Slovenská Technická Univerzita v Bratislave
Fakulta Informatiky a Informačných Technológií

Monitorovanie a vyhodnocovanie fyziologických procesov človeka [StressMonitor]

Dokumentácia k riadeniu

Vedúci tímu: Ing. Katarína Jelemenská PhD.
Členovia tímu: Bc. Tomáš Bako, Bc. Martin Baláž, Bc. Matúš Brandýs, Bc. Patrik Husár, Bc. Patrik Krupa, Bc. Matúš Matula, Bc. Kristián Ostatník, Bc. Peter Vašek
Školský rok: 2016/2017

Obsah

1 ÚVOD

Predmet Tímový projekt má za úlohu ponúknuť skupine študentov možnosť pracovať na konkrétnej softvérovej úlohe spoločne a podnietiť tímové myslenie. Práca podlieha agilnej metóde vývoja známeho ako Scrum, ktorý má svoje jednoznačné dané prísne pravidlá. V Scrume sa práca delí do šprintov. V rámci predmetu Tímový projekt trvali šprinty 2 týždne. Po každom šprinte sa konalo vyhodnotenie a tzv. retrospektíva, v ktorej sa hľadali nedostatky, ktoré si tím zaumienil do najbližšieho šprintu potlačiť alebo úplne eliminovať. Pre vybrané dôležité veci, pri ktorých bolo potrebné sa v rámci tímu ujednotiť a zadefinovať si pravidlá, bol napísaný ich súhrn – tzv. metodika. Metodík bolo niekoľko, týkali sa manažmentu niektorej konkrétnej oblasti v tíme.

Tím sa skladá z viacerých členov a práca sa delí medzi nich. Každý člen dostal v konkrétnom šprinte priradenú úlohu. Na základe preferencií a dominujúcich úloh boli členom tímu priradené určité roly. V nasledujúcej kapitole sa opisujú úlohy a tímové roly členov tímu.

V ďalších kapitolách sa opisuje manažment konkrétnych dôležitých oblastí, jednotlivé šprinty vrátane ich výstupu a retrospektívy. Samostatná kapitola je venovaná sumarizačnej globálnej retrospektíve celého semestra.

2 ROLE ČLENOV TÍMU A PODIEL PRÁCE

2.1 Predstavenie členov tímu a ich roly

Tím vznikol pred zahájením semestra. Väčšia časť tímu bola vzájomne dohodnutá na spolupráci v tímovom projekte, traja členovia boli do tímu pridelení vedením predmetu Tímový projekt. Žiadna rola (okrem scrum mastera) nebola prirodzene pridelená konkrétnemu členovi na začiatku semestra. Roly vyplynuli podľa toho, ktorý člen primárne na akých úlohách robil. Názvy niektorých rolí sa zadefinovali až postupom času.

2.1.1 Bc. Tomáš Bako

Je absolventom bakalárskeho štúdia na FIIT STU v odbore Informatika. Zaujíma sa najmä o oblasť dátovej analytiky (v súvislosti s ktorou mal aj bakalársku prácu) a databáz. Vďaka jeho pracovným skúsenostiam, sa prehĺbila jeho znalosť v oblasti databáz. Z jazykov preferuje primárne Javu, no má skúsenosti aj s inými – PL/SQL, Python, C#.

V tíme zastáva rolu **scrum mastera**. Táto rola vyplynula najmä z jeho odhodlania a záujmu mentorovať (a nepriamo aj viesť) tím pri vývoji softvérového produktu. Okrem toho sa primárne podieľal na písaní dokumentácií a ďalších dôležitých materiálov k tímovému projektu. Vypracoval väčšinu metodík.

2.1.2 Bc. Martin Baláž

Martin je takisto absolventom bakalárskeho štúdia na FIIT STU v odbore Informatika. Zameriava sa primárne na vývoj mobilných aplikácií (primárne pre Android). Okrem toho ho zaujíma aj oblasť počítačovej grafiky. Z jazykov ovláda najmä Javu a C.

Jeho rola je **Android developer** – Martin teda programoval aplikáciu na OS Android. Jeho rola vyplynula už od prvého šprintu najmä na základe jeho záujmu o vývoj mobilných aplikácií.

2.1.3 Bc. Matúš Brandýs

Matúš je ďalším absolventom bakalárskeho štúdia na FIIT STU v odbore Informatika. Primárne sa zaujíma o siete, zabezpečenie sietí a mnoho ďalších vecí z tejto oblasti. Z toho dôvodu veľmi dobre ovláda konfiguráciu router-ov a switch-ov, s tým súvisia aj jeho administrátorské znalosti s Linuxom. Okrem toho je skúsený vývojár v jazykoch Java a Python.

Matúšovi podľa pôsobnosti možno priradiť dve roly, ktoré spolu súvisia. Sú to **dev-ops** a **full-stack developer**. Rola dev-ops súvisí najmä s rozbehávaním a konfiguráciou nevyhnutných serverov, ktoré sa používali. Možno k tomu okrem iného dodať aj vytváranie Docker kontajnerov, vďaka ktorým bolo možné vytvoriť nezávislé skupiny

zdrojového kódu, ku ktorým sa automaticky stiahla databáza aj „ovládače“[1]. Úloha full-stack vývojára u Matúša súvisela s vývojom webovej aplikácie. Jeho predchádzajúce skúsenosti s jazykom Python a jeho frameworkom Django, ktorý slúži na tvorbu webových stránok, boli pri tvorbe webovej aplikácie veľmi vítané.

2.1.4 Bc. Patrik Husár

Patrik podobne, ako vyššie uvedení, absolvoval bakalársky stupeň štúdia na FIIT STU v odbore Počítačové a komunikačné systémy a siete. Zaujíma sa najmä o hardvér a vývoj v tejto oblasti. Zručnosti, ktoré má s hardvérom, sú široko vítané vzhľadom na riešený projekt. Z hľadiska programovacích jazykov najlepšie ovláda jazyky na nižšej úrovni – najmä jazyk C. Patrik bol primárnym iniciátorom záujmu o tému, ktorú tím rieši.

Patrikova rola je **databázový špecialista**. Vyplýva najmä z jeho záujmu o zlepšenie sa v databázových technológiách. V rámci tímu sa podieľal na návrhu dátového modelu pre databázu a tiež na získavaní dát z testovania.

2.1.5 Bc. Patrik Krupa

Patrik je ďalším spomedzi absolventov odboru Informatika v bakalárskom stupni štúdia na FIIT STU. Výborne sa orientuje najmä v .NET technológiách od spoločnosti Microsoft. Z týchto technológii ovláda na pokročilej úrovni programovanie a použitie jazykov a frameworkov C#, VB.NET, ASP.NET, LINQ, WPF a mnohé iné. Vo všeobecnosti sa zaujíma o oblasť softvérového inžinierstva a nových technológií.

Patrikova primárna rola súvisí s administrovaním manažovacieho nástroja TFS, z toho vyplýva aj názov jeho roly – **správca TFS**. Táto rola vyplynula aj z jeho predchádzajúcich pozitívnych skúseností s daným nástrojom. Okrem toho je aj **testerom** a vzhľadom na tvorbu mobilnej aplikácie aj **Android developerom**.

2.1.6 Bc. Matúš Matula

Matúš je absolventom bakalárskeho stupňa štúdia na FIIT STU v odbore Informatika. Výborne sa orientuje v Java technológiách. Okrem nich má skúsenosti s REST-ovým rozhraním, aplikačnými servermi a vo všeobecnosti s vývojom kvalitného softvéru. Zaujíma sa o vývoj softvéru a softvérové inžinierstvo ako také.

