Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Tímový projekt
Robocup
Dokumentácia robotického hráča Jim
Riadenie

Bc. Peter Filípek
Bc. Metod Rybár
Bc. Michal Segeč
Bc. Juraj Šimek
Bc. Martin Vrabec
Bc. Miroslav Wolf

Tím č. 8:	Infinity
Vedúci projektu:	Ing. Ivan Kapustík
Predmet:	Tímový projekt I
Ročník:	2014/2015
Akademický rok:	1.
Mailový kontakt:	team8@centrum.sk

Log zmien v dokumentácií

Dátum zmeny	Názov sekcie	Autor	Verzia
12. október 2014	Ako čítať a písať túto dokumentáciu	Metod Rybár	0.1
12. október 2014	Popis základných častí	Metod Rybár	0.1
12. október 2014	Monitorovanie zmien v dokumentácií	Metod Rybár	0.1
15. október 2014	Úvod 1	Metod Rybár	0.2
15. október 2014	Role členov tímu a podiel práce	Metod Rybár	0.2
15. október 2014	Aplikácie manažmentov	Metod Rybár	0.2
15. október 2014	Sumarizácie šprintov	Metod Rybár	0.2
15. október 2014	Používané metodiky	Metod Rybár	0.2
15. október 2014	Globálna retrospektíva ZS/LS	Metod Rybár	0.2
15. október 2014	Titulná strana	Metod Rybár	0.2
21. október 2014	Popis základných častí	Metod Rybár	0.3
3. november 2014	Odstránenie návodu k dokumentácií	Metod Rybár	0.4
3. november 2014	Úvod 1	Metod Rybár	0.4
3. november 2014	Odstránenie prázdych kapitol	Metod Rybár	0.4
16. november 2014	Manažment kvality 13	Juraj Šimek	0.5
16. november 2014	Konvencie písania kódu 15	Juraj Šimek	0.5
16. november 2014	Manažment komunikácie 12	Michal Segeč	0.6
16. november 2014	Manažment podpory vývoja a integrácie 11	Martin Vrabec	0.6
16. november 2014	Manažment rizík 10	Peter Filípek	0.6

16. november 2014	Manažment dokumentovania 9	Metod Rybár	0.6
17. november 2014	Manažment dokumentovania 9	Metod Rybár	0.7
17. november 2014	Manažment dokumentovania v tíme 9.1	Metod Rybár	0.7
17. november 2014	Vedenie tímu 7	Metod Rybár	0.7
19. november 2014	Úprava konvencií písania ZK 15	Juraj Šimek	0.8
19. november 2014	Manažment rozvrhu 8	Miroslav Wolf	0.9
19. november 2014	Metodika pre vytváranie vetiev v prostredí Bitbucket a importovanie projektu v prostredí Eclipse 11.6	Michal Segeč	0.9
19. november 2014	Metodika značkovania kódu v CodeReview 11.7	Miroslav Wolf	0.9
19. november 2014	Retrospektívy k šprintom 5	Metod Rybár	0.9
19. november 2014	Retrospektíva k prvému šprintu 3.1	Michal Segeč	0.9
19. november 2014	Retrospektíva k druhému šprintu 3.2	Peter Filípek	0.9
19. november 2014	Retrospektíva k tretiemu šprintu 3.3	Miroslav Wolf	0.9
19. november 2014	Globálna retrospektíva k zimnému semestru 4	Michal Segeč	0.9
19. november 2014	Úvod 1	Metod Rybár	1.0
19. november 2014	Podiel jednotlivých členov tímu na projekte 2	Metod Rybár	1.0

8. december 2014	Metodika pre vytváranie automatických testov v nástroji Bamboo 11.1	Michal Segeč	1.1
8. december 2014	Metodika tvorby testov v projekte JIM 14	Peter Filípek	1.2
9. december 2014	Úpravy manažmentu kvality 13	Juraj Šimek	1.2
9. december 2014	Retrospektíva k štvrtému šprintu 3.4	Metod Rybár	1.2
9. december 2014	Retrospektíva k piatemu šprintu 3.5	Juraj Šimek	1.2
9. december 2014	Plán pre zimný semester **??**	Miroslav Wolf	1.3
9. december 2014	Zhrnutie zimného semestra **??**	Miroslav Wolf	1.3
9. december 2014	Plán pre letný semester **??**	Miroslav Wolf	1.3
9. december 2014	Podiel jednotlivých členov tímu na projekte 2	Metod Rybár	1.4
10. december 2014	Presunutie plánov do dokumentácie k inžinierskemu dielu	Metod Rybár	1.5
2. máj 2015	Manažment kvality v letnom semestri 13.8	Juraj Šimek	1.6
2. máj 2015	Retrospektívy k šprintom v letnom semetri 5	Juraj Šimek	1.7
2. máj 2015	Retrospektíva k šiestemu šprintu 5.1	Michal Segeč	1.7
2. máj 2015	Retrospektíva k siedmemu šprintu 5.2	Miroslav Wolf	1.8

2. máj 2015	Retrospektíva k ôsmemu šprintu 5.3	Martin Vrabec	1.9
2. máj 2015	Retrospektíva k deviatemu šprintu 5.4	Peter Filípek	1.9
2. máj 2015	Retrospektíva k desiatemu šprintu 5.5	Metod Rybár	1.9
2. máj 2015	Globálna retrospektíva k letnému semestru 6	Miroslav Wolf	1.9
3. máj 2015	Formátovanie dokumentácie	Juraj Šimek	2.0
4. máj 2015	Retrospektíva k jedenástemu šprintu 5.6	Juraj Šimek	2.1
6. máj 2015	Zmazanie exportov úloh, pridanie manažérskych úloh a tabuľky s podielom prác	Juraj Šimek	2.2
13. máj 2015	Retrospektíva k jedenástemu šprintu 5.6 - pridaný burndown chart	Juraj Šimek	2.3
17. máj 2015	Podiel prác v LS 2 - pridaný burndown chart	Juraj Šimek	2.4

Obsah

1 Úvod

Posledná úprava	Metod Rybár
Platné od	3. november 2014
Poznámky	

RoboCup je medzinárodná súťaž autonómnych robotov, ktorá je určená na podporu výskumu a výučby robotiky a umelej inteligencie.

Každý tím sa v rámci obmedzení, určených pravidlami hry futbal a špecifikami simulačného prostredia, snaží vytvoriť čo najlepšieho hráča. Mužstvo, vytvorené z takýchto hráčov, by malo vyhrať nad mužstvom súpera. O súťaži a doterajšej činnosti je možné dozvedieť sa na stránke STU turnaj v simulovanom robotickom futbale (www.fiit.stuba.sk/robocup).

Simulácia futbalu pôvodne prebiehala iba v dvoch rozmeroch. Pre zvýšenie reálnosti simulácie bolo vytvorené 3D simulačné prostredie, ktoré rozširuje možnosti hry. 3D simulačné prostredie sa pomerne výrazne líši od doposiaľ používaného 2D prostredia, a to jednak spôsobom simulácie, ale hlavne možnosťami ktoré poskytuje hráčom. Na rozdiel od 2D simulácie ide o simuláciu jednotlivých pohybov humanoidného robota.

Na fakulte informatiky a informačných technológií sa vývinu robota venuje tím zložený z jej študentov v rámci predmetu Tímový projekt už od roku 2000. Do roku 2006 vývoj robota prebihal v 2D prostredí, od roku 2006 prebieha vývoj v 3D prostredí.

Hlavným cieľom projektu je vylepšiť doterajšieho hráča pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím.

Táto dokumentácia obsahuje popis postupu práce na projekte, výsledky jednotlivých šprintov, ich priebeh a zhodnotenie. Vyhodnocujú sa v nej problémy s úlohami a ich splnenie.

Ďalej táto dokumentácia obsahuje popis úloh jednotlivých členov tímu, metodiky vytvorené k práci na projekte a zhodnotenie ich aplikovania. Dokumentácia sa nevenuje technickej časti projektu.

Nasledovný zoznam uvádza úlohy jednotlivých členov tímu a s nimi súvisiace kapitoly dokumentácie, ktoré vypracovali a za ktoré sú zodpovední. V rámci týchto kapitol sú uvedené aj metodiky týkajúce sa danej oblasti manažmentu:

- Metod Rybár - Vedeniu tímu 7

- Miroslav Wolf - Manažment rozvrhu 8
- Metod Rybár - Manažment dokumentovania 9
- Peter Filípek - Manažment rizík 10
- Martin Vrabec - Manažment podpory vývoja a integrácie 11
- Michal Segeč - Manažment komunikácie 12
- Juraj Šimek - Manažment kvality 13

2 Podiel jednotlivých členov tímu na projekte

Posledná úprava	Metod Rybár
Platné od	19. november 2014
Poznámky	

Tabuľka 2: Percentuálny podiel práce v zimnom semestri

Peter Filípek	**Metod Rybár**	**Michal Segeč**	**Juraj Šimek**	**Martin Vrabec**	**Miroslav Wolf**
16,5	19	15	21,5	12	16

Tabuľka 3: Percentuálny podiel práce v letnom semestri

Peter Filípek	**Metod Rybár**	**Michal Segeč**	**Juraj Šimek**	**Martin Vrabec**	**Miroslav Wolf**
20,54	14,59	17,12	19,82	11,53	16,4

3 Retrospektívy k šprintom v zimnom semestri

Posledná úprava	Metod Rybár
Platné od	19. november 2014
Poznámky	

3.1 Retrospektíva k prvému šprintu

Posledná úprava	Michal Segeč
Platné od	19. november 2014
Poznámky	

V tomto šprinte sme sa hlavne sústreďovali na vytvorenie stabilných základov pre projekt, čiže oboznámenie sa so základnými nástrojmi, ktoré budeme využívať, importovaním projektu, vytvorením web stránky, kde budeme umiestňovať všetky dokumenty a vytvorenie konvencie pre dokumentáciu, ktorú budeme upravovať počas celého trvania projektu. Taktiež sme urobili základnú analýzu zdrojového kódu minulých ročníkov (tab. 4). Všetky úlohy sa podarilo splniť v požadovanom čase.

3.1.1 ROBOCUPTP-2 - Konvencia pre dokumentáciu

Úloha bola pridelená Bc. Metodovi Rybárovi, ktorý bol riešiteľom a zodpovedným za splnenie úlohy.

Pri vytváraní štandardu dokumentácie sa myslelo hlavne na kompatibilitu a prehľadnosť zmien, keďže na projekte sa pracuje dlhodobo a neustále sa mení. Preto sme sa rozhodli využiť technológiu Latex, ktorá zabezpečuje prenositeľnosť dokumentácie medzi platformami. Na monitorovanie zmien sme zaviedli verziovanie a zaznamenávanie zmien. Rovnako sme vytvorili štandardy pre nadpisy, tabuľky a obrázky. Štandard dokumentácie sa bude priebežne vyvíjať a dopĺňať tak, aby zabezpečoval všetky aktuálne potreby tímu. Samotnú dokumentáciu neuvádzame, keďže jej finálna forma bude dokumentáciou celého projektu a bude zverejnená priebežne. Priebežná verzia je uverejnená na web stránke nášho tímu.

Tabuľka 4: Úlohy pre prvý šprint

Číslo úlohy	**Zhrnutie úlohy**	**Riešiteľ**	**Zodpovedný**
ROBOCUPTP-2	Konvencia pre dokumentáciu	Metod Rybár	Metod Rybár
ROBOCUPTP-6	Rozbehať Jiru	Miroslav Wolf	Miroslav Wolf
ROBOCUPTP-8	Prezrieť diplomové práce a zahraničné tímy	Všetci	Michal Segeč
ROBOCUPTP-13	Príprava web stránky tímu	Martin Vrabec	Martin Vrabec
ROBOCUPTP-14	Zoznámiť sa s gitom a importovať si projekt	Všetci	Michal Segeč
ROBOCUPTP-15	Identifikácia a vymazanie nepotrebných častí - odstrániť balík garbage	Juraj Šimek	Juraj Šimek
ROBOCUPTP-16	Vytvoriť zoznam zakomentovaného kódu	Peter Filípek, Miroslav Wolf, Michal Segeč	Peter Filípek
ROBOCUPTP-17	Úprava logovania (funkcionalita, dokumentácia, vybrané balíky)	Juraj Šimek, Metod Rybár	Juraj Šimek

V tomto šprinte sme sa hlavne sústreďovali na vytvorenie stabilných základov pre projekt, čiže oboznámenie sa so základnými nástrojmi, ktoré budeme využívať, importovaním projektu, vytvorením web stránky, kde budeme umiestňovať všetky dokumenty a vytvorenie konvencie pre dokumentáciu, ktorú budeme upravovať počas celého trvania projektu. Taktiež sme urobili základnú analýzu zdrojového kódu minulých ročníkov.

3.1.2 ROBOCUPTP-6 - Rozbehať Jiru

Úloha bola pridelená Bc. Miroslavovi Wolfovi, ktorý bol riešiteľom a zodpovedným za splnenie úlohy.

Bolo potrebné oboznámiť sa s používaním tohto nástroja a vybaviť všetky práva, aby bolo možné vytvoriť projekt pre náš tím a aby všetci členovia tímu mohli narábať so všetkými funkciami ako pridávanie taskov, editovanie atď. Úlohy sme do Jiry úspešne pridali a nalogovali časy, ktoré sme strávili pri vykonávaní jednotlivých úloh. Výsledkom šprintu je burndown chart, ktorý avšak neodzrkadľuje skutočné plnenie úloh, keďže pre tento šprint to pre nás bola experimentálna fáza, kde sme nevedeli zadávať jednotlivé story pointy a časy pre úlohy a tak krivka grafu nemenila svoju podobu. Je preto potrebné naučiť sa zadávať správne úlohy aj s vyhradeným časom pre danú úlohu a priebežne zaznamenávať odrobený čas na daných úlohách.

3.1.3 1.3 ROBOCUPTP-8 - Prezrieť diplomové práce a zahraničné tímy

Cieľom úlohy bolo analyzovať bakalárske a diplomové práce študentov bývalých ročníkov a taktiež zahraničné tímy, ktoré sa zúčastňujú každoročnej celosvetovej súťaže Robocup. Každá analýza obsahuje nami vybrané zaujímavé časti, ktoré nás, či už v prácach, alebo pri zahraničných tímoch zaujali a ktoré by sa potenciálne dali využiť pri našom upravovaní agenta.

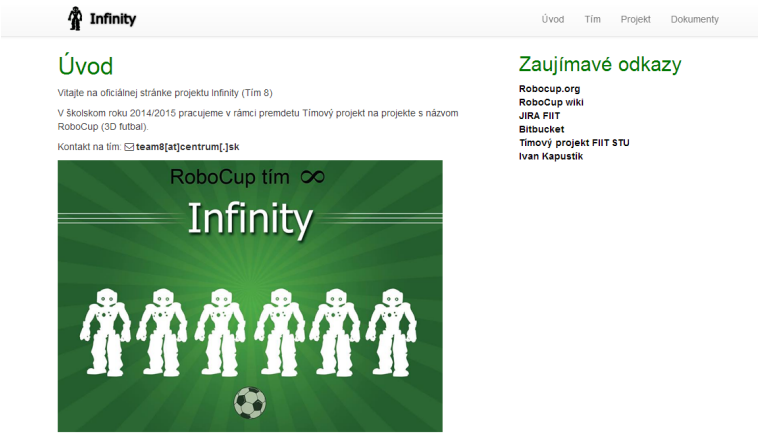
Úloha bola určená pre celý tím, zodpovedným za splnenie úlohy bol Bc. Michal Segeč. Analýzy môžte vidieť v Dokumentácií k inžinierskemu dielu.

3.1.4 ROBOCUPTP-13 – Príprava web stránky tímu

Úloha bola pridelená Bc. Martinovi Vrabcovi, ktorý bol zároveň aj zodpovedným za splnenie úlohy.

Bolo potrebné vybaviť prístupy na server, ktoré Martin dohodol s Ing. Petrom Lackom, PhD. Stránka je už úspešne zavesená na webovom sídle.

Obsahuje všetky náležité informácie o tíme, o jej členoch, stručný popis problému, ktorá riešime a plán na zimný semester. Sú uvedené taktiež zaujímavé odkazy, týkajúce sa Robocupu, ale taktiež na nástroje, ktoré využívame pri tomto projekte. Stránka je priebežne dopĺňaná, každý týždeň sa pridáva nová zápisnica z najaktuálnejšieho stretnutia tímu s vedúcim projektu a vždy po vytvorení dôležitého dokumentu je daný dokument zverejnený na stránke. (obr. 1). Odkaz na stránku: `http://labss2.fiit.stuba.sk/TeamProject/2014/team08is-si/`.



Obr. 1: Webová stránka tímu

3.1.5 ROBOCUPTP-14 – Zoznámiť sa s gitom a importovať si projekt

Úloha bola určená pre celý tím a zodpovedný za jej splnenie bol Bc. Michal Segeč.

Prvým dôležitým krokom bolo si importovať projekt, aby sme mohli vykonávať jednotlivé úlohy stanovené pre tento šprint. Bc. Juraj Šimek vytvoril krátky návod ako si importovať projekt. Viac k postupom si možno prečítať v 11.

Pri práci s projektom využívame taktiež Bitbucket a Git. Preto aby sme dokázali projekt upravovať a sprístupniť upravenú verziu pre ostatných sme potrebovali sa naučiť pracovať s Gitom. Za týmto zámerom sme si dohodli tímové stretnutie, kde nám Bc. Metod Rybár vysvetlil základné funkcie tohto nástroja. Ďalšie hlbšie študovanie git-u sme si ponechali na každého osobné uváženie.

3.1.6 ROBOCUPTP-15 - Identifikácia a vymazanie nepotrebných častí - odstrániť balík garbage

Úloha bola pridelená Bc. Jurajovi Šimekovi, ktorý bol zároveň zodpovedný za jej splnenie.

V projekte Jim sa nachádzal balík sk.fiit.jim.garbage. Tento balík obsahoval všetky balíky, ktoré neboli potrebné pre ďalšie fungovanie agenta. Súbory v tomto balíku boli zastarané a používali sa v čase keď veľká časť projektu Jim bola implementovaná v jazyku Ruby. Slúžili hlavne na integráciu častí napísaných v Ruby a častí napísaných v Jave. Nakoľko súčasná implementácia agenta Ruby nepoužíva, rozhodli sme sa balík odstrániť a zvýšiť prehľadnosť zdrojového kódu. Balík garbage vznikol ako činnosť tímu Gitmen, ktorí prepisovali súčasti agenta z Ruby do Javy a do tohto balíku vložili balíky, ktoré už nebude potrebné používať. Neboli však odstránené závislosti medzi týmito balíkmi a ostatnými balíkmi, ktoré používa agent implementovaný v projekte Jim. Jednoduché odstránenie balíka nebolo možné, nakoľko vyvolalo sériu chýb. Museli sme preto analyzovať závislosti medzi používanými balíkmi a tými balíkmi, ktoré boli v balíku garbage. Následne bolo treba odstrániť všetky tieto závislosti a až potom bolo možné balík garbage odstrániť.

3.1.7 ROBOCUPTP-16 - Vytvoriť zoznam zakomentovaného kódu

Úloha bola pridelená Bc. Petrovi Filípekovi, Bc. Michalovi Segečovi a Bc. Miroslavovi Wolfovi. Zodpovedný za splnenie úlohy bol Bc. Peter Filípek.

Cieľom úlohy bolo postupne prejsť celý projekt Jim a zdokumentovať časti kódu, ktoré sú zakomentované. Dôvod, prečo to bolo potrebné urobiť je ten, že zdrojový kód obsahuje veľa balíčkov a v nich vnorených tried, a kód je veľmi neprehľadný. Pri vytvorenom zozname sme sa snažili aj zistiť, či daný kus zakomentovaného kódu je možné vymazať, alebo nie. Mnohokrát to nebolo možné jednoznačne určiť. Výstupom úlohy je zoznam, pričom každý kúsok kódu má svoj samostatný záznam vo forme balíček -> trieda -> popis zakomentovaného kódu.

Dokument momentálne neuvádzame, keďže sme si stanovili úlohu pre druhý šprint, v ktorej budeme pokračovať vymazávaním týchto nepotrebných časti kódu. Následne bude vytvorený dokument, ktorý bude spájať úlohy z týchto 2 šprintov.

3.1.8 ROBOCUPTP-17 - Úprava logovania (funkcionalita, dokumentácia, vybrané balíky)

loha bola pridelená Bc. Metodovi Rybárovi a Bc. Jurajovi Šimekovi. Zodpovedný za splnenie úlohy bol Bc. Juraj Šimek.

Cieľom úlohy bolo vytvoriť nový logger, keďže doterajšie logovanie v projekte Jim nebolo jednotné. K novému loggeru bola vytvorená dokumentácia a bolo upravených niekoľko vybraných balíkov, kde sa otestoval nový logger. V druhom šprinte sa chceme zamerať na kompletné prerobenie starého loggera v projekte Jim na nový. Bol vytvorený nový logger, ktorý ako základ využíva štandardný logger z JavaAPI. Ten je obalený triedou JLog. Na formátovanie výstupu do konzoly a do HTML súboru sa používajú triedy JimConsoleFormatter a JimHtmlFormatter, ktoré sú odvodené od triedy Formatter slúžiacej na formátovanie výstupu loggera. Viac môžte vidieť v Dokumentácií k inžinierskemu dielu.

3.2 Retrospektíva k druhému šprintu

Posledná úprava	Peter Filípek
Platné od	19. november 2014
Poznámky	

V tomto šprinte sme sa zamerali najmä na úlohy ktoré nadväzujú na úlohy z predošlého šprintu. Úlohou bolo dokončiť loger tak, aby vyhovoval potrebám logovania. V súlade s touto úbohou bola do šprintu vybraná aj ďalšia úloha tykajúca sa logera. Táto úloha bola zameraná na odstránenie starých správ pre loger, ich úpravu pre potreby nového logera, prípadne logovanie nových správ. Celý tento proces úpravy logera a správ určených pre loger bol navrhnutý za účelom zjednotenia a sprehľadenia týchto správ. Ďalšia z vybraných úloh bola zameraná na odstránenie, alebo prípadné ponechanie a označenie zakomentovaných časti kódu, ktorých význam sa analyzoval v predošlom šprinte. Cieľom tejto úlohy bolo zlepšiť čitateľnosť kódu a to tým, že sa tieto časti odstránia alebo vhodne označia a zaradia v kóde. Ďalšia úloha, ktorá bola vybraná pre tento šprint, nadväzovala na analýzy diplomových prác a zahraničných tímov z predošlého šprintu. Cieľom tejto úlohy bolo vybrať z týchto analýz zaujímavé časti, vhodne ich zanalyzovať a popísať, aby bolo možné v ďalších šprintoch niektoré úlohy implementovať (tab. 5).

Tabuľka 5: Úlohy pre druhý šprint

Číslo úlohy	**Zhrnutie úlohy**	**Riešiteľ**	**Zodpovedný**
ROBOCUPTP-69	Konvencia pre dokumentáciu a úpravu logovania	Juraj Šimek	Juraj Šimek
ROBOCUPTP-70	Pokračovanie prác v úprave logovania	Všetci	Metod Rybár
ROBOCUPTP-71	Odstránenie nepotrebných častí zakomentovaného kódu	Michal Segeč, Peter Filípek, Miroslav Wolf	Peter Filípek
ROBOCUPTP-72	Výber zaujímavých častí z analýzy záverečných prác a projektov	Všetci	Michal Segeč

3.2.1 ROBOCUPTP-69 - Konvencia pre dokumentáciu a úpravu logovania

Úloha bola pridelená Bc. Jurajovi Šimekovi, ktorý bol riešiteľom a zodpovedným za splnenie úlohy.

Táto úloha priamo nadväzuje na úlohu ROBOCUPTP-17 - Úprava logovania (funkcionalita, dokumentácia, vybrané balíky) 3.1.8 z predošlého šprintu. Loger bol rozšírený o úrovne pre logovanie a k tomu bola vytvorená príslušná metodika, aby logy boli zaradené do správnej úrovne. Toto rozšírenie logera bolo doplnené aj do dokumentácie, ktorá bola aktualizovaná a zverejnená na webe.

3.2.2 ROBOCUPTP-70 - Pokračovanie prác v úprave logovania

Na riešení úlohy sa podieľali všetci členovia tímu, Bc. Metod Rybár bol zodpovedný za splnenie úlohy.

Táto úloha priamo nadväzuje na úlohu ROBOCUPTP-17 - Úprava logovania (funkcionalita, dokumentácia, vybrané balíky) 3.1.8 z predošlého šprintu a úlohu ROBOCUPTP-69 - Konvencia pre dokumentáciu a úpravu logovania 3.2.1 z aktuálneho šprintu. Cieľom úlohy bolo prejsť jednotlivé

triedy v projekte Jim a upraviť logy pre starý loger tak, aby sa logovali do nového logera a zodpovedali zavedenej konvencii. Ďalším cieľom úlohy bolo prejsť system.outprint v projekte Jim a vhodné prepísať tak, aby sa logovali do logera podľa príslušnej úrovne a nevhodné odstrániť.

3.2.3 ROBOCUPTP-71 - Odstránenie nepotrebných častí zakomentovaného kódu

Úloha bola pridelená Bc. Petrovi Filípkovi, Bc. Michalovi Segečovi a Bc. Miroslavovi Wolfovi. Zodpovedný za splnenie úlohy bol Bc. Peter Filípek.

Táto úloha priamo nadväzuje na úlohu ROBOCUPTP-16 - Vytvoriť zoznam zakomentovaného kódu z predošlého šprintu. Cieľom úlohy bolo sprehľadniť kód tým, že sa odstránia nepotrebné zakomentované časti kódu. V prípade, že zakomentované časti kódu majú potenciál pri rozširovaní projektu, zostanú zachované, a budú vhodne označené čo zakomentované časti robia a na čo by sa dali využiť. Pri prechádzaní zakomentovaného kódu projektu Jim boli odstránené všetky testovacie výpisy, nastavovania premenných a hodnôt pre testovacie účely a logovanie stavov, ktoré boli zbytočné. Niektoré logy stavov boli ponechané a prepísané pre nový loger. Došlo aj k odstráneniu niekoľkých funkcií a tried, ktoré boli buď nepoužívané, alebo boli nahradené novou verziou. K úlohe vznikla aj dokumentácia, kde sú podrobnejšie popísané vykonané zmeny. Dokumentácia je zaradená do dokumentácie k inžinierskemu dielu.

3.2.4 ROBOCUPTP-72 - Výber zaujímavých častí z analýzy záverečných prác a projektov

Na riešení úlohy sa podieľali všetci členovia tímu, Bc. Michal Segeč bol zodpovedný za splnenie úlohy.

Táto úloha priamo nadväzuje na úlohu ROBOCUPTP-8 - Prezrieť diplomové práce a zahraničné tímy z predošlého šprintu. Cieľom úlohy bolo z analyzovaných diplomových prác a tímov vybrať zaujímavé časti, ktoré by sa dali implementovať do nášho projektu. Výsledkom úlohy sú dokumenty, ktoré podrobnejšie popisujú konkrétne časti z predošlej analýzy a zahŕňajú popis použitých algoritmov, prípadne aj kroky ktoré bude nutné podniknúť pri implementácii danej funkcie do nášho projektu. Tieto dokumenty sú zahrnuté v dokumentácii inžinierskeho diela.

3.2.5 Zhodnotenie druhého šprintu

Úlohy sa riešili priebežne počas celých dvoch týždňov vyhradených pre šprint. Na týždňovom stretnutí uprostred šprintu sme si prešli stav úloh ktorý je uvedená v tab. 6.

Tabuľka 6: Stav úloh pre druhý šprint v polovici šprintu

Číslo úlohy	**Zhrnutie úlohy**	**Stav úlohy**
ROBOCUPTP-69	Konvencia pre dokumentáciu a úpravu logovania	Splnené
ROBOCUPTP-70	Pokračovanie prác v úprave logovania	Čiastočne splnené
ROBOCUPTP-71	Odstránenie nepotrebných častí zakomentovaného kódu	Čiastočne splnené
ROBOCUPTP-72	Výber zaujímavých častí z analýzy záverečných prác a projektov	Čiastočne splnené

Väčšina úloh bola v rozpracovanom stave, okrem úlohy ROBOCUPTP-69 - Konvencia pre dokumentáciu a úpravu logovania, ktorá bola dokončená už v priebehu prvého týždňa, toto bolo podstatné vzhľadom na to že úloha ROBOCUPTP-70 - Pokračovanie prác v úprave logovania na ňu nadväzovala. Do konca šprintu boli všetky úlohy dokončené a aj zdokumentované okrem ROBOCUPTP-70 - Pokračovanie prác v úprave logovania.

Na poslednom stretnutí, kde sa uzatváral šprint, sa preberali aj návrhy na zlepšenia do ďalšieho šprintu. Problém bol najmä pri vytváraní úloh v Jire a logovaní práce na úlohách, čo malo negatívny vplyv na výsledok Burn down chart-u. Návrhy prijaté po prvom šprinte mali čiastočne pozitívny vplyv, ale výsledok nezodpovedal našej predstave a očakávaniu. Novo navrhnuté zmeny pre ďalší šprint by mali pomôcť odstrániť tento nedostatok. Ďalší nedostatok, ktorý sa odhalil pri uzatváraní šprintu bol ten, že nám chýbali exporty z týždňových stretnutí, ktoré by podrobne ukazovali progres práce v priebehu šprintu.

3.3 Retrospektíva k tretiemu šprintu

Posledná úprava	Miroslav Wolf
Platné od	19. november 2014
Poznámky	

Tabuľka 7: Úlohy pre tretí šprint

Číslo úlohy	**Zhrnutie úlohy**	**Riešiteľ**	**Zodpovedný**
ROBOCUPTP-73	Implementácia zero moment point	Metod Rybár	Miroslav Wolf
ROBOCUPTP-74	Zavedenie automatického testovania a značkovania kódu	Michal Segeč, Miroslav Wolf	Michal Segeč
ROBOCUPTP-75	Refaktoring kódu	Všetci okrem Metoda	Juraj Šimek
ROBOCUPTP-76	Finalizácia dokumentácie pre kontrolný bod	Všetci	Metod Rybár

V tomto šprinte sme sa zamerali najmä na úpravu kódu do konvencie, ale aj implementácie novej časti z analyzovaných prác a to zero moment point z diplomovej práce Jána Hudeca - Motorika hráca simulovaného futbalu. Okrem toho sme dokončovali všetky potrebné súčasti pre odovzdanie projektu pre kontrolný bod, kde každý z členov tímu napísal dokument k manažmentu svojej role, vytvorili sa metodiky a dokončovali sa aj ďalšie dokumenty k úlohám, ktoré ešte neboli dokončené. Keďže sme ešte nemali nástroj na značkovanie kódu a automatické testovanie, jedna z úloh bolo aj rozbehanie týchto nástrojov a vytvorenie metodík k nástrojom. (tab. 7)

3.3.1 ROBOCUPTP-73- Implementácia zero moment point

Úloha bola pridelená Bc. Metodovi Rybárovi, ktorý bol riešiteľom. Zodpovedným za splnenie úlohy bol Bc. Miroslav Wolf.

Táto úloha nadväzovala na úlohu z druhého šprintu, v ktorej sme z analyzovaných diplomových prác a zahraničných tímov vyberali zaujímavé časti. Z

diplomovej práce Jána Hudeca (Motorika hráca simulovaného futbalu), sme sa rozhodli implementovať zero moment point. Implementovať túto metódu sa do istej miery podarilo, no úlohu by bolo dobré ešte dopracovať, keďže niektoré funkcie boli písané v jazyku Ruby a tie by bolo dobré prepísať do Javy. Okrem toho úloha nebola ani otestovaná preto sa v dopracovaní tejto úlohy bude pokračovať aj ďalší šprint.

3.3.2 ROBOCUPTP-74- Zavedenie automatického testovania a značkovania kódu

Na riešení úlohy sa podieľali členovia tímu, Bc. Miroslav Wolf a Bc. Michal Segeč, ktorý bol aj zodpovedný za splnenie úlohy.

Túto úlohu sme si rozdelili na dve pod-úlohy, pričom Miroslav sa venoval zavedeniu značkovania kódu a Michal zavedeniu automatických testov.

Značkovanie kódu bolo zavedené, na značkovanie budeme používať nástroj CodeReview, ktorý vznikol minulý rok v rámci tímového projektu. Ku značkovaniu kódu bola vytvorená aj metodika, ktorá opisuje používanie jednotlivých značiek. Táto dokumentácia sa nachádza v dokumentácii riadenia.

Pre automatické testy nám bol vytvorený projekt v nástroji Bamboo, ktorý bol aj prepojený s nástrojom Jira. Automatické testy sa dajú vytvárať, ale testy sa nedajú spúšťať nad jednotlivými konkrétnymi triedami a tak táto časť ešte musí byť dopracovaná. Úloha bola teda čiastočne splnená, keďže značkovanie kódu bolo zavedené a čiastočne aj automatické testovanie.

