii.

11.2.2 Taskbar notifikácia

11.2.2.1 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[6], 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.

11.2.2.2 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.

11.2.2.3 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 udalost (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.

[6] https://bitbucket.org/hardcodet/notifyicon-wpf

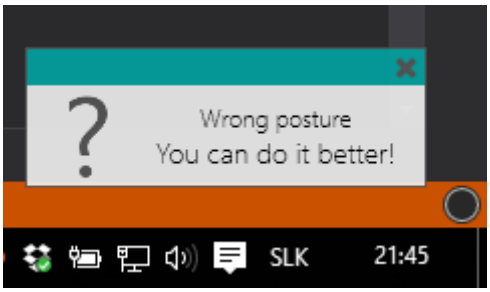
11.2.3 Popup notifikácia

11.2.3.1 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.

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



Obrázok 8 Pop-up notifikácia

11.2.3.3 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 notifikácii. To znamená, že sama o sebe nezachytáva udalosť novej evaluácie. Spracováva

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

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

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

iba udalosť zmeny polohy. Vtedy mení obrázok znázorňujúci používateľovu polohu. Na to sa zameralo aj unit testovanie. Ukážka časti testu:

11.2.4 Podpora kamery Intel F200

11.2.4.1 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.

11.2.4.2 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.

11.2.4.3 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 nemanažovanú 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.

11.2.5 Implementácia podpory webkamier nezávislá od intel runtime

11.2.5.1 Analýza

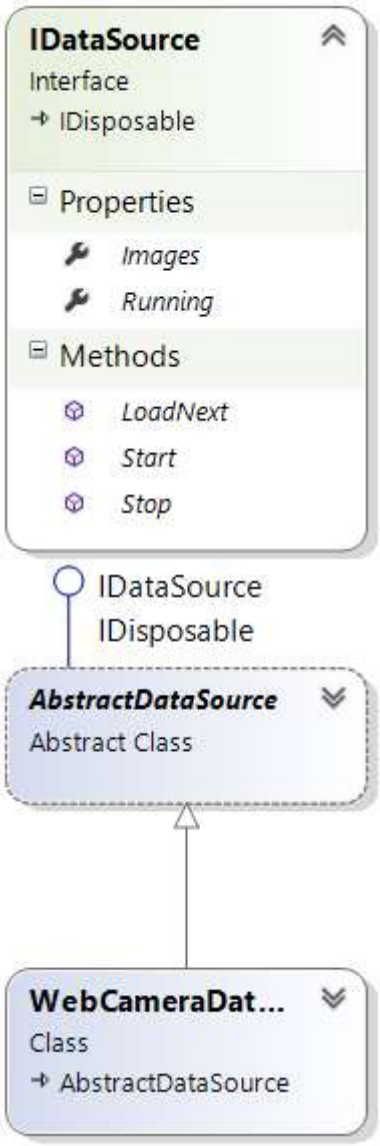
V doterajšej implementácií využívame intel SDK Runtime Distributable na získavanie obrazu z web kamery. Ten však je príliš veľký (558MB) a je nutné ho dodatočne inštalovať. Cieľom je implementovať získavanie údajov za pomoci menších a bežne dostupných knižníc.

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

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

11.2.5.2 Návrh

Funkcionalitu budeme implementovať v triede WebCameraDataSource.



Obrázok 9 WebCameraDataSource

11.2.5.3 Implementácia

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

Pri implementácií 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);
```

11.3 Šprint 3 - Škrečok

11.3.1 Plánovanie notifikácií

11.3.1.1 Analýza

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

Pri module na plánovanie notifikácii 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 nastaví v aplikácii. Modul postupne vyberá notifikácie podľa toho, ako boli nastavené ich časové limity v nastaveniach.

11.3.1.2 Návrh

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

Modul na plánovanie notifikácii sa skladá z týchto hlavných častí:

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

11.3.1.3 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.

11.3.2 Logujeme používanie aplikácie

11.3.2.1 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. Naviac, logy vygenerované pri používaní aplikácie chceme použiť na analýzu používania aplikácie používateľmi.

11.3.2.2 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

11.3.2.3 Návrh

Rozhodli sme sa použiť logovací framework nlog, pretože je aktívne vyvíjaný, 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 neriešie 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.

11.3.2.4 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)
{
[LogMethodCall]
public bool TryStartDataSource(IDataSource dataSource)
{
    ...
}
```

Použitie anotácie pre logovanie

11.3.3 Spúšťanie a riadenie strážcov

11.3.3.1 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.