Matúšova rola v tíme je **back-end developer**. Jeho náplňou je najmä vývoj serverovej časti aplikácie a REST-ových služieb pre prácu s databázou. Matúšove pracovné skúsenosti s Javou boli pri jeho rozbehaní na serveri veľmi cenné.

[1] Pod pojmom „ovládače“ sa v tomto prípade myslia knižnice konkrétneho jazyka, ako napr. Python, alebo Java.

2.1.7 Bc. Kristián Ostatník

Kristián je absolventom bakalárskeho programu Informačné technológie na VUT v Brne. Vzhľadom na jeho štúdium z programovacích jazykov najlepšie ovláda C, celkovo má skúsenosti s hardvérom. Zaujíma sa najmä o oblasť big data a umelú inteligenciu.

Kristiánova rola v tíme je **full-stack developer**. Súvisí to najmä s jeho prácou na webovej stránke tímu a taktiež na vývoji webovej aplikácie, pri ktorej sa chcel naučiť lepšie programovať v jazyku Python.

2.1.8 Bc. Peter Vašek

Peter je absolventom bakalárskeho štúdia v odbore Informatika na FIIT STU. Zaujíma sa najmä o oblasti big data, strojového učenia, umelej inteligencie. Je skúseným v oblasti paralelizácie výpočtov a spracovania dát. Aj vďaka tomu je skúsený v tvorbe v jazykoch Java a C++.

Peter zastáva rolu **databázového analytika**. Táto rola súvisí najmä s jeho skúsenosťami so spracovaním dát a tiež jeho záujmu o skúmanie významu dát získavaných z meracích zariadení.

2.2 Podiel práce na dokumentácii

V nasledujúcej tabuľke je uvedený podiel práce jednotlivých členov tímu na tomto dokumente. Je dôležité podotknúť, že tento podiel je daný pre tento konkrétny dokument a v žiadnom prípade sa nedá dať do korelácie vo všeobecnosti s prácou na tímovom projekte.

Tab. 1: Podiel členov tímu na vypracovaní tohto dokumentu

Meno	Percentuálny podiel práce	Náplň práce na dokumente
Bc. Tomáš Bako	70%	spísanie manažmentu komunikácie a dokumentácie, metodík a finálneho dokumentu
Bc. Martin Baláž	2%	revízia finálneho dokumentu a vybraného manažmentu
Bc. Matúš Brandýs	2%	revízia finálneho dokumentu a vybraného manažmentu
Bc. Patrik Krupa	9%	revízia finálneho dokumentu a vybraného manažmentu, práca na metodike práce s Git-om, konzultácie
Bc. Matúš Matula	1%	revízia finálneho dokumentu
Bc. Kristián Ostatník	15%	spísanie manažmentu testovania, plánovania a rizík, práca na metodike práce s Git-om
Bc. Peter Vašek	1%	revízia vybraného manažmentu

3 APLIKÁCIE MANAŽMENTOV

V tejto kapitole sú napísané manažmenty vo viacerých oblastiach v rámci tímového projektu, ktorými sme sa riadili.

3.1 Manažment plánovania

Plánovanie sa vykonáva spravidla na začiatku každého šprintu. Pozostáva z viacerých krokov, ktoré budú popísané v tejto podkapitole. Náš tím si vybral nástroj TFS na zaznamenanie a riadenie naplánovaných úloh, ktorý taktiež dovoľuje sledovanie vykonanej práce, vytvoreného kódu a jeho testovanie.

3.1.1 Product backlog

Plánovanie sa začína s napĺňaním tzv. **product backlog**-u za prítomnosti vlastníka produktu (tzv. **product owner**). Product backlog je číslovaný zoznam (čísla znamenajú prioritu, poradie určuje product owner), v ktorom jednotlivé položky (tzv. **user story** – používateľské príbehy) reprezentujú jednu konkrétnu funkčnú, grafickú, ale aj nefunkčnú časť vytváraného produktu (napr. dokumentácia), ktorá sa dokončením pridáva k vytváranému produktu. Každá položka sa ohodnocuje podľa náročnosti, ktorá sa tiež môže vyjadriť ako úsilie, ktoré potrebuje vynaložiť tím na jej dokončenie.

Ohodnotenie položiek sa vykonáva pomocou techniky **planning poker**. Každý člen tímu si vyberá číslo (ohodnotenie úlohy) podľa vlastného uváženia. Ohodnotenie je úspešné, ak si každý člen tímu vyberie rovnaké číslo, v opačnom prípade tím diskutuje o vybratých číslach a nastáva nové hodnotenie až pokým sa nezhodnú všetci členovia. Každá položka tiež obsahuje popis a cieľ, kedy môže byť úloha prehlásená za dokončenú, čo určuje product owner.

3.1.2 User story

User story je hodnota pre product owner-a, ktorú chce mať zahrnutú vo svojom produkte. Aby s ním mohol tím pracovať, potrebuje ju rozdeliť na menšie časti – tzv. **task**-y (úlohy), ktoré pre tím predstavujú hodnotu. Každý task je potom ohodnotený tímom podľa počtu hodín, ktoré treba na dokončenie danej úlohy. Taktiež sa pridáva popis a definícia cieľa, kedy môže byť task prehlásený za dokončený. Aby sa mohol task prehlásiť za dokončený musí sa vykonať tzv. **review** (revízia). V prípade implementačných záležitostí sa vykonáva **code review** a v ostatných prípadoch (najmä pri písaní dokumentácie) sa kontrolujú vytvorené dokumenty.

3.1.3 Priraďovanie úloh

Počet úloh sa vyberá podľa tzv. **velocity** tímu, ktorá sa vyjadruje číslom, aké množstvo práce je tím schopný vykonať za jeden šprint. Dané číslo je v závislosti s ohodnoteniami

user stories, a to tým spôsobom, že by sa do jedného šprintu malo dať také množstvo úloh, že ich súčet náročností z backlog-u bude približne rovnaký, ako je velocity. Ak je súčet podľa priorít z backlogu neprípustne väčší, tak sa so súhlasom product ownera môže zobrať aj user story s menšou prioritou. Na záver plánovania sa priraďujú úlohy jednotlivým členom tímu. Každý člen si vyberá úlohy podľa vlastného záujmu, skúseností, vedomostí a ochoty pracovať s novými technológiami. Taktiež sa priraďujú recenzenti (reviewers) k daným úlohám, ku ktorým sa hlásia členovia podľa rovnakých kritérií ako pri výbere úloh.

3.2 Manažment rizík

Manažment rizík slúži na predchádzanie a riadenie chaosu, ktorý by nastal pri krízových situáciách. V počiatočnej fáze projektu sa vykonáva analýza na identifikovanie rizikových situácií a vytvárajú sa prvotné rizikové scenáre opisujúce ich rozsah a následky. V našom prípade opisujeme taktiež:

- reakciu, t.j. bezprostredné kroky, ktoré sa majú vykonať v prípade, že nastane riziko,
- dopad, ktorý je mierou závažnosti negatívnych vplyvov na projekt a pravdepodobnosti výskytu rizika,
- príznaky, ktoré môžu napovedať, že hrozí nastanie rizika,
- prevenciu, t.j. kroky, ktoré tím musí dodržovať, čím by sa mohlo predísť daným rizikám.

V ďalších fázach projektu sa naďalej skúmajú riziká a vytvárajú nové rizikové scenáre. Doteraz identifikované riziká sú popísané v nasledujúcich tabuľkách.