3.3.3 ROBOCUPTP-75- Refaktoring kódu

Na riešení tejto úlohy sa podieľali členovia tímu, Bc. Peter Filípek, Bc. Michal Segeč, Bc. Juraj Šimek a Bc. Martin Vrabec. Zodpovedný za túto úlohu bol Bc. Juraj Šimek.

V tejto úlohe sme prešli všetky balíčky projektu Jim a upravovali ich do konvencie. Kód sme upravovali podľa zavedenej konvencie pre písanie kódu. Tam, kde to bolo možné sme pridávali dokumentačné komentáre, snažili sme sa opísať najmä metódy, aby bolo jasné, ktorá na čo slúži. Okrem toho sme robili drobné úpravy, ktoré môžu napomôcť čitateľnosti kódu a zlepšenie kvality kódu. Napríklad telá podmienených príkazov a cyklov sme uvádzali do blokov, ak v blokoch neboli, čo mohlo viesť ku chybám. Kód sme aj lepšie formátovali pridaním medzier a odsadení. Úloha bola splnená.

3.3.4 ROBOCUPTP-76 - Finalizácia dokumentácie pre kontrolný bod

Na riešení tejto úlohy sa podieľali všetci členovia tímu. Zodpovedný za túto úlohu bol Bc. Metod Rybár.

V rámci tejto úlohy sa spravili dokumentácie k roliam v tíme, metodiky pre rôzne nástroje a dokončili sa ostatné nedokončené dokumenty. Boli vytvorené nasledovné dokumenty k roliam:

- Dokumentácia k vedeniu tímu 7
- Manažment rozvrhu 8
- Manažment dokumentovania 9
- Manažment rizík 10
- Manažment podpory vývoja a integrácie 11
- Manažment komunikácie 12
- Manažment kvality 13

Okrem toho boli dokončené ostatné dokumenty k úlohám, ktoré doteraz v dokumentácii inžinierskeho diela chýbali a rovnako aj metodiky k dokumentácii riedenia tímu. Dokumentácia k riadeniu a dokumentácia inžinierskeho diela boli tak pripravené pre odovzdanie kontrolného bodu. Úloha bola splnená.

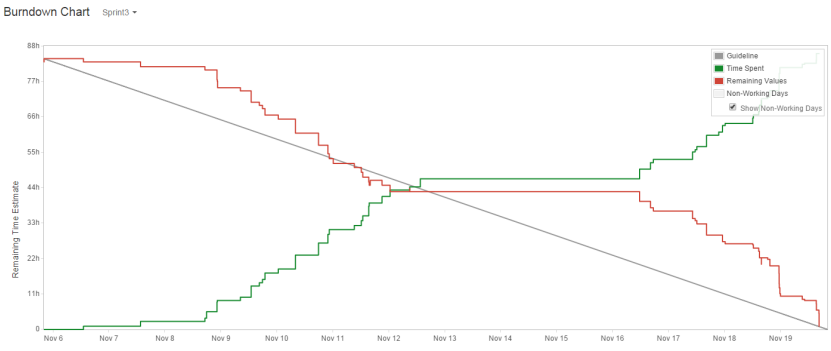
3.3.5 Zhodnotenie tretieho šprintu

Na úlohách sa pracovalo priebežne. Po prvom stretnutí tímu boli všetky úlohy v rozpracovanom stave. Úloha ROCOBUPTP-74 3.3.2 už bola v stave, kedy bola takmer hotová. Pre značkovanie už bol zavedený CodeReview a pre automatické testy Bamboo. Pri Bamboo však ešte vznikli nejaké nedostatky a k obom nástrojom bolo ešte nutné spraviť metodiky. Ostatné úlohy boli v stave rozpracovania.

Pri uzatváraní šprintu sme si zhodnotili priebeh práce tímu a prebrali nedostatky a návrhy do ďalšieho šprintu. Zhodnotenie úloh prebehlo nasledovne: Úlohy ROBOCUPTP-75 3.3.3 a ROBOCUPTP-76 3.3.4. Úloha ROBOCUPTP-73 3.3.1 nebola dokončená. Pre dokončenie tejto úlohy je

nutné prerobiť niektoré funkcie z jazyka Ruby do Javy a celú metódu otestovať. To sa nám počas tohto šprintu nepodarilo stihnúť, no úloha je v rozpracovanom stave a veľká časť je už implementovaná. Úloha ROBOCUPTP-74 3.3.2 bola čiastočne splnená - značkovanie kódu pomocou nástroja CodeReview sa úspešne podarilo zaviesť a bola k nemu aj vytvorená metodika ktorá sa nachádza v dokumentácii riadenia. Automatické testovanie v nástroji Bamboo však zatiaľ nefunguje pre jednotlivé triedy, bol nám však vytvorený projekt a všetky testy sa dajú spúšťať, len ešte treba doriešiť spúšťanie niektorých testov samostatne.

Po zhodnotení úloh sme prebrali aj nedostatky šprintu. Medzi hlavné nedostatky tento krát patril aj odhadovaný čas, kde sme pri úlohách ROBOCUPTP-73 3.3.1 a ROBOCUPTP-76 3.3.4 odhadli málo času, pričom úlohu ROBOCUPTP-73 3.3.1 sa nepodarilo dokončiť a na úlohe ROBOCUPTP-76 3.3.4 tím celkovo pracoval dlhšie ako bolo odhadované. Okrem toho sme sa zhodli, že dokumentácie k úlohám by mali byť vytvárané priebežne v rámci danej úlohy a nie dodávané až dodatočne, na čo bude potrebné si na úlohu vyhradiť aj dodatočný čas pre vytvorenie dokumentácie. Ďalej vznikli drobné nedostatky pri komunikácii tímu, pri značení prejdených balíčkoch pri refaktoringu v úlohe ROBOCUPTP-75 3.3.3, čo však našťastie nemalo väčšie následky. Zhodli sme sa aj na tom, že niektoré úlohy by bolo lepšie viac rozložiť na menšie časti vďaka čomu by sme aj lepšie vedeli odhadnúť čas pre vypracovanie úloh.



Obr. 2: Tretí šprint po druhom týždni - Burndown chart

V tomto šprinte sa nám podarili aj isté zlepšenia a to najmä v zapisovaní práce v nástroji Jira. Tím si priebežne logoval vykonanú prácu a Jira bola už lepšie nastavená a úlohy vhodnejšie vytvorené, čím sa nám podarilo vyriešiť

aj problém s burndown chartom a tento krát sa na ňom už dala sledovať práca tímu.

3.4 Retrospektíva k štvrtému šprintu

Posledná úprava	Metod Rybár
Platné od	9. december 2014
Poznámky	

Tabuľka 8: Úlohy pre štvrtý šprint

Číslo úlohy	**Zhrnutie úlohy**	**Riešiteľ**	**Zodpovedný**	**Stav úlohy**
ROBOCUPTP-77	Dokončenie ZMP a testovanie	Metod Rybár	Metod Rybár	Splnené
ROBOCUPTP-78	Automatické testy - Bamboo	Michal Segeč	Michal Segeč	Splnené
ROBOCUPTP-79	Oddelenie testovania	Peter Filípek	Miroslav Wolf	Splnené
ROBOCUPTP-80	Odstraňovanie nepoužívaných časti kódu	Juraj Šimek	Juraj Šimek	Splnené
ROBOCUPTP-81	Analýza Code Detectoru	Martin Vrabec, Peter Filípek, Miroslav Wolf	Martin Vrabec	Splnené
ROBOCUPTP-83	Odstraňovanie warning hlásení	Michal Segeč, Miroslav Wolf, Martin Vrabec	Miroslav Wolf	Splnené

Vo štvrtom šprinte sme sa zamerali na odstraňovanie nepotrebných častí hráča a rôznych chýb v kóde, ktoré by mohli v budúcnosti spôsobovať problémy a boli aj vývojárskym prostredím označené ako varovania. Zároveň sme nasadzovali automatické testy a dokončovali sme implementáciu Zero moment point, ktorá sa v predchádzajúcom šprinte ukázala komplikovanejšia ako sme plánovali.

3.4.1 ROBOCUPTP-77- Dokončenie ZMP a testovanie

Úloha bola pridelená Metodovi Rybárovi, ktorý na implementácií Zero moment point pracoval aj v predchádzajúcom šprinte. Po tom ako sa implementovali hlavné výpočtové funkcie sa v tomto šprinte tvoril nový HighSkill, ktorý Zero moment point využíval. Implementácia bola úspešná, ale ukázalo sa, že pri prechode z Ruby na Javu sa zmenili niektoré veci, kvôli ktorým v novom HighSkille napríklad hráč automaticky nevstal po tom ako spadol. Hráčova chôdza bola však očividne stabilnejšia, a preto v budúcnosti treba HighSkill upraviť tak, aby fungoval v Javovskej implementácií a zároveň sa Zero moment point môže využívať aj pri iných HighSkilloch.

3.4.2 ROBOCUPTP-78- Automatické testy - Bamboo

Úloha bola pridelená Michalovi Sgečovi. Nástroj bol úspešne nasadený, avšak pri refraktoringu kódu boli niektoré testy zakomentované a po ich odkomentovaní automatické testy prestali fungovať. Je preto nutné v ďalšom šprinte tento problém odstrániť.

3.4.3 ROBOCUPTP-79- Oddelenie testovania

Tejto úlohe sa venoval Peter Filípek. Testy presunul do balíčku tests a zároveň aj upravil classpath. Juraj Šimek našiel jeden test, ktorý však nebol Unit test a teda nebol ani presunutý. Zároveň sa pri tom odhalil problém so zakomentovanými testami spomínaný v ROBOCUPTP-78- Automatické testy - Bamboo 3.4.2.

3.4.4 ROBOCUPTP-80- Odstraňovanie nepoužívaných časti kódu

Úlohe sa venoval Juraj Šimek a v rámci nej odstránil najmä nevyužívane HighSkilly. Zároveň upravil niektoré triedy, v ktorých CodeDetector našiel problém a odstránil duplicity na výpočet vzdialenosti vektora. Kód celkovo zreduloval o približne 2000 riadkov kódu. V rámci úlohy vytvoril dokumentáciu k HighSkillom.

3.4.5 ROBOCUPTP-81- Analýza Code Detectoru

Úlohu mal na starosti Martin Vrabec a venovali sa jej Martin Vrabec, Peter Filípek a Miroslav Wolf. Najväčšiu časť odstránil Miroslav Wolf a bola

splnená rýchlejšie ako sa predpokladalo. Kód sa zanalyzoval pomocou pluginu CodeDetector do IDE Eclipse, ktorý slúži na alalýzu kódu a odhaľuje napríklad nevolané metódy. Tieto metódy sa skontrolovali a označili sa buď na odstránenie alebo ponechanie. Pri nerozhodných sa diskutovalo s tímom a rovnako sa o nich rozhodlii. Následne sa nepotrebný kód odstránil.

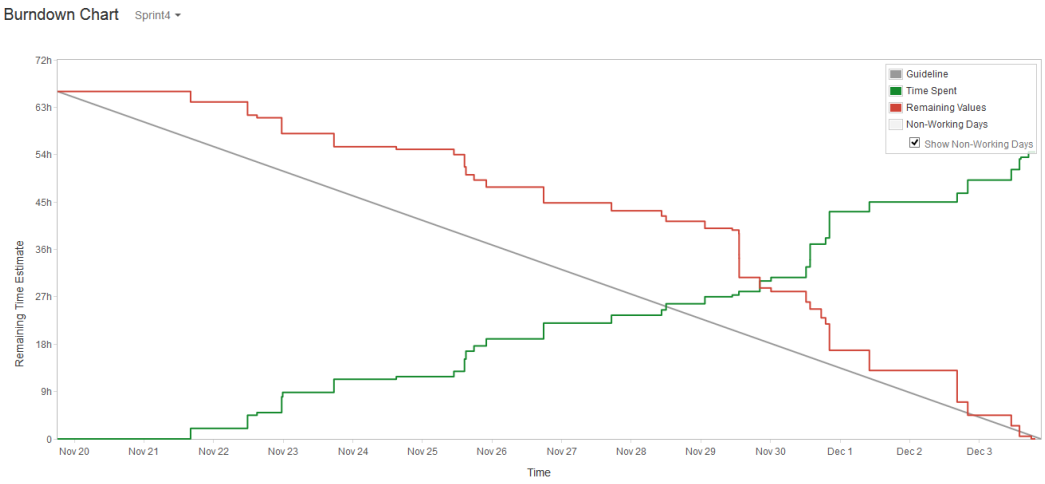
3.4.6 ROBOCUPTP-83- Odstraňovanie warning hlásení

Úlohu mal na starosti Miroslav Wolf a pracovali na nej Michal Segeč, Miroslav Wolf a Martin Vrabec. Na základe warningov detekovaným IDE Eclipse sa vykonala analýza ku a odstraňovali sa warningy, ktorá by v budúcnosti mohli spôsobovať problémy. Najčastejšie problémy boli nepoužité metódy alebo premenná, prípadne ich viditeľnosť. V agentovi sa odstránili všetky warningy, RoboCup Library a TestFramework boli nezmenené, kedže sa na nich ešte nepracovalo a môžu sa podstatne upravovať. Zredukovaný počet warningov bol z 936 na 582.

3.4.7 Zhodnotenie štvrtého šprintu

V rámci šprintu sa nám podarilo podstatne zredukovať počet riadkov kódu odstránením nepotrebných častí, implementovať Zero moment point, nasadiť automatické testovanie a rovnako sme zredukovali warningy a tým šancu budúcich problémov.

Ukázalo sa, že sme príliš podcenili implementáciu Zero moment point, a to najmä kvôli nášmu nepresnému odhadu urobených zmien pri prechode z Ruby do Javy minulý rok. Rovnako sa vyskytli problémy s pri presúvaní testov a automatickom testovaní, ktoré sa budú musieť v budúcnosti odstrániť.



Obr. 3: Štvrtý šprint po druhom týždni - Burndown chart

3.5 Retrospektíva k piatemu šprintu

Posledná úprava	Juraj Šimek
Platné od	9. december 2014
Poznámky	

V tomto šprinte sme sa zamerali najmä na finalizáciu dokumentácie pred odovzdaním výstupu tímového projektu zo zimného semestra. Okrem toho sme sa zamerali aj na nahranie wikipédie projektu na náš webový server, opravy automatického testovania a úpravy nastavení tak, aby ich bolo možné načítavať z konfiguračného súboru. Jednotlivé úlohy pre paity šprint sú uvedené v tabuľke 9.

3.5.1 ROBOCUPTP-130 – Úprava dokumentácie pre odovzdávanie

Keďže vytlačenie dokumentácie k riadeniu musí prebehnúť pred uzavretím šprintu, táto úloha v čase písania tejto retrospektívy nie je plne dokončená. Venovali sme sa najmä dokumentovaniu riadenia v tíme. Bc. Juraj Šimek upravil značnú časť kapitoly Manažment kvality a rozpracoval technickú dokumentáciu k nastaveniam, bola pridaná aj metodika testovania od Bc. Petra

Tabuľka 9: Úlohy pre tretí šprint

Číslo úlohy	**Zhrnutie úlohy**	**Riešiteľ**	**Zodpovedný**
ROBOCUPTP-130	Úprava dokumentácie pre odovzdávanie	Všetci okrem Martina	Miroslav Wolf
ROBOCUPTP-137	Oprava testov v Bamboo	Michal Segeč	Michal Segeč
ROBOCUPTP-138	Nahranie media wiki na server	Peter Filípek, Martin Vrabec	Martin Vrabec
ROBOCUPTP-141	Úprava Settings	Juraj Šimek	Juraj Šimek

Filípka. Bc. Miroslav Wolf vytvoril plán pre letný semester a v čase písania tejto retrospektívy mal rozpísané celkové zhrnutie semestra. Bc. Metod Rybár sa venoval finálnej úprave dokumentácie a má rozpracovanú dokumentáciu k Zero Moment Point. Celkovo sa neočakávajú problémy pri tejto úlohe a do konca šprintu bude určite splnená celá, nakoľko chýbajú už len časti z technickej dokumentácie.

3.5.2 ROBOCUPTP-137 – Oprava testov v Bamboo

Kvôli presunu testov do nového zdrojového priečinka (tests) sa poškodila činnosť automatického testovania v prostredí Bamboo. Túto skutočnosť mal zanalyzovať a opraviť Bc. Michal Segeč, ktorý tak aj spravil a automatické testovanie je znova plne funkčné.

3.5.3 ROBOCUPTP-138 - Nahranie media wiki na server

Wikipédia fakultného hráča bola úspešne nahratá na náš webserver a je všetkým prístupná. O nahratie wikipédie sa staral Bc. Peter Filípek a o nahratie mediawiki a úpravu servera tak, aby wikipédia fungovala sa staral Bc. Martin Vrabec.

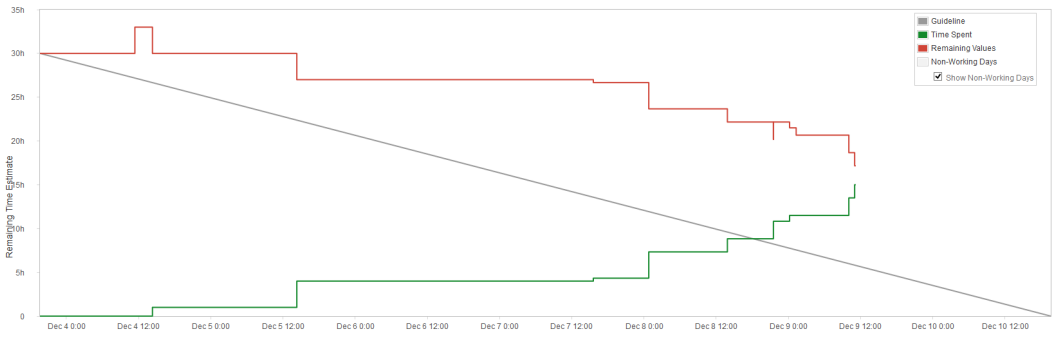
3.5.4 ROBOCUPTP-141 – Úprava Settings

Bc. Juraj Šimek úspešne reimplementoval nastavenia v projekte Jim. Teraz je možné načítavať nastavenia z konfiguračného súboru. Z triedy Settings.java

odbudlo množstvo zbytočných metód, ktoré boli implementované kvôli neaktuálnej spolupráci s jazykom Ruby.

3.5.5 Zhodnotenie piateho šprintu

Keďže tento šprint trval len jeden týždeň, vyberali sme si také úlohy, ktoré za tento týždeň stihneme dokončiť. Išlo najmä o úlohy pre sfinalizovanie odovzdávaného diela, ktoré vzniklo v rámci predmetu Tímový Projekt I. Podarilo sa nám dokončiť všetky úlohy. V čase písania tejto dokumentácie ešte nebola dokončená úloha úprav dokumentácie, nakoľko prezentácia z Manažmentu Informačných Systémov vyžaduje finálnu verziu dokumentácie k riadeniu. Z toho dôvodu nie je burndown chart na obrázku 4 kompletný.



Obr. 4: Piaty šprint v čase spísania retrospektívy.

4 Globálna retrospektíva k zimnému semestru

Posledná úprava	Michal Segeč
Platné od	19. november 2014
Poznámky	

1.kontrolný bod pozostáva z doposiaľ 3 šprintov. V rámci týchto 3 šprintov sme si celkovo stanovili 16 úloh, ktoré bolo treba vykonať. Ostatné úlohy boli zatiaľ zaradené do backlogu. Z týchto 16 vybraných úloh bolo splnených 14 a 2 boli čiastočne splnené.

V rámci 1.šprintu sme sa zamerali najmä na činnosti, ktoré využívame v rámci nášho projektu. Bola vytvorená dokumentácia, ktorá je priebežne dopĺňaná pri každom šprinte. Bolo taktiež potrebné získať práva a naučiť sa pracovať s nástrojom Jira, ktorý slúži na vedenie informácií o jednotlivých úlohách, vykonávaných v rámci projektu a taktiež čas, ktorý strávili všetci členovia pri plnení úloh. Analyzovali sme aj zahraničné tímy a diplomové práce študentov minulých rokov, aby sme sa inšpirovali a našli veci, ktoré by sa dali využiť a implementovať aj v rámci nášho projektu. Pripravili sme web stránku nášho tímu, kde každý týždeň pravidelne dopĺňame zápisnice z formálnych stretnutí ale aj ostatné dokumenty, ktoré vytvoríme pri práci na jednotlivých úlohách. Na stránke sa nachádza aj dokumentácia k riadeniu projektu a dokumentácia k inžinierskemu dielu. V rámci neformálneho stretnutia sme sa stretli s ostatnými členmi tímu, kde sme si prebrali základné funkcie nástroja git, ktoré už teraz aj využívame. Výzvou bolo aj importovanie projektu, ktoré zahŕňalo inštaláciu množstva potrebných, pre prácu na tímovom projekte však nevyhnutných programov. Z implementačnej časti sme sa sústredili na vymazanie nepotrebných časti kódu, pričom sa odstránil celý balík Garbage z projektu Jim. Zamerali sme sa taktiež na analýzu zdrojového kódu, pričom sme vytvorili zoznam zakomentovaného kódu a snažili sme sa analyzovať funkciu, ktorú mal zakomentovaný kód plniť. Poslednou úlohou bola úprava logovania, kde sme vytvorili nový logger. Logger bol zdokumentovaný a boli upravené vybrané balíky, kde sa starý logger prerobil na nový. V rámci tohto šprintu boli splnené všetky stanovené úlohy.

V rámci 2. šprintu sa pokračovalo v práci na úprave logovania. Dokumentácia pre logger bola upravená a doplnená o úrovne logovania, ktoré boli vytvorené, aby jednotlivé údaje, ktoré sú zalogované, boli zatriedené do odpovedajúcej kategórie. Tieto úrovne sú bližšie popísané v dokumentácií. Keďže

týmto bol nový logger definitívne hotový, začali sme aj s refaktoringom starého loggera v rámci projektu na nový logger aj s odpovedajúcimi úrovňami logovania. V rámci tohto šprintu bol starý logger kompletne prerobený na nový. V ďalšej úlohe sme nadviazali na výstup úlohy z predchádzajúceho šprintu, v ktorej sme analyzovali zakomentovaný kód v rámci projektu Jim. Tieto časti zakomentované kódu, ktoré sme označili ako nepotrebné, či už z dôvodu duplicity, nekompletnosti, alebo jednoducho nefunkčnosti sme odstránili a tieto úpravy zdokumentovali pre prípad budúceho možného využitia. Taktiež v ďalšej úlohe sme pokračovali v úlohe z minulého šprintu, kde sme každý mali za úlohu bližšie analyzovať vybrané zaujímavé časti z analyzovaných zahraničných tímov a diplomových prác študentov minulých ročníkov, ktoré by sa potenciálne dali využiť v rámci nášho tímového projektu. Jednotlivé analýzy sú súčasťou dokumentácie k inžinierskemu dielu. Všetky stanovené úlohy v rámci tohto šprintu boli splnené.

V 3.šprinte sme sa venovali implementácií ZMP – zero movement pointu. Túto úlohu sme splnili čiastočne. Zatiaľ čo základ ZMP je úspešne implementovaný a okomentovaný, je potrebné ešte reimplementovať funkciu pre rýchlu chôdzu z Ruby do Javy a pridať testovanie. Ďalšou úlohou bolo zavedenie automatického testovania a značkovania kódu. Značkovanie kódu bolo úspešné. Pre značkovanie využívame nástroj Code Review. Bola vytvorená aj metodika pre používanie tohto nástroja spolu s vysvetlením jednotlivých značiek, kedy je ich vhodne použiť. Čo sa týka zavedenia automatického testovania pomocou nástroja Bamboo, úloha je splnená čiastočne. V rámci nástroja Bamboo bol vytvorený projekt pre náš tím, ktorý bol prepojený s nástrojom Jira. Je avšak potrebné zamerať sa na vytvorenie testov, ktoré budú automaticky testovať náš projekt. V tomto šprinte sme sa z veľkej časti venovali refaktoringu, pričom sme sa zameriavali na metódy, ktoré nie sú volané v rámci projektu ale taktiež na prerábanie zdrojového kódu do konvencie, ktorá bola vytvorená pre náš tím. Úspešne sme prerobili všetky balíky projektu Jim do konvencie. Keďže úloha bola rozsiahla, treba ešte pokračovať v analyzovaní nepotrebných častí kódu a ich vymazávaní v prípade zbytočnosti. Poslednou úlohou, ktorú sme stanovili pre tento šprint bola finalizácia dokumentácie pre tento kontrolný bod. Bolo potrebné vytvoriť dokumentácie pre úlohy, ktoré predtým neboli zdokumentované a taktiež sa sústrediť na vytvorenie metodík a jednotlivých opisov manažérskych rolí všetkých členov tímu. Z daných úloh boli 2 splnené a 2 čiastočne splnené, pričom sa v čiastočne splnených úlohách bude pokračovať v ďalšom šprinte.

Keďže na projekte robotického futbalu Robocup doteraz pracovalo mnoho

tímov ale taktiež bakalárov a diplomantov po minulé roky, zdrojový kód sa stal veľmi neprehľadným a vyžaduje si mnoho úsilia porozumieť mu. Predtým než sa pustíme do samotnej hĺbkovej implementácie projektu je potrebné preto upraviť kód tak, aby ho bolo možné v budúcnosti bez problémov rozširovať.

Kód obsahoval množstvo duplicity, nevyužitých premenných, metód, zakomentovaného kódu a bol tak nekonzistentný. Taktiež sa využívali v rámci projektu 2 loggery, čo bolo zbytočné. V rámci týchto 3 šprintov sme sa sústreďovali preto na zjednocovanie všetkých častí zdrojového kódu, ale taktiež na analýzu jednotlivých časti kódu. Aby kódu bolo možné lepšie porozumieť, bolo potrebné okomentovať všetky triedy, public aj private metódy, prípadne premenné v rámci projektu Jim a upraviť ich do konvencie, aby spĺňali jednotnú formu. Taktiež sa v kóde vyskytovalo mnoho zbytočných výpisov, ktoré boli v prípade užitočnosti prerobené na logované hodnoty, alebo boli odstránené. Aby sme odstránili aj zbytočnosť používania 2 loggerov, bol vytvorený jeden jednotný logger, ktorý využívame v celom projekte.

Pre úplne prerobenie kódu do formy, ktorá bude jednoducho rozširovateľná je avšak potrebné ešte vykonať viaceré úpravy, ako napr. odstránenie varovných hlásení, ktorých sa v projekte vyskytujú stovky a odstrániť nevyužívané časti kódu, ktoré zostali. Na tieto úlohy sa zameriame ešte v budúcej práci.

V rámci plnenia úloh sa často vyskytli nové neočakávané problémy a tak sa stalo, že nám vznikla nová úloha, ktorú bolo treba zaradiť medzi budúce úlohy. Tieto úlohy sme zaradili do backlogu k ostatným úlohám na vykonanie.

Agent vo svojom momentálnom stave je nedokonalý a jeho pohyb je veľmi labilný a nepoužiteľný pre turnaj. Po vykonaní celkových úprav nad zdrojovým kódom sa chceme venovať samotným implementačným úpravám nižších skillov a prípadne vyšších skillov (t.j. stratégia a taktika) ktoré ovplyvnia celkový pohyb agenta a zlepšia ho natoľko, aby bolo možné sa aj v budúcich rokoch prihlásiť na turnaj v robotickom futbale. V budúcej prácu sa chceme taktiež zamerať na vytvorenie unit testov pre projekt, na stabilizáciu a finalizáciu pohybov a iné vylepšenia.

V rámci šprintov si každý člen v nástroji Jira zaznamenával úlohy, ktoré vykonal spolu s časom, ktorý strávil pri plnení danej úlohy. V úvodnom šprinte sme neboli dostatočne oboznámený s nástrojom Jira a burndown chart, ktorý mal zobrazovať krivku plnenia úloh nezobrazuje reálne strávenie času nad jednotlivými úlohami. Z tohto problému sme sa poučili a všetky úlohy sú v systéme už zadávané správne. Problém avšak vzniká s logovaním

časov jednotlivých úloh. Nie každý si zaznamenávame presne čas, kedy sme pracovali na danej úlohe a potom aj graf odzrkadľujúci čas strávený nad úlohami je skreslený. Je preto nevyhnutné, na čo si dáme pozor aj v budúcich šprintoch, aby si každý priebežne zaznamenával strávený čas na úlohe.

Čo sa týka odhadovania zložitosti vykonania jednotlivých úloh v šprinte, podarilo sa nám čas odhadovať pomerne presne. V prípade, že člen tímu, ktorý mal vyhradenú prácu na konkrétnej úlohe a podarilo sa mu ju ukončiť v predstihu, zaznamenal čas odrobený na úlohe v nástroji Jira a zvyšný čas doplnil na nulu, takže bolo potrebné zaznamenať aj keď úloha bola splnená za kratší čas. V prípade, že vykonávanie úlohy trvalo dlhší čas ako bol stanovený, člen tímu si zaznamenal aj čas, ktorý strávil nad plnením úlohy navyše. Tieto prípady sa stávali avšak ojedinele. Chybu, ktorú sme ako tím urobili bolo odhadovanie náročnosti jednotlivých úloh, pričom sme nerátali s vytváraním dokumentáciu pre každú z úloh. Tak sa nám tieto dokumenty nahromadili pri poslednom šprinte a museli sme ich dokončovať. Preto v nasledujúcich šprintoch je potrebné do vykonávania úlohy zahrnúť aj čas potrebný pre spísanie dokumentácie k vykonanej úlohe.

V nástroji Jira sme zistili taktiež, že pri vytváraní jednotlivých úloh v rámci šprintov sme neznamenali správne člena tímu, ktorému bola pridelená úloha a člena tímu, ktorý bol zodpovedný za splnenie úlohy. V niektorých prípadoch môže obidve funkcie zastávať rovnaký člen tímu, alebo pre rovnomerné rozdelenie zodpovednosti sa tieto funkcie rozdelia. Problém, ktorý vznikol bol, že neustále bol ako zodpovedný za úlohu defaultne jeden člen tímu, pričom túto funkciu mal zastávať vždy iný člen. Preto sa musíme sústrediť aj na napravenie tohto problému.

Takmer všetky úlohy sa nám podarilo splniť v stanovenom čase až na 2 úlohy. Pri týchto úlohách sa vyskytli okolnosti, pre ktoré v úlohách budeme pokračovať v ďalšom šprinte. Jednou z nich je implementácia ZMP, kde existujúce metódy pre chôdzu sú implementované v Ruby a je potrebné ich prerobiť do Javy. Treba taktiež doplniť testy pre otestovanie správnosti riešenia. Druhou úlohou je zavedenie automatického testovania v nástroji Bamboo. V rámci tohto nástroja bol vytvorený projekt pre náš tím a prepojený s nástrojom Jira. Avšak samotné vytvorenie testov sa nedá vykonať jednoducho pre jednotlivé triedy a je potrebné vytvorenie scriptov pre testovanie. Taktiež je vhodné následne vytvoriť metodiku pre využívanie automatického testovania.

S ďalšími výraznými problémami sme sa nestretli a pri ďalšej práci sa sústredíme na odstránenie doterajších nedostatkov.