11.3.3.2 Návrh

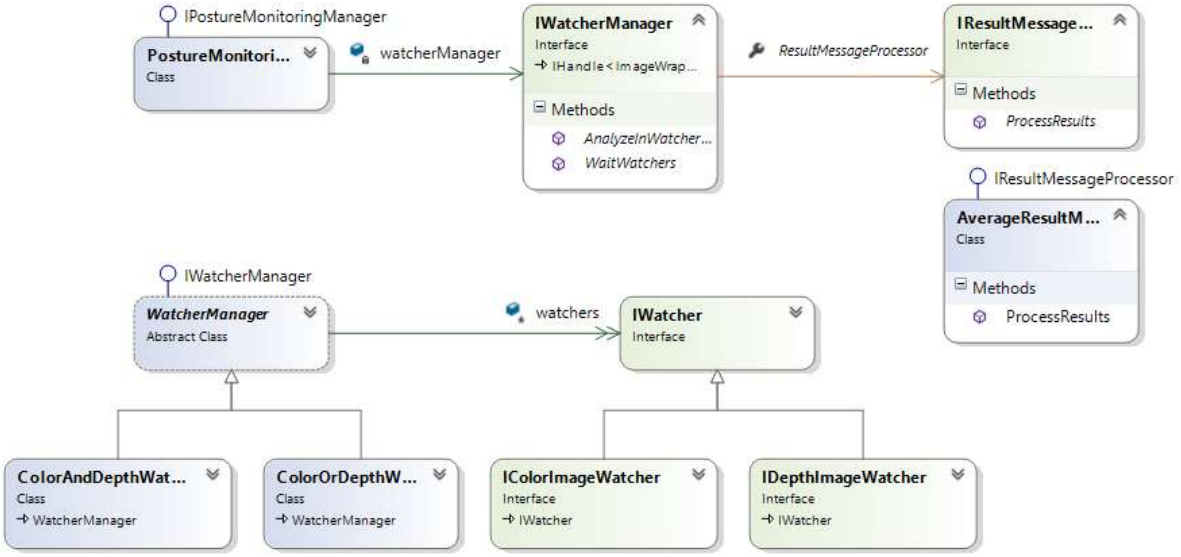
Analýza jednotlivých strážcov sa bude vykonávať paralelne vo vlastnom vlákne. Analýza v strážcoch sa spustí na základe toho či sú k dispozícii hĺbkový a farebný obraz. Na výpočet celkového sedenia používateľa sa výsledky z jednotlivých strážcov spriemerujú.

11.3.3.3 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 10 Diagram tried pre riadenie a spúšťanie strážcov

11.4 Šprint 4 – Morské prasiatko

11.4.1 Stmavenie obrazovky

11.4.1.1 Analýza

Stmavenie obrazovky je jeden zo spôsobov notifikácie v našej aplikácii. Cieľom je, aby táto notifikácia bola čo najmenej rušivá. Po zobrazení tejto notifikácie má obrazovka počíta nabrať efekt, akoby sa postupne stmavovala od jej rohov.

Požiadavky na notifikáciu:

- Zobrazuje sa cez celú obrazovku.
- Okno notifikácie je vždy navrchu. Teda notifikácia sa neskryje po prekliknutí na iné okno.
- Je možné zrušiť notifikáciu pomocou klávesovej skratky.
- Napriek tomu, že okno je vždy navrchu, je možné ovládať aplikácie pod ním.

11.4.1.2 Návrh

Na zabezpečenie všetkej potrebnej funkcionality je potrebné komunikovať s Windows API (skrátene WinAPI). Konkrétne sa využívajú funkcie, ktoré okno s notifikáciou spravia „click-through“, čiže je možné ovládať okná pod ním. Takisto, keďže okno môže stratiť focus, nestačí vytvoriť klávesovú skratku v rámci okna notifikácie, ale je nutné vytvoriť globálnu klávesovú skratku.

11.4.1.3 Implementácia

Funkcie komunikujúce s WinAPI sú umiestnené v samostatných triedach. Konkrétne ide o WindowHelper, ktorý obsahuje funkcie na prácu s oknami v rámci WinAPI a InputHelper, ktorý pomáha pri vytváraní globálnej klávesovej skratky.

Na zabezpečenie, aby okno bolo na celú obrazovku, vždy navrchu a nezobrazovalo sa

```
<Window
        WindowStyle="None"
        ShowInTaskbar="False"
        ShowActivated="False"
        Topmost="True">
```

v taskbare, stačilo použiť atribúty, ktoré poskytuje XAML:

11.4.2 Zareportovanie nesprávneho sedenia

11.4.2.1 Analýza

V aplikácií chceme poskytnúť možnosť používateľovi nahlásiť nesprávne monitorovanie a odoslať si potrebné informácie aby sme mohli zlepšiť algoritmy na vyhodnocovanie správnosti sedenia.

Požiadavky:

- jednoduché odoslanie – stlačenie jedného tlačidla.
- odoslanie všetkých potrebných informácií
    - aktuálny snímok
    - kalibračný snímok
    - aktuálne vyhodnotenie
- odoslanie bez ďalšej interakcie

11.4.2.2 Návrh

Na odoslanie reportu o nesprávnom sedení postačuje implementácia pomocou odoslania emailu. V produkčnej verzii nepredpokladáme použitie tejto funkcionality. V prílohe emailu bude kalibračný a aktuálne vyhodnocovaný obrázok ktoré je možné serializovať.

11.4.2.3 Implementácia

Pridanie tlačidla do MainWindow okna. Tieto časti okna sú v aplikácií stále viditeľné – aj po prekliknutí na iné políčko v menu. Tlačidlo sa aktivuje/deaktivuje na základe spustenia/vypnutia monitorovania sedenia.

Po stlačení je na pozadí odoslaný mail pomocou triedy MailSender. Do emailu sú vložené dve prílohy. Prílohy sú vytvorené rovnakým spôsobom ako sa ukladá kalibrácia na pevný disk. Do textu prílohy je vložené aktuálne vyhodnotenie merania…

12 Prílohy

Celkový diagram tried