Tab. 2: Opis rizika pre nesprávny odhad času alebo nesprávne stanovenie požiadaviek

Názov	**Nesprávny odhad času/stanovenie požiadaviek**
Popis	nesprávne ohodnotená náročnosť úlohy, ktorú zodpovedný člen nebude môcť dokončiť v plánovanom šprinte
Reakcia	- konzultácia s tímom - hľadanie partnera, s ktorým by sa úloha dala rozdeliť
Dopad	7
Pravdepodobnosť	0,3
Príznaky	slabá špecifikácia úlohy, nedostatok vedomostí
Dôsledok	neúspešný šprint
Prevencia	- pri plánovaní šprintu sa dostatočne špecifikuje každá úloha (čiastočné kroky, cieľ, tzv. definition of ready) - člen tímu si berie úlohu, len ak má potrebné vedomosti v danej problematike, poprípade sa dohovorí s ďalším členom, s ktorým bude spolupracovať

Tab. 3: Opis rizika pre stratu člena tímu

Názov	**Strata člena tímu**
Popis	člen tímu končí v štúdiu na fakulte
Reakcia	rozdelenie úloh a rolí bývalého člena pomedzi ostatných členov
Dopad	8
Pravdepodobnosť	0,15
Príznaky	-
Dôsledok	neúspešný šprint, časový sklz
Prevencia	Častá komunikácia v tíme ak aj nedokáže predísť danému riziku, môže pomôcť včas identifikovať nastanie problému.

Tab. 4: Opis rizika pre práceneschopnosť člena tímu

Názov	**Práceneschopnosť člena tímu**
Popis	člen tímu zo zdravotných alebo osobných dôvodov nemôže pracovať na pridelených úlohách
Reakcia	rozdelenie úloh medzi ostatnými členmi
Dopad	5
Pravdepodobnosť	0,5
Príznaky	zlý zdravotný stav člena, osobné problémy
Dôsledok	väčšie zaťaženie ostatných členov tímu
Prevencia	Komunikácia – možnosť predpovedania.

Tab. 5: Opis rizika pre použitie novej technológie

Názov	**Použitie novej technológie**
Popis	-
Reakcia	diskutovanie s tímom, hľadanie skúsenejších ľudí, ktorí by mohli poradiť
Dopad	6
Pravdepodobnosť	0,4
Príznaky	chýbajúce skúsenosti, objav novej technológie, nedostatočná dokumentácia danej technológie
Dôsledok	časový sklz, poprípade nedokončenie všetkých úloh v šprinte
Prevencia	Rozdelenie úlohy na dva šprinty. Prvý šprint sa bude venovať len analýze novej technológie.

Tab. 6: Opis rizika pre prestoj

Názov	**Prestoj**
Popis	stratový čas, keď je člen tímu nútený stáť s danou úlohou, napr. z dôvodu čakania na dokončenie nadväzujúcej úlohy, výpadok elektriny, servera
Reakcia	-
Dopad	4
Pravdepodobnosť	0,3

Príznaky	na seba nadväzujúce úlohy
Dôsledok	väčšie zaťaženie čakajúceho člena
Prevencia	- predchádzanie plánovania nadväzujúcich úloh; ak to nie je možné, zodpovedný člen, na ktorého je treba čakať, je povinný dokončiť úlohu čo najskôr - priebežná kontrola úspešnosti plnenia úloh

Tab. 7: Opis rizika pre konflikty v tíme

Názov	**Konflikty v tíme**
Popis	nespokojnosť s rozdelením úloh, s prácou alebo časovým rozdelením iného člena tímu
Reakcia	stretnutie/diskutovanie s tímom a vedúcou tímu
Dopad	3
Pravdepodobnosť	0,2
Príznaky	-
Dôsledok	časový sklz, v extrémnom prípade odchod člena z tímu
Prevencia	- častá komunikácia - udržovanie dobrých vzťahov v tíme pomocou SCRUM mastera

3.3 Manažment testovania

Testovanie je jednou z najdôležitejších úloh pri vývoji softvérového produktu. Pri správnom návrhu metódy testovania, ktorý postupne odhaľuje chyby počas implementácie produktu, sa môže ušetriť množstvo času, ktoré sa následne môže použiť na zvýšenie kvality. Testovanie nielen urýchľuje vývoj a zaručuje validáciu produktu, ale aj umožňuje predísť rôznym logickým a skrytým chybám, na ktoré programátor pri vývoji nemyslel.

3.3.1 Systém riadenia revízií

Na vytváranie a ukladanie jednotlivých verzií vytvoreného kódu sme sa rozhodli využívať systém Git. Slúži na zálohovanie, distribúciu a kontrolu verzií. Výhodou zachovávania a práce s jednotlivými verziami je, že pri vyskytnutí straty kódu alebo pri vykonaní viacerých zmien, ktoré by používateľ nevedel napraviť, sa dá ľahko vrátiť k predchádzajúcej verzii. Git a komunikačný nástroj (v našom prípade Slack) sú prepojené, preto nemusíme explicitne informovať všetkých členov tímu o pridaní kódu alebo vytvorení pull request-u.

3.3.2 Testovanie pomocou bielej skrinky

Testovanie pomocou bielej skrinky prebieha vo fáze code review. Je aplikovaný na testovanie webovej aplikácie a serveru. Počas 2. šprintu sme sa rozhodli využiť softvér **Docker**, ktorý umožňuje spustenie aplikácie na ľubovoľnej platforme. To nám umožňuje

testovanie a prechádzanie vytvoreného kódu na vlastných počítačoch bez manuálnej inštalácie závislých alebo chýbajúcich programov. Zodpovedný člen tímu za code review v rámci tejto fázy testovania postupuje podľa krokov metodiky testovania.

3.3.3 Testovanie mobilnej aplikácie

Na testovanie Android aplikácie je potrebná inštalácia nástroja **Android studio** od spoločnosti JetBrains a ideálne vlastnenie zariadenia s OS Android, na ktorom bude možné danú aplikáciu otestovať. Z týchto dôvodov boli určení dvaja testeri, ktorí splňujú tieto podmienky. Testovanie prebieha podobne ako pri bielej skrinke. Tester si stiahne zdrojové kódy k dokončenému task-u, ktorý spúšťa/nahráva do zariadenia. Pri testovaní sa zameriava na funkcionalitu a GUI rozhranie. Pri identifikovaní chyby a vykonávaní code review postupuje rovnako ako bolo v predchádzajúcich častiach spomenuté.

3.4 Manažment dokumentácie

K softvérovému procesu bez pochýb patrí aj tvorba dokumentácie a ďalších materiálov takéhoto charakteru. Počas práce na tímovom projekte sa preto vytváralo množstvo rozličných dokumentov. Najväčšími z nich sú **Dokumentácia k riadeniu** a **Dokumentácia inžinierskeho diela**.

V súvislosti s agilnou metódou Scrum, ktorá sa uplatňuje v našom tíme, sa po každom stretnutí tímu v daný týždeň spísala zápisnica, ktorej úlohou bolo vytvárať prehľad o jednotlivých stretnutiach tímu. Zápisnicu mal na starosti vždy vopred určený člen tímu, pre jeden šprint ten istý. Každá zápisnica obsahovala základné informácie o stretnutí – kedy a kde sa konalo, kto sa ho zúčastnil a body, ktoré sa prerokovali.

V každom šprinte bolo množstvo rozličných úloh. Pri plnení úloh sa tím zaviazal dodržiavať určité pravidlá, ktoré boli pre prehľadnosť a ľahkú dostupnosť spísané do viacerých metodík. Sú to nasledovné metodiky:

- metodika práce v tíme,
- metodika práce s Git-om,
- metodika testovania,
- metodika písania zdrojového kódu.