5 Retrospektívy k šprintom v letnom semestri

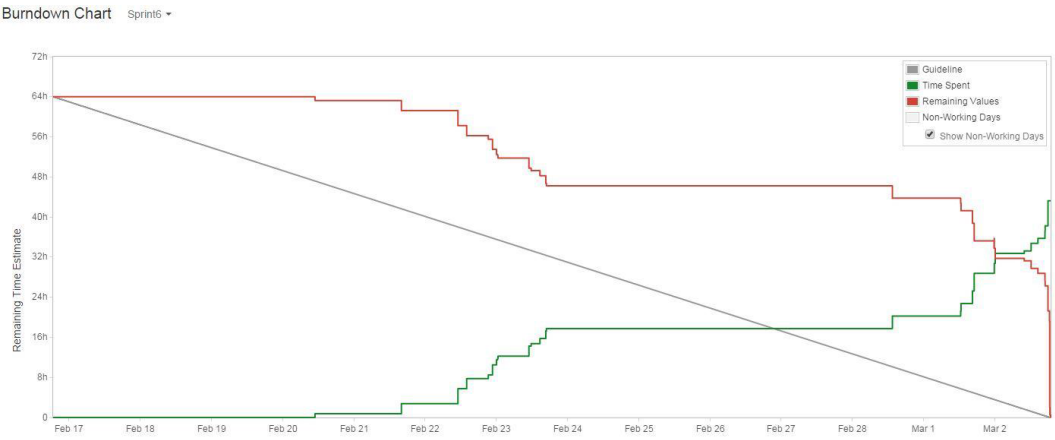
Posledná úprava	Juraj Šimek
Platné od	2. máj 2015
Poznámky	

5.1 Retrospektíva k šiestemu šprintu

Posledná úprava	Michal Segeč
Platné od	2. máj 2015
Poznámky	

V rámci úloh pre letný semester sme sa rozhodli zamerať sa na pokračovanie v refaktoringu zvyšných projektov robocupu do konvencií, a to projektov RobocupLibrary a TestFramework. Tu bude potrebné vykonať mnoho úprav, či už z dôvodu duplicity alebo zbytočnosti. Testovací framework kvôli mnohým úpravám nie je plne funkčný, tak sme sa zamerali najskôr na jeho analýzu. S touto úlohou je samozrejmé je samozrejme spojené aj automatické testovanie, ktoré zlyháva pri vykonaní 2 testoch. Bolo potrebné tieto testy preskúmať a rozhodnúť sa čo s nimi urobiť. Testy boli vyhodnotené ako nepotrebné a odstránené. Okrem týchto úloh sa pokračovalo aj v aplikácií ZMP do pohybu, ktoré bolo dokončené aj keď s problémom, kde agent pri vykonaní otáčavého pohybu padol, takže sme sa rozhodli zamerať sa na opravu tohto pohybu do ďalšieho šprintu. Vector3 bol úspešne prerobený na Vector3D. V medzišprinte sme zistili, že sa občas stane, že zlyhá pri automatickom testovaní aj ďalší test, ktorý keď vráti nesprávnu hodnotu, tak zlyhá, čo taktiež bude treba prešetriť. Taktiež sme zistili, že v testovacom frameworku nie je možné pridávať nového hráča na ihrisko, čo bude potrebné dorobiť. Padla aj myšlienka, že možno bude treba celý testovací framework prerobiť, na čo ale zatiaľ nepomýšľame. V budúcom šprinte bude teda potrebné zamerať sa na pokračovanie v prácach na testovacom frameworku, ako aj opraviť otáčanie agenta, aby nepadal. Ďalšou úlohou bude pridávanie nových hráčov na ihrisko, aby bolo možné lepšie simulovať hru. Budeme taktiež pokračovať refaktoring RobocupLibrary projektu. Vzhľadom na blížiacu sa konferenciu IIT.SRC je potrebné pripraviť plagát a videá pre prezentáciu

projektu. Taktiež bude potrebné pokúsiť sa opraviť padanie hráča, ktoré je príliš časté. Je potrebné sa sústrediť na funkcie, ktoré zisťujú, či je agent na zemi, alebo či padá. V šprinte sme pokračovali rovnako ako v zimnom semestri. Zápisnicu píše každý týždeň iný člen tímu. Rovnako striedame aj scrum mastera pre každý šprint. Čo sa zmenilo je vytváranie úloh v nástroji Jira. Úlohy si rozdeľujeme či už na základe toho, či už niekto mal skúsenosti s danou úlohou, alebo na nej chce jednoducho pracovať. Následne pre každú úlohu vytvoríme story. Rozdielom je, že už nevytvárame substory pre každého člena tímu, ktorému bola pridelená konkrétna úloha, ale v rámci jednej story si viacerí členova tímu logujú svoj čas, odrobený na danej úlohe. Tento spôsob nám prišiel ako efektívnejšie riešenie na rozdiel od vytvárania substory. Všetky úlohy v rámci šprintu boli úspešne splnené, pričom na niektoré z nich sa bude nadväzovať v práci v nasledujúcom šprinte. Z výslednej burndown chart krivky je možné pozorovať, že priebeh plnenia úloh bol v poriadku, až na konci vzhľadom na zlý odhad času krivka prudko spadla. Tejto situácií sa pokúsime v budúcnosti vyvarovať.



Obr. 5: Burndown chart po ukončení 6.šprintu.

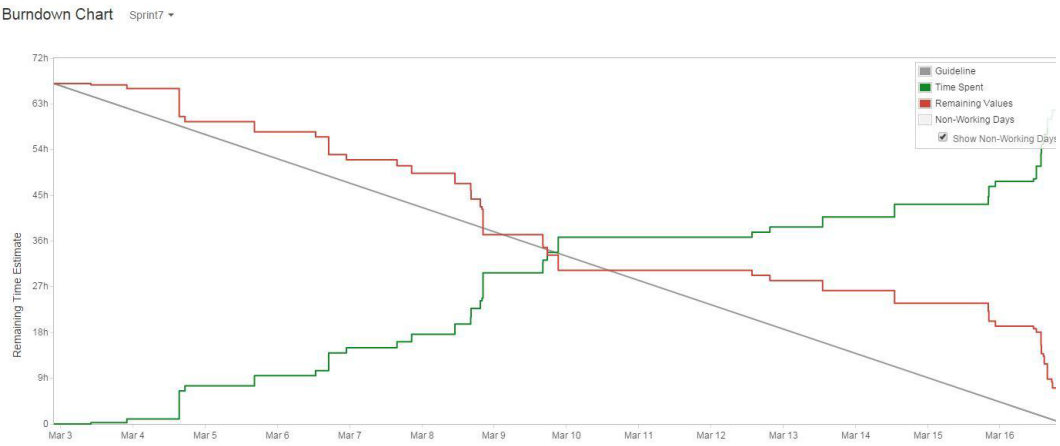
5.2 Retrospektíva k siedmemu šprintu

Posledná úprava	Miroslav Wolf
Platné od	2. máj 2015
Poznámky	

Opis šprintu: V tomto šprinte sme sa zamerali na prípravu materiálov pre konferenciu IIT.SRC, ale aj na opravenie a refaktoring TestFrameworku, refaktoring RobocupLibrary a na opravenie vzniknutých chýb po implementácii ZMP. Pre tento šprint sme si vybrali viacero úloh, ako v predošlých šprintoch. Až na dve úlohy sa nám všetky úlohy podarilo splniť, keďže nám nečakane ochorel jeden člen tímu a boli problémy s analyzovaním a testovaním chýb najmä pri úlohách analýzy pádu agenta pri otáčaní a analýzy detekcie pádu agenta. Úlohy sme si rozdelili tak, aby každý člen tímu mal približne rovnako časovo náročné úlohy a tak aby každý člen tímu vedel čo má robiť a teda aj podľa toho, či má s danou úlohou nejaké skúsenosti. Úlohy sme vyberali z backlogu, prípadne nové úlohy sme v Jire vytvorili. Niektoré úlohy nám vyplynuli z predošlého šprintu a na týchto úlohách pokračovali členovia tímu, ktorí pracovali aj na pôvodných úlohách v predošlom šprinte. Pre úlohy sme mali v Jire ako v predošlom šprinte vytvorené story, pričom sme nevytvárali substory, ale každý člen tímu, ktorý na danej úlohe pracoval si pod danú story zalogoval prácu s popisom, čo vykonal a koľko času pri tom strávil. Keďže sme doteraz robili vo vlastných branchoch a veľa sme ich nemergovali s master branchom, tak bolo nutné vykonať aj túto úlohu. Bolo tak nutné vyriešiť niekoľko konfliktov. V polovici šprintu sme si zhodnotili náš postup, pričom dve z úloh už boli splnené a ostatné boli v štádiu rozpracovania a tak vyzeralo že pomerne stíhame.

Zhrnutie šprintu: Pre tento šprint sme si vytvorili viacero úloh a navyše nám ochorel jeden člen tímu. Okrem toho vznikli aj iné problémy pri testovaní a tak sa dve úlohy nestihli dokončiť. Medzi nestihnuté úlohy patrí refaktoring testframeworku, kde sa stihlo zrefaktorovať 11 balíkov z celkových 19 a analýza pádu agenta pri otáčaní. V úlohe detekcia padania hráča sa podarilo vytvoriť nové funkcie pre detekciu padania, no ešte nefungujú úplne korektne, keďže je nutné vhodne nastaviť hraničné hodnoty niektorých hodnôt. V ďalšom šprinte teda bude ešte nutné hlbšie analyzovať jed-

notlivé logy a sledovať, ako sa menia hodnoty pri kráčaní a padnutí hráča a podľa toho vhodnejšie nájsť hraničné hodnoty. Okrem spomínaného sa nám v tomto šprinte podarilo opraviť pridávanie hráča pomocou testframeworku, pripraviť videá a plagát na IIT.SRC konferenciu, zrefaktorovať projekt RoboCupLibrary a mergli sme doterajšiu prácu s master branchom. Zhodli sme sa aj na niekoľkých zmenách, ktoré by bolo vhodné zlepšiť. V budúcnosti bude vhodné branche častejšie mergovať s master branchom, aspoň po každom šprinte, aby pri mergovaní nevznikalo veľa konfliktov. Narazili sme aj na problém s testovaním a analyzovaním logov, kde pri úpravách a analýze pádu agenta pri otáčaní a pri úpravách detekcií pádov bolo ťažké sa vo veľkom množstve logov vyznať. Logger loguje mnoho údajov a tak hľadať v logoch to čo nás zaujíma môže zabrať dlhý čas a je to nepraktické. Tento problém sme sa rozhodli vyriešiť tým, že sme si do ďalšieho šprintu zaradili úlohu pre vytvorenie grafického rozhrania pre logger, v ktorom sa bude dať upravovať čo chceme loggovať. Dohodli sme sa tiež, že stretnutia bude viesť vždy ten člen tímu, ktorý naposledy robil zápisnicu zo stretnutia.

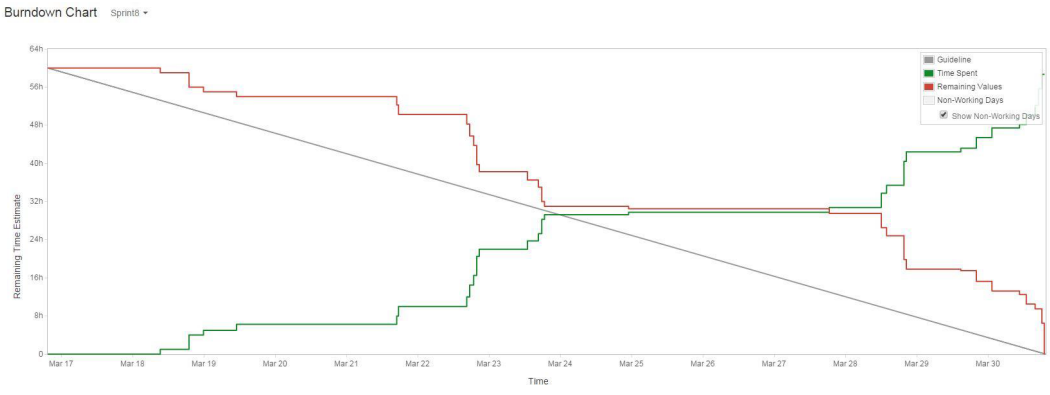


Obr. 6: Burndown chart po ukončení 7. šprintu.

5.3 Retrospektíva k ôsmemu šprintu

Posledná úprava	Martin Vrabec
Platné od	2. máj 2015
Poznámky	

Opis šprintu: V tomto šprinte sme väčšinou pokračovali v úlohách z minulého šprintu, čo znamenalo pokračovanie v refaktoringu TestFrameworku, doladenie funkcií pre zisťovanie pádu, finalizáciu highskillov a analyzovanie RoboCup turnaja. Až na finalizáciu highskillov sa podarilo všetky úlohy úspešne dokončiť a táto čiastočne dokončená úloha bude predmetom ďalšej činnosti tým. Pri refaktoringu TestFrameworku zostalo pre posledný týždeň zrefaktorovať dva balíčky, ktoré boli v druhom týždni šprintu aj úspešne dokončené. Pri tejto úlohe boli oddelené testy a nepoužívané triedy do nových balíčkov. Pri dolaďovaní funkcií na detekciu pádu agenta sa podarilo implementovať funkcie na detekciu pádu agenta na brucho a na chrbát. Pri finalizácii highskillov sa zistilo, že problém s prepínaním highskillov je spôsobený nesprávným plánovaním a finalizovaním a nie implementáciou ZMP, čo bude treba analyzovať v niektorom z ďalších šprintov. Pri analyzovaní taktík používaných na fakultný RoboCup turnaj boli zistené niektoré nedostatky, ktoré boli aj opravené. Zároveň boli implementované niektoré nové taktiky a upravené už existujúce.



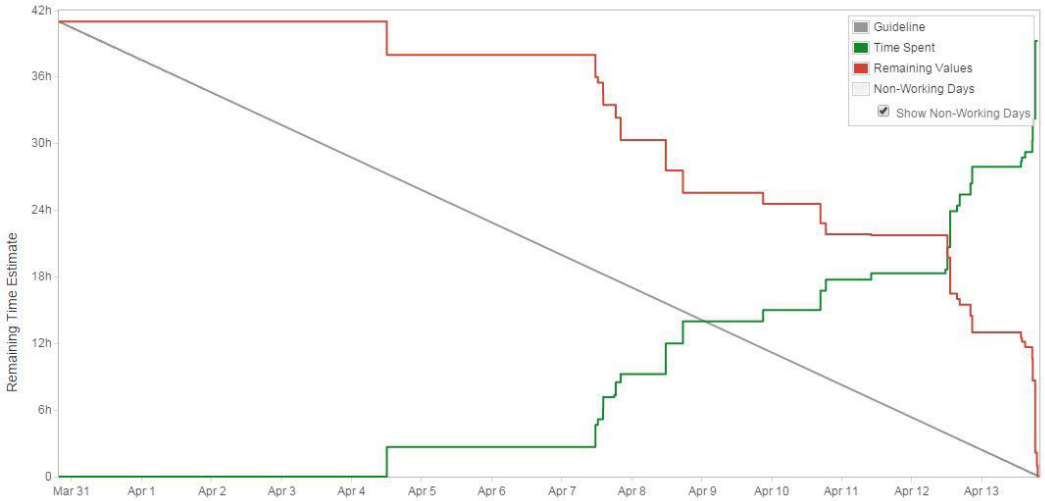
Obr. 7: Burndown chart po ukončení 8. šprintu.

5.4 Retrospektíva k deviatemu šprintu

Posledná úprava	Peter Filípek
Platné od	2. máj 2015
Poznámky	

Opis šprintu: Tento šprint bol prerušený sviatkami a dekanským voľnom. Z tohto dôvodu bola znížená potrebná odpracovaná práca u pôvodných 10h na 7h za šprint. Kvôli sviatku sme boli nútení presunúť tímové stretnutie zo štandardného pondelku na stredu. Jeden z členov tímu sa nezúčastnil stretnutia pre pracovné povinnosti a druhý sa zúčastnil len virtuálne cez Skype kvôli cestovaniu. Jednalo sa o výnimku vzhľadom na sviatky a neobvyklý termín stretnutia. Stretnutie sme sa miesto úplného vynechania rozhodli radšej presunúť aby sa prípadné problémy prediskutovali a bol čas na ich riešenia a súčasne aby sa s prácou nezačalo príliš neskoro. Toto rozhodnutie sa nakoniec ukázalo správne, ako je vidno z grafov, počas sviatkov sa na úlohách nepracovalo, potom intenzita mierne narástla a tesne pred stretnutím dosahovala maximum.

Na tento šprint sme vyberali najmä úlohy, ktoré súviseli z predošlými a bolo nutné dopracovať niektoré časti ako napr. úlohy Doplnenie dokumentácie a Vylepšenie GUI loggera. Ďalej sme vyberali úlohy, ktoré sa týkali novej oblasti a bolo nutné zanalyzovať danú problematiku pre ďalšie úlohy, analýze sa venovali úlohy Analýza planela, Analýza nefunkčnosti anotácií v TestFrameworku a čiastočne aj úloha Zisťovanie údajov v TestFrameworku v rámci ktorej sa začali prípravy na RoboCup turnaj na FIIT. Ďalšia úloha, ktorá sa venovala príprave na turnaj bola Príprava taktík na turnaj. Posledná úloha sa venovala tiež TestFrameworku, konkrétne išlo o Implementáciu pohybu lopty v TestFrameworku.



Obr. 8: Burndown chart po ukončení 9. šprintu.

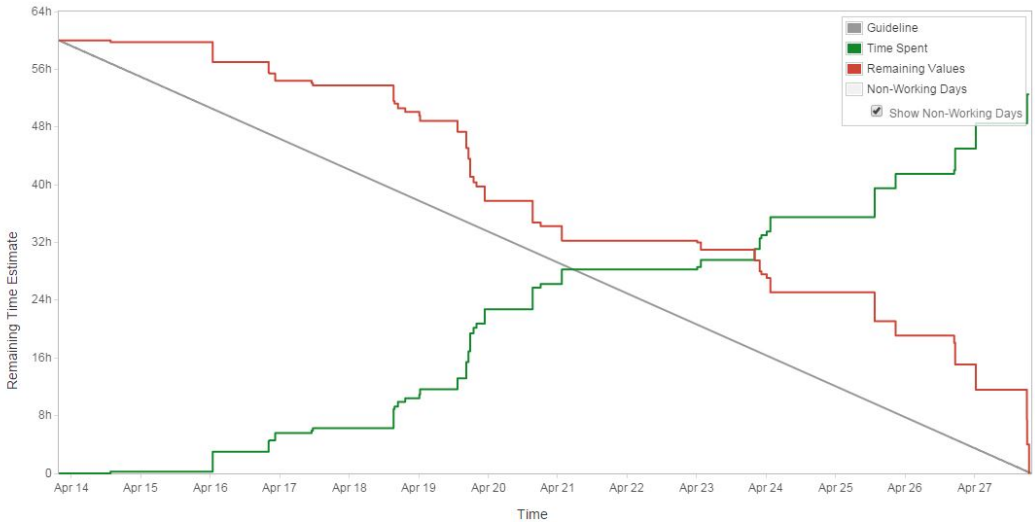
5.5 Retrospektíva k desiatemu šprintu

Posledná úprava	Metod Rybár
Platné od	2. máj 2015
Poznámky	

Opis šprintu: Na priebežnom stretnutí šprintu 20. 4. 2015 sa nezúčastnili dvaja členovia kvôli chorobe, a to Juraj Šimek a Martin Vrabec. Keďže sme však po predchádzajúcich skúsenostiach mali nastavené postupy, dodali nám postup svojej práce, takže sme ho mohli na stretnutí prebrať. Toto následne pomohlo aj pri uzatváraní šprintu, kedy sme sa k už prediskutovaným veciam nemuseli príliš vracať.

Projekt sa pripravoval v tomto šprinte na odovzdanie na testovanie a na web. Keďže sme náš projekt nechali testovať viacerým bakalárom a diplomantom už dlhodobejšie, s týmto nebol problém. Rovnako bolo jednoduché určiť si úlohy potrebné pre dokončenie produktu na odovzdanie, keďže sme podobnú skúsenosť mali už z predchádzajúceho semestra a naše úlohy v týme už boli zabehnuté.

Šprint sme ukončili preto úspešne a produkt bol odovzdaný na čas.

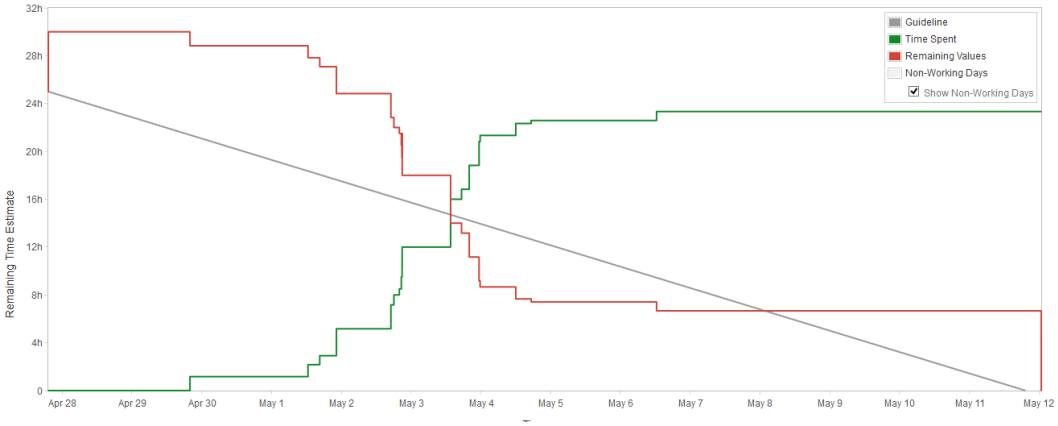


Obr. 9: Burndown chart po ukončení 10. šprintu.

5.6 Retrospektíva k jedenástemu šprintu

Posledná úprava	Juraj Šimek
Platné od	4. máj 2015
Poznámky	

Opis šprintu: Posledný šprint sme sa venovali najmä finalizácii projektu a jeho dokumentácie. Juraj Šimek a Metod Rybár vhodne formátovali a dopĺňali dokumentáciu k inžinierskemu dielu a dokumentáciu k riadeniu tímu obsahom, ktorí bol priebežne vkladaný na spoločný Dropbox účet. Rozhodli sme sa z dokumentácie k riadeniu vyhodiť exporty úloh, nakoľko majú byť odovzdané len vo formáte PDF. Peter Filípek a Michal Segeč pracovali na doladení meraní časov pre turnaj tak, aby nebolo potrebné stále reštartovať server. Túto úlohu sa im podarilo splniť. Nepodarilo sa však detegovať, či má agent nohy na zemi, túto úlohu by bolo potrebné riešiť pomocou komunikácie cez TFTP server. Miroslav Wolf spísal globálnu retrospektívu a dokument pre ďalšie tímy, kde opísal, čomu sme sa venovali a čomu nie, čo sme nestihli, na aké problémy sme narazili a čo by bolo vhodné riešiť v budúcnosti. Martin Vrabec sa nakoniec venoval úprave webového sídla pre jeho transformáciu na statickú verziu.



Obr. 10: Burndown chart po ukončení 11. šprintu.

6 Globálna retrospektíva k letnému semestru

Posledná úprava	Miroslav Wolf
Platné od	2. máj 2015
Poznámky	

Letný semester pozostával zo 6tich šprintov, pričom dĺžka jedného šprintu bola štandardne 2 týždne, no posledný 11ty šprint bol skrátený na 1 týždeň. Počas týchto šprintov sme sa zamerali najmä na opravu, doplnenie funkcionalít a úpravu do konvencií TestFrameworku. Okrem toho sme však robili aj zmeny v projekte Jim, kde sme tiež ladili niektoré chyby spojené najmä s pridaním nového pohybu ZMP (Zero Moment Point), upravovali sme funkciu pre detekciu pádu a dopĺňali funkcie pre pohodlnejšie logovanie. V jednom zo šprintov sme sa zamerali aj na projekt RoboCupLibrary, ktorý sme zrefaktorovali podľa konvencií. Keďže sa náš tím chystá aj na fakultný turnaj RoboCupu, niektoré z úloh boli zamerané práve na prípravu na tento turnaj. Zamerali sme sa teda najmä na to, aby sa dali sledovať a merať rôzne disciplíny turnaja prostredníctvom TestFrameworku a aby poskytoval svoju základnú funkcionalitu.

Výber úloh

Pri výbere úloh do šprintu sme sa riadili najmä vlastníkom produktu (pánom Kapustíkom), no úlohy sme vyberali aj samy. Manažér rizík na základe konverzácie s členmi tímu stanovoval predpokladaný čas, ktorý sme mali za šprint odrobiť. Po výbere hlavných úloh sme spoločne odhadovali časovú náročnosť úloh a v prípade, že celkový odhadovaný čas bol nižší ako stanovený čas manažérom rizík, tak sme doplnili, alebo vytvorili ďalšie úlohy do šprintu, aby každý člen tímu mal pridelenú nejakú úlohu a aby sa za šprint stihli úlohy dokončiť.

Vytváranie úloh

Pri vytváraní úloh sme sa držali vytvorenej metodiky pre vytváranie úloh. Nie vždy sme však úlohám v Jire napísali opis – čo sa v danej úlohe bude robiť, no úlohy sú opísané v zápisniciach, prípadne čo sa v úlohe vykonávalo je možné vidieť v logoch práce, prípadne v dokumentácii k úlohe. Pri logovaní práce sme sa snažili dodržiavať istú úroveň opisu práce, aby bolo jasné na čom daný člen tímu pracoval a čo sa mu podarilo, prípadne na čo je nutné sa

ešte pozrieť. Pre úlohy bola vždy vytvorená aj dokumentácia, v ktorej bola opísaná vykonaná práca, opísaná nová funkcionalita a prípadne návod na jej použitie.

Stručný opis šprintov

V šiestom šprinte (prvom v letnom semestri) sme sa zamerali na analýzu TestFrameworku, keďže niektoré jeho časti neboli funkčné. Okrem tejto úlohy sme riešili zlyhávanie dvoch testov, problémy spojené so ZMP a prerobenie Vector3 na Vector3D. Pri vytváraní tohto šprintu sme sa rozhodli zmeniť spôsob vytvárania úloh v nástroji Jira, kde pre každú úlohu vytvoríme story, no nevytvárame subtasky pre každého člena tímu, ktorí na úlohe pracuje. Prácu si tak logujú viacerí členovia tímu môžu logovať do jednej story. Tento spôsob sme zvolili pretože sa nám zdal jednoduchší, rýchlejší a efektívnejší. Úlohy medzi členov tímu sme rozdeľovali podľa toho, či člen tímu mal s úlohou skúsenosti, alebo jednoducho na nej chce pracovať. V ostatných veciach týkajúcich sa šprintu sme pokračovali podobne ako v zimnom semestri. Úlohy sa nám podarilo splniť, no zhodli sme sa že sme zle odhadli časovú náročnosť úloh.

V siedmom šprinte (druhom v letnom semestri) sme začali s prípravou materiálov na konferenciu IIT.SRC – bolo potrebné pripraviť plagát a videá. Okrem toho sme však po analýze TestFrameworku zo šiesteho šprintu pokračovali jeho refaktoringom, rozhodli sme sa zrefaktorovať aj projekt RoboCupLibrary a opravovali a analyzovali sa chyby vzniknuté po implementácii ZMP. Pri ZMP vznikal problém s detekciou pádu, tak sme sa rozhodli ju analyzovať a prípadne navrhnúť nové riešenie detekcie pádu. Pred začatím šprintu sme sa zhodli, že by sme mali častejšie mergovat branche, keďže robíme vo viacerých oddelených branchoch a dlhšie sme nemergovali, museli sme riešiť mnoho konfliktov. Niektoré úlohy nadväzovali na úlohy z predošlého šprintu a tak týmto úlohám sa venovali prevažne rovnaký členovia tímu, ako pôvodným úlohám. Pre tento šprint sme si naplánovali viac úloh ako obvykle a v polovici šprintu vyzeralo, že pomerne stíhame. V druhej polovici nám však ochorel jeden člen tímu a pri analýze a opravovaní chýb v ZMP sme narazili na niekoľko problémov. Dve úlohy sa teda nestihli dokončiť a pokračovalo sa na nich aj v nasledujúcom šprinte. Jedným z problémov na ktoré tím počas tohto šprintu narazil bol tiež problém z testovaním, kde sme sa zhodli že logy sú neprehľadné a tak sme sa rozhodli v ďalšom šprinte tento problém vyriešiť. Okrem toho sme nemali report od chorého člena tímu a tak sme sa rozhodli zaviesť pravidlo, že ak niekto ochorie pošle krátku správu o

tom čomu sa venoval a čo dokončil, prípadne čo nestihol.

V ôsmom šprinte (treťom v letnom semestri) sa pokračovalo na dvoch nedokončených úlohách zo siedmeho šprintu, dolaďovanie navrhnutých funkcií pre detekciu pádu, implementáciu GUI loggera pre prehľadnejšie a utriedeniejšie logy, finalizáciu highskillov a analýzu fakultného RoboCup turnaja, kde sme zisťovali čo všetko pre turnaj projekt obsahuje a čo bude potrebné doplniť. Úlohy sa nám podarilo úspešne dokončiť.

V deviatom šprinte (štvrtom v letnom semestri) sme sa zamerali na vylepšovanie GUI pre logger, analyzovali sme planner highskillov s ktorým boli problémy, analyzovali sa nefunkčné anotácie v TestFrameworku, navrhlo sa niekoľko meraní do TestFrameworku a pripravovali sa rôzne taktiky pre fakultný RoboCup turnaj a do TestFrameworku bola tiež implementovaná simulovaná streľba, ktorá môže slúžiť napríklad pre otestovanie brankára. Šprint bol počas sviatkov a tak sme sa snažili vybrať jednoduchšie úlohy. Kvôli sviatkom sme mali tiež presunuté stretnutie. Úlohy sa nám podarilo včas splniť.

V desiatom šprinte (piatom v letnom semestri) sme pokračovali v pripravovaní turnajových taktík a meraní pre fakultný turnaj, ako aj hlbšou analýzou anotácií výsledkom ktorej je dokument určený pre budúce tímy. V rámci šprintu sme tiež pripravovali projekt pre finálne odovzdanie a teda sme vytvárali používateľskú príručku na inštaláciu, spustenie a otestovanie projektu, ktorá bude slúžiť pre budúce tímy, ako aj pre kohokoľvek kto bude chcieť náš projekt vyskúšať, prípadne otestovať. Zamerali sme sa tiež na ukončovanie highskillov, doladenie otáčania a jeho implementácia do highskillu. Počas tohto šprintu nám ochoreli dvaja členovia tímu, no obaja dali vopred vedieť, že sa nezúčastnia stretnutia a ako sme sa dohodli poslali aj report o svojej vykonanej práci a tak bol zbytok tímu informovaný. Počas tohto šprintu sa tiež konala konferencia IITSRC a tak sme si tiež rozdelili kto pôjde pre plagát a kto prichystá ukážkové videá. Okrem toho sme v rámci odovzdania projektu aktualizovali web, dali naň aktuálnu dokumentáciu a projekt sme sprístupnili – všetko sme mergli do master branchu, ku ktorému má prístup viacero študentov. Úlohy sme stihli úspešne dokončiť.

V poslednom jedenástom šprinte (šiestom v letnom semestri) sme sa zamerali na dokončovanie a kontrolu dokumentácií, úpravu a sprehľadnenie webového sídla a doladeniu meraní pre turnaj. Tento šprint je iba týždňový a keďže je aj posledný tak sme si vybrali úlohy spojené najmä s ukončením práce a s dokumentovaním. V rámci tohto šprintu bol tiež vypracovaný dokument pre budúce tímy, čomu by bolo vhodné sa v budúcnosti venovať, čomu

sa náš tím nestihol venovať a prípadne chyby, ktoré zatiaľ neboli odstránené. Po ukončení šprintu sme zhodnotili našu prácu – čo sa stihlo a čo nie. Veci ktorým sme sa nestihli venovať sú opísané v spomínanom dokumente pre budúce tímy, ktorý sa nachádza v dokumente inžinierskeho diela tímu.

Zhrnutie

V letnom semestri sa nám podarilo splniť väčšinu úloh, ktoré sme mali naplánované. Niektoré úlohy sme na začiatku semestra v backlogu nemali a vytvorili sa až počas semestra. Backlog sme postupne upravovali a úlohy, ktorým sme sa nestihli venovať sme opísali v dokumente pre budúce tímy, čomu sa možno v budúcnosti venovať. Úlohy sa nám pomerne úspešne darilo plniť, s výnimkou siedmeho šprintu. Na začiatku semestra – v šiestom šprinte sme mali tiež problém s odhadnutou časovou náročnosťou, no v ďalších šprintoch sa nám podarilo tento problém odstrániť. V tíme sa počas semestra zlepšila aj organizácia, kde keď člen tímu ochorel, tak poslal report o svojej vykonanej práci. Oproti zimnému semestru sme tiež častejšie a podrobnejšie opisovali logy práce v Jire a vďaka tomu boli členovia tímu lepšie koordinovaný a informovaný o práci ostatných kolegov. Zlepšila sa aj koordinácia členov tímu v Jire a tak nám vytváranie úloh a exporty zabrali na stretnutiach menej času.

7 Vedenie tímu

Posledná úprava	Metod Rybár
Platné od	17. november 2014
Poznámky	

Každý člen tímu má na starosti časť manažovania práce na projekte a je zodpovedný za vedenie tímu v rámci svojich zodpovedností. Okrem týchto pozícií existujú ale ešte dve zastrešujúce pozície, a to vedúci tímu a scrum master.

Na úspešné vykonávanie vedenia sa využívajú najmä prostriedky spomínané v manažmente komunikácie 12. Tieto prostriedky vedúci tímu a scrum master využívajú na koordináciu práce ostatných členov tímu.

Úlohou vedúceho tímu je najmä kontrolovanie ostatných členov tímu a dohliadanie na plnenie náležitostí prislúchajúcim k jednotlivým manažérskym pozíciám v tíme počas celej doby projektu.

Medzi úlohy vedúceho tímu patrí aj sledovanie požiadaviek k projektu, ich zaznamenávanie a vyhodnotenie s ostatnými členmi tímu.

Vedúci tímu však nenahrádza scrum mastera pre jednotlivé behy. Teda sa nezaoberá priamo kontrolovaním práce na výkone jednotlivých úloh v danom behu. Jeho úlohou je udržiavanie dlhodobejšieho plánu a úloh pre projekt.