Všetky dokumenty vrátane zápisníc a metodík sú prístupné na webovej stránke tímu.

3.5 Manažment komunikácie

Komunikácia v tímovom projekte patrí k jedným z kľúčových faktorov. Od nej môže veľmi závisieť úspech alebo neúspech daného projektu. V našom tíme sa použilo viacero kanálov na komunikáciu. Sú uvedené a vysvetlené v jednotlivých podčastiach tejto kapitoly.

3.5.1 Team Foundation Server (TFS)

TFS je manažovací nástroj pre správu úloh v tíme, ktorý podlieha metóde agilného vývoja Scrum. V TFS je možné zadefinovať si projekt, vytvárať šprinty, počas ktorých sa vytvára produkt iteratívno-inkrementálne. Ďalej sa v ňom dajú vytvoriť používateľské príbehy pre konkrétny šprint (angl. user stories), ku ktorým sa dajú priradiť jednotlivé úlohy (angl. tasks). K úlohám ako aj k príbehom je možné priradiť konkrétnu osobu zodpovednú za úspešné dokončenie danej činnosti.

Pre výber TFS sa rozhodlo na základe toho, že patrí medzi pomerne známe nástroje, niektorí členovia tímu s ním už mali skúsenosti a tiež na kvôli jeho podpore na fakulte – pre študentov fakulty je voľne dostupný v on-line verzii na *https://tfs.fiit.stuba.sk/* . Vďaka tomu majú zjednodušenú prácu pri rozbehávaní a nemusia si nič inštalovať lokálne na svoj osobný počítač. Veľkou výhodou TFS je najmä možnosť integrácie s Git-om (ktorý sa používa ako verziovací nástroj). Táto integrácia sa dá ďalej posunúť na posielanie správ do komunikačného nástroja Slack – napr. správy o pridaní nových súborov pomocou Git-u alebo posielanie tzv. pull-request-ov, ktoré súvisia s pridaním novej funkcionality do výsledného zdrojového kódu.

Pri práci s TFS trvalo približne 3 týždne, kým sa tím dostatočne oboznámil, ako s ním pracovať, a zároveň sa naučil využívať všetky jeho vymoženosti a výhody. V sumári je zrejmé, že aj vďaka tomu je TFS v tíme hodnotený pozitívne.

3.5.2 Slack

Pre tímovú komunikáciu sa používal známy nástroj Slack. V tíme sa všetci členovia zhodli na tom, že bude dobré používať tento nástroj. Poskytoval nám dostatočnú podporu vzhľadom na možnosť vytvárať osobitné „vlákna“ komunikácie – napr. podľa určitej témy. Zároveň poskytoval možnosť komunikovať aj osobne – s konkrétnym členom tímu (napr. ak by dvaja členovia spolupracovali na konkrétnej úlohe, resp. si dokázali navzájom vypomôcť). Základné vlákna v Slack-u boli nasledovné:

- **General** – v tomto vlákne sa riešili najmä organizačné záležitosti a veci, o ktorých mal byť informovaný každý člen tímu,
- **TFS Notifications** – do tohto vlákna chodili správy o pridaní súborov zdrojového kódu do Git-u a pull request-ov vzhľadom na synchronizáciu s nástrojom TFS,
- **Mobilná aplikácia** – v tomto vlákne sa riešili nejasnosti ohľadom implementácie mobilnej aplikácie,
- **Web stránka** – v tomto vlákne sa spresňovali nejasnosti ohľadom webovej stránky tímu a zároveň i webovej aplikácie, ktorá primárne slúžila na pokročilejšiu vizualizáciu nameraných údajov,
- ďalšie vlákna – počas práce na tímovom projekte sa okrem vyššie spomenutých vlákien vytvárali aj mnohé ďalšie, ktoré boli dôležité len dočasne – počas toho, ako

sa riešila určitá problematika (napr. keď sa navrhoval plagát, dohadovali sa preferencie v rozvrhu a v témach tímového projektu u každého člena tímu, atď.).

3.5.3 Tímové stretnutia

Počas semestra sa každý týždeň konali tímové stretnutia. Mali dvojaký charakter:

- *inicializačný*
    - za inicializačný sa dalo nazvať také stretnutie, v ktorom sa inicializoval šprint, doplňoval sa product backlog, časovo sa ohodnocovali jednotlivé používateľské príbehy a úlohy priradené týmto príbehom,
    - tieto úlohy sa potom prideľovali jednotlivým členom tímu,
    - počas tohto stretnutia sa okrem inicializácie šprintu konalo aj ukončenie šprintu – členovia tímu zhrnuli, čo urobili počas šprintu, ako si splnili svoje úlohy,
    - s ukončením šprintu súvisela aj retrospektíva, v nej sa spätne hľadali klady a zápory šprintu, na základe ktorých si tím vytýčil, v čom sa chce zlepšiť do najbližšieho šprintu,
- *konzultačný*
    - spravidla sa tieto stretnutia konali týždeň po inicializácii šprintu
    - členovia tímu zhrnuli, čo sa im podarilo za uplynulé obdobie, aby si v prípade problémov mohli navzájom poradiť.

Okrem týchto pravidelných stretnutí sa raz počas týždňa uskutočnil aj tzv. stand-up, niekedy fyzický, inokedy virtuálny. Počas stand-up-u každý člen tímu zhrnul 3 veci:

- na čom pracoval,
- na čom bude pracovať,
- či má s niečím problémy, alebo či stojí na niečom.

Stand-up mal primárne za úlohu sprehľadniť prácu jednotlivých členov tímu a zároveň včas odhaliť problémy s konkrétnymi úlohami a poskytnúť riešenie v prípade, že sa nejaké vyskytli.

3.5.4 Gmail

Tím mal okrem iného vytvorené aj konto na Gmail-i. E-mailová adresa tímu bola *tpteam23@gmail.com*. Táto adresa sa používala najmä na komunikáciu s externým dodávateľom meracích zariadení a tiež na oficiálnu komunikáciu s vedúcou tímu.

3.5.5 Git

O Git-e sa píše v tomto dokumente vo viacerých kapitolách. Za kanál komunikácie ho možno považovať z toho hľadiska, že ku kódu, ktorý sa pomocou Git-u posiela do spoločného repozitára, pridávajú komentáre. Tieto komentáre určitým spôsobom

poskytujú informáciu ostatným členom tímu o tom, čo sa vykonalo a aký progres sa udial v danej úlohe.

4 POUŽÍVANÉ METODIKY

Ako už bolo v predchádzajúcej kapitole spomenuté, tím si pre úlohy podobného charakteru vytvoril metodiky, ktoré mali za úlohu zefektívniť a zjednodušiť prácu, aby výsledky aspoň do určitej miery spĺňali nejaký štandard a celkovo sa dalo s nimi lepšie pracovať. Všetky metodiky sú uvedené v Prílohe A.

4.1 Metodika práce v tíme

Po prvom šprinte aj vzhľadom na komplikácie, ktoré sa v ňom odohrali, si tím určil viaceré všeobecné pravidlá, ktoré sa primárne týkajú informovanosti o úlohách v danom šprinte a spôsobe odovzdávania výsledkov práce.

4.2 Metodika práce s Git-om

Zdrojové kódy výsledného softvérového produktu podliehajú verziovaniu pomocou nástroja Git. Nakoľko viacerí členovia tímu mali s Git-om menšie skúsenosti, skúsení členovia vytvorili metodiku, ako postupovať pri pridávaní zdrojového kódu do spoločného repozitára vrátane postupnosti krokov od začiatku celého procesu.