Vedenie tímu v jednotlivých behoch preberá scrum mater a on zodpovedá za úlohy v danom behu, kontrolu ich priebehu a prípadné riešenie problémov počas daného behu.

8 Manažment rozvrhu

Posledná úprava	Miroslav Wolf
Platné od	19. november 2014
Poznámky	

8.1 Manažment rozvrhu v tíme

Posledná úprava	Miroslav Wolf
Platné od	19. november 2014
Poznámky	

Úlohou manažéra rozvrhu je odhadovanie časov, sledovanie termínov odovzdania a sledovanie činnosti práce tímu. Manažér rozvrhu sleduje najmä to, či sa do termínu odovzdania stíhajú ukončiť všetky požadované úlohy a ako tím na úlohách priebežne pracuje, prípadne či boli časy úloh vhodne odhadnuté. S manažmentom rozvrhu súvisí aj manažment rizík, pričom pri odhadovaní časov je vhodné rátať aj so vznikom pravdepodobných rizík a snažiť sa tak vhodnejšie čas odhadnúť, prípadne rizikám predísť, či ich minimalizovať.

Pojmy:

- Jira - nástroj na plánovanie, evidenciu úloh a organizáciu práce v tíme
- SCRUM - metodika riadenia projektu
- SCRUM Master - člen tímu, ktorý je zodpovedný za vedenie šprintu. Táto úloha sa strieda medzi členmi nášho tímu každý šprint.
- Šprint - jeden cyklus iterácie práce v projekte pri riadení metodikou SCRUM
- Burndown chart - grafická reprezentácia postupu práce tímu počas prebiehajúceho šprintu. Znázorňuje vykonanú prácu počas daného času.

8.1.1 Metodika plánovania

Ako metodiku náš tím používa techniku SCRUM. SCRUM je agilná metodika s iteratívno-inkrementálnym spôsobom vývoja. V tejto metóde tím pracuje spolu, aby dosiahli splnenie spoločného cieľa. Kľúčovým je aj predvídanie, že zákazník môže počas projektu zmeniť svoje požiadavky a rovnako sa môže meniť aj rozsah projektu. Ďalšou charakteristikou SCRUM-u je, že projekt je síce dodávaný po častiach, ale pravidelne.

8.1.2 Stretnutia tímu

Náš tím sa stretáva každý týždeň na spoločných stretnutiach, kde si najskôr zhrnieme priebeh práce a následne si určíme postup a úlohy do ďalšieho týždňa. Každé stretnutie je zaznamenané v zápisnici, pričom zapisovateľ sa strieda každý týždeň a teda sa vystriedajú všetci členovia tímu. Podľa burndown chart-u vidíme ako tím postupuje v práci a či sa požadované úlohy stíhajú včas ukončiť, prípadne či je tím v predstihu.

8.1.3 Roly účastníkov tímu spojené s prípravou šprintu

- Vlastník produktu - zákazník, alebo jeho zástupca, ktorí schváli finálne zadanie a prevedenie úlohy
- Zapisovateľ stretnutia - zapíše najdôležitejšie spomenuté poznatky zo stretnutia, poznačí nové úlohy ktoré je nutné pridať do zoznamu úloh a vytvorí zápisnicu
- SCRUM Master - zodpovedný za začatie a ukončenie šprintu, vytvorí retrospektívu šprintu
- Manažér rozvrhu - sleduje plnenie plánu a odhadnutý čas a či sa stihnú všetky úlohy včas ukončiť
- Manažér rizík - identifikuje riziká, na základe ktorých sa môže zmeniť odhadovaný čas pre splnenie úloh
- Riešiteľ - každý z členov týmu, musí mať vybratú úlohu zo šprintu, zaznamenávať si svoju činnosť
- Tester - testuje výstup úlohy a potvrdí správnosť výstupu

8.1.4 Použité nástroje

Na plánovanie si náš tím zvolil nástroj JIRA od spoločnosti Atlessian. V tomto nástroji evidujeme úlohy a stav ich plnenia, celkový stav projektu, vykonanú prácu a postup tímu pri plnení úloh. Taktiež si cez tento nástroj vieme pozrieť rôzne diagramy ako napríklad burndown chart, z ktorého môžeme vidieť postup práce tímu a či je tím v časovej tiesni alebo naopak v predstihu. Nástroj obsahuje veľa funkcií a je prístupný cez web a podporuje aj prepojenie s ďalšími nástrojmi od spoločnosti Atlassian.

O vytváranie úloh a sledovanie postupu tímu sa stará najmä manažér rozvrhu, no každý z členov tímu si môže vytvárať úlohy a uzatvárať ich podľa potreby. Členovia tímu si aj zapisujú strávený čas nad jednotlivými úlohami, na čo tiež dohliada manažér rozvrhu. Keďže pri vytváraní úloh priraďujeme každej úlohe aj odhadovaný čas, z celkového stráveného času na úlohách tak vidíme, či bol tento čas odhadnutý dobre.

Pre tento nástroj bola vytvorená aj metodika - Metodika pre vytváranie úloh v nástroji Jira 8.2.

8.1.5 Zahájenie šprintu

Na začiatku každého šprintu vyberie vlastník produktu úlohy zo zoznamu úloh, ktoré sa budú počas šprintu riešiť. Tím diskutuje o úlohách a ich náročnosti, prípadne niektoré úlohy rozdelí na menšie pod-úlohy, aby sa vedel presnejšie určiť odhad náročnosti. K náročnosti a odhadom sa vyjadrí aj manažér rizík, ktorý identifikuje možnosti vzniku rizík. Následne sa pre jednotlivé úlohy sa odhaduje náročnosť, ktorá zodpovedá časovej náročnosti. Ak pri hlasovaní vzniknú odlišnosti, každý člen tímu sa vyjadrí, prečo odhadol danú náročnosť, na základe čoho sa buď hlasuje znovu, alebo sa spriemeruje hlasovanie a určí sa tak náročnosť úlohy.

Po odhadnutí náročnosti úloh sa manažér rizík a manažérom rozvrhu vyjadria či sa dané úlohy stíhajú počas šprintu dokončiť, prípadne či niektorá z úloh bude rozdelená do viacero šprintov. Ak je úloh málo, tím, alebo vlastník produktu si zvolí ďalšiu úlohu zo zoznamu, pre ktorú sa znova určí náročnosť.

Ak sú už odhadnuté náročnosti pre všetky úlohy, následne si tím úlohy rozdelí, pričom každý člen tímu by mal za šprint odpracovať približne rovnaký čas a každý člen tímu musí mať pridelenú úlohu. Každá úloha má aj zodpovedného za splnenie danej úlohy, ktorý riešiteľa úlohy kontroluje. Po rozdelení úloh , manažér rozvrhu zapíše úlohy do nástroja Jira, vytvorí šprint

a zaradí úlohy do daného šprintu.

8.1.6 Ukončenie šprintu

Pri ukončovaní šprintu tím zhrnie všetky úlohy zo šprintu a ich stav - Splnená, Čiastočne splnená, Nesplnená.

- Splnená - Úloha je kompletne splnená a otestovaná. K úlohe existuje aj vypracovaný dokument. Riešitelia úlohy sa k jej riešeniu vyjadria, prípadne spomenú vzniknuté problémy a opíšu a vysvetlia jej funkčnosť ostatným členom tímu.
- Čiastočne splnená - Úloha je splnená, no niektoré časti úlohy ešte treba dopracovať, prípadne otestovať a vytvoriť k úlohe dokument. Riešitelia úlohy sa k jej riešeniu vyjadria, opíšu čo bolo splnené a čo treba ešte dopracovať, prípadne sa vyjadria k vzniknutým problémom.
- Nesplnená - Úlohu sa nepodarilo dokončiť a je v neúplnom stave. Riešitelia sa vyjadria prečo úloha nie je splnená, kde vznikol problém a čo na úlohe treba dopracovať a čo bolo v rámci úlohy vypracované.

Po prejdení úloh sa každý člen tímu vyjadrí, čo si o postupe tímu myslí a svoje pripomienky. SCRUM Master zhrnie postup tímu, ako si tím počínal a čo je nutné do ďalšieho šprintu vylepšiť. Následne šprint uzavrie a vytvorí retrospektívu šprintu.

8.2 Metodika pre vytváranie úloh v nástroji Jira

Posledná úprava	Miroslav Wolf
Platné od	19. november 2014
Poznámky	

Táto metodika je určená pre vytváranie úloh v nástroji Jira. Bude sa ňou riadiť najmä SCRUM Master, manažér rozvrhu a každý člen tímu, ktorí bude vytvárať úlohy v nástroji Jira. Pokrýva akcie spojené s vytváraním úloh v nástroji Jira, rozhodovanie aký typ úlohy bude vytvorený a aká je jej priorita a čo má byť nastavené v jednotlivých poliach pri vytváraní úloh. Metodika neznázorňuje presný opis pri prechádzaní obrazoviek v nástroji

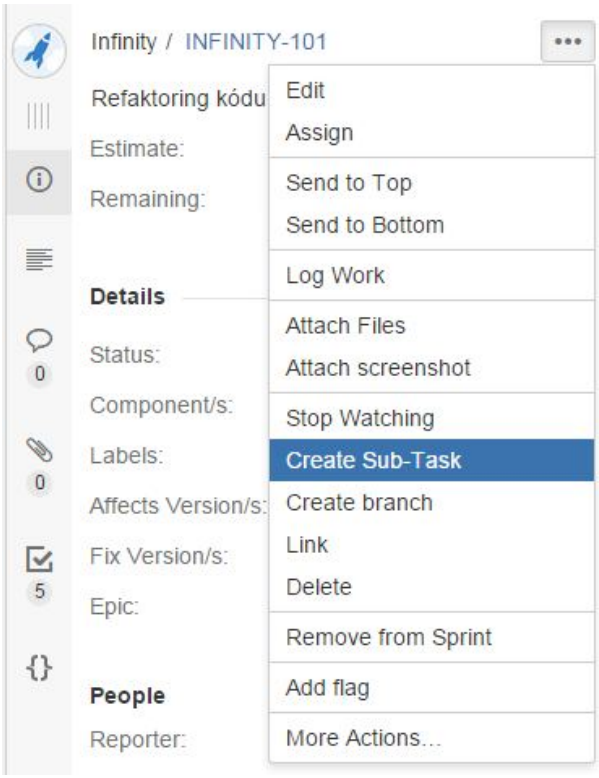
Jira a ani nevysvetľuje, ako sa tím má riadiť pri rozdeľovaní úloh medzi členov tímu, ako dekomponovať jednotlivé úlohy, zapisovať vykonanú prácu a ani ostatnými akciami spojenými s vytváraním a ukončovaním šprintov. Metodika úzko súvisí s dokumentom k manažmentu rozvrhu a plánovania, kde je opísaný postup tímu pri vytváraní šprintov, rozdeľovaní úloh a riadenie tímu metodikou SCRUM.

Základné pojmy:

- úloha - práca ktorá sa bude vykonávať, môže sa skladať z viacerých pod-úloh
- story - úloha, predstavujúca väčšiu časť projektu, ktorá sa rozdeľuje do viacerých pod-úloh
- pod-úloha - menšia časť z úlohy, časť z práce úlohy
- zoznam úloh - nahromadené nesplnené úlohy, ktoré sa postupne zaraďujú do šprintu a následne vykonávajú
- funkčnosť projektu - projekt je funkčný, ak všetky jeho súčasti fungujú správne
- priorita úlohy - určuje dôležitosť splnenia úlohy, úlohy s väčšou prioritou by mali byť splnené skôr, ako ostatné úlohy

8.2.1 Vytváranie úloh v Jire

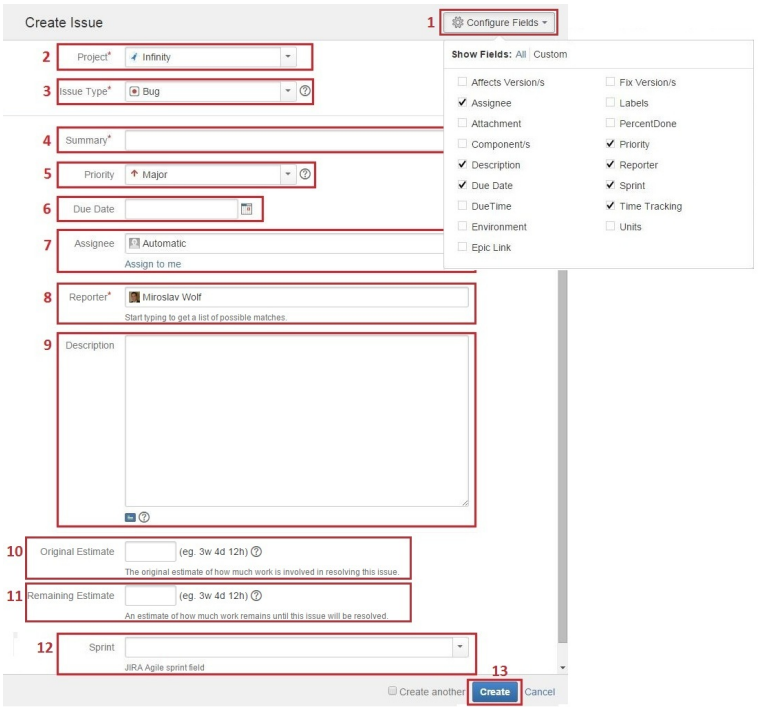
V nástroji Jira prejdite na obrazovku so všetkými úlohami, v hornom menu cez *Agile* a následne vyberte tabuľu so zoznamom úloh pre váš projekt (v pravom hornom rohu musíte byť prepnutý do *Plan*). Pri vytváraní úloh je dôležité si overiť, či sa daná úloha už nenachádza v zozname úloh, prípadne či nie je súčasťou inej väčšej úlohy. Pokiaľ ide o úlohu ktorá je súčasťou inej úlohy, nastavte sa na túto úlohu a vytvorte pre ňu pod-úlohu (*Create Sub-Task*), kliknutím na menu úlohy v pravej hornej časti obrazovky 11. Pri vytváraní pod-úloh sa riaďte rovnakými pokynmi, ako pri vytváraní ostatných úloh.



Obr. 11: Vytváranie podúlohy

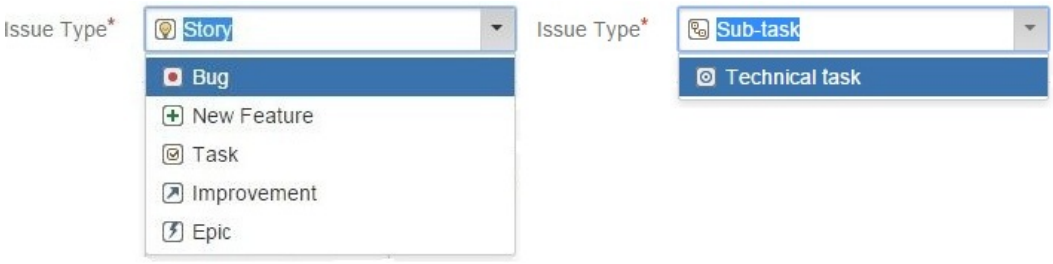
8.2.2 Postup pri vytváraní úloh

Nasledujúce pokyny sa týkajú vytvárania všetkých typov úloh.



Obr. 12: Ukážka polí pri vytváraní úloh

1. Configure Fields - podľa obr. 12 vyberte polia, ktoré budú zobrazené pri vytváraní úloh.
2. Project - vyberte tím, pre ktorý idete úlohu vytvárať.
3. Issue Type (obr. 13)



Obr. 13: Ukážka výberu typu úlohy a typu podúlohy

- Pre úlohy:

  Story - vyberte v prípade, že ide o komplexnú úlohu alebo väčšiu časť prjektu, ktorá bude zložená z viacerých pod-úloh.

  Bug - vyberte v prípade, že ide o opravu nájdenej, alebo nahlásenej chyby v projekte.

  New Feature - tento typ úlohy nevytvárajte.

  Task - vyberte v prípade, že ide o bežnú úlohu, ktorá má byť splnená.

  Improvement - vyberte v prípade, že ide o vylepšenie, alebo rozšírenie už existujúcej funkcie projektu.

  Epic - tento typ úlohy nevytvárajte.

- Pre pod-úlohy:

  Sub-task - vyberte v prípade, že ide o pod-úlohu inej úlohy.

  Technical task - tento typ úlohy nevytvárajte.

4. Summary - do tohto textového poľa zadajte názov úlohy. Názov musí byť stručný a opisovať danú úlohu. Pri opise činnosti používajte slovesá v neurčitku. V prípade, že ide o úlohu, ktorú bude vykonávať viacero členov tímu, každý bude mať vytvorenú svoju úlohu, pričom musia byť rozlíšené názvom. Ak to bude nutné konkretizujte názov úlohy (napíšte konkrétne balíčky/triedy v ktorých pracujete, alebo meno člena tímu ktorý bude na úlohe pracovať).

5. Priority:

Obr. 14: Ukážka nastavenia priority úlohy

- Blocker - nastavte v prípade, že ide o úlohu ktorá blokuje vývoj, alebo testovanie projektu a bez vyriešenia tejto úlohy sa v projekte nedá pokračovať.

- Critical - nastavte v prípade, že ide o úlohu, ktorá zasahuje do funkčnosti projektu a môže brániť v pokračovaní prác na niektorých úlohách.
- Major - nastavte v prípade, že ide o úlohu, ktorá zasahuje do funkčnosti projektu a môže ovplyvniť aj ostatné úlohy, no nebráni v ich plnení.
- Minor - nastavte v prípade, že ide o úlohu, ktorá nezasahuje do funkčnosti projektu a nebráni v pokračovaní projektu a ani nepredstavuje v projekte zmeny, ktoré by mohli ovplyvniť ostatné úlohy.
- Trivial - nastavte v prípade, že ide o triviálnu úlohu, ktorá nezasahuje do funkčnosti projektu a ktorá predstavuje iba "kozmetické"úpravy.

6. Due date - do tohto textového poľa nastavte dátum, do kedy musí byť úloha dokončená. Dátum nastavte kliknutím na ikonu kalendára () a vybratím dátumu, do kedy má byť úloha dokončená. V prípade, že sa na stretnutí nerozhodne inak, nastavte toto pole na dátum konca šprintu.

7. Assignee - do tohto textového poľa zadajte meno člena tímu, ktorí bude na úlohe pracovať.

8. Reporter - do tohto textového poľa zadajte meno člena tímu, ktorí za úlohu zodpovedá.

9. Description - do tohto textového poľa napíšte stručný opis úlohy. Z opisu musí byť jasné, čo sa v úlohe bude vykonávať a aký ma cieľ. V prípade, že úloha blokuje prácu na ostatných úlohách, napíšte, ktoré úlohy nemôžu byť bez dokončenia danej úlohy vykonávané a prečo. Ak úlohe musia predchádzať niektoré iné úlohy, tak napíšte ktoré a prečo.

10. Original Estimate - do tohto textového poľa zadajte odhadnutý čas pre splnenie úlohy, ktorý bol odhlasovaný na stretnutí. V prípade, že vytvárate úlohu počas prebiehajúceho šprintu a nebol pre túto úlohu odhadnutý čas, nechajte toto pole prázdne. Čas píšte v nasledujúcom formáte: Xw Xd Xh Xm, pričom X predstavuje hodnotu, w predstavuje pracovný týždeň, d predstavuje pracovné dni, h predstavuje hodiny a m predstavuje minúty (1w = 5d, 1d = 8h).

11. Remaining Estimate - toto textové pole nechajte prázdne.

12. Sprint - pokiaľ vytvárate úlohu do zoznamu úloh nechajte toto pole prázdne. Ak chcete úlohu zaradiť do vytvoreného šprintu vyberte daný šprint. Ak ide o pod-úlohu, toto pole bude automaticky nastavené podľa úlohy z ktorej je pod-úloha vytvorená.

13. Create - po vyplnení všetkých požadovaných polí, týmto tlačidlom vytvorte úlohu.

9 Manažment dokumentovania

Posledná úprava	Metod Rybár
Platné od	16. november 2014
Poznámky	

9.1 Manažment dokumentovania v tíme

Posledná úprava	Metod Rybár
Platné od	17. november 2014
Poznámky	

Hlavná dokumentácia pre tímový projekt sa vytvára v prostredí LaTeXa to najmä kvôli kompatibilite a prenositeľnosti. Vzhľadom na to, že sa hráč Jim vyvíja dlhodobo je dobré, aby sa dala dokumentácia modifikovať a prenášať aj v budúcnosti.

Dokumentáciu tvoria všetci členovia tímu. Každý člen sa venuje hlavne dokumentovaniu tej časti, ktorú ma v tíme na úlohu a dokumentovaniu úloh na ktorých pracuje.

V rámci dokumentácie riadenia preto každý člen tímu tvorí napríklad metodiky a postupy pre riadenie k svojej manažérskej pozícií v tíme. Ide o tieto pozície:

- Vedúci tímu
- Manažér rizík
- Manažér dokumentovania
- Manažér komunikácie
- Manažér kvality
- Manažér podpory vývoja
- Manažér rozvrhu

V rámci dokumentácie inžinierskeho diela členovia tímu tvoria dokumentáciu k úlohám na ktorých pracujú. Dokumentácia obsahuje čo je cieľom úlohy, ako sa na nej pracovalo a aký je výsledný stav po jej ukončení.

Dokumentácia sa tvorí podľa metodiky uvedenej v 9.2, ktorá si dáva za cieľ najmä predchádzanie konfliktov pri citovaní, odkazovaní a udržanie jednotného formátu a štýlu dokumentácie.

9.2 Metodika pre dokumentovanie v prostredí LaTeX

Posledná úprava	Metod Rybár
Platné od	16. november 2014
Poznámky	

9.2.1 Základné nastavenia

Zdrojový kód dokumentácie musí mať na začiatku zadefinovaný slovenský jazyk, veľkosť písma 12 a štýl dokumentu *article*. Ďalej musí využívať nasledovné balíky *t1enc, inputenc, babel, graphicx, url, setspace, hyperref, listings a biblatex*. Balík *inputenc* musí mať nastavené kódovanie *utf8*. Balík *babel* jazyk *slovak* a balík *biblatex backend=bibtex*. K dokumentácií prislúcha *bib* súbor s citáciami, ktorý sa zahrnie pomocou názvu súboru s *addbibresource*. Začiatok dokumentu by mal vyzerať nasledovne.

```
\documentclass[slovak,12pt]{article}
\usepackage{t1enc}
\usepackage[utf8]{inputenc}
\usepackage[slovak]{babel}
\usepackage{graphicx}
\usepackage{url}
\usepackage{setspace}
\usepackage{hyperref}
\usepackage{listings}
\usepackage[backend=bibtex]{biblatex}
\addbibresource{nazov_bib_suboru}
```

9.2.2 Hranice dokumetnu

Na začiatku zadefinujte začiatok a koniec dokumentu. Prvú stranu dokumentu bude tvoriť titulná strana. Za ňou nasleduje automaticky generovaný obsah, ktorý sa vloží pomocou *\tableofcontents*. Za obsahom zadefinujete počítanie strán od 1 pomocou *\setcounter{page}{1}*.

Základná štruktúra dokumentu by mala vyzerať nasledovne.

```
\begin{document}

\begin{titlepage}
Titulná strana 9.2.3
\end{titlepage}

\tableofcontents
\setcounter{page}{1}

\end{document}
```

9.2.3 Titulná strana

Na titulnej strane uveďte aktuálny názov školy bez skratiek. Z názvu dokumentácie musí byť jasné o čom je, teda napríklad či sa jedná o dokumentáciu k riadeniu alebo k inžinierskemu dielu, prípadne ide o iný typ dokumentácie. Pod názov uveďte všetkých autorov aj s titulmi. V pätičke uveďte názov tímu, meno vedúceho s titulmi, predmet pre ktorý je dokumentácia tvorená, aktuálny ročník, akademický rok v tvare RRRR/RRRR a mailový kontakt.

Názov školy, názov dokumentácie a autorov zarovnajte na stred, pätičku zarovnajte doprava. Použite nasledovnú šablónu tak, aby bolo všetko zarovnané. Potrebné množstvo medzier označuje také množstvo medzier koľko je potrebných, aby všetky údaje za dvojbodkami boli spolu zarovnané. Medzeru vložte ako ~.

```
\begin{center}
\large{{Názov školy}}
\end{center}
\vspace*{\fill}
```

```
\begin{center}
\Large{Názov dokumentácie}\\
\vspace*{\fill}
\normalsize{Meno Autora}
\end{center}
\vspace*{\fill}
\begin{tabbing}
(potrebné množstvo medzier)\=\\
Tím č. : \> Názov tímu\\
Vedúci projektu: \> Meno vedúceho\\
Predmet: \> Názov predmetu\\
Ročník: \> 1.\\
Akademický rok: \> RRRR/RRRR \\
Mailový kontakt: \> email
\end{tabbing}
\end{titlepage}
```

9.2.4 Delenie na sekcie

Existujú tri základné úrovne sekcií, *section, subsection a subsubsection*, pričom každá sekcia je automaticky číslovaná. Hlavné sekcie sú číslované v poradí a následné aj každá podsekcia je označená číslom sekcie do ktorej patrí a poradím v danej sekcií. Sekcia sa vytvára pre väčšie celky dokumentácie, ako je napríklad analýza podobných prác alebo časť dokumentácie venujúca sa používateľskému rozhraniu.

Podsekcia sa vytvára pre časť danej sekcie, ako analýza, návrh, implementácia a overenie. *Subsubsection* sa používa na rozdelene podsekcie v prípade, ak napríklad analýza obsahuje analýzu rôznych samostatných jednotiek.

Zo všetkých názvov musí byť jasné o čom daná časť je, a preto nepoužívajte ako názvy len slová *Analýza, Návrh* atď. ale použite napríklad *Analýza zahraničných tímov*.

9.2.5 Labels a referencovanie

Pre časti dokumentácie na ktoré sa treba odkazovať, ako nadpisy kapitol, obrázky, tabuľky atď. je nutné aby ste zadefinovali *\label{názov labelu}*.

Názov *labels* by mali byť prvé tri slová z názvu danej časti bez medzery.

Spojky a predložky sa pritom nepočítajú za slová. V prípade, že by dve kapitoly začínali rovnakými troma slovami alebo by konflikt vznikol s tabuľkami alebo obrázkami, zmeňte názov kapitoly, obrázka alebo tabuľky tak, aby ste tento konflikt odstránili.

Ak sa chceme odkázať na URL, môžeme použiť príkaz *\url{linka}*, ktorý automaticky spraví linku klikateľnou.

Label je povinný pri *section, subection, subsubsection* a pri vkladaných obrázkoch a tabuľkách.

Na označkované časti sa odkazujte pomocou *\ref{názov labelu}*, pričom ak je dokument správne vytvorený, automaticky sa vytvorí klikateľný odkaz na odkazovanú sekciu, obrázok alebo tabuľku. Ak sa odkaz hneď nevytvorí, vyskúšajte dokument skompilovať znova. Následne otestujte či budete po kliknutí na referenciu odkázaný na správnu časť v texte. Ak nie, opravte referenciu.

V prípade, že sa nevytvorí automatický odkaz ani po opätovnej kompilácií, skontrolujte, či neexistuje viacero *labels* s rovnakým názvom. Ak tomu tak je, zmeňte niektorý z nich ale nezabudnite upraviť aj názov obrázka, tabuľky alebo kapitoly.

9.2.6 Citácie a odkazovanie sa na ne

Na citácie použite Bibtex. Samotné citované texty a publikácie sa vkladajte do súboru s koncovkou *.bib* a to v nasledovnom formáte

```
@article{berghman_efficient_solutions,
title={Efficient solutions for Mastermind using genetic algorithms},
author={Berghman, Lotte and Goossens, Dries and Leus, Roel},
journal={Computers & operations research},
volume={36},
number={6},
pages={1880–1885},
year={2009},
publisher={Elsevier}
}
```

prípadne s využitím iných polí a typov definovaných v štandarde Bibtexu. Prvý riadok slúži ako *label*. Uveďte v ňom priezvisko prvého autora a prvé dve slová z názvu publikácie, pričom sa spojky a predložky za slová nepovažujú.

Súbor *bib* pomenujte rovnako ako sa volá súbor tex a tento názov uveďte na začiatku zdrojového súboru (tak ako je uvedené v **??** pomocou *\addbibresource{nazov_bib_suboru}*, pričom názov uveďte bez koncovky. Tento súbor vložte do rovnakého priečinka ako zdrojový súbor dokumentácie.

Citujte pomocou *\cite{nazov* s uvedením *labels* z citácií v súbore *.bib*. Na konci zdrojového súboru pred značku *\end{document}* uveďte *\printbibliography*, ktorý pri kompilácií vloží do dokumentácie citované texty.

9.2.7 Vkladanie obrázov a tabuliek

Pri každej časti treba dodržiavať pravidlá uvedené v Labels a referencovanie 9.2.5.

Na vkladanie obrázkov použite nasledovný formát. Šírka obrázkov musí byť pre celý dokument rovnaká a obrázky nesmú presahovať okraje. Vkladané obrázky pomenujte podľa ich *labels* a umiestnite ich k zdrojovému kódu.

```
\begin{figure}[ht!]
\centering
\includegraphics[width=10cm]{subor.jpg}
\caption{Popis obrázku}
\label{popis_obrazku}
\end{figure}
```

Pre vkladanie tabuliek použite nasledovný formát s tým, že sa počet riadkov, stĺpcov a vertikálnych a horizontálnych čiar môže meniť podľa potreby. Názov tabuľky umiestnite nad tabuľku pomocou *\caption*. Ak kvôli dlhému textu začne tabuľka pretekať, v stĺpci ktorý to spôsobuje rozdeľte text do viacerých riadkov pomocou \\. Každý stĺpec a riadok pomenujte tak, aby bolo jasné čo dané pole obsahuje.

```
\begin{table}[ht!]
\centering
\caption{Popis tabuľky}
\label{popis_tabulky}
\begin{tabular}{| l || r | r | }
\hline
riadok1 & a & b\\ \hline
riadok2 & c & d\\
```

\hline
\end{tabular}
\end{table}

9.2.8 Monitorovanie zmien v dokumentácií

Pre monitorovanie zmien v dokumentácií je nutná na začiatok každej *section a subsection* uveďte nasledovnú tabuľku, ktorá nemá klasické polia pre popis a *label*.

Posledná úprava	Meno autora poslednej zmeny v časti alebo podčasti
Platné od	Dátum poslednej zmeny
Poznámky	Prípadné poznámky k aktuálnej verzií

Za titulnú stranu vložte tabuľku *Log zmien*, ktorá bude mať nasledovné stĺpce.

Dátum zmeny	Názov sekcie	Autor	Verzia

K názvu sekcie uveďte aj referenciu. Informácie k referenciám sa nachádzajú v časti Labels a referencovanie 9.2.5.

Pri každej zmene alebo doplnení dokumentácie je treba verziu zväčšiť o 0.1 a vytvoriť nový dokument.

10 Manažment rizík

Posledná úprava	Peter Filípek
Platné od	16. november 2014
Poznámky	

Manažment rizík má za úlohu v projekte odhaliť potenciálne riziká, ktoré pri vývoji produktu môžu vzniknúť. Následne navrhnúť kroky na minimalizáciu šance vzniku rizika, prípadne navrhnúť kroky, ktoré minimalizujú následky vzniknutého rizika. Správne odhalene rizika a návrh opatrení je dôležitou súčasťou projektu a zvyšuje šance na jeho úspešné dokončenie.

V našom projekte sme identifikovali viacero rizík, ktoré sme následne zoradili podľa priority do tabuľky. Hodnoty pravdepodobnosti, že riziko vznikne sme si stanovili od najpravdepodobnejšej po najmenej pravdepodobnú takto: často, pravdepodobne, príležitostne. Hodnoty závažnosti, keď riziko vznikne sme si stanovili od najzáväznejšieho po najmenej záväzné takto: veľká, stredná, malá.

Tabuľka 10: Identifikované riziká

Riziko	Pravdepodobnosť	Závažnosť
Nerealistické rozvrhy a plány	pravdepodobne	veľká
Nepochopenie a nedostatky externe vytvorených	často	stredná modulov
Nedostatok požadovaných znalostí	príležitostne	stredná
Konflikty v rámci riešenia úloh	príležitostne	malá

10.0.1 Nerealistické rozvrhy a plány

Toto riziko je jedno z najčastejšie sa vyskytujúcich rizík. Pravdepodobnosť vzniku tohto rizika je obzvlášť vysoká pri novo založených tímoch, alebo pri prijatí nových členov do tímu. V prípade nášho tímu je pravdepodobnosť vzniku tohto rizika ešte vyššia, vzhľadom na to, že členovia tímu pracujú na viacerých školských projektoch súbežne. Vzhľadom na fakt, že členovia na ostatných projektoch pracujú oddelene, je aj vyťaženie členov tímu rôzne. Pri rozsiahlejších úlohách to môže spôsobiť, že úloha bude v určitom bode prerušená, neskôr bude na ňu naviazané a následne naviazanie už na rozpracovanú úlohu zaberie určitý čas, s ktorým treba pri plánovaní počítať.