4.3 Metodika testovania

Testovanie patrí k jedným z najdôležitejších pri tvorbe softvéru. Čím skôr sa začne testovať, tým je väčšia šanca skoršieho odhalenia možných chýb, vďaka čomu sa môže ušetriť určité množstvo času. Metodika testovania bola vytvorená, aby sa testovalo jednotným spôsobom, t.j. aby sa nestal ten prípad, že tester nevykoná test dostatočne kvalitne. Vtedy by sa v súvislosti s neotestovanou alebo nedostatočne otestovanou funkcionalitou mohli objaviť chyby. Lepšie je chybám prejsť, daná metodika vytvára paradigmu, ako sa to dá.

4.4 Metodika písania zdrojového kódu

Náš softvérový projekt pozostáva z viacerých častí. Jednotlivé programové komponenty sú napísané vo viacerých programovacích jazykoch – webová aplikácia v jazyku Python, zároveň používa framework Django. Serverový back-end podobne, ako aj mobilná Android aplikácia je naprogramovaná v jazyku Java. Nakoľko na zdrojovom kóde pracujú viacerí členovia tímu, pre zjednodušenie refaktorizácie a zlepšenia čitateľnosti sa vytvorila metodika písania zdrojového kódu, ktorá obsahuje viacero pravidiel. Pravidlá boli zvlášť napísané pre jazyk Java, Python a C.

5 SUMARIZÁCIA ŠPRINTOV

V tejto kapitole je sumár jednotlivých šprintov v priebehu semestra – opis situácie šprintu, aké úlohy boli do šprintu priradené, čo bolo cieľom šprintu a aké sú jeho výsledky vrátane retrospektívy po šprinte. Pre potreby vizualizácie priebehu práce na jednotlivých úlohách bol z nástroja TFS vyexportovaný tzv. **burndown chart** (graf, ktorý znázorňuje, koľko práce na šprinte k určitému bodu zostáva).

5.1 Šprint č.1

5.1.1 Opis šprintu a úloh

Prvý šprint sa dá považovať za inicializačný šprint celého projektu. V tomto šprinte sa tím spoznával a prvýkrát si vyskúšal vzájomnú spoluprácu, ktorá ešte nemala svoje jasne stanové pravidlá. Postupom času, ako sa členovia spoznávali a začali spolupracovať, sa zároveň aj etablovali a zvykli si na prácu v tíme. Napriek tomu si to vyžadovalo čas, úsilie i trpezlivosť členov tímu.

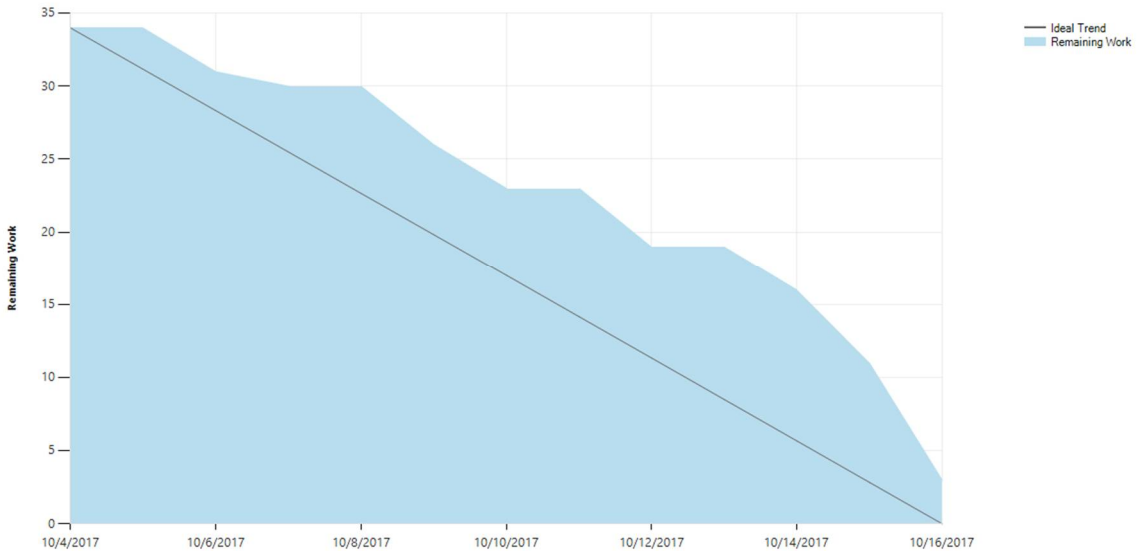
Z hľadiska úloh sa riešili úlohy z rôznych kategórií, najdôležitejšie boli infraštruktúrne úlohy, ktoré súviseli s rozbehávaním servera a inštalovaním dôležitých komponentov, bez ktorých by sa na projekte nedalo pracovať, nakoľko by jednotlivé komponenty projektu nemohli bežať – webový server, vďaka ktorému bude môcť bežať webová stránka tímu, Java pre back-end aplikáciu a databázu MariaDB, do ktorej sa budú ukladať dáta. Ďalšie úlohy sa týkali jednoduchého prototypovania, resp. inicializácie, čo malo slúžiť pre vyskúšanie si vývoja na platformu Android. Ďalej to bolo vytvorenie dátového modelu a konfigurácia manažovacieho nástroja. Za taký sme si vybrali TFS, nakoľko s ním už niektorý člen mal skúsenosti a vedel dobre využívať jeho pokročilé vlastnosti.

Počas prvého šprintu sa prebrali meracie dosky plošných spojov od externého dodávateľa – Juraja Brenkúša. V súvislosti s nimi sa vytvoril vhodný testovací scenár (aj podľa určitých odporúčaní od mnohých psychológov a manažérov zo sveta), ktorý dáva predpoklad získania širokej škály hodnôt dát o testovanom človeku. Do šprintu boli zaradené aj úlohy súvisiace s otestovaním dosiek.

V šprinte sa až na jednu úlohu podarilo všetky úspešne dokončiť. Prvý šprint aj napriek tomu, že bol inicializačný, sa dá považovať za relatívne úspešný. Predpokladal sa totiž horší finálny výsledok, napriek zbrusu novému štýlu projektu sa tím vedel dostatočne prispôsobiť podmienkam.

5.1.2 Burndown chart

Priebeh plnenia úloh prehľadne zobrazuje tzv. **burndown chart**, ktorý ukazuje, koľko hodín ešte nebolo odpracovaných v danom čase.



Obr. 1: Burndown chart prvého šprintu

Z burndown chartu je zrejmé, že na úlohách sa pracovalo s výkyvmi. Najviac práce sa vykonalo v posledných 2 dňoch daného šprintu.

5.1.3 Retrospektíva

V prvom šprinte, ako to býva pravdepodobne pri väčšine tímov, sa vyskytli viaceré chyby a nedostatky. Jednotliví členovia za najväčšie chybu považovali nedostatočnú komunikáciu v tíme. Ako spôsob riešenia sa dohodlo na pravidelnejších stand-up-och (krátkych neformálnych stretnutiach, o ktorých sa píše v časti Manažment komunikácie) a na zlepšení komunikácie v rámci tímu – aby sa člen tímu nebál ozvať sa v prípade chýb. Ďalšie konkrétne podnety od jednotlivých členov tímu na zlepšenie tímovej spolupráce boli nasledovné:

- skoršie písanie dokumentácií a popisov k jednotlivým úlohám v manažovacom nástroji MS TFS,
- skoršie plánovanie aktivít ako napr. pravidelných stand-up-ov,
- pravidelnejšie stand-up-y,
- celkovo zlepšenie komunikácie v rámci tímu,
- vytvorenie určitých metodík práce v tíme,
- vytváranie akceptačných kritérií pre jednotlivé úlohy.

5.2 Šprint č.2

5.2.1 Opis šprintu a úloh

Charakter druhého šprintu sa dá považovať za analyticko-implementačný. Veľmi dôležitou úlohou v tomto šprinte totiž bola analýza meraných údajov a spôsob, akým sa dajú interpretovať. Bez toho, aby sme vedeli, čo vlastne meriame nemôžeme vedieť, akým spôsobom môžeme používateľovi aplikácie dať nejakú pridanú hodnotu. V šprinte bolo

viacero implementačných úloh, ako napr. inicializácia webovej aplikácie (prihlásenie do nej) a s ňou spojená vizualizácia nameraných dát. Podobne tomu bolo aj pre mobilnú aplikáciu, pre ktorú bolo potrebné implementovať vizualizáciu. Pre server bolo nevyhnutné vytvoriť funkčný back-end pre zapisovanie a výber dát.

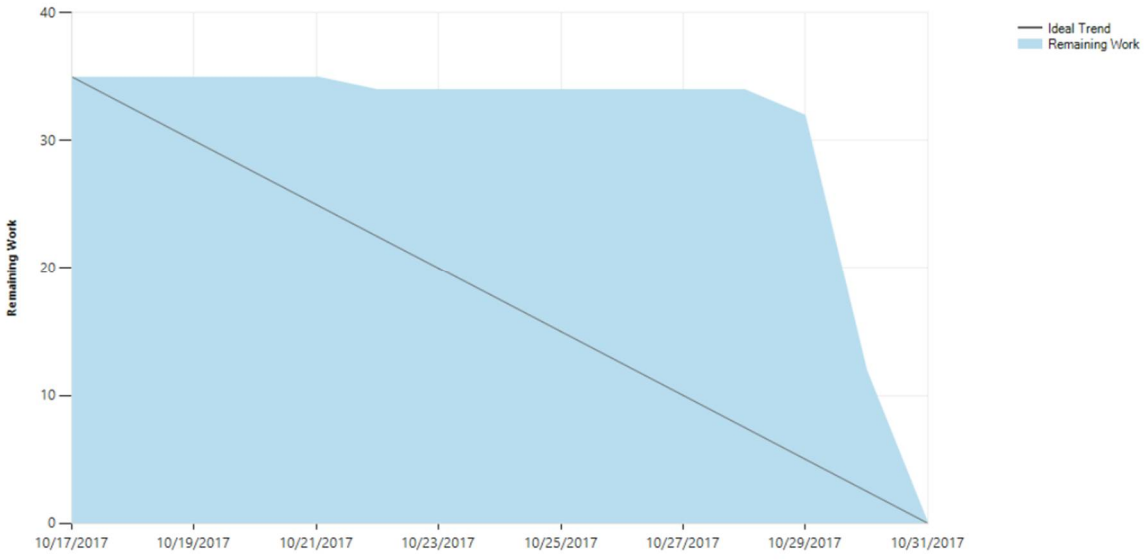
V pokročilej fáze šprintu došlo k zmene plánov, keď sa v tíme na odporúčanie od iných kolegov z fakulty dohodlo na použití nástroja Docker, ktorý umožňuje vytvárať replikovateľné kontajnery zdrojových kódov. Napriek tomu, že s ňou nebola spojená žiadna úloha, rozhodlo sa, že finálne zdrojové kódy sa budú dávať do Docker-a, aby si ich mohol hocikto jednoducho a efektívne stiahnuť a spustiť. Rozbehávanie Docker-a a s ňou spojená konfigurácia projektov a hotových zdrojových kódov skomplikovala priebeh šprintu. Napriek tomu sa podarilo úspešne vyriešiť všetky úlohy v šprinte.

5.2.2 Burndown chart

Z nasledovného burndown chart-u je vidieť, že priebeh dokončovania úloh bol dosť nerovnomerný. Burndown chart sa v tomto prípade dá považovať za veľmi zlý. Dôvodov môže byť niekoľko:

- zlý odhad potrebného času práce,
- komplikácia v priebehu riešenia úloh,
- veľa rozpracovaných úloh naraz a ich úspešné vyriešenie až ku koncu šprintu.

Napriek tomu je pozitívom, že sa vyriešili všetky úlohy do konca šprintu.



Obr. 2: Burndown chart druhého šprintu

5.2.3 Retrospektíva

V retrospektíve tím za najväčšie pozitívum vyzdvihlo veľké zlepšenie celkovej komunikácie. Naopak, za veľký problém sa považovalo nedostatočné sledovanie diania v manažovacom nástroji TFS – viacerí členovia mali dlhší prestoj, keď museli zbytočne

čakať na revíziu napr. zdrojového kódu. Z tohto problému vyplynul aj záväzok a dohoda na konkrétnom riešení revízie jednotlivých vecí – ku každej úlohe bude priradený konkrétny človek na vykonanie revízie.

Ďalším problémom v súvislosti s TFS bol nemenný stav úloh aj napriek tomu, že boli rozpracované. Preto sa dohodlo na určitých pravidlách, ktoré sa budú dodržiavať – na začiatku riešenia úlohy každý človek nastaví stav úlohy na „IN PROGRESS“, aby každý člen vedel, že sa na danej úlohy pracuje. Zároveň riešiteľ úlohy doplní popis pre danú úlohu v prípade, že popis chýba.

Pre prehľadnosť sa tiež dohodlo na zlepšení rozdelenia úloh – veľké úlohy sa rozdelili na viacero menších úloh, vďaka tomu sa bude dať lepšie sledovať priebeh práce.

5.3 Šprint č.3

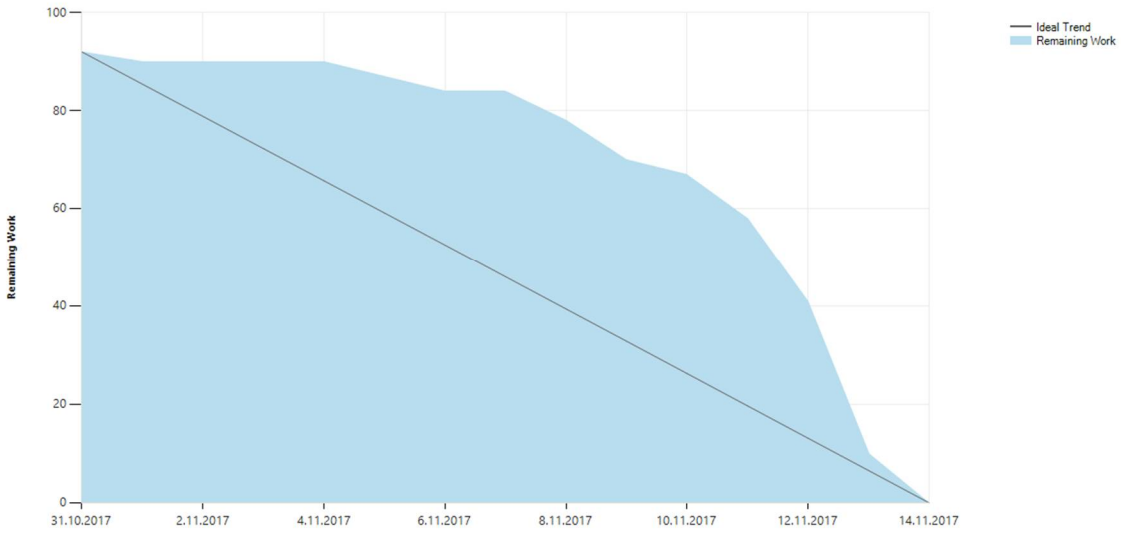
5.3.1 Opis šprintu a úloh

V treťom šprinte sa riešilo veľa úloh biznis charakteru – najmä písanie rozličných dokumentov. Medzi ne patrili rôzne manažmenty, metodiky, a napokon aj dokumentácie k rôznym častiam projektu. Do osobitnej kategórie by sa dali zaradiť „veľké“ dokumentácie k manažmentu projektu a k inžinierskemu dielu. Manažmenty a metodiky boli kľúčové pre dokončenie oboch „veľkých“ dokumentácií. Tie boli dôležité aj z hľadiska hodnotenia predmetu Tímový projekt, nakoľko sa odovzdávali po ukončení tohto šprintu.