Aby sa toto riziko minimalizovalo, na zostavovaní plánu a vyberaní úloh pre ďalší šprint sa podieľa celý tím. Využívame pri tom klasickú techniku ohodnocovania úloh pre scrum pomocou Fibonačiho kartičiek, v spojení s diskusiou o problematike riešenia danej úlohy. Pri diskusii je dôležité zhodnotiť pravdepodobnosť vzniku iných rizík pri plnení úlohy a s ohľadom na to postupovať pri tvorbe plánu. Pre spresnenie plánu počítame aj s tým, že môže dôjsť k prerušeniu úlohy pre povinnosti v rámci iných projektov. Z týchto dôvodov je pri zostavovaní 2-týždenného plánu zarátaná aj 2-hodinová rezerva pre každého člena tímu. Dôležité pre minimalizáciu následkov tohto rizika je, aby sa na úlohách začalo pracovať čo najskôr. Preto je ku každej úlohe pridelený zodpovedný člen tímu, ktorý má za úlohu dohliadnuť na včasné plnenie úlohy. V prípade, že sú v šprinte aj kritické úlohy, je nutné začať s riešením týchto úloh.

10.0.2 Nepochopenie a nedostatky externe vytvorených modulov

Toto riziko nastáva vždy, keď sa využívajú časti kódu, ktoré neboli priamo implementované v tíme. S využitím takýchto modulov podstupujeme riziko, že nemuseli byt korektne vytvorené alebo neporozumieme dostatočne ich vlastnostiam pri implementácii. Toto riziko môže mať za následok neočakávane výsledky, prípadne znefunkčnenie celého programu. Špecifické pre náš projekt je to, že pokračujeme v práci na projekte, ktorý je už dlhé roky vo vývoji a pracovalo na ňom veľa rôznych tímov. Výsledkom toho je rozsiahly projekt, ktorého dokumentácia nie je konzistentná a kompletná. Vzhľadom na povahu nášho projektu budenutné niektoré časti upraviť, čím nám vzniká ďalšie riziko, ak dostatočne nepochopíme prevzatý projekt.

Na odstránenie tohto rizika nám poslúžia dokumentácie predošlých tímov, ktoré sa na projekte podieľali. Ale vzhľadom na to, že dokumentácie nie sú konzistentné a kompletné, zostáva v našom prípade riziko stále veľké. Pre lepšie pochopenie projektu sme nadviazali kontakt s predošlým tímom, ktorý sa na projekte podieľal. Na stretnutí nám predali svoje informácie a znalosti o projekte a niekoľko rád ako postupovať a vyhnúť sa chybám. Toto nám pomohlo čiastočne minimalizovať riziko. Vzhľadom na rozsah projektu a jeho nekonzistentnosť sme sa zo začiatku venovali úlohám spojených s analýzou a refaktoringom projektu. Tieto úlohy nám pomohli k lepšiemu pochopeniu projektu a ďalšiemu minimalizovaniu tohto rizika. Vzhľadom na to, že sa venujeme aj implementácii nových funkcií do projektu, treba dbať na to, aby aj potrebné prebrané časti zostali zachované a funkčné. S týmto nám po-

môže pravidelné testovanie pomocou unit testov a vytváranie nových testov pre časti projektu, ktoré nie sú testami pokryté. Ďalej nám pri tom môžu pomôcť simulácie rôznych situácii a ich skúmanie, či sa vykonávajú správne. Vzhľadom na fakt, že toto riziko sa nedá odstrániť, využívame prostriedky na verziovanie kódu, ktoré výrazne minimalizujú škody keď toto riziko vznikne.

10.0.3 Nedostatok požadovaných znalostí

Rôzna úroveň znalostí v tíme je častá a preto treba s ňou rátať. Na začiatku, pri tvorbe tímu je často spôsobená rôznym zameraním členov tímu, prípadne ich schopnosťami. Tieto rozdiely sa časom zmenšujú, ako sa členovia postupne oboznamujú s projektom. Napriek tomu, že tento rozdiel v úrovni znalostí sa zmenšuje, môže aj narastať. K nárastu prichádza pri deľbe úloh medzi členov tímu, vzhľadom na fakt, že nie vždy sa celý tím podieľa na danej úlohe.

Vzhľadom na fakt, že náš tím prebral prácu po iných tímoch, sme členom s nižšou úrovňou znalostí pridelili úlohy, ktoré sú vhodné na oboznámenie sa s projektom a jeho vlastnosťami. Jednalo sa najmä o úlohy refaktoringu, pri ktorých členovia tímu analyzovali hlavné časti projektu, čím nadobudli užitočné znalosti. Vzhľadom na to, že v tíme je malo úloh, na ktorých sa podieľa celý tým, je nutné vytvárať dokumentáciu k daným úlohám a počas stretnutí priebežne informovať ostatných členov s postupom a stavom danej úlohy. Úlohou členov tímu, ktorí sa na úlohe nepodieľali, je v prípade nejasností počas stretnutí pýtať sa a oboznámiť sa s dokumentáciou k danej úlohe.

Toto riziko treba zohľadňovať najmä pri plánovaní a prideľovaní úloh.

10.0.4 Konflikty v rámci riešenia úloh

Riziko že nastanú konflikty pri riešení úloh môže nastať, ak sa úlohy navzájom prekrývajú alebo na jednej úlohe pracuje viacero ľudí naraz. Môžu nastať rôzne problémy, ako konflikty pri spájaní kódu, alebo zbytočné zdvojenie práce.

Pre minimalizáciu tohto rizika je dôležité plánovanie a dobrá komunikácia medzi členmi tímu. Pri vyberaní úloh do šprintu je vhodné vyberať také úlohy, ktoré sú najmenej náchylné kolíziám pri spájaní kódu, t.j. tie, čo sú najviac separátne. V prípade, že na úlohe pracuje viacero ľudí, je nutné si stanoviť postup práce a určiť, ktorý člen vykoná ktorú časť. Pre minimalizáciu rizika vzniku konfliktov je vhodné vytvoriť komunikačný kanál pre členov

tímu, ktorí úlohu riešia, kde sa bude značiť aktuálny stav práce. Za účelom minimalizácie aj tohto rizika, sú k úlohám prideľované zodpovedné osoby, ktoré v prípade že na úlohe pracuje viacero ľudí, by mali mať prehľad o stave vykonanej práce, aby sa zamedzilo zbytočným konfliktom a duplicitám.

V našom tíme využívame viacero nástrojov a komunikačných prostriedkov, ktoré nám pomáhajú toto riziko minimalizovať. Ktoré nástroje a kanály sa budú používať, sa určí na základe typu úlohy.

11 Manažment podpory vývoja a integrácie

Posledná úprava	Martin Vrabec
Platné od	8. december 2014
Poznámky	

11.1 Metodika pre vytváranie automatických testov v nástroji Bamboo

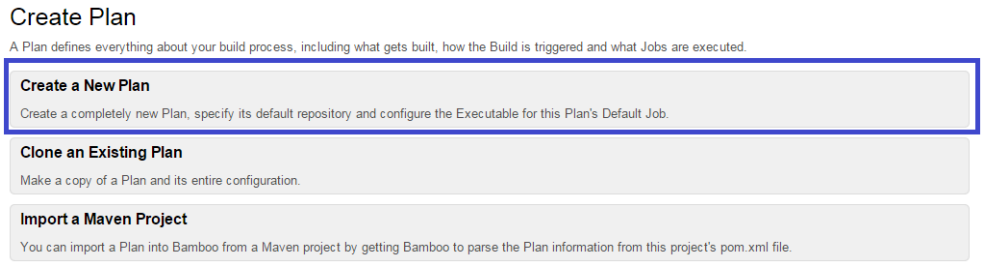
Posledná úprava	Michal Segeč
Platné od	8. december 2014
Poznámky	

Táto metodika sa zaoberá vytváraním automatických testov v nástroji Bamboo. Zavedením automatických testov zaniká povinnosť ručne testovať jednotlivé programy na funkčnosť, pretože tieto testy sa vykonávajú pravidelne. Ako pravidelne sa vykonajú tieto testy treba už nastaviť v samotnom nástroji.

Je potrebné mať vytvorený projekt v nástroji Bamboo. Keďže využívame nástroj v rámci našej univerzity, je potrebné získať administratorské práva, aby bolo možné vytvárať testy. Je vhodné využívať aj ostatné nástroje, na ktoré ma univerzita licenciu a to Jiru, pre vedenie záznamov o úlohách v rámci jednotlivých šprintov, ale taktiež Bitbucket, ako repozitár zdrojového kódu. Po získaní administratorských práv bude vytvorený projekt v systéme, ktorý je prepojený s vytvoreným projektom v Jire.

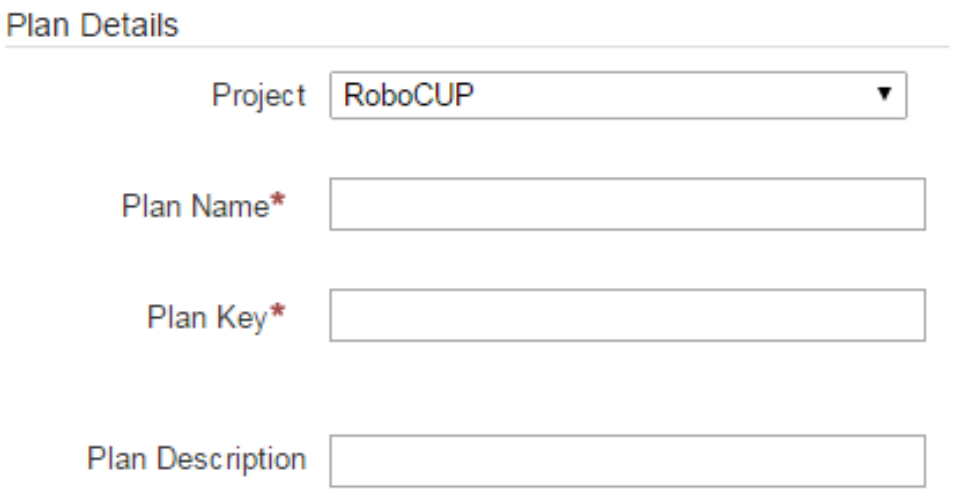
Keď už existuje vytvorený projekt v nástroji Bamboo, môže sa prejsť na vytváranie samotných testov. Každé testovanie sa vykonáva v rámci nejakého vytvoreného plánu. V rámci vývoja každého projektu sa vytvorí v repozitári zdrojového kódu niekoľko vetiev. Aby boli jednotlivé testovania prehľadné, vytvárajte vždy práve jeden plán pre testovanie jednej vetvy. Samozrejme, ak je potrebné testovať viacero vetiev v rámci jedného plánu, alebo jedna vetva v rámci viacerých plánov, je to možné, ale nedoporučuje sa to. Pre vytvorenie nového plánu, kliknite v nástroji Bamboo na tlačidlo v pravom hornom rohu – Create Plan. Otvorí sa Vám nové okno, v rámci ktorého máte na výber z 3 možností. Kliknite na prvú možnosť vytvorenia nového plánu

(Obr. 15). Druhou možnosťou je vytvorenie kópie už existujúceho plánu – tuto možnosť je možné si zvoliť, ak už máme existujúci plan a nový plan je iba trochu odlišný od pôvodného plánu. Treťou možnosťou je importovanie Mave projektu. Touto možnosťou sa v tejto metodike nezaoberáme.



Obr. 15: Vytvorenie nového plánu

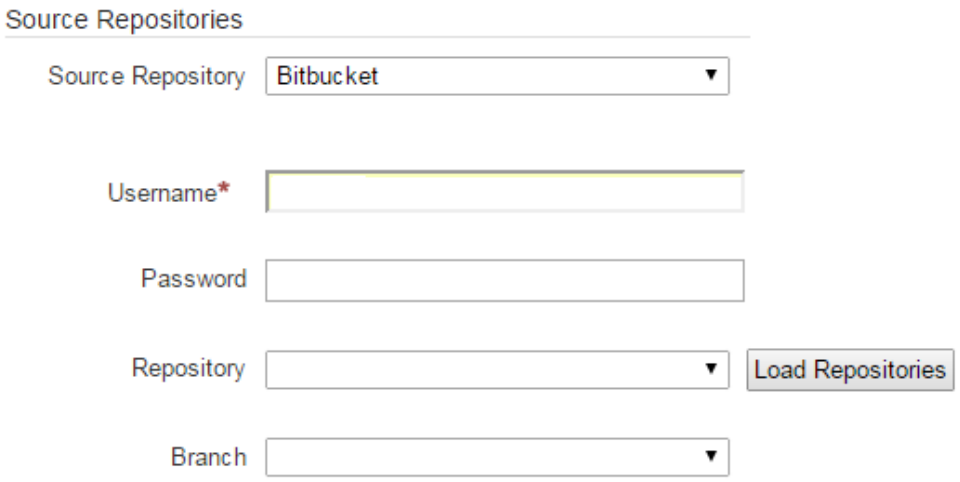
Otvorí sa Vám nové okno (Obr. 16), v ktorom je potrebné vyplniť pole projekt (Project), v rámci ktorého sa vytvorí plán. Druhé pole nazvané Plan Name označuje identifikátor plánu, čiže ako bude nazvaný. Do poľa Plan Key vložte projektu a musíte ho zadať veľkými písmenami. Najlepšie je zvoliť taký názov, ktorý najlepšie vystihuje celý plán.



Obr. 16: Detaily plánu

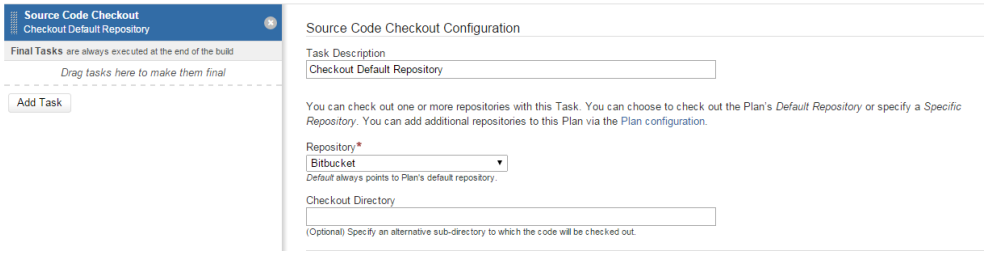
Nasledujúce polia slúžia pre výber repozitára zdrojového kódu. Nasleduje pole Source Repository, kde si zvoľte repozitár, kde sa nachádzajú vaše zdrojové kódy (Obr.17). V tomto prípade je to repozitár Bitbucket. Do ďalších polí – Username a Password vložte svoje prihlasovacie údaje do nástroja Bitbucket. Po vyplnení týchto údajov kliknite na tlačidlo Load Repositories, kde prebehne autentifikácia, či ste zadali správne údaje. Ak ste ich zadali správne, vytvorí sa spojenie a sprístupnia sa vám jednotlivé repozitáre, ktoré máte vytvorené v nástroji Bitbucket. Zvoľte si požadovaný repozitár v poli Repository. Po zvolení repozitára sa vám v poli Branch zobrazia všetky vetvy, ktoré sú vytvorené v danom repozitári. Tu si zvoľte vetvu, ktorú chcete testovať.

Posledným krokom pri vytváraní samotného plánu je zadať spúšťač, ktorým zabezpečíme to, ako často sa budú púšťať jednotlivé testy. Výber spúšťačov je na vašom uvážení. Je avšak vhodné kontrolovať projekt po každom upravení zdrojového kódu, čím je potom jednoduchšie odhaliť, kde nastala chyba a testy zlyhali. Túto možnosť si zvolíte v poli Trigger type, ako Repository triggers the build when changes are commited. Touto možnosťou sa budú testy spúšťať vždy, keď niekto použije príkaz commit a tým upraví zdrojový kód. Pole Trigger IP Addresses nechajte prázdne. Ak ste vyplnili všetky údaje, kliknite na tlačidlo Configure Tasks.



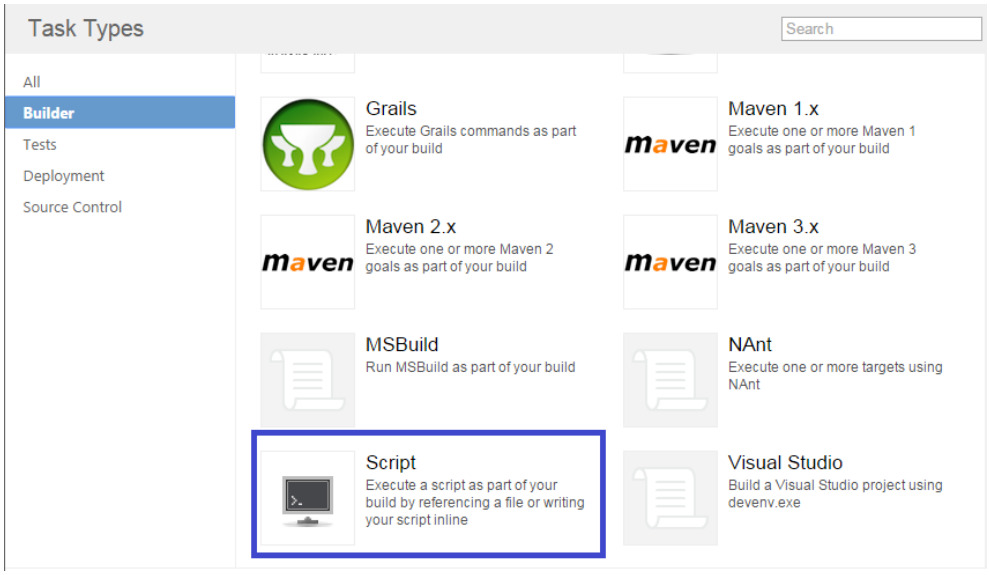
Obr. 17: Nastavenie zdrojového repozitára a spúšťačov

Teraz prichádza ten najdôležitejší krok a to určenie, ktoré testy sa vykonajú. Defaultne je nastavená úloha, ktorá sa vykoná pre každý plán, a to je Source Code Checkout (Obr. 18). Táto úloha natiahne kód z repozitára zdrojového kódu a pripraví ho pre spracovanie jednotlivými úlohami. Pre prípad, že by chceli ste pracovať s viacerými repozitármi naraz, je možné pridať do tejto úlohy ďalší repozitár, kde je dôležité zvoliť najmä umiestnenie repozitára – pole Repository. Dodatočné polia slúžia pre vytvorenie popisu úlohy – Task Description a pre možnosť zvolenia podadresára, kam sa uloží pripravený kód.



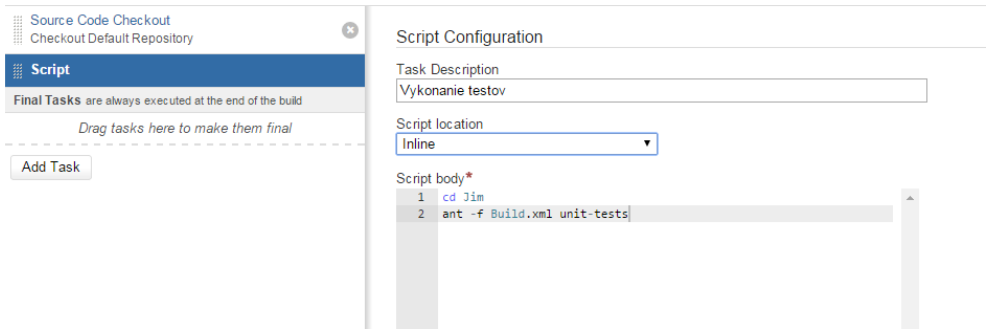
Obr. 18: Pridanie úloh do plánu

Pre pridanie novej úlohy, kliknite na tlačidlo Add Task. Tu už chceme zabezpečiť spúštanie jednotlivých testov. Minuloročný tím (Gitmen - ak.rok: 2013/2014) vytvoril súbor v jazyku ant – Build.xml, ktorý existuje pre všetky 3 projekty. Tento súbor obsahuje základ pre vykonanie unit testov. V prípade, že chcete vykonať testy nad určitými triedami, alebo triedou, je potrebné upraviť tento súbor, aby sa vedelo, ktoré testy sa majú vykonať. Tým je prvá časť úlohy splnená. Teraz treba vytvoriť skript, ktorý zabezpečí spustenie tohto súboru. Kliknite na tlačidlo Script (Obr. 19).



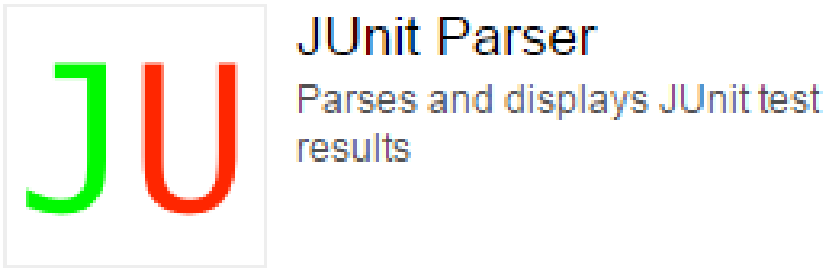
Obr. 19: Voľba druhov testovania

Pre vytvorenie skriptu si zvoľte názov tohto skriptu – Task Description. Následne si môžete zvoliť, či sa skript bude písať priamo v okne, alebo je možné ho načítať do súboru. Vzhľadom na jednoduchosť skriptu je jednoduchšie ho priamo napísať do okna, ako vidíte na (Obr. 20). Opíšte tieto 2 riadky, ako je uvedené v časti Script body. Skript vykoná, že sa prejde do priečinka Jim a spustí sa súbor Build.xml nachádzajúci sa v projekte Jim. Prepínač –f umožňuje zvolenie si build súboru. Slovné spojenie unit-tests spúšťa target, čiže cieľovú časť bloku definovanom v súbore Build.xml, v ktorom sú definované, ktoré testy sa vykonajú a kam sa má odoslať výstup vykonaných testov. Kliknite na tlačidlo Save, aby sa uložila konfigurácia.



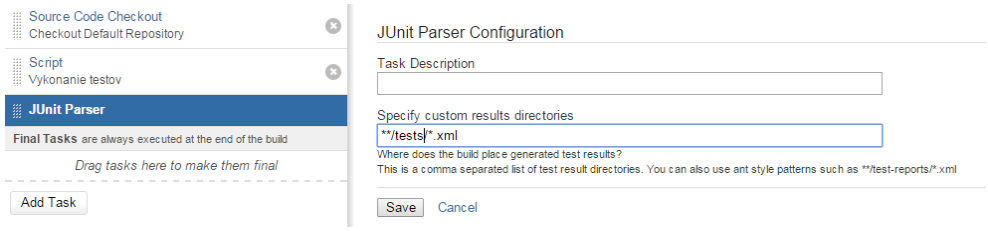
Obr. 20: Konfigurácia skriptu

Teraz je potrebné zabezpečiť, aby sa testy vhodne vyhodnotili a bolo ich možné zobraziť. Pre túto možnosť opäť kliknite na tlačidlo Add Task a namiesto tlačidla Script, ktoré ste volili v prechádzajúcom kroku ako je uvedené na (Obr. 19) zvoľte možnosť JUnit Parser, ktorý testy zanalyzuje a umožní zobrazenie ich výsledkov (Obr. 21).



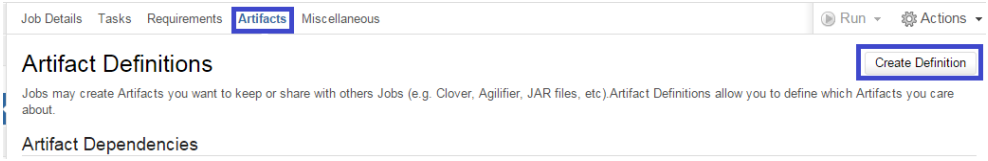
Obr. 21: Výber možnosti pre spracovanie testov

Tu je potrebné zmeniť iba jednu vec. V súbore Build.xml sa v časti bloku, ktorú sme spomínali už vyššie – unit-tests nachádza súbor, kam sa majú ukladať analyzované testy. Tento priečinok je nazvaný ako tests. Preto vložte do poľa Specify custom results directories (Obr. 22) priečinok tests, kam sa budú ukladať výstupy analýzy testov. Kliknite na tlačidlo Save pre uloženie konfigurácie.



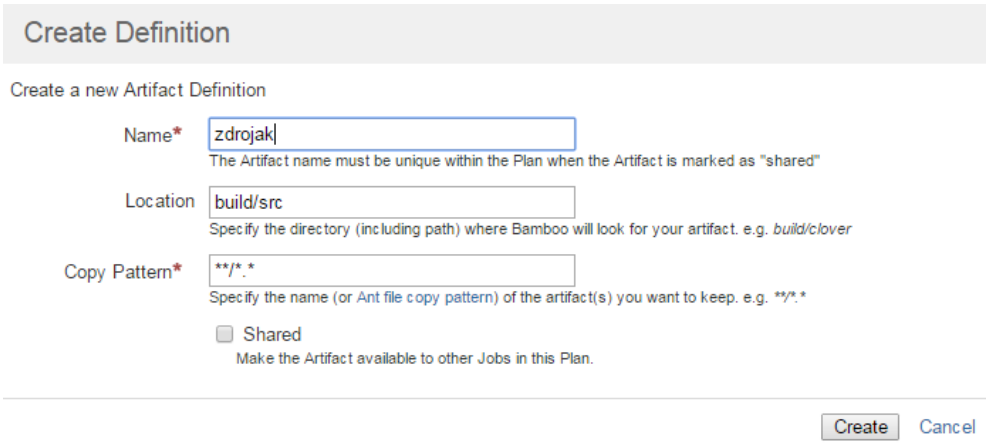
Obr. 22: Zvolenie cieľového priečinka

Testovanie je tým skoro ukončené. Preto, aby bolo možné ešte zobraziť výsledky testov, je potrebné vytvoriť tzv. Artefakt. Kliknite na tlačidlo Artifacts (Obr. 23) a následne na tlačidlo Create Definition, pre vytvorenie nového artefaktu.



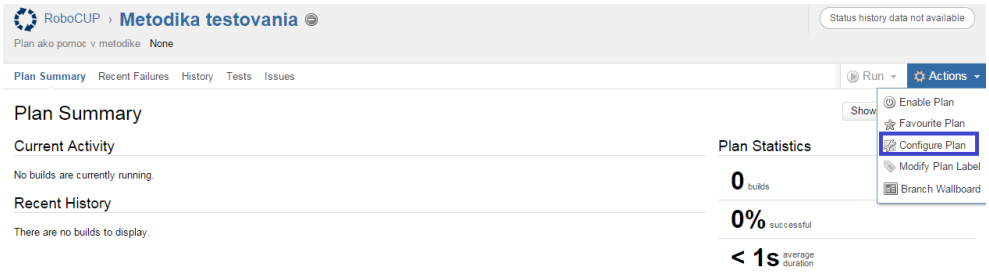
Obr. 23: Definície artefaktu

Je potrebné vyplniť názov artefaktu – pole Name. Za poľom Location sa ukrýva cesta, kde má byť umiestnený artefakt. Posledným krokom je špecifikovať, ktoré artefakty sa uchovajú – pole Copy Pattern. Zadajte konfiguráciu ako je zobrazené na (Obr. 24). Tlačidlom Share sa dá artefakt zdieľať medzi ďalšími úlohami v rámci tohto plánu. Nechajte toto pole teraz voľné. Uložte konfiguráciu kliknutím na tlačidlo Save. Týmto je artefakt vytvorený a je možné už spustiť testovanie.

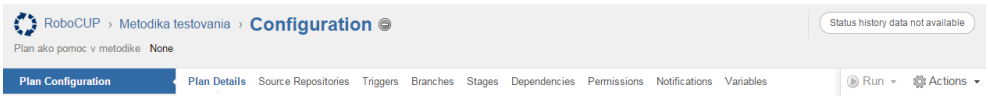


Obr. 24: Vytvorenie artefaktu

Ako je možné vidieť na (Obr. 25), plan je vytvorený a ešte neboli vykonané žiadne testovania. Aby bolo možné testovania vykonať, je treba zapnúť vytvorený plan. Predtým je však ešte možnosť nastaviť ďalšie funkcie tohto plánu, ktoré je taktiež možné upravovať, keď už je plán spustený. Kliknite na tlačidlo Actions a následne na tlačidlo Configure Plan, čím sa dostanete do konfigurácie plánu.



Obr. 25: Konfigurácia plánu



Obr. 26: Možnosti konfigurácie plánu

Na (Obr. 26) môžete vidieť viaceré možnosti, ktoré môžete nastaviť vo svojom plane.

Source Repostitories – voľba repozitárov pre plán

Triggers – spúšťače, ktoré zabezpečujú vykonanie plánu. Ich počet môže byť rôzny. V rámci jedného plánu môže existovať aj viacero spúšťačov.

Branches – pridávanie vetiev do plánu

Stages – stanovenie fáz vykonania testovania

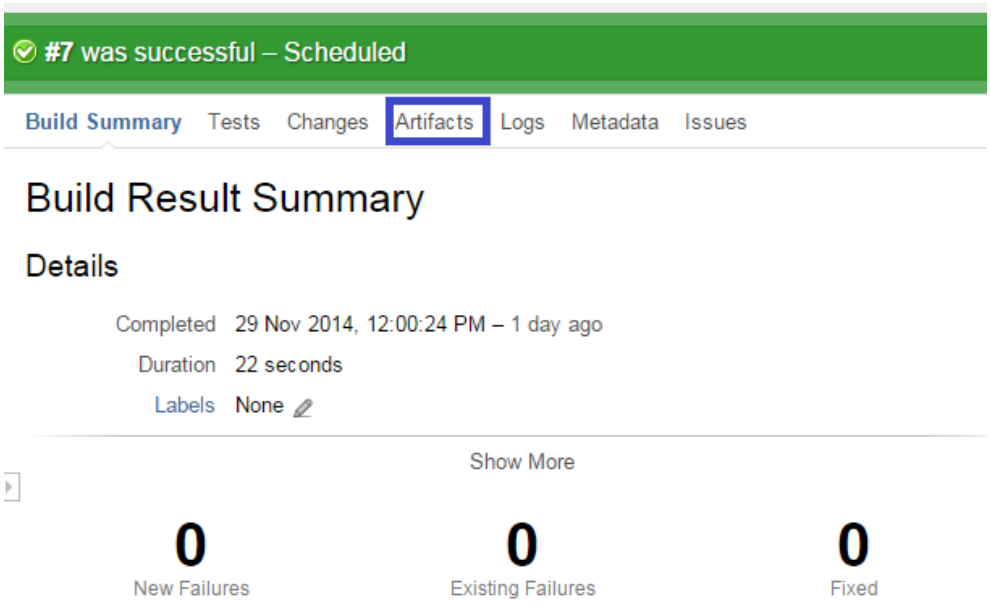
Dependencies – nastavenie závislosti medzi plánmi

Permissions – nastavenie prístupu k danému plánu

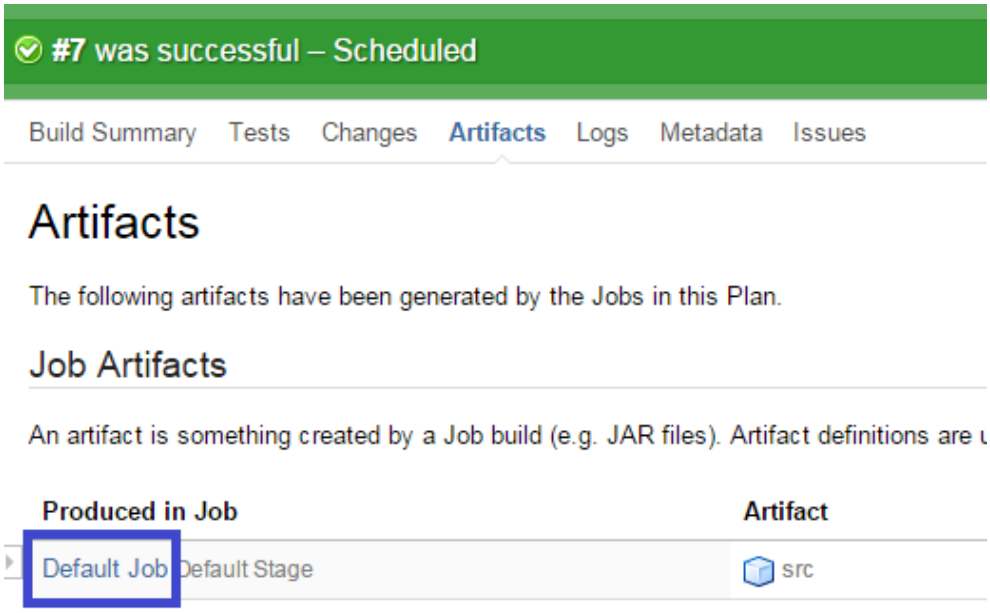
Notifications – nastavenie upozornení pre daný plan

Variables – konfigurácia premenných plánu

Pokiaľ ste skončili s upravovaním plánu, kliknite opäť na tlačidlo Actions (Obr 27 28) a následne na tlačidlo Enable Plan. Plán je pripravený na vykonanie. Ak chcete manuálne vykonať testovanie, stlačte tlačidlo Run (Obr. 26), čím sa vykoná testovanie. Tieto testovania sa ďalšie razy vykonajú na základe nastavených spúšťačov.