Ďalšie úlohy sa týkali viacej vytváranej aplikácie. Pre možnosť efektívneho a jednoduchého replikovania a rozbehnutia jednotlivých častí bolo pre ne žiaduce vytvoriť tzv. Docker kontajnery. Ďalej bolo kľúčové navrhnúť spôsob spracovania a vyhodnotenia nameraných dát, bez ktorých v podstate celý projekt nemá zmysel – bola pre to vytvorená aj úloha. Aby sa s dátami dalo efektívnejšie, jednoduchšie, a aj na diaľku pracovať, vytvorili sa RESTové služby pre prácu s nimi. Pre ich otestovanie bolo zároveň dôležité naplniť databázu dátami. Všetky úlohy boli úspešne vyriešené do konca šprintu.

5.3.2 Burndown chart

Burndown chart pre 3. šprint je oproti predošlému šprintu podstatne lepší. Dokončenie viacerých úloh sa robilo ku koncu šprintu, čo ale neodráža skutočný priebeh práce na nich, nakoľko boli rozpracované skôr. Daný vývoj, aj vďaka tomu, že tím sa postupne v čase organizačne zlepšuje a dokáže lepšie a lepšie spolupracovať, dáva pozitívny predpoklad pre ďalší šprint.



Obr. 3: Burndown chart tretieho šprintu

5.3.3 Retrospektíva

V retrospektíve tretieho šprintu sa opäť vyzdvihla komunikácia a vzájomná spolupráca medzi jednotlivými členmi tímu. Tento aspekt vysoko ocenila aj vedúca tímu, dr. Jelemenská. Napriek tomu sa ale objavili aj určité nedostatky a dohodlo sa na vylepšení vecí, ktoré sú s nimi spojené. Napr. sa dohodlo na virtuálnom stand-up-e pomocou aplikácie **Hangouts**, nakoľko pri poslednom stand-up-e došlo k menšiemu prestoju vzhľadom na nedohodnutie sa na spôsobe vykonania. Riešil sa aj problém so zlými komentármi pri jednotlivých úlohách, pri ktorých sa pripomenulo pravidlo z metodiky práce v tíme. Toto pravidlo hovorí o tom, že člen tímu, ktorý ide riešiť konkrétnu úlohu, má k nej napísať výstižný komentár. Patrik Krupa navrhol, aby sa dokumenty, na ktorých písaní alebo revízií sa bude podieľať viacero členov tímu, dali na **Google Drive** za použitia **Google Dokumentu**. Tento návrh bol zároveň aj odsúhlasený.

PRÍLOHA A – METODIKY

A.1 Metodika práce v tíme

Pre zefektívnenie, zjednodušenie a celkovo zlepšenie práce v tíme sa v rámci tímu pristúpilo k dodržiavaniu niekoľkých pravidiel. Vďaka týmto pravidlám sa sprehľadní práca tímu a jednotliví členovia budú mať prehľad o aktuálnom dianí.

Pravidlá sú nasledovné:

- v prípade, že člen tímu začne pracovať na niektorej úlohe, v nástroji TFS preradí danú úlohu do kategórie „**IN PROGRESS**“ (t.j. rozpracované) - je to dôležité, aby ostatní mali prehľad, na čom sa práve robí,
- každý člen tímu je povinný na začiatku práce svojej úlohy skontrolovať, či daná úloha má nejaký popis, čoho sa bude týkať, resp. náplň, a čo sa pri jej splnení vlastne dosiahne. V prípade, že nemá, je povinný doplniť popis danej úlohy,
- po dokončení úlohy v závislosti od charakteru úlohy sa môže postupovať nasledovne:
    - ak je úloha implementačná – člen tímu pomocou GITu z novej branch-e (s názvom **TASK####**) pošle pull-request do branch-e **develop**,
    - ak má úloha administratívny charakter (nie programátorský, teda nie implementačný) – vypracovanie dokumentu, spísanie metodiky, zápisnice, štúdium materiálov a ich spísanie do stručných poznámok, člen tímu priloží vypracovaný súbor k danej úlohe. V prípade, že úloha má priradeného človeka na revíziu, skontaktuje ho a požiada o vykonanie revízie daného dokumentu,
- autor dokumentov, ktoré sú výstupom pre konkrétnu úlohu, je povinný po označení danej úlohy ako splnenej upload-núť dokument:
    - a) k svojej danej úlohe (pokiaľ už bola upload-nutá, v prípade potreby nahradiť ju aktuálnou verziou),
    - b) k zoznamu dokumentov pre projekt (v hornom paneli zvoliť kategóriu **Doc**, v nej si vybrať kategóriu dokumentu a nahrať pomocou **Upload document**)
- kódy z pull-request-ov musí člen tímu, ktorý je k danej revízii priradený, zhodnotiť a posúdiť (buď vráti naspäť na prerobenie, alebo schváli). V prípade, že k úlohe nie je nikto priradený na vykonanie revízie, autor daného kódu skontaktuje ľubovoľného člena tímu a požiada o vykonanie revízie kódu a otestovanie

- ak sú pull-request-y schválené aspoň jedným členom tímu, kódy sa môžu merge-núť. Podmienkou je, že ten jeden člen musí byť zodpovedný za revíziu pre daný zdrojový kód. Merge-nutie kódu vykonáva Patrik Krupa,
- v prípade nejasností a problémov s priradenou úlohou má člen tímu povinnosť konzultovať s tímom, aby sa úloha do konca šprintu stihla.

A.2 Metodika práce s Git-om

Prehľad základných príkazov, ktoré sú potrebné pre prácu s Git-om:

- **Nastavenie globálnych parametrov:**

```
$ git config --global user.name "ais_login"
$ git config --global user.email "lubovolny@email.com"
```

- **Naklonovanie a stiahnutie existujúceho projektu:**

```
$ git clone <adresa projektu>
$ git pull
```

- **Práca s branch-ami (vetvy) – vytvorenie, prepínanie:**

```
$ git branch TASK<číslo>
$ git checkout TASK<číslo>
```

- **Pridanie nových súborov, commit a nahranie do repozitára:**

```
$ git add .
$ git commit -m "TaskXY fixed overflow"
$ git push
```

V tíme sme sa dohodli na niektorých pravidlách, ktoré budeme dodržiavať:

- novú úlohu ne-commit-ovať do **master** alebo **develop** vetvy,
- každú úlohu vytvárať v samostatnej vetve, s názvom TASKxxxx (xxxx – číslo úlohy),
- pridávanie commit-ov primerane často,
- commit vždy doplniť stručným, ale výstižným komentárom,
- po dokončení úlohy vytvoriť tzv. **pull request** do develop vetvy,
- vytvárať pull request čo najskôr, aby sa mohla dôkladne vykonať revízia a testovanie kódu,
- informovať testera, ktorý bol priradený k úlohe, o vytvorení pull request-u, aby vykonal revíziu kódu.