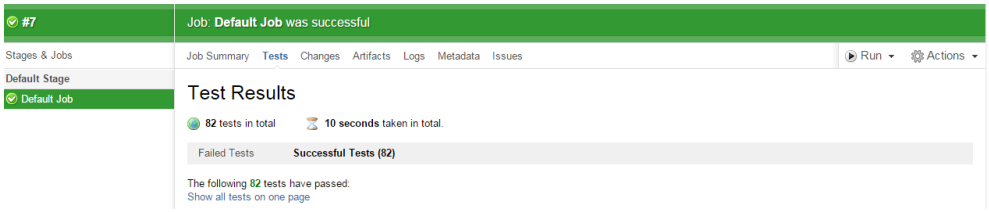


Obr. 27: Zobrazenie výsledkov testovania A



Obr. 28: Zobrazenie výsledkov testovania B

Po prekliknutí sa do artefaktov si môžete zobraziť výstupy testovania, t.j. všetky úspešné a neúspešné testy (Obr 29).



Obr. 29: Zobrazenie úspešných vs. neúspešných testov

11.2 Metodika manažmentu verziovania

Posledná úprava	Martin Vrabec
Platné od	16. november 2014
Poznámky	

Táto metodika určuje pravidlá a postupy pri organizácii práce s nástrojom na správu verzií. Určuje postupy a konvencie, ako s týmto nástrojom zaobchádzať v rámci práce v tíme. Podrobne opisuje postup pri práci s vetvami, pomenovávanie a písanie commit správ a postupy pri synchronizácii so vzdialeným serverom.

Pojmy:

- git – voľne dostupný distribuovaný systém na správu verzií projektu
- vetva (branch) – vývojová línia umožňujúca vývoj oddelene od iných vetiev, vetvy je možné spájať aj rozdeľovať
- master – hlavná vetva projektu
- commit – odoslanie zmien v zdrojovom kóde do lokálneho repozitára
- merge – zlúčenie vývojových vetiev, ktoré zahŕňa aj riešenie konfliktov toho istého súboru v rôznych vetvách
- rebase – získanie zmien vykonaných na inej vetve za účelom aktualiácia aktuálnej vetvy
- push – vloženie vetvy z lokálneho vývoja na inú vetvu vzdialeného repozitára
- Jira – nástroj na plánovanie a organizáciu práce v tíme

11.2.1 Role a zodpovednosti členov tímu

- Vývojár

  Správa lokálneho úložiska

  Vývoj nových funkcionalít, prípadne zmena existujúceho zdrojového kódu

  Oprava chýb

- Manažér podpory vývoja

  Správa vzdialeného úložiska

11.2.2 Správa vzdialeného úložiska

Zodpovedný: manažér podpory vývoja

Pre správu verzií zdrojového kódu sme si zvolili nástroj git, ktorý umožňuje prácu viacerými vývojármi, sledovanie zmien v zdrojovom kóde ale aj prácu s vetvami. Ako vzdialený repozitár sme použili už existujúci repozitár na adrese `https://bitbucket.org/robocup_tp09/`, ktorý používali predchádzajúce tímy RoboCupu. Okrem toho tento repozitár používajú aj riešitelia bakalárskych a diplomových prác s témou robotického futbalu. Z týchto dôvodov sa pracuje vo vlastnom vlákne TP_2014, ktorý považujeme za naše master vlákno.

11.2.3 Správa lokálneho úložiska

Lokálny repozitár si každý vývojár spravuje samostatne na svojom počítači. Tento proces zahŕňa nasledovné aktivity:

- Inštalácia nástroja git
- Výber adresára pre git a vytvorenie git repozitára
- Vytvoriť používateľské konto pre vzdialený repozitár – Bitbucket
- Klonovať vzdialený repozitár do lokálneho repozitára

Postup:

- Stiahnite si lokálneho klienta pre git z adresy `http://git-scm.com/downloads`
- Inicializujte lokálny repozitár vo vybratom priečinku príkazom:

  ```
  git init
  ```

- Pomocou svojho konta na Bitbucket naklonujte do pripraveného repozitára vzdialený repozitár:

  ```
  git clone https://<login>@bitbucket.org/robocup_tp09/agent.git
  ```

Posledná úprava	Martin Vrabec
Platné od	16. november 2014
Poznámky	

11.3 Vývoj novej funkcionality

Pre každú úlohu v rámci Jiry, v rámci zmeny zdrojového kódu, bude vytvorená jedna vetva, kde budú vývojári riešiť menšie úlohy. Tieto vetvy bude vždy vytvárať manažment podpory vývoja

11.3.1 Vetvenie (branch projektu)

Novú vetvu do vzdialeného repozitára pridáva vždy len manažér podpory vývoja. Každá jedna vetva nad tímovou vetvou (TP_2014) identifikuje jednu úlohu v rámci Jiry.

- Pred začatím prác na úlohe, spravidla po začiatku šprintu, vývojár stiahne do svojho repozitáru nové vetvy zo vzdialeného repozitáru:

  ```
  git fetch
  ```

- Nastaví sa do vetvy úlohy, na ktorej vykonáva pridelenú úlohu, pomocou príkazu:

  ```
  git checkout <nazov vetvy>
  ```

11.3.2 Ukladanie zmien v zdrojovom kóde

Pre vytváranie zmien v zdrojovom kóde, prosím, dodržujte nasledovné pravidlá:

- Jedna zmena (commit) rieši len jeden problém. Ďalší problém zahrňte do ďalšej aktualizácie
- V rámci jednej aktualizácie je možné do zmeny zahrnúť aj viac súborov. Takto zahrnuté súbory sa však musia týkať jedného problému.
- Aktualizáciu urobte len pri všetkých ukončených zmenách.

  Pri problémoch, ktoré si vyžadujú kompletné prehľadanie projektu, aktualizujte zmeny len pri vyriešenom probléme v rámci jedného balíčka. Neaktualizujte rozpracované súbory.

- Pred ukončením práce na projekte v daný deň je potrebné vykonať aktualizáciu zmien (commit). Táto zmena musí spĺňať všetky vyššie popísané pravidlá, funkčnosť však nemusí byť hotová. V tomto prípade autor musí zapísať tieto skutočnosti do správy aktualizácie.

  Toto pravidlo je zavedené hlavne kvôli riešeniu problému viacerými autormi.

- Pri vytváraní aktualizácie zmeny (commit) je nutné aktualizáciu popísať podľa pravidiel uvedených nižšie v tomto dokumente.

Pred vykonaním zmeny je nutné do pripravovanej aktualizácie pridať súbory s obsiahnutou zmenou. Prehľad vykonaných zmien sa získa zadaním príkazu:

```
git status
```

Pre pridanie súboru do aktualizácie zadajte nasledovný príkaz

```
git add <nazov suboru>
```

Pre pridanie všetkých zmenených súborov do aktualizácie zadajte príkaz

```
git add .
```

Potom, čo sú do pripravovanej aktulizácie zahrnuté všetky súbory podľa všeobecných pravidiel popísaných vyššie, vykonajte aktualizáciu nasledovným príkazom:

```
git commit
```

Dva posledné príkazy je možné zlúčiť a tak je možné zahrnúť všetky zmenené súbory do zmeny a vytvoriť zmenu jedným príkazom:

```
git commit -a
```

11.4 Správa zmeny v zdrojovom kóde

Posledná úprava	Martin Vrabec
Platné od	16. november 2014
Poznámky	

Po spustení príkazu git commit spustíte textový editor, v ktorom sa zadáva popisovacia správa.

- Správu píšte v slovenskom jazyku v trpnom rode a bez diakritiky
- Jeden riadok správy musí obsahovať kvôli prehľadnosti maximálne 80 znakov, v prípade prekročenia pokračujte na druhom riadku

Správa musí byť dostatočne vysvetľujúca, aby bolo zrejmé, k akej zmene došlo. Tvar správy musí byť nasledovný:

```
<ID ulohy> [<typ ulohy>] <predmet aktualizacie>
<<Autor zmeny>>
<Sprava>
<Paticka>
```

11.4.1 Hlavička správy

ID úlohy - Identifikačné číslo úlohy. Zadávajte identifikátor z Jiry v tvare INFINITY <číslo úlohy>. Priradené číslo musí byť zhodné s číslom čiastkovej úlohy (sub-task), na ktorej v rámci aktualizácie zmien pracujete. Príklad: INFINITY 112

Typ úlohy - Pri type úlohy vyjadrite, ktorý typ daná aktualizácia zmien (commit) rieši. Povolené sú len tieto typy:

- Add – Nová implementácia v zdrojovom kóde
- Fix – Oprava chyby v zdrojovom kóde
- Refactor – Refaktoring zdrojového kódu
- Test – Vytvorenie testu
- Merge – Pri opravovaní chyby pri zlučovaní vetiev
- Iná aktualizácia, ktorá sa netýka zdrojového kódu

Typ úlohy uvádzajte v hranatých zátvorkách. Príklad: [Add]

Predmet aktualizácie (commit) - Predmet aktualizácie súborov píšte tak, aby vyjadroval, čo ste pre danú aktualizáciu zmien urobili. Dbajte na to, že predmet musí byť stručný. Podrobne popísať zmeny môžete v tele správy. Pri písaní predmetu sa riaďte nasledovnými pravidlami:

- Predmet aktualizácie musí mať spolu ID úlohy a typom úlohy dĺžku práve jedného riadku a maximálne 80 znakov

- Prvé slovo predmetu začínajte veľkým písmenom, ostatné okrem názvom píšte malým písmom
- V prípade, že sa jedná o neukončený logický celok v rámci úlohy (napr. pri refaktorovaní tried v rámci balíčka zostane ešte niekoľko tried, ktoré neboli refaktorované), uveďte na záver predmetu medzerou oddelene od posledného slova rímskou číslicou oddelenie časti. Príklad: Uprava baliku sk.fiit.jim.agent.highskill podla konvencie I a Uprava baliku sk.fiit.jim.agent.highskill podla konvencie II

Za hlavičkou správy vynechajte prázdny riadok

Autor zmeny - Autorom aktualizácie zmeny v zdrojovom kóde je pisateľ tejto aktualizácie. Sem teda napíšte vaše meno a priezvisko v zátvorkách <>. Príklad: <Ján Matejka>

11.4.2 Správa

Súvislý text s popisom vykonaných zmien v rámci zdrojového kódu. Zmeny, ktoré ste vyrobili sa snažte opísať čo možno najstručnejšie a všeobecne, tak aby bol tento text sprievodnou správou pre daný commit. Riaďte sa týmito pravidlami:

- Správu píšte ako súvislý text.
- Maximálny počet znakov na jeden riadok je 80. Počet riadkov je však neobmedzený.
- V správe neuvádzajte žiadne skratky
- Môžete sa odvolávať na predchádzajúce commit správy

Za správou vynechajte prázdny riadok.

11.4.3 Pätička správy

V pätičke správy môžete písať dodatočné správy o vykonaných zmenách. Ak zmena opravuje nejaké chyby, každú takúto chybu napíšte na jeden riadok, začínajte vždy FIX. Príklad: FIX Komunikácia agenta so serverom

Do pätičky môžete zadávať aj vlastné poznámky. V takých prípadoch každý riadok začínajte znakom +

```
INFINITY-112 [Refactor] Uprava logovania v balickoch jim.*
<Martin Vrabec>

V balickoch popisanych v poznamkach (+) bol odstraneny stary
logger, inicializovany novy logger a logovania upravene podla
neho

+ balicky: jim.annotations - jim.gui,
jim.agent.communication, jim.agent.highskill
- jim.agent.models

ion, jim.agent.highskill - jim.agent.models
```

Po vykonaní zmeny je potrebné túto zmenu previesť aj na vzdialený (spoločný) repozitár. Tento úkon sa vykoná príkazom:

```
git push
```

11.5 Spájanie vetiev

Posledná úprava	Martin Vrabec
Platné od	16. november 2014
Poznámky	

Keď sú práce na vetve ukončené, zvyčajne pri ukončení šprintu, táto vetva sa spojí s hlavnou vetvou pre náš tím: TP_2014

1. Nastavte sa do hlavnej vetvy projektu: TP_2014

```
git checkout TP_2014
```

2. Zlúčte vetvu, na ktorej bola vykonávaná úloha, s aktuálnou vetvou TP_2014:

```
git merge TP_2014
```

11.5.1 Riešenie konfliktov pri spájaní vetiev

Ak sa obidve vetvy obsahujú rôzne artefakty jedného súboru, pri ich vzájomnom zlučovaní dôjde ku chybe, ktorú je potrebné odstrániť manuálne.

1. Otvorte súbor, kvôli ktorému vznikol konflikt, alebo viac konfliktov, pre textový editor vim je tento príkaz

   ```
   vim <cesta k suboru>
   ```

2. Blok kódu, ktorý spôsobuje konflikt je vyznačený spolu s názvom vetvy, ku ktorej patrí

3. Vyberte si verziu kódu, ktorú chcete ponechať a odstráňte verziu kódu z ostatnej vetvy

4. Kvôli vykonanej zmene v kóde je potrebné vykonať commit, pridajte opravený súbor pre commit a vykonajte commit podľa nasledovného príkladu (pre zlučovací commit stačí vytvorenie krátkej správy s príznakom -m):

   ```
   git add .
   git commit –m "[Merge] <Nazov zlucovanej vetvy>
   ```

5. Opakujte pokus o zlúčenie vetiev

11.6 Metodika pre vytváranie vetiev v prostredí Bitbucket a importovanie projektu v prostredí Eclipse

Posledná úprava	Michal Segeč
Platné od	19. november 2014
Poznámky	

Táto metodika sa zaoberá vytváraním vetiev v prostredí Bitbucket, stiahnutím požadovanej vetvy do lokálneho repozitára počítača a následným importovaním projektu z danej vetvy.

Metodika je určená primárne pre celý tím, ale taktiež ju môže využiť každý, kto využíva na uchovávanie zdrojového kódu niektorý z dostupných distribučných revíznych systémov typu Github a Bitbucket, avšak s drobnými obmenami, keďže sa v metodike zameriavame primárne na potreby nášho tímu. Zároveň metodika môže byť vítaným prínosom pre tých, kto uprednostňujú narábanie s nástrojom git pomocou grafického rozhrania namiesto príkazového riadku. V našom tíme ako repozitár zdrojového kódu

využívame prostredie Bitbucket, takže metodika je venovaná práci s ním. Projekt vykonávame v programovacom prostredí Eclipse, v jazyku Java.

Metodika nepracuje s príkazovým riadkom, ale s grafickým rozhraním a nevyužívajú sa teda priamo príkazy nástroja git. Tieto príkazy sú zabudované priamo v nástrojoch.

Použité pojmy:

- vetva – angl. branch – nezávislá časť vyvíjaného projektu, úpravy v nej sa neprenášajú na ostatné vetvy
- Git, Bitbucket – distribučné voľne dostupné revízne nástroje, slúžiace na uchovávanie zdrojových kódov
- repozitár – miesto uloženia zdrojového kódu
- master vetva – hlavná vetva vyvíjaného projektu
- Jira – software pre sledovanie procesu vytvárania produktu

Hlavné kroky metodiky:

- vytvorenie vetvy pre požadovanú úlohu
- vytvorenie lokálneho repozitára z vytvorenej vetvy
- importovanie projektu z lokálneho repozitára

Pri vytváraní vetvy treba postupovať nasledovne:

Počítame s tým, že je v prostredí Bitbucket vytvorený repozitár so zdrojovým kódom projektu. Pre celý projekt tak existuje jedna hlavná master vetva.

Pri každom upravení programu môže nastať nejaký omyl a program je tak nevykonateľný. Aby ostala pôvodná verzia programu zachovaná, vytvárajte pre každú úlohu v rámci tímového projektu novú vetvu, v ktorej sa budú vykonávať všetky úpravy v rámci danej úlohy. Táto novo vytvorená vetva obsahuje všetky súbory ako master vetva ale s rozdielom, že jej úprava neovplyvní master vetvu a tak pôvodná verzia ostane zachovaná.

V rámci tímového projektu vykonávate množstvo úloh, pričom každá úloha má svoje označenie. Všetky úlohy, ktoré sa majú vykonať v rámci projektu sa nachádzajú v nástroji Jira. Každá takáto úloha má svoje označenie [názov_tímu-číslo_úlohy]-popis_úlohy. Je nutné, aby pre každú úlohu

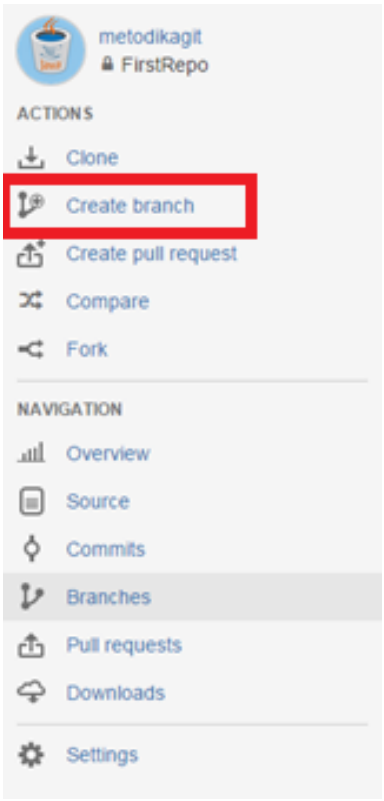
bola vytvorená práve jedna vetva. V prípade, že by bolo vytvorených viac vetiev na jednu úlohu, mohlo by dochádzať ku konfliktom. Aj keď máte ako viacerí členovia tímu pracovať na spoločnej úlohe, všetci pracujte v rámci vytvorenej jednej vetvy pre danú úlohu.

1.krok – V rámci repozitára, vytvoreného v prostredí Bitbucket kliknite na tlačidlo Create branch, nachádzajúceho sa na ľavej lište stránky. (Obr. 30))

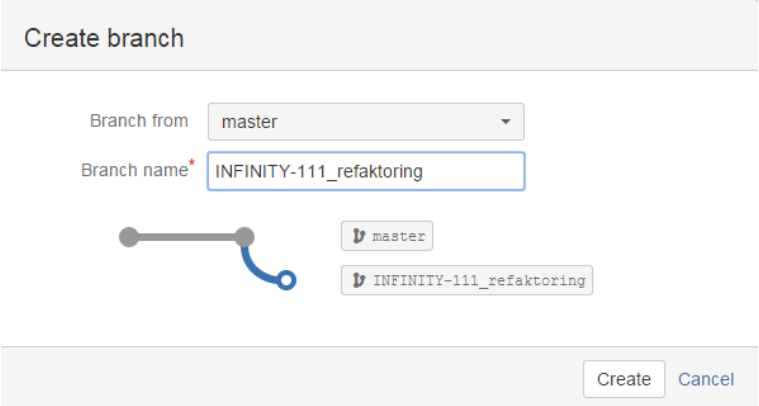
2.krok – Otvorí sa vám nové okno, kde si v poli popísanom ako Branch from máte zvoliť vetvu, z ktorej sa nová vetva má vytvoriť. Zvoľte master vetvu, od ktorej sa odvodí nova vetva so všetkými súbormi hlavnej master vetvy. (Obr. 31))

Druhá voľba, ktorú treba vyplniť je pole popísané ako Branch name. Do tohto poľa máte vyplniť názov vetvy. Pre lepšiu konzistenciu uvádzajte názov vetvy s ohľadom na názov úloh v prostredí Jira v tvare:

Nazov_tímu-číslo_ulohy_nazov_ulohy INFINITY-111_refaktoring



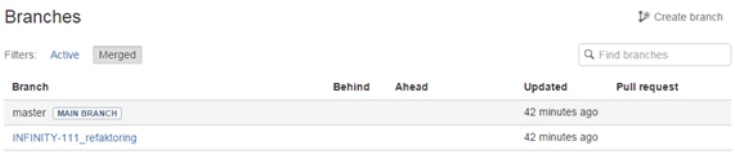
Obr. 30: Vytváranie vetvu A



Obr. 31: Vytváranie vetvu B

3.krok – Keď ste vyplnili obe polia, kliknite na tlačidlo Create, ktoré vytvorí novú vetvu.

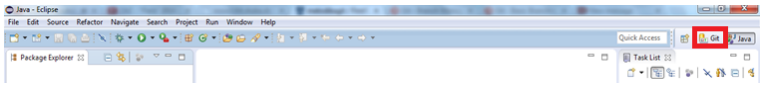
4.krok – Skontrolujte, či je vetva naozaj vytvorená. Po kliknutí na tlačidlo Branches, ktoré sa nachádza opäť v ľavej lište, ako ste vytvárali vetvu, sa vám zobrazí okno so všetkými vytvorenými vetvami v rámci projektu. Ako vidíte na obrázku, vetvy môžu byť zaradené do kategórie Merged alebo Active. Novo vytvorená vetva je automaticky zaradená do Merged (zlúčené) skupiny. Až keď upravíte túto vetvu v rámci vykonávania úlohy, je vetva zaradená medzi aktívne vetvy – Active. (Obr. 32)



Obr. 32: Zoznam vytvorených vetiev

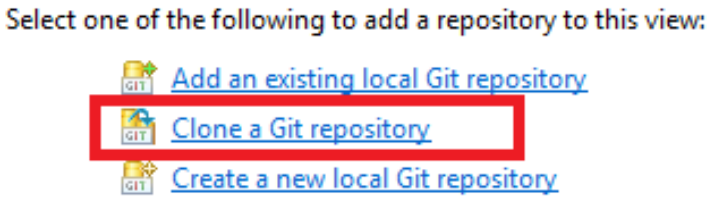
5.krok – Keďže ste práve vytvorili novú vetvu pre požadovanú úlohu, môžete začať s načítaním repozitára z prostredia Bitbucket do lokálneho repozitára na vašom zariadení. Pre tento účel spustite vašu nainštalovanú verziu vývojového prostredia Eclipse.

V prostredí Eclipse, kliknite na tlačidlo Git, umiestnené v pravom hornom rohu okna. (Obr. 33)



Obr. 33: Zvolenie nástroja Git v prostredí Eclipse

Zobrazí sa vám ponuka pre prácu s repozitárom. V ľavej časti okna sa nachádza záložka Git repositories, v ktorej uvidíte 3 možnosti, ktoré môžete vykonať s repozitárom. Vás momentálne zajíma 2.voľba a to Clone a Git repository. (Obr. 34) Touto voľbou získate namapovanie vášho repozitára z Bitbucket prostredia do vášho lokálneho repozitára. Kliknite na ňu.



Obr. 34: "Klonovanie"git repozitára do lokálneho repozitára

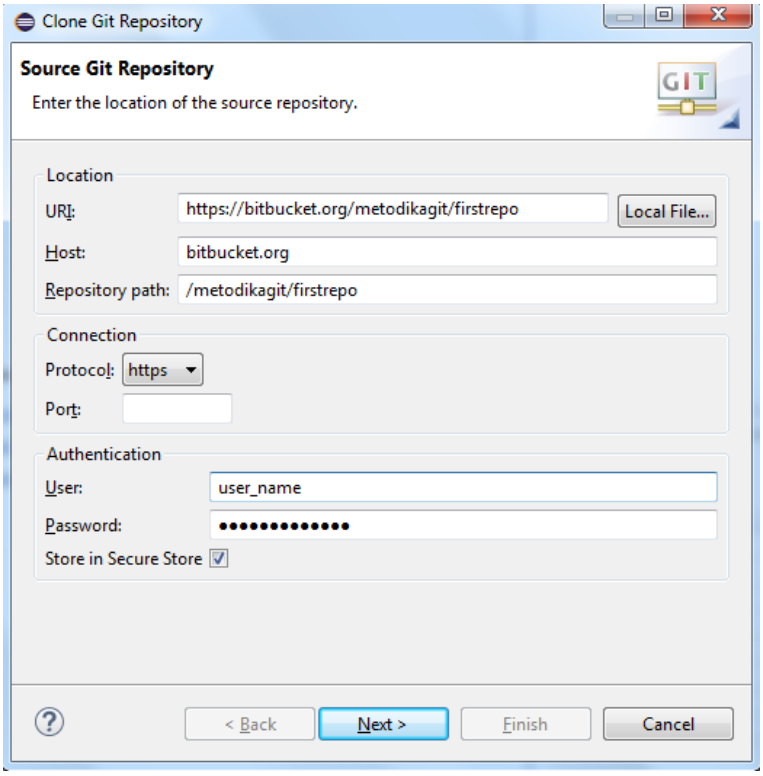
6.krok – Po kliknutí na tlačidlo Clone a git repository sa vám zobrazí okno, do ktorého musíte zadať požadované údaje pre vytvorenie lokálneho repozitára. (Obr. 35)

- URI – musíte zadať adresu repozitára, ktorý sa má skopírovať. Ak nie ste vo Vami vytvorenom repozitári, choďte najprv do neho a následne skopírujte webovú adresu z prehliadača do poľa URI

  Host – adresa sa automaticky doplní

  Repository path – sa taktiež vyplní na základe umiestnenia repozitára

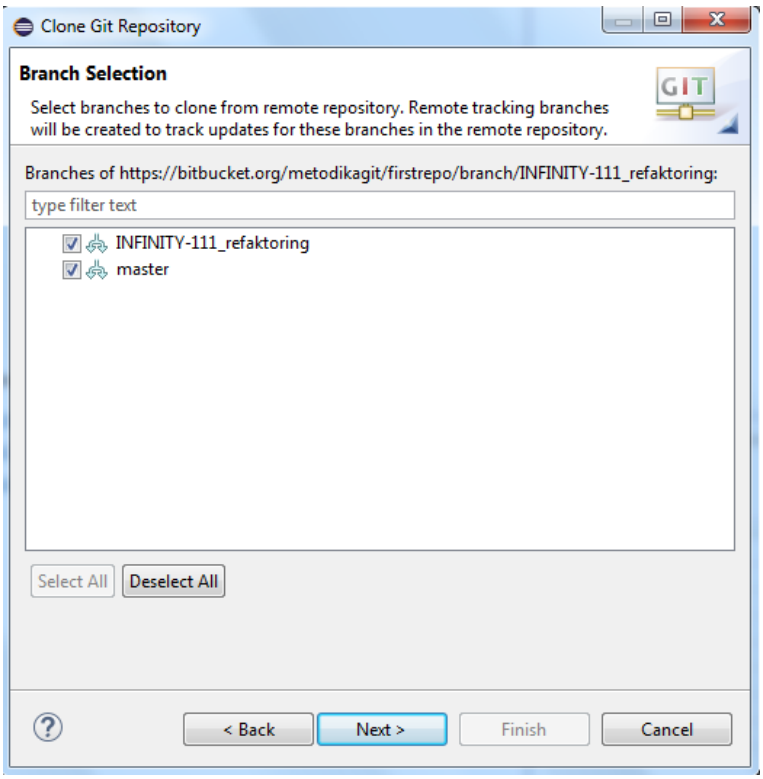
- Polia Protocol a Port neupravujte, nedopĺňajte.
- Polia User a Password slúžia na autentifikáciu používateľa. Zadajte svoje prihlasovacie údaje do prostredia Bitbucket.
- Kliknite na tlačidlo Next



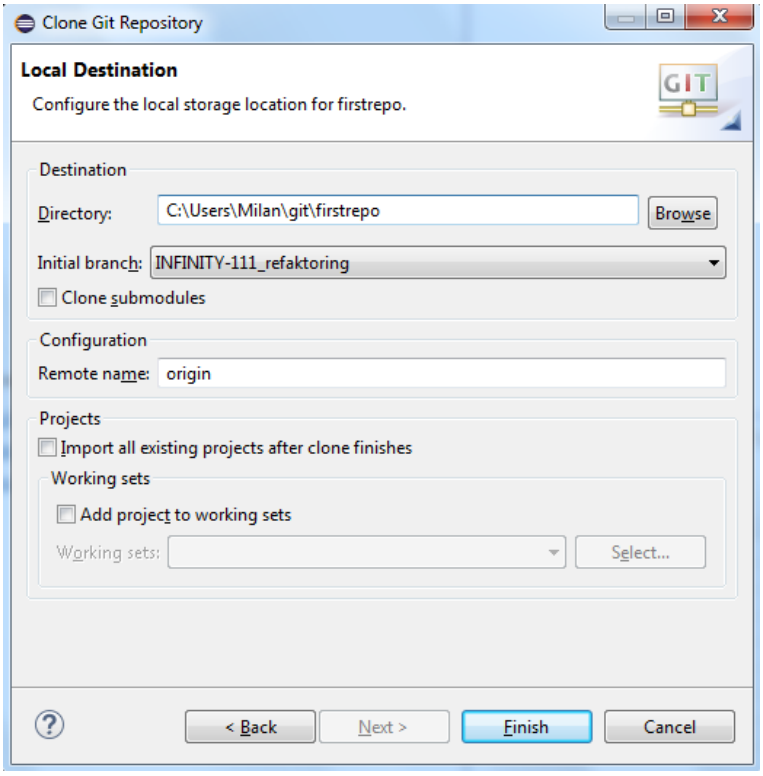
Obr. 35: "Klonovanie"git repozitára do lokálneho repozitára

7.krok - Zobrazí sa vám okno, v ktorom si máte zvoliť vetvu, z ktorej si chcete vytvoriť repozitár. Na začiatku sú označené všetky vetvy. Stlačte tlačidlo Deselect All, aby ste zrušili označenie vetiev. Teraz si zvoľte tú vetvu, ktorú aktuálne potrebujete. V tomto prípade zvolíme vetvu INFINITY-111_refaktoring. Kliknite na tlačidlo Next. (Obr. 36))

8.krok - V tomto poslednom kroku nahrávania repozitára máte na výber, či si zmeníte umiestnenie repozitára, alebo ponecháte adresu ako je v poli Directory. Ostatné polia ponechajte ako sú, netreba ich upravovať. Kliknite na tlačidlo Finish. (Obr. 37))

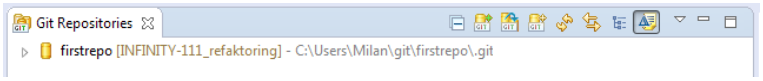


Obr. 36: Zvolenie vetvy a umiestnenia lokálneho repozitára A



Obr. 37: Zvolenie vetvy a umiestnenia lokálneho repozitára B

9.krok – Opäť v záložke Git repositories následne vidíte vytvorený lokálny repozitár s názvom vetvy, ktorá je na neho namapovaná a cestu k tomuto repozitáru. (Obr. 38)



Obr. 38: Vytvorený lokálny repozitár

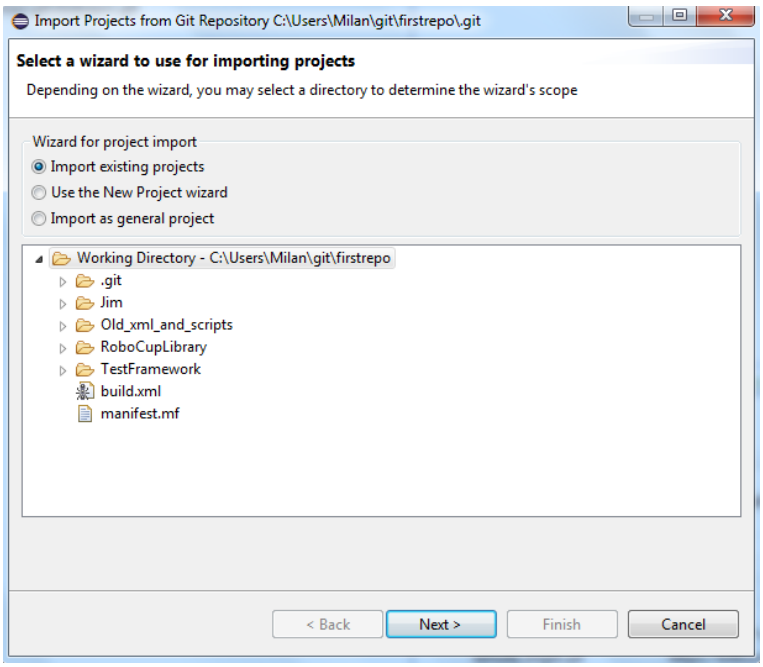
10.krok - Týmto ste vytvorili lokálny repozitár z požadovanej vetvy. Teraz už musíte iba importovať projekty z vytvoreného lokálneho repozitára.

11.krok – Kliknite pravým tlačidlom na váš lokálny repozitár v záložke Git repositories a zvoľte si možnosť importovania projektov – Import projects.