A.3 Metodika testovania

Zodpovedný člen tímu za code review v rámci tejto fázy testovania vykonáva tieto kroky a dodržiava pravidlá:

- stiahnutie príslušného task-u z verziovacieho systému Git, ktorý obsahuje tzv. Docker image,

- zadanie príkazov na vytvorenie a spustenie aplikácie:
    - `docker-compose build`,
    - `docker-compose up`,
- vykonanie testovania – code review, kontrola funkcionality, hľadanie chýb,
- prehliadku kódu vykonáva člen tímu, ktorý daný kód nevytváral a je dostatočne skúsený na posúdenie,
- pri priraďovaní úloh sa automaticky priradí aj člen tímu na posúdenie,
- k nájdenému problému v kóde pridať komentár, ktorého obsahom je výstižný popis problému alebo spresňujúca otázka.
- komunikovať s autorom zdrojového kódu úlohy ohľadom nájdených problémov, prípadne snažiť sa o nájdenie riešenia problémov,
- po vyriešení problému zmeniť stav problému z **Active** na **Resolved**,
- kontrolovať zmeny v zdrojovom kóde vykonané na základe prehliadky kódu, po schválení zmeniť stav problému na **Closed**, inak sa proces opakuje.

A.4 Metodika písania zdrojového kódu

V rámci tímu sa do behu zaviedla metodika písania zdrojového kódu, v ktorej je napísaných niekoľko pravidiel. Tieto pravidlá by sa dali rozdeliť do troch častí, podľa jazyka, v ktorom sa programuje – serverová časť a mobilná aplikácia v Jave, webová aplikácia v Pythone a program na mikroprocesore v C. Nasledujúce kódové konvencie pre kódy písané v oboch jazykoch majú dvojaký cieľ:

- obmedziť potrebný počet refaktorizácií a úprav finálneho zdrojového kódu,
- zefektívniť vykonávanie revízie kódu,
- zjednodušiť čítanie a študovanie kódu cudzou osobou.

Pravidlá pre jazyk Java:

1. Všetky názvy premenných, tried a aj všetky komentáre písať VÝHRADNE v anglickom jazyku.
2. Triedy (classes) a rozhrania (interfaces) pomenovávať vo formáte *UpperCamelCase*.
3. Metódy pomenovávať vo formáte *lowerCamelCase*.
4. Premenné triedy definovať na začiatku definície triedy a dodržiavať pri nich nasledujúce pravidlá:
    a. Premenné pomenovať vo formáte *lowerCamelCase*.
    b. Premenné, ktoré majú modifikátor prístupu *private* a *non-static* pomenovať s písmenom *m* na začiatku.
    c. Premenné s modifikátormi prístupu *private* a *static* pomenovať s písmenom *s* na začiatku.

    d. Premenné, ktoré sú konštanty (majú modifikátor prístupu *final* a *static*) pomenovať veľkými písmenami s podčiarkovníkom pre oddelenie slov (*ALL_CAPS_WITH_UNDERSCORES*).
5. Vždy špecifikovať modifikátor prístupu.
6. Bloky kódu sa odsadzujú na začiatku tabulátorom (tabulátor má predstavovať 4 medzery)
7. Dĺžka riadku by nemala presahovať 100 znakov.
8. Presahujúce riadky odsadzovať na začiatku nového riadku ôsmimi medzerami.
9. Otváracie zátvorky v definícii tried, metód a blokov kódu písať na konci daného riadku.
10. Časti tried písať v poradí:
    a. Konštanty
    b. Polia
    c. Konštruktory
    d. „Override“ metódy a „callbacks“
    e. Public metódy
    f. Private metódy
    g. Vnorené triedy a rozhrania
11. Pred a po znakoch + - * / = == != > >= < <= použiť medzeru.
12. Ak sa používajú lokálne premenné v cykloch, deklarovať ich výhradne v tele alebo hlavičke cyklu (riadiaca premenná pre FOR cyklus).
13. V parametroch funkcie definovať *Context* na prvej pozícii a *Callback* na poslednej pozícii.
14. Komentáre písať vo formáte *Javadoc*.

<u>Pravidlá pre jazyk C</u>:

1. Všetky názvy premenných, tried a aj všetky komentáre písať VÝHRADNE v anglickom jazyku.
2. Procedúry (resp. funkcie) a premenné písať vo formáte *this_piece_of_junk* (oddeľovačom jednotlivých slov nech je podčiarknik).
3. Používať dostatočne výstižné a stručné mená pre premenné aj procedúry.
4. Pred a po znakoch + - * / = == != > >= < <= { vždy písať medzeru.
5. Pri deklarácii pointra sa znak * píše spolu s názvom premennej a nie spolu s typom, na ktorý ukazuje.
6. Názvy globálnych premenných by mali mať prefix *g_* .
7. Názvy globálnych konštánt a makier by sa mali písať veľkým písmenom a slová oddeľovať podčiarkníkom.

8. Pre číselníkové typy (enum) by mali byť názvy premenných písané veľkým písmenom.
9. Za ukončujúcou množinovou zátvorkou } sa môžu dať komentáre vo formáte /* .. */.
10. V riadku by malo byť najviac 78 znakov.
11. Ak sa dá, nepoužívať príkaz *goto*.
12. Príkazy vetvenia *else* a *else if* písať po medzere, ktorá je za ukončovacou množinovou zátvorkou }.
13. Ak sa vo vetvení pomocou *switch* v konkrétnej možnosti vyskytne viacero príkazov, treba tieto príkazy (okrem príkazu *break*) ohraničiť množinovými zátvorkami do bloku {}.
14. V jednom riadku by mal byť len jeden výraz.
15. Ak je zrejmé, že sa bude cast-ovať medzi *enum* a iným typom, je lepšie *enum* radšej nepoužiť.
16. Nepoužívať makro, ak to nie je vyslovene nutné (môžu s ním byť spojené viaceré problémy). Makro sa odporúča použiť, ak by bolo telo k nej ekvivalentnej funkcie krátke.
17. Všetky premenné by sa vždy a za každých okolností mali inicializovať.
18. Komentáre by mali byť vo forme „príbehu“ opisujúceho systém.

Pravidlá pre jazyk Python:

1. Používať výhradne verziu *Python 2*.
2. Súbory kódovať vo formáte UTF-8.
3. Pre *indentation level* (oddeľovanie logických celkov – ekvivalent pre oddeľovanie v Jave sú {}) používať tabulátor (pozn. tabulátor musí korešpondovať 4 medzerám).
4. Dĺžka riadku by nemala presiahnuť 80 znakov. Maximum je 120 znakov.
5. Importovanie modulov písať v oddelených riadkoch.
6. Definície funkcií oddeľovať dvoma prázdnymi riadkami.
7. Pri definovaní reťazca znakov prioritne používať úvodzovky, apostrofy iba v nevyhnutných prípadoch, kedy si to situácia vyžaduje.
8. Dokumentačné komentáre uzavrieť tromi pármi úvodzoviek.
9. Každá metóda by *mala mať* dokumentačný komentár.
10. Moduly pomenovávať malými písmenami.
11. Triedy pomenovávať vo formáte *UpperCamelCase*.
12. Premenné, metódy a funkcie pomenovávať malými písmenami a slová oddeľovať podčiarkovníkmi.
13. V definícii argumentov inštančných metód je argument *self* vždy na prvej pozícii.
14. V definícii argumentov statických metód je argument *cls* vždy na prvej pozícii.
15. Konštanty pomenovávať veľkými písmenami a slová oddeľovať podčiarkovníkmi.