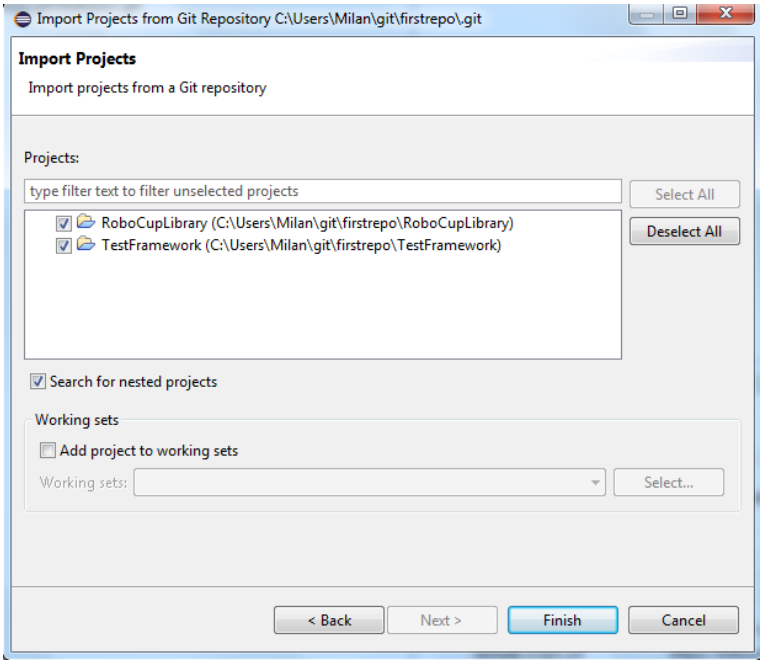
12.krok – V tomto kroku vidíte všetky súbory, ktoré sa nachádzajú vo vašom lokálnom repozitári. Vy chcete importovať už existujúce projekty z

tohto repozitára, takže ponechajte zaškrtnutú voľbu Import existing projects a kliknite na tlačidlo Next. (Obr. 39))

13.krok – Zobrazia sa vám všetky projekty, ktoré sa nachádzajú v rámci repozitára. Tu si zvoľte, ktoré projekty chcete importovať. Keď ste zaškrtli všetky projekty, ktoré chcete importovať, kliknite na tlačidlo Finish. (Obr. 40))



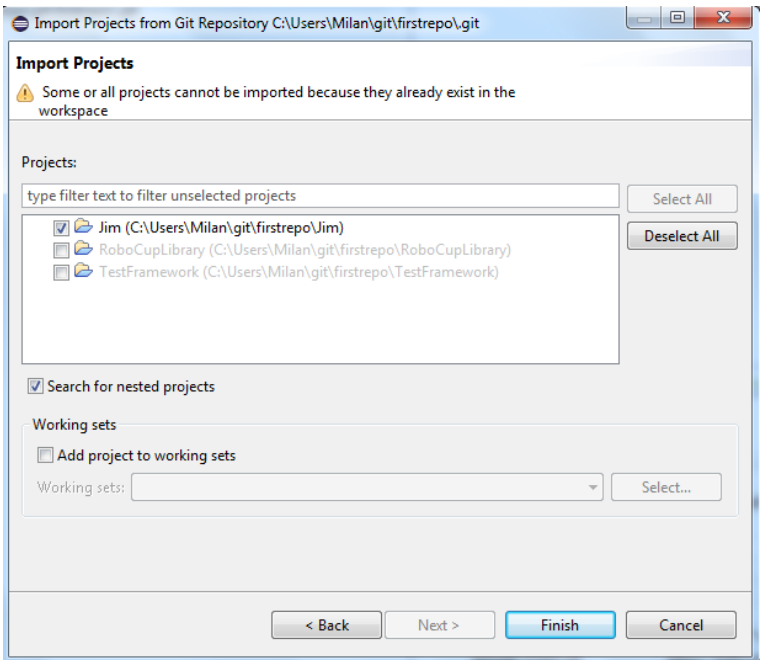
Obr. 39: Importovanie projektov z lokálneho repozitára A



Obr. 40: Importovanie projektov z lokálneho repozitára B

14.krok – Tu však nastane problém, pretože projekt Jim neobsahuje súbor .project a tak ho nemôžete vidieť medzi projektmi na importovanie. Choďte preto do priečinku, kde máte váš lokálny repozitár a skopírujte súbor .project buď z projektu RoboCupLibrary alebo TestFramework a vložte ho do priečinku projektu Jim. Súbor následne otvorte a upravte tag <name> v súbore do tvaru <name>Jim</name>. Project Jim týmto krokom pribudne medzi projekty na importovanie.

15.krok – Teraz pokračujte znovu krokom č.7. Tu môžete vidieť, že medzi projekty na importovanie naozaj pribudol aj projekt Jim. Označte ho a kliknite na tlačidlo Finish. (Obr. 41)



Obr. 41: Importovanie projektov z lokálneho repozitára C

Týmto pádom sú projekty úspešne naimportované a môžete s nimi začať pracovať.

11.7 Metodika značkovania kódu v CodeReview

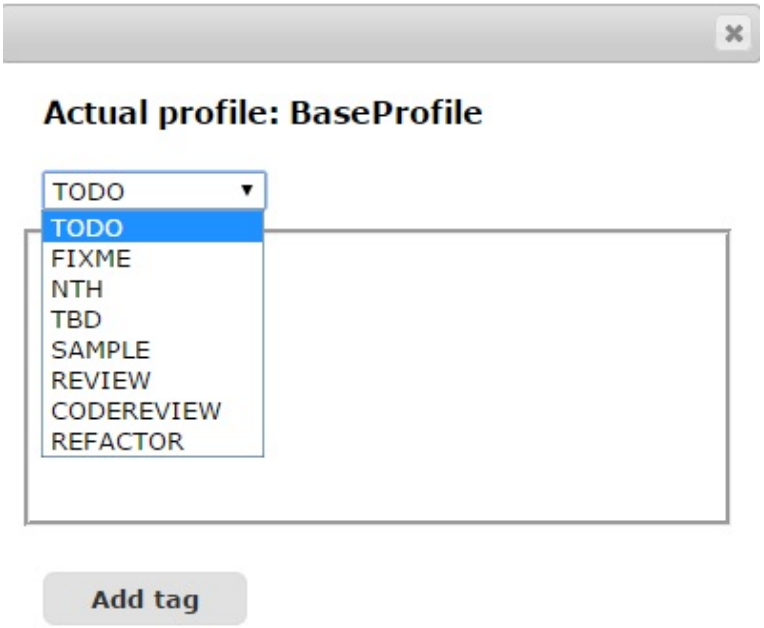
Posledná úprava	Miroslav Wolf
Platné od	19. november 2014
Poznámky	

Táto metodika je určená pre značkovanie kódu pomocou nástroja CodeReview. Je určená najmä pre manažéra kvality, ale aj pre každého člena tímu, ktorý bude pridávať značky do kódu. V metodike nie je opísané, ako postupovať pri prechádzaní kódu a ako kód udržiavať, ani ako sa prepája CodeReview s repozitárom. Slúži najmä pre rozhodnutie kedy a akú značku je vhodné pridať a ako ju opísať.

Predtým ako idete pridávať značky do kódu, uistite sa že máte nastavený branch v ktorom chcete značky pridávať. Následne prejdite v pravej

hornej časti obrazovky do menu na ÜPDATE REPOSITORYä stlačte "Pull repository", vďaka čomu budete mať najaktuálnejšiu verziu kódu. V ľavej časti vidíte štruktúru projektu, tu si vyberte triedu, do ktorej idete značky pridávať.

Typy značiek: (Obr. 42)

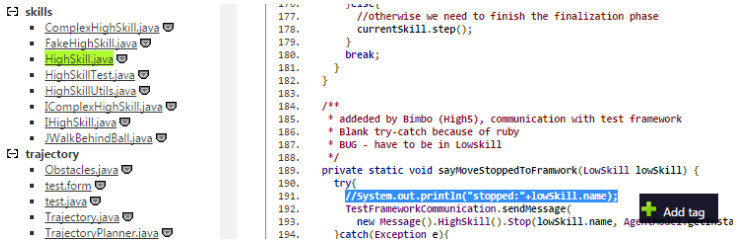


Obr. 42: Typy značiek

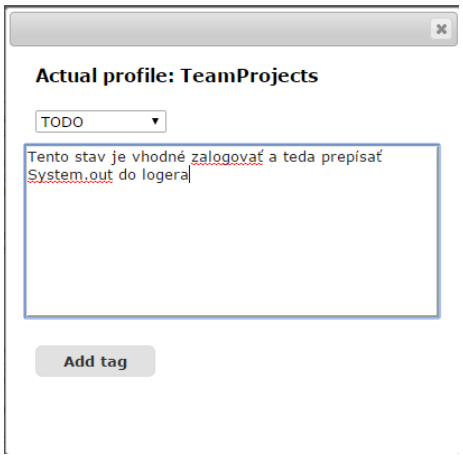
- TODO - túto značku pridajte, ak niečo musí byť v kóde spravené
- FIXME - túto značku pridajte, ak bola nájdená chyba v označenom kóde
- NTH - túto značku pridajte, ak má byť v označenom kóde niečo (re)implementované
- TBD - túto značku pridajte, ak je označený kód vo vývoji
- SAMPLE - túto značku pridajte, ak je označený kód iba príkladom
- REVIEW - túto značku pridajte, ak je nutné označený kód prejsť a skontrolovať

- CODEREVIEW - túto značku pridajte, ak je nutné označený kód refaktorovať
- REFACTOR - túto značku pridajte, ak označený kód nedodržuje konvenciu písania kódu

Každú značku vhodne opíšte. Opis musí obsahovať dôvod, prečo bola značka pridaná a z opisu musí byť jasné, čo je nutné v kóde vykonať, alebo zmeniť. Pokiaľ zmeny v označenom kóde môžu ovplyvniť funkčnosť aj inej časti projektu, uveďte do opisu, ktorá časť môže byť ovplyvnená a prečo. Príklad obr. 43 a obr. 44.



Obr. 43: Príklad vloženia značky A



Obr. 44: Príklad vloženia značky B

12 Manažment komunikácie

Posledná úprava	Michal Segeč
Platné od	16. november 2014
Poznámky	

Cieľom tejto kapitoly je poukázať na potreby riadenia komunikácie v rámci tímového projektu. Preto aby tím dokázala fungovať plnohodnotne, je dôležité venovať náležité množstvo času komunikácií. Tá môže byť priama a nepriama. Priama komunikácia sa uskutočňuje buď v rámci pravidelných formálnych stretnutí vo vyhradenom čase, alebo v rámci nepravidelných stretnutí, ak je potrebné prejednať určité náležitosti týkajúce sa zadaných úloh. Nepriama komunikácia zahŕňa použitie komunikačných prostriedkov, kde nedochádza k osobnému stretnutiu medzi členmi tímu. Pre úspešnosť projektu je nevyhnutné, aby všetci členovia tímu dodržiavali isté základy komunikácie a využívali potrebné nástroje na komunikáciu.

Použité pojmy:

- manažment - je proces plánovania, organizovania, vedenia a kontroly organizačných činností zameraných na dosiahnutie organizačných cieľov
- komunikácia – základ sociálnej interakcie, výmena myšlienok, názorov, tvrdení medzi 2 a viacerými živými bytosťami
- šprint – časovo vyhradené obdobie, ktoré cieľom je splnenie stanovených úloh
- task – úloha, ktorá sa vykonáva v rámci šprintu
- scrum – typ agilného vývoja softvéru, ktoré cieľom je efektívne riadenie komplexných projektov

12.1 Priama komunikácia

O priamej komunikácií hovoríme, ak účastníci istej konverzácie sú prítomní v tom istom čase na tom istom mieste. Je to najvhodnejší spôsob komunikácie, ak je potrebné riešiť určitú úlohu v dohodnutom čase. Účastníci konverzácie si tak navzájom môžu dať okamžitý feedback a môžu rýchlejšie dôjsť k riešeniu problému.

Posledná úprava	Michal Segeč
Platné od	16. november 2014
Poznámky	

12.1.1 Formálne stretnutia

Formálne stretnutia tímu sú stretnutia, ktoré sa konajú každý týždeň v určitý deň vo vyhradenom čase v Jobsovom softvérovom štúdiu. Stretnutie sa koná v prítomnosti všetkých členov tímu a vedúceho projektu (vlastníka produktu). Každé stretnutie vedie scrum master, určený pre daný šprint.

Každé stretnutie má svojho zapisovateľa, ktorý spisuje zo stretnutia zápisnicu. Zápisnica má svoju určitú štruktúru, ktorej forma sa dodržiava počas celého trvania projektu. Musí obsahovať údaje o prítomných členoch tímu, o pridelených úlohách pre daný šprint, zhodnotenie splnenia úloh a stručný obsah stretnutia, aké problémy sa riešili a na čo je vhodné sa zamerať neskôr. Každé stretnutie teda obsahuje diskusiu k projektu, kde každý člen tímu zhodnotí, ake úlohy vykonal. V prípade, že sa niektorý člen tímu nemôže zúčastniť stretnutia, je povinný oznámiť svoju neúčasť prostredníctvom niektorého z komunikačných kanálov, ktorých opis je uvedený v ďalšej časti.

12.1.2 Neformálne stretnutia

Neformálne stretnutia sú stretnutia, ktoré nie sú určené v presne vyhradenom čase a v úplnom zložení tímu. Uskutočňujú sa či už v rámci školských priestorov, alebo mimo v čase, ktorý vyhovuje členom tímu, ktorí sa potrebujú stretnúť za účelom riešenia úloh projektu. Výhodou stretnutí je odľahčená atmosféra a možnosť voľného priebehu komunikácie, bez predpísanej formy. Tento typ komunikácie je vhodný, ak viacerí členovia tímu riešia určitý vzniknutý problém a je najvýhodnejšie stretnúť sa osobne, aby sa mohlo čo najrýchlejšie dospieť k určitému riešeniu. Taktiež nie vždy sa dá vzhľadom na zložitosť a povahu problém opísať pomocou komunikačných prostriedkov, kde členovia nie sú osobne prítomní. Mnohokrát sa stane, že aj keď je úloha pridelená jednému členovi tímu, nemusí sa mu dariť ju úspešne vyriešiť a potrebuje pomoc od ďalšieho člena, ktorý má skúsenosti s daným riešením. V tomto prípade je dôležitá komunikácia, kde člen tímu, ktorý potrebuje pomoc skontaktuje druhého a je tak väčšia pravdepodobnosť, že úlohu, ktorá by pravdepodobne skončila ako nesplnená sa podarí načas splniť a tak úspešne pokročiť pri riešení spoločného projektu.

12.2 Nepriama komunikácia

Posledná úprava	Michal Segeč
Platné od	16. november 2014
Poznámky	

Je typ komunikácie, pri ktorej členovia tímu nemusia byť osobne prítomní na rovnakom mieste v určitom čase. Tento spôsob komunikácie je vhodný, ak členovia tímu nemajú dostatok času na osobné stretnutia z rozličných dôvodov a je tak možnosť riešiť vzniknuté problémy prostredníctvom rôznych komunikačných prostriedkov. Nevýhodou niektorých prostriedkov je, že odozva jedného člena tímu na problém druhého člena môže byť okamžitá, ale môže trvať aj neurčité množstvo času.

12.2.1 E-mail

Tento prostriedok slúži ako pre jednotlivých členov tím, tak aj pre celý tím. Ako tím využívame spoločný mail, ku ktorému majú všetci členovia prístup. Tento mail sa využíva, ak nás chce skontaktovať napr. vedúci projektu a nemusí sa tak obťažovať písať každému členovi zvlášť, ale každý si môže prečítať danú správu. Aby sa predchádzalo neprečítaným správam, každý deň niektorý z členov tímu skontroluje mail.

E-mail avšak môžu využívať aj členovia tímu za účelom riešenia problémov, keď nie je možné osobné stretnutie. Výhodou je, že člen tímu nemusí byť dostupný, ale môže si správu prečítať neskôr. Takto je možné aj dlhšie uchovať konverzáciu pre prípad jej potrebného znovupoužitia.

12.2.2 Skype

Ďalším používaným prostriedkom pre komunikáciu je skype, ktorý umožňuje okrem klasického chatu aj videohovor, prípadne iba hovor bez videa. Skype sa najčastejšie využíva pri kolaborácií viacerých členov tímu pri riešení problému. Výhodou je, že člen tímu nemusí byť osobne prítomný a môže komunikovať aj z domu v prípade, že je chorý, alebo mu nie je možné z nejakého dôvodu umožnené dostaviť sa na stretnutie. Taktiež výhodou je možnosť okamžitej odpovede, čiže tímu nemusí čakať na odpoveď od druhého ako je to v prípade napr. e-mailu. Konverzácia je taktiež uchovávaná a je možné si ju spätne prezrieť.

12.2.3 Telefón

Keďže každý z nás vlastní telefón, je to najrýchlejší možný spôsob komunikácie. Telefón využívame výhradne ak treba súrne riešiť nejaký problém. V opačnom prípade využívame ostatné komunikačné prostriedky.

12.2.4 Dropbox

Tento prostriedok slúži ako úložisko a využívame ho uchovávanie všetkých dokumentov, ktoré v rámci tímového projektu vytvoríme, t.j. zápisnice, dokumentáciu, aj čiastkové výstupy jednotlivých úloh. Vždy keď sa zverejní dokument na dropboxe, oznámi člen tímu prostredníctvom Facebooku, že pridal daný dokument. Tieto dokumenty sú v rámci tímu zdieľané a môže si ich sprístupniť tým pádom každý člen tímu.

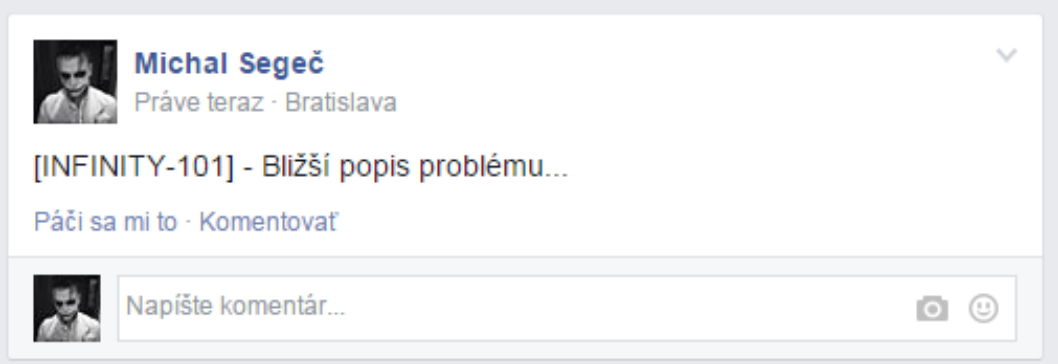
12.2.5 Facebook

Facebook, ďalej len FB, tvorí náš hlavný a najčastejšie používaný prostriedok nepriamej komunikácie. V rámci tímového projektu máme vytvorenú skupinu – Tímový projekt, kde prejednávame všetky potrebné záležitosti, týkajúce sa riešenia úloh. Na komunikáciu teda využívame všemožné funkcie, ktoré FB ponúka. Jednou z nich je pridanie príspevku na nástenku. Aby nedochádzalo k zmätočným situáciám, každý príspevok musí byť označený názvom tímu a za ním musí nasledovať číslo úlohy, ktoré je pre úlohu pridelené v JIRE a popis problému, ktorý je treba riešiť. Je vhodné, aby každý člen tímu klikol na tlačidlo páči sa mi to a tým pádom potvrdil, že si príspevok prečítal. V prípade, že ide o oznam, že bola pridaná zápisnica, alebo nejaký iný dokument, nemusí byť príspevok špeciálne označený 45.

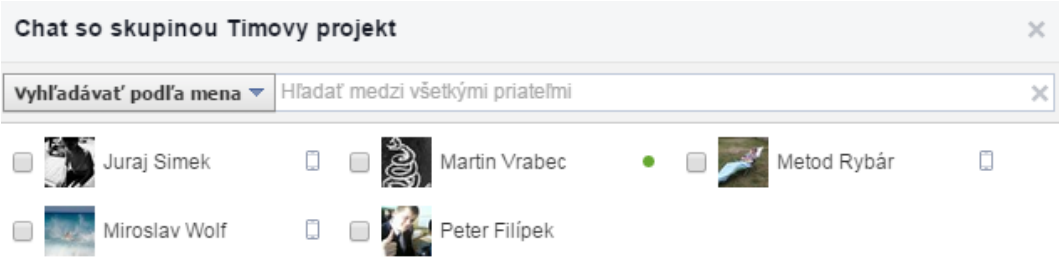
V prípade, že pôjde o riešenie komplexnejšieho problému je vhodné problém riešiť v rámci konverzácie. Môžeme si tak zvoliť jednotlivých členov tímu, s ktorými treba riešiť daný problém, alebo aj celý tím. Konverzácia na FB ostáva uchovaná a dá sa k nej hocikedy vrátiť. Uprednostnením chatu pred písaním príspevkov priamo na nástenku umožní, aby skupina ostala prehľadná a nástenka nebola zahltená veľa príspevkami 46.

12.2.6 Redmine

Redmine je fórum, ktoré slúži na riešenie problémov ohľadom nástrojov, ktoré využívame v rámci tímového projektu. Bolo potrebné riešiť práva a vytvore-



Obr. 45: Príklad statusu vo facebookovej skupine



Obr. 46: Vytváranie skupinového chatu

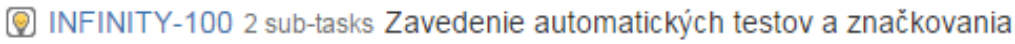
nie projektu v Jire, ale taktiež vybavenie administrátorských práv v rámci využitia Bamboo, pre zavedenie automatických testov zdrojového kódu projektu. Každý, kto potrebuje riešiť nejaký problém vytvorí nový Issue a pridelí ho zodpovednej osobe, ktorá má pomôcť s riešením problému. Výsledkom je rozriešenie problému, prípadne rada a uzavretie Issue.

12.2.7 Google Drive

Google drive je ďalším dôležitým prostriedkom, ktorý využívame pre kolaboráciu. V prípade, že je potrebné, aby viacerí členovia tímu pracovali na spoločnom dokumente v rovnaký čas, alebo aby ho mohli priebežne dopĺňať je umožnené zdieľanie tohto dokumentu medzi členov tímu. Je pritom možné zabezpečiť, aby ho mohli vidieť a upravovať len oprávnení členovia.

12.2.8 Jira

Tento nástroj slúži ako scrum nástroj, ktorý uchováva údaje o všetkých úlohách, ktoré riešime v rámci tímového projektu. Všetky úlohy si prideľujeme na formálnych stretnutiach a na konci si z backlogu vyberieme tasky, ktoré budeme riešiť v danom šprinte. Každá úloha je pridelená svojmu riešiteľovi a zodpovednej osobe za splnenie úlohy. Každá úloha je označená názvom projektu a číslom, ktoré systém pridelí úlohe. Tieto označenia využívame ako sme už spomínali aj pri pridávaní príspevku na nástenku FB. Týmto dosiahneme konzistenciu dát a vieme si príspevok na FB spojiť s úlohou v Jire 47.



Obr. 47: Ukážka úlohy v Jire

12.3 Zhrnutie manažmentu komunikácie

Posledná úprava	Michal Segeč
Platné od	16. november 2014
Poznámky	

Komunikácia zohráva v rámci riešenia tímového projektu kľúčovú úlohu. Je potrebné, aby všetci členovia tímu boli oboznámený s jej priebehom a správnym používaním, aby sa dosiahlo efektívne riadenie v rámci projektu, ale taktiež aby sa úspešne splnili všetky zadané úlohy. Vzhľadom na to, že používame množstvo komunikačných prostriedkov, je nutné sa vedieť rozhodnúť, ktorý prostriedok je vhodný na riešenie ktorej úlohy. S ohľadom na doterajší priebeh projektu môžeme zhodnotiť, že dané komunikačné metódy sú vhodne zvolené a používané a je možné ich použiť na riešenie takéhoto typu projektu.

13 Manažment kvality

Posledná úprava	Juraj Šimek
Platné od	2. máj 2015
Poznámky	

Táto kapitola popisuje všetky procesy týkajúce sa manažmentu kvality v našom tíme. Manažment kvality siaha do všetkých aspektov riešenia a riadenia projektu. Keďže zdrojový kód robotického hráča, ktorý sme dostali len ťažko možno nazvať kvalitným, je potrebné zdvihnúť kvalitu tohto kódu a zabezpečiť tak jeho lepšiu čitateľnosť a pomôcť jeho udržateľnosti. V tejto kapitole preto opisujeme, aké kroky sme podnikli na poli zvýšenia a udržania kvality v projekte hráča robotického futbalu.

13.1 Proces schvaľovania a hodnotenia vykonanej práce

Na začiatku každého šprintu sme si vhodne zadefinovali *používateľské príbehy* (story) a tieto sme rozdelili do úloh tak, že každá úloha je pridelená jednému členovi tímu. Používateľské príbehy a úlohy sme následne vložili do systému *Jira* tak, ako to opisuje Metodika pre vytváranie úloh v nástroji Jira **??**. Člen tímu, ktorému je daná úloha priradená následne úlohu prevedie do stavu *In Progress* a začne ju riešiť. Keď je úloha vyriešená z pohľadu riešiteľa, to znamená, že riešiteľ je s finálnym stavom úlohy spokojný a náležite ju otestoval, prevedie ju do stavu *Resolved*. Vtedy požiada iného člena tímu, aby vykonal nad touto úlohou revíziu. Tento člen tímu – *overovateľ*, skontroluje prácu riešiteľa, otestuje ju a vyjadrí svoj postreh ku stavu úlohy. V prípade, že úlohu akceptuje, prevedie ju do stavu *Closed*, v prípade, že nie, uvedie v Jire *komentár*, čo treba opraviť, znovu ju otvorí a upovedomí riešiteľa o nutnosti zmien. Po tom, ako riešiteľ vykoná zmeny prebehne celý schvaľovací proces odznova až kým nie je úloha zatvorená. Kontrola riešiteľa prebieha i počas riešenia, kedy sa dohliada na to, aby všetky unit testy fungovali tak, ako majú.

Keďže počas zimného semestra sme implementovali iba značne obmedzené množstvo novej funkcionality, tento proces sme dôsledne nedodržiavali. Dodržiavali sme ho však pri kritických úlohách, ako implementácia nového Loggera, ZMP, nastavení, a pri premazávaní kritických častí zdrojového kódu kde mohlo dôjsť ku chybám.

V procese schvaľovania a hodnotenia vykonanej práce sa teda stretávame s nasledovnými úlohami:

- *Riešiteľ*: Človek, ktorému je pridelená úloha a je zodpovedný za jej vyriešenie. Ak riešiteľ mení zdrojový kód (teda vystupuje aj v úlohe programátora) je potrebné, aby tento kód prešiel všetkými definovanými unit testami. Ak riešiteľ vytvára nový zdrojový kód, je potrebné, aby pre kritické úseky tohto kódu napísal test. V prípade úpravy zdrojového kódu riešiteľ považuje úlohu za hotovú, ak naprogramoval všetky požadované zmeny, zakomponoval ich do projektu vo svojom počítači, napísal vhodné testy a úspešne ich vykonal a po integrácii zmeny do celkového projektu zbehli aj všetky ostatné testy. Navyše musia zbehnúť všetky automatizované testy v systéme Bamboo po nahratí zmien na Bitbucket účet. Vykonané zmeny musia pritom presne kopírovať požiadavky, ktoré boli na ne kladené a riešiteľ sa riadi všetkými metodikami, ktoré súvisia s udržiavaním kvality kódu. V prípade, keď sa nič neprogramuje (analýzy, dokumentovanie, ...) je výstupom úlohy dokument popisujúci vykonané činnosti. Takáto úloha je riešiteľom považovaná za hotovú, keď dokument opisuje všetky náležitosti, ktoré boli úlohou požadované a v takej podrobnosti, ako to táto úloha žiadala.

- *Overovateľ*: Človek, ktorého úlohou je skontrolovať prácu riešiteľa úlohy. Táto osoba je poverená riešiteľom na to, aby skontrolovala jeho výstup. V prípade zmeny zdrojového kódu, overovateľ vykoná prehliadku kódu, zistí, či boli dodržiavané konvencie písania zdrojového kódu, overí a spustí testy. Ak testy prešli úspešne a overovateľ zmenu akceptuje, overovanú úlohu uzavrie (prevedie ju do stavu *Closed*) a zvyčajne ju bližšie nekomentuje. Ak našiel nejaké nedostatky, v Jire uvedie komentár so zistením nedostatkov, znova otvorí úlohu (uvedie ju do stavu *In Progress*) a dodatočne (telefonicky, mailom, pomocou sociálnej siete) upovedomí riešiteľa o nedostatku jeho riešenia. Pri komunikácii sa riadi postupmi opísanými v Manažmente komunikácie 12.

- *Product owner*: Schvaľuje finálny stav úlohy na konci šprintu. Ak ju akceptuje úloha je skutočne hotová, ak nie, pre nový šprint sa vytvorí story, ktorá odstráni nedostatky.

13.2 Metodika písania zdrojového kódu

V súvislosti s manažmentom kvality bola vypracovaná *metodika písania zdrojového kódu* 15. Keďže náš tím prevzal existujúci projekt, ktorý riešilo pred nami zhruba 10 tímov a rôzni iní jednotlivci, zdrojovému kódu v projekte chýbal jednotný štýl. Za účelom zjednotenia štýlu písania kódu sme vytvorili starostlivo vypracovanú metodiku písania zdrojového kódu, ktorá podrobne popisuje rôzne situácie, ktoré môžu nastať v procese písania zdrojového kódu a opisuje to, ako ich jednotne riešiť. Metodika opisuje formátovanie zdrojového kódu, to, ako treba nazývať premenné, konštanty, metódy, triedy, rozhrania a balíky. Ďalej táto metodika opisuje štýl komentovania zdrojového kódu a štýl písania dokumentačných komentárov. Značná časť metodiky sa venuje logovaniu, ktoré bolo v projekte značne nejednotné. Môžeme povedať, že prechodom na nový jednotný logger sa výrazne zvýšila kvalita zdrojového kódu, nakoľko sa k logovaniu namiesto troch rôznych prístupov používa jeden prístup.

Úlohou a zodpovednosťou každého člena tímu je dodržiavať štýl písania zdrojového kódu. Tím si predsavzal upraviť aj už existujúci kód tak, aby spĺňal metodikou predpísaný štýl. Nakoniec bude metodika sprístupnená pomocou *Wikipedie* projektu aj ostatným vývojárom, ktorí sa v budúcnosti budú podieľať na vývoji agenta hrajúceho robotický futbal, pričom bude zdôraznená nutnosť túto metodiku dodržiavať.

13.3 Testovanie zmien v zdrojovom kóde

Keďže sme doposiaľ nevyvíjali rozsiahlejšie nové triedy, ani sme nepridávali novú funkcionalitu (okrem loggera, nastavení a ZMP), nebolo potrebné vytvárať množstvo nových testov. Po každej zmene sme sa spoliehali na už vytvorené testy. Keďže sme hlavne refaktorovali pôvodný kód, spúšťali sme už pripravené testy. Ak testy dopadli dobre, predpokladali sme, že sme počas úprav zdrojového kódu nič nepoškodili a zmeny sme akceptovali. Hráča sme priebežne testovali i v simulačnom prostredí. Testy sme vytvárali pomocou Metodiky tvorby testov v projekte JIM 14.

Počas 3. šprintu sme zaviedli systém automatického testovania *Bamboo* a spárovali sme ho s účtom na Bitbuckete. Systém Bamboo využívame na automatické testovanie. Systém Bamboo bol nakonfigurovaný tak, aby každý deň spúšťal testy nachádzajúce sa v súbore TestAll.java. Konfiguráciou bol poverený Bc. Michal Segeč a je bližšie opísaná v retroepsktívach šprintov.

13.4 Značkovanie zdrojového kódu

V priebehu 3. šprintu sme zaviedli systém PerConIk, ktorý slúži na značkovanie zdrojového kódu. K značkovaniu bola vytvorená metodika Metodika značkovania kódu v CodeReview 11.7, ktorá opsuje použitie jednotlivých začiek. Systém je prepojený s účtom na Bitbuckete, preto si vie aktualizovať zdrojové kódy. V zimnom semestri sme však značkovanie nepoužívali, nakoľko aj na miestach, kde by to bolo možné (konkrétne išlo o opis varovaní z analýzy nástroja Unnecessary Code Detector v úlohe refaktorovania nepoužívaných častí kódu), to nebolo vhodné, lebo by sme strácali zbytočne čas, keďže označkovaný kód by sme aj tak z veľkej časti vymazali.

13.5 Kvalita dokumentácie

Počas celého semestra sme sa pri vypracovaní dokumentácie riadili metdikami Manažmentu dokumentovania 9. Metodika písania dokumentácie je napíasná jednoznačne a výslednú dokumentáciu vždy kontroloval Bc. Metod Rybár. Aj vďaka použitiu systému LaTeX sú zmeny v dokumentácii rýchle a výstup kvalitný aj z estetického hľadiska. Obsahová kvalita dokumentácie je zaistená pokrytím všetkých častí riešenia a riadenia projektu a ich schválením členmi tímu.

13.6 Metriky zdrojového kódu

Počas 3. šprintu sme do vývojového prostredia Eclipse inštalovali 2 pluginy. Jedným z nich je plugin *Unnecessary Code Detector*, ktorý odhalí triedy, metódy a polia, ktoré nie sú používané, alebo sú volané len v testoch. Keďže v zdrojovom kóde agenta hrajúceho robotický futbal je prítomných veľa duplicitných i nepotrebných metód a tried, v treťom šprinte sme sa venovali analýze výstupu tohto pluginu. Vymazanie duplicitných a nepoužívaných metód, či tried výrazne zvýši čitateľnosť zdrojového kódu agenta.

Ďalším inštalovaným pluginom bol plugin *Code Metrics for Eclipse*, ktorý počíta rôzne metriky zdrojového kódu agenta a pomáha nám v identifikácii problémových častí. Plugin počíta napríklad cyklomatickú zložitosť, počet parametrov metód, hĺbku vhniezdených blokov, počet odvodených tried, počet prekrytých metód, počet atribútov, počet metód, počet tried a rozhraní, počet balíkov a celkovú dĺžku zdrojového kódu v projekte, ako aj počet riadkov kódu v metódach. Pomocou tohto pluginu môžeme reálne opísať aj to,

či v priebehu refaktorovania projektu došlo ku zmene v kvalite kódu a to, ku akej zmene došlo. Tento plugin hrá dôležitú úlohu pri opise toho, ako sa zlepšovala kvalita medzi jednotlivými šprintami.

13.7 Kvalita zdrojového kódu v priebehu riešenia projektu v zimnom semestri

Táto časť dokumentuje to, ako sa menila kvalita celkového zdrojového kódu v priebehu riešenia projektu.

Obrázok 48 znázorňuje výstup pluginu Code Metrics projektu Jim v čase pred vykonaním akejkoľvek zmeny našim tímom. Ako možno vidieť, balík annotation.data trpí značnou zložitosťou, v triede XMLParser je vysoká cyklomatická zložitosť a hlboké vnorenia, trieda XMLCreator obsahuje dlhé metódy. Počet riadkov zdrojového kódu je 16900, z čoho 9948 riadkov kódu je v metódach. Počet balíkov je 46, počet tried 311 a počet metód 1168.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,584	4,802	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,556	0,822	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,439	0,952	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		12,5	24,546	146	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		5,196	7,036	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,498	0,337	1	/Jim/src/sk/fiit/jim/tests	
Abstractness (avg/max per packageFragment)		0,078	0,174	1	/Jim/src/sk/fiit/jim/garbage/code_review	
Normalized Distance (avg/max per packageFra		0,444	0,318	1	/Jim/src/sk/fiit/jim/enums	
Depth of Inheritance Tree (avg/max per type)		1,653	0,883	6	/Jim/src/sk/fiit/jim/agent/trajectory/test.java	
Weighted methods per Class (avg/max per typ	3444	11,074	17,92	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	144	0,463	3,8	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	51	0,164	0,483	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,207	0,388	1,833	/Jim/src/sk/fiit/jim/agent/moves/Joint.java	
Number of Attributes (avg/max per type)	773	2,486	4,353	27	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	378	1,215	2,11	15	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1168	3,756	6,114	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	170	0,547	1,611	13	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,133	0,529	3	/Jim/src/sk/fiit/jim/garbage/plan/PlanTournamentKi...	
Number of Classes (avg/max per packageFragr	311	6,761	7,645	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	8	0,174	0,433	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	46					
Total Lines of Code	16900					
Method Lines of Code (avg/max per method)	9948	7,463	20,676	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 48: Výstup z Code Metrics pred riešením projektu.

Obrázok 49 znázorňuje výstup pluginu Code Metrics projektu Jim po druhom šprinte. Do druhého šprintu sme sa venovali prerábaniu logovania, mazaniu nepotrebných balíkov a nepotrebných komentárov v kóde. Zlá cyklomatická zložitosť preto ostala nevyriešená. Boli sme však schopní zmenami vykonanými v druhom šprinte znížiť počet balíkov o tri balíky, počet riadkov zdrojového kódu o vyše 1000 riadkov (čo je 6,15% pôvodného počtu riadkov),

vymazať 28 nepotrebných tried a to pri zachovaní pôvodnej funkcionality projektu. Výrazne sa tým zdvihla kvalita zdrojového kódu.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,585	4,759	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,569	0,819	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,462	0,961	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		12,93	23,848	134	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		5,093	7,249	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,483	0,346	1	/Jim/src/sk/fiit/jim/tests/decision	
Abstractness (avg/max per packageFragment)		0,082	0,181	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,465	0,328	1	/Jim/src/sk/fiit/jim/enums	
Depth of Inheritance Tree (avg/max per type)		1,629	0,832	6	/Jim/src/sk/fiit/jim/agent/trajectory/test.java	
Weighted methods per Class (avg/max per typ	3164	11,18	18,512	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	134	0,473	3,949	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	41	0,145	0,487	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,211	0,388	1,833	/Jim/src/sk/fiit/jim/agent/moves/Joint.java	
Number of Attributes (avg/max per type)	726	2,565	4,5	27	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	370	1,307	2,176	15	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1085	3,834	6,343	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	144	0,509	1,516	13	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,049	0,196	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	283	6,581	7,83	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	7	0,163	0,428	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	43					
Total Lines of Code	15860					
Method Lines of Code (avg/max per method)	9424	7,699	21,399	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 49: Výstup z Code Metrics po 2. šprinte.

Obrázok 50 znázorňuje výstup pluginu Code Metrics projektu Jim po treťom šprinte. V treťom šprinte dochádzalo k refaktorovaniu zdrojového kódu do konvencií písania zdrojového kódu 15, čo spolu s čiastočnou implementáciou ZMP viedlo k miernemu nárastu počtu riadkov v projekte v porovnaní s druhým šprintom. Aj napriek tomu, že mnoho riadkov v procese úprav kódu do konvencie bolo vymazaných, pribudlo aj niekoľko nových, obzvlášť vďaka dokumentačným komentárom, ktoré sme pridali všade, kde sme boli schopní tak, ako to definuje konvencia písania zdrojového kódu.

Obrázok 51 znázorňuje výstup pluginu Code Metrics projektu Jim po štvrtom šprinte. Došlo k výraznému poklesu počtu riadkov zdrojového kódu, kvôli tomu, že boli zmazané mnohé jeho časti, ktoré boli nepotrebné a nikde sa nepoužívali. Na identifikovanie týchto častí sme použili Unnecessary Code Detector. Došlo najmä k odstráneniu mnohých nepotrebných highskillov, preto si možno všimnúť aj výrazný pokles tried. Vo výstupe však nie je zohľadnený zdrojový kód ZMP.

Obrázok 52 znázorňuje výstup pluginu Code Metrics projektu Jim po poslednom, piatom šprinte. Kvôli reimplementácii triedy Settings.java došlo k poklesu počtu riadkov o 500, nakoľko sa výrazne zjednodušila i napriek pridanej funkcionalite, pomocou ktorej možno nastavenia čítať zo súboru.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,582	4,756	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,568	0,818	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,515	0,991	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		12,533	23,297	133	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,867	7,145	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,446	0,293	1	/Jim/src/sk/fiit/jim/decision/tactic/oldgoalie	
Abstractness (avg/max per packageFragment)		0,08	0,18	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,508	0,278	1	/Jim/src/sk/fiit/jim/enums	
Depth of Inheritance Tree (avg/max per type)		1,629	0,832	6	/Jim/src/sk/fiit/jim/agent/trajectory/test.java	
Weighted methods per Class (avg/max per typ	3166	11,187	18,513	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	135	0,477	3,959	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	41	0,145	0,487	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,211	0,387	1,833	/Jim/src/sk/fiit/jim/agent/moves/Joint.java	
Number of Attributes (avg/max per type)	726	2,565	4,5	27	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	371	1,311	2,175	15	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1086	3,837	6,35	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	145	0,512	1,516	13	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,049	0,196	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	283	6,289	7,748	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	7	0,156	0,419	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	45					
Total Lines of Code	15938					
Method Lines of Code (avg/max per method)	9515	7,761	21,053	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 50: Výstup z Code Metrics po 3. šprinte.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,409	4,418	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,567	0,819	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,508	0,992	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		10,388	19,793	115	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,102	6,175	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,449	0,267	1	/Jim/test/sk/fiit/jim/decision/tactic/attack	
Abstractness (avg/max per packageFragment)		0,076	0,174	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,503	0,251	1	/Jim/src/sk/fiit/jim/log	
Depth of Inheritance Tree (avg/max per type)		1,609	0,844	6	/Jim/src/sk/fiit/jim/agent/trajectory/test.java	
Weighted methods per Class (avg/max per typ	2778	10,444	17,916	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	135	0,508	4,082	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	43	0,162	0,513	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,186	0,371	1,667	/Jim/src/sk/fiit/jim/agent/highskill/kick/KickHighSkil...	
Number of Attributes (avg/max per type)	611	2,297	4,416	27	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	344	1,293	1,905	11	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1007	3,786	7,115	73	/Jim/src/sk/fiit/jim/agent/models/AgentModel.java	
Number of Static Methods (avg/max per type)	148	0,556	1,572	13	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,054	0,204	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	266	5,429	6,934	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	7	0,143	0,404	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	49					
Total Lines of Code	14579					
Method Lines of Code (avg/max per method)	8675	7,524	20,775	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 51: Výstup z Code Metrics po 4. šprinte.

13.8 Kvalita zdrojového kódu v priebehu riešenia projektu v letnom semestri

Nakoľko sme sa v letnom semestri venovali všetkým trom projektom venujúcim sa robotickému futbalu, menovite projektom *Jim, TestFramework a RoboCupLibrary*, opíšeme zmeny v nich z hľadiska kvality v samostatných

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,438	4,499	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,576	0,821	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,522	1,004	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		10,347	19,771	115	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,061	6,182	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,448	0,269	1	/Jim/test/sk/fiit/jim/decision/tactic/attack	
Abstractness (avg/max per packageFragment)		0,076	0,174	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,505	0,254	1	/Jim/src/sk/fiit/jim/log	
Depth of Inheritance Tree (avg/max per type)		1,615	0,847	6	/Jim/src/sk/fiit/jim/agent/trajectory/test.java	
Weighted methods per Class (avg/max per typ	2679	10,225	17,17	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	135	0,515	4,113	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	41	0,156	0,504	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,19	0,381	1,833	/Jim/src/sk/fiit/jim/agent/moves/Joint.java	
Number of Attributes (avg/max per type)	576	2,198	4,043	26	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	335	1,279	1,877	11	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	961	3,668	6,169	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	140	0,534	1,51	12	/Jim/src/sk/fiit/jim/agent/parsing/Perceptors.java	
Specialization Index (avg/max per type)		0,053	0,203	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	262	5,347	6,877	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	7	0,143	0,404	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	49					
Total Lines of Code	14032					
Method Lines of Code (avg/max per method)	8325	7,575	21,111	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 52: Výstup z Code Metrics po 5. šprinte.

podkapitolách. Celkovo môžeme konštatovať, že v priebehu letného semestra sa nám podarilo udržať kvalitu projektu, dokonca sme ju v mnohých prípadoch zdvihli, najmä čo sa týka kvality zdrojového kódu. Kód sme vhodne okomentovali, opravili sme mnohé nedostatky a do projektu sme doplnili mnohé vylepšenia, ktoré určite pomôžu tímom, ktoré budú na projekte pracovať po nás. Počas riešenia tímového projektu sme sa aj v letnom semestri riadili metodikami opísanými v predchádzajúcich častiach tejto kapitoly.

13.8.1 Kvalita zdrojového kódu projektu Jim

Obrázok 53 zachytáva stav zdrojového kódu projektu Jim po prvom semestri. Kvalita kódu je v porovnaní so stavom, v akom sme projekt dostali značne na vyššej úrovni. To má za následok refaktorovanie projektu v zimnom semestri. Obrázok 54 zachytáva stav projektu po šiestom šprinte, počas ktorého došlo k opravám a odstráneniu niektorých nefunkčných testov a prerobeniu triedy Vector3 na Vector3D. To malo za následok aj menší dodatočný refaktoring kódu, čo viedlo k zníženiu počtu riadkov zdrojového kódu a k zníženiu počtu tried. Pokračujúca implementácia a vylepšovanie ZMP mali tiež na tento jav vplyv. V siedmom šprinte bola reimplementovaná detekcia pádu hráča za kvalitnejšiu metódu detekcie, ktorá lepšie deteguje, kedy hráč spadol. Táto zmena však mala na celkovú kvalitu zdrojového kódu len minimálny vplyv. V ôsmom šprinte bolo skvalitnené používanie loggera pomocou

implementácie jeho GUI a boli doladené funkcie pre detekciu pádu agenta. V deviatom šprinte došlo k vylepšeniu GUI pre logger, ktoré si teraz pamätá svoju predchádzajúcu konfiguráciu. Počas celého riešenia projektu prebiehala implementácia ZMP, čo skvalitnilo pohyby hráča. V desiatom šprinte sme vykonali posledné úpravy v ZMP a dokončili sme implementáciu nového otáčania hráča. Táto implementácia je kvalitnejšia ako predchádzajúce pokusy. Agent sa vie rýchlo a stabilne otočiť k želanému bodu s toleranciou 5 stupňov. Stav projektu po desiatom šprinte vidno na obrázku 55, kde projekt obsahuje 14959 riadkov zdrojového kódu v 274 triedach. Kvôli refaktoringu a s prihliadnutím na fakt, že červené čísla v metrikách sa týkajú len XML parsera, ktorý zbehne len na začiatku programu a načíta anotácie a lowskilly, môžeme konštatovať, že sa nám podarilo výrazne zdvihnúť kvalitu zdrojového kódu projektu Jim a s ním spojenej dokumentácie, čo výrazne uľahčí prácu budúcim tímom. Rovnako sme im poskytli nové možnosti pohybu robota pomocou ZMP a vylepšené otáčanie. Podarilo sa nám implementovať aj viaceré taktiky pre univerzitný robocup turnaj. Merania pomocou opraveného test frameworku ukazujú, že mnohé z týchto taktík sú implementované dostatočne kvalitne a spĺňajú požiadavky dané turnajom.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,427	4,405	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,568	0,815	6	/Jim/src/sk/fiit/jim/agent/models/BodyPart.java	BodyPart
Nested Block Depth (avg/max per method)		1,511	0,989	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		10,143	20,03	122	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,184	6,288	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,455	0,284	1	/Jim/src/sk/fiit/jim/agent/trajectory	
Abstractness (avg/max per packageFragment)		0,078	0,175	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,469	0,271	1	/Jim/src/sk/fiit/jim/log	
Depth of Inheritance Tree (avg/max per type)		1,658	0,942	7	/Jim/src/sk/fiit/jim/gui/LevelCheckBox.java	
Weighted methods per Class (avg/max per typ	2920	10,618	17,818	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	121	0,44	3,769	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	54	0,196	0,564	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,203	0,377	1,667	/Jim/src/sk/fiit/jim/agent/highskill/kick/KickHighSkil...	
Number of Attributes (avg/max per type)	693	2,52	4,838	29	/Jim/src/sk/fiit/jim/agent/models/BodyPart.java	
Number of Static Attributes (avg/max per type	354	1,287	1,873	11	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1056	3,84	7,053	73	/Jim/src/sk/fiit/jim/agent/models/AgentModel.java	
Number of Static Methods (avg/max per type)	149	0,542	1,559	11	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,065	0,215	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	275	5,612	7,088	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	8	0,163	0,421	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	49					
Total Lines of Code	15360					
Method Lines of Code (avg/max per method)	9190	7,639	20,917	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 53: Výstup z Code Metrics pre projekt Jim po zimnom semestri.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,491	4,53	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,569	0,805	6	/Jim/src/sk/fiit/jim/agent/server/TFTPServer.java	TFTPServer
Nested Block Depth (avg/max per method)		1,53	1,007	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		9,837	18,49	109	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,082	6,174	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,449	0,28	1	/Jim/src/sk/fiit/jim/agent/trajectory	
Abstractness (avg/max per packageFragment)		0,077	0,175	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,476	0,271	1	/Jim/src/sk/fiit/jim/log	
Depth of Inheritance Tree (avg/max per type)		1,658	0,946	7	/Jim/src/sk/fiit/jim/gui/LevelCheckBox.java	
Weighted methods per Class (avg/max per typ	2815	10,465	17,21	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	121	0,45	3,81	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	48	0,178	0,537	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,196	0,374	1,667	/Jim/src/sk/fiit/jim/agent/highskill/kick/KickHighSkil...	
Number of Attributes (avg/max per type)	639	2,375	4,422	26	/Jim/src/sk/fiit/jim/agent/highskill/move/Walk.java	
Number of Static Attributes (avg/max per type	340	1,264	1,852	11	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	993	3,691	6,156	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	139	0,517	1,465	11	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,061	0,212	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	269	5,49	6,896	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	7	0,143	0,404	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	49					
Total Lines of Code	14734					
Method Lines of Code (avg/max per method)	8868	7,848	21,486	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 54: Výstup z Code Metrics pre projekt Jim po 6. šprinte.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,439	4,428	106	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Number of Parameters (avg/max per method)		0,579	0,822	6	/Jim/src/sk/fiit/jim/agent/models/BodyPart.java	BodyPart
Nested Block Depth (avg/max per method)		1,518	0,994	8	/Jim/src/sk/fiit/jim/annotation/data/XMLParser.java	parse
Afferent Coupling (avg/max per packageFragn		10,041	19,464	117	/Jim/src/sk/fiit/jim/agent/models	
Efferent Coupling (avg/max per packageFragm		4,184	6,288	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Instability (avg/max per packageFragment)		0,455	0,284	1	/Jim/src/sk/fiit/jim/agent/trajectory	
Abstractness (avg/max per packageFragment)		0,078	0,175	1	/Jim/src/sk/fiit/jim/code_review	
Normalized Distance (avg/max per packageFra		0,469	0,271	1	/Jim/src/sk/fiit/jim/log	
Depth of Inheritance Tree (avg/max per type)		1,661	0,943	7	/Jim/src/sk/fiit/jim/gui/LevelCheckBox.java	
Weighted methods per Class (avg/max per typ	2851	10,405	17,056	160	/Jim/src/sk/fiit/jim/agent/models/TacticalInfo.java	
Number of Children (avg/max per type)	121	0,442	3,775	59	/Jim/src/sk/fiit/jim/decision/situation/Situation.java	
Number of Overridden Methods (avg/max per	54	0,197	0,565	3	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Lack of Cohesion of Methods (avg/max per typ		0,203	0,377	1,667	/Jim/src/sk/fiit/jim/agent/highskill/kick/KickHighSkil...	
Number of Attributes (avg/max per type)	671	2,449	4,651	29	/Jim/src/sk/fiit/jim/agent/models/BodyPart.java	
Number of Static Attributes (avg/max per type	352	1,285	1,852	11	/Jim/src/sk/fiit/jim/agent/AgentInfo.java	
Number of Methods (avg/max per type)	1022	3,73	6,123	52	/Jim/src/sk/fiit/jim/annotation/data/Annotation.java	
Number of Static Methods (avg/max per type)	149	0,544	1,561	11	/Jim/src/sk/fiit/jim/Settings.java	
Specialization Index (avg/max per type)		0,065	0,216	1,5	/Jim/src/sk/fiit/jim/agent/highskill/move/WalkFast.ja...	
Number of Classes (avg/max per packageFragr	274	5,592	7,088	40	/Jim/src/sk/fiit/jim/decision/situation/octan	
Number of Interfaces (avg/max per packageFra	8	0,163	0,421	2	/Jim/src/sk/fiit/jim/agent/skills	
Number of Packages	49					
Total Lines of Code	14959					
Method Lines of Code (avg/max per method)	8905	7,618	20,836	375	/Jim/src/sk/fiit/jim/annotation/data/XMLCreator.java	createXML

Obr. 55: Výstup z Code Metrics pre projekt Jim po 10. šprinte.

13.8.2 Kvalita zdrojového kódu projektu RoboCupLibrary

Obrázok 56 zachytáva stav zdrojového kódu projektu RoboCupLibrary po prvom semestri. Keďže sme s týmto projektom nijako nemanipulovali, zachytáva pôvodný stav projektu. Projekt má dokopy 959 riadkov zdrojového

kódu v 21 triedach. Kvalita projektu nie je dostatočná. Zdrojový kód nie je vhodne formátovaný a nie je okomentovaný. Mnohé triedy vykonávajú duplicitné akcie. Po šiestom šprinte došlo k zmazaniu triedy Vector3 a jej migráciu na triedu Vector3D, ktorá bola vhodne naformátovaná a okomentovaná. To malo za následok pokles riadkov zdrojového kódu na 924. Odstránením duplicitných častí kódu a jeho dôsledným refaktorovaním v priebehu siedmeho šprintu došlo ku zvýšeniu kvality zdrojového textu programu, pričom projekt teraz obsahuje 901 riadkov kódu v 21 triedach. Stav zdrojového kódu po siedmom šprinte vidno na obrázku 57. Od tohto šprintu sa stav tohto projektu už nezmenil.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		1,282	0,799	6	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	minCircle
Number of Parameters (avg/max per method)		0,718	0,972	4	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	minCircle
Nested Block Depth (avg/max per method)		1,169	0,504	4	/RoboCupLibrary/tests/sk/fiit/robocup/library/math/...	reasonableNoiseCovaria...
Afferent Coupling (avg/max per packageFragn		24,6	42,749	110	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Efferent Coupling (avg/max per packageFragn		1,6	1,356	3	/RoboCupLibrary/src/sk/fiit/robocup/library/math	
Instability (avg/max per packageFragment)		0,31	0,291	0,75	/RoboCupLibrary/tests/sk/fiit/robocup/library/geom...	
Abstractness (avg/max per packageFragment)		0,2	0,4	1	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Normalized Distance (avg/max per packageFra		0,49	0,347	1	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Depth of Inheritance Tree (avg/max per type)		0,762	0,426	1	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Weighted methods per Class (avg/max per typ	159	7,571	10,022	46	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Children (avg/max per type)	0	0	0	0	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Number of Overridden Methods (avg/max per	8	0,381	0,722	3	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Lack of Cohesion of Methods (avg/max per typ		0,149	0,264	0,806	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Attributes (avg/max per type)	40	1,905	3,038	11	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Static Attributes (avg/max per type	7	0,333	1,084	5	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Methods (avg/max per type)	117	5,571	7,932	37	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Static Methods (avg/max per type)	11	0,524	1,052	3	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Specialization Index (avg/max per type)		0,039	0,068	0,2	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Classes (avg/max per packageFragr	21	4,2	1,72	7	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Number of Interfaces (avg/max per packageFra	0	0	0	0	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Number of Packages	5					
Total Lines of Code	959					
Method Lines of Code (avg/max per method)	508	4,097	5,184	29	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	rotateOver

Obr. 56: Výstup z Code Metrics pre projekt RoboCupLibrary na začiatku letného semestra.

13.8.3 Kvalita zdrojového kódu projektu TestFramework

Obrázok 58 zachytáva stav zdrojového kódu projektu TestFramework po prvom semestri. Keďže sme s týmto projektom predtým nijako nemanipulovali, zachytáva aj pôvodný stav projektu. Projekt má dokopy 10652 riadkov zdrojového kódu v 136 triedach. Cyklomatická zložitosť je vysoká, rovnako aj počty vnorených blokov a to v prípade parsera pre anotácie. Na konci šiesteho šprintu došlo len k menším zmenám súvisiacim s refaktorovaním triedy Vector3 a s jej migráciou na triedu Vector3D, čo je opísané v dokumentácii k

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		1,273	0,803	6	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	minCircle
Number of Parameters (avg/max per method)		0,719	0,973	4	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	minCircle
Nested Block Depth (avg/max per method)		1,165	0,487	4	/RoboCupLibrary/tests/sk/fiit/robocup/library/math/...	reasonableNoiseCovaria...
Afferent Coupling (avg/max per packageFragn		15,2	26,91	69	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Efferent Coupling (avg/max per packageFragm		1,6	1,356	3	/RoboCupLibrary/src/sk/fiit/robocup/library/math	
Instability (avg/max per packageFragment)		0,403	0,333	0,75	/RoboCupLibrary/tests/sk/fiit/robocup/library/geom...	
Abstractness (avg/max per packageFragment)		0,2	0,4	1	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Normalized Distance (avg/max per packageFra		0,397	0,331	1	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Depth of Inheritance Tree (avg/max per type)		0,762	0,426	1	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Weighted methods per Class (avg/max per typ	154	7,333	9,125	41	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Children (avg/max per type)	0	0	0	0	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Number of Overridden Methods (avg/max per	8	0,381	0,722	3	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Lack of Cohesion of Methods (avg/max per typ		0,148	0,264	0,806	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Attributes (avg/max per type)	40	1,905	3,038	11	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Static Attributes (avg/max per type	4	0,19	0,499	2	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Methods (avg/max per type)	114	5,429	7,372	34	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Static Methods (avg/max per type)	11	0,524	1,052	3	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Specialization Index (avg/max per type)		0,04	0,069	0,2	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	
Number of Classes (avg/max per packageFragr	21	4,2	1,72	7	/RoboCupLibrary/src/sk/fiit/robocup/library/geometry	
Number of Interfaces (avg/max per packageFra	0	0	0	0	/RoboCupLibrary/src/sk/fiit/robocup/library/annotati...	
Number of Packages	5					
Total Lines of Code	901					
Method Lines of Code (avg/max per method)	467	3,86	4,703	26	/RoboCupLibrary/src/sk/fiit/robocup/library/geomet...	getCircleIntersection

Obr. 57: Výstup z Code Metrics pre projekt RoboCupLibrary po 7. šprinte.

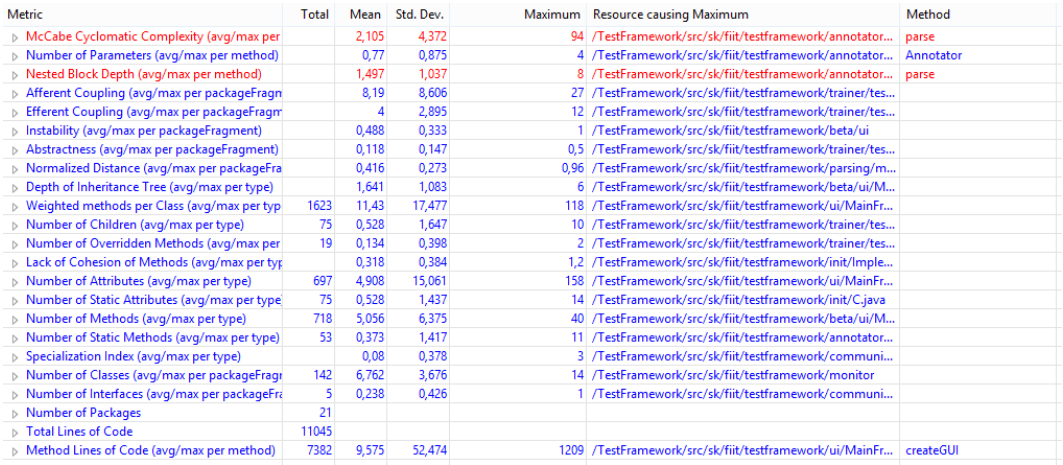
inžinierskemu dielu. V siedmom a ôsmom šprinte sme refaktorovali testovací framework. V tomto období sme aj opravili pridávanie hráčov do testovacieho frameworku, čo znova umožnilo využívať testovací framework. Stav zdrojového kódu projektu po ôsmom šprinte možno z hľadiska metrík vidieť na obrázku 59. Napokon sme sa v deviatom a desiatom šprinte venovali implementácii meraní pre turnaj, vďaka čomu možno využiť testovací framework na hodnotenie turnajových disciplín. Stav zdrojového kódu projektu po desiatom šprinte znázorňuje obrázok 60. Projekt mal koncom desiateho šprintu 11045 riadkov v 142 triedach. Všetky zmeny, ktoré sme vykonali v testovacom frameworku výrazne zvýšili jeho kvalitu. Na začiatku riešenia projektu bol framework nepoužiteľný, teraz v ňom vieme pridávať a analyzovať hráčov a využiť ho na vyhodnotenie niektorých turnajových disciplín. Nepodarilo sa nám však opraviť pridávanie anotácií, práve s nimi súvisí už spomínaná vysoká cyklomatická zložitosť. Vykonali sme však pomerne rozsiahlu analýzu anotovania a pravdepodobných príčin jeho nefunkčnosti.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,089	4,375	94	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Number of Parameters (avg/max per method)		0,78	0,88	4	/TestFramework/src/sk/fiit/testframework/annotator...	Annotator
Nested Block Depth (avg/max per method)		1,497	1,031	8	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Afferent Coupling (avg/max per packageFragn		8,333	8,632	27	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Efferent Coupling (avg/max per packageFragm		3,714	2,897	12	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Instability (avg/max per packageFragment)		0,468	0,332	1	/TestFramework/src/sk/fiit/testframework/beta/ui	
Abstractness (avg/max per packageFragment)		0,125	0,153	0,5	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Normalized Distance (avg/max per packageFra		0,421	0,278	0,96	/TestFramework/src/sk/fiit/testframework/parsing/m...	
Depth of Inheritance Tree (avg/max per type)		1,654	1,101	6	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Weighted methods per Class (avg/max per typ	1573	11,566	17,147	104	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Children (avg/max per type)	75	0,551	1,68	10	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Number of Overridden Methods (avg/max per	17	0,125	0,373	2	/TestFramework/src/sk/fiit/testframework/incomplet...	
Lack of Cohesion of Methods (avg/max per typ		0,327	0,385	1,2	/TestFramework/src/sk/fiit/testframework/init/Imple...	
Number of Attributes (avg/max per type)	665	4,89	13,762	136	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Static Attributes (avg/max per type	73	0,537	1,465	14	/TestFramework/src/sk/fiit/testframework/init/C.java	
Number of Methods (avg/max per type)	702	5,162	6,405	40	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Number of Static Methods (avg/max per type)	51	0,375	1,445	11	/TestFramework/src/sk/fiit/testframework/annotator...	
Specialization Index (avg/max per type)		0,078	0,381	3	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Classes (avg/max per packageFragr	136	6,476	3,567	14	/TestFramework/src/sk/fiit/testframework/ui	
Number of Interfaces (avg/max per packageFra	5	0,238	0,426	1	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Packages	21					
Total Lines of Code	10652					
Method Lines of Code (avg/max per method)	7143	9,486	49,413	1087	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	createGUI

Obr. 58: Výstup z Code Metrics pre projekt TestFramework na začiatku letného semestra.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,106	4,381	94	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Number of Parameters (avg/max per method)		0,762	0,874	4	/TestFramework/src/sk/fiit/testframework/annotator...	Annotator
Nested Block Depth (avg/max per method)		1,501	1,038	8	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Afferent Coupling (avg/max per packageFragn		8,143	8,626	27	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Efferent Coupling (avg/max per packageFragm		3,952	2,768	12	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Instability (avg/max per packageFragment)		0,489	0,334	1	/TestFramework/src/sk/fiit/testframework/beta/ui	
Abstractness (avg/max per packageFragment)		0,118	0,147	0,5	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Normalized Distance (avg/max per packageFra		0,415	0,274	0,96	/TestFramework/src/sk/fiit/testframework/parsing/m...	
Depth of Inheritance Tree (avg/max per type)		1,645	1,086	6	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Weighted methods per Class (avg/max per typ	1605	11,383	16,956	104	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Children (avg/max per type)	75	0,532	1,653	10	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Number of Overridden Methods (avg/max per	19	0,135	0,399	2	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Lack of Cohesion of Methods (avg/max per typ		0,317	0,386	1,2	/TestFramework/src/sk/fiit/testframework/init/Imple...	
Number of Attributes (avg/max per type)	675	4,787	13,564	136	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Static Attributes (avg/max per type	74	0,525	1,442	14	/TestFramework/src/sk/fiit/testframework/init/C.java	
Number of Methods (avg/max per type)	710	5,035	6,333	40	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Number of Static Methods (avg/max per type)	52	0,369	1,421	11	/TestFramework/src/sk/fiit/testframework/annotator...	
Specialization Index (avg/max per type)		0,081	0,379	3	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Classes (avg/max per packageFragr	141	6,714	3,588	14	/TestFramework/src/sk/fiit/testframework/monitor	
Number of Interfaces (avg/max per packageFra	5	0,238	0,426	1	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Packages	21					
Total Lines of Code	10846					
Method Lines of Code (avg/max per method)	7221	9,476	49,164	1087	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	createGUI

Obr. 59: Výstup z Code Metrics pre projekt TestFramework po 8. šprinte.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		2,105	4,372	94	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Number of Parameters (avg/max per method)		0,77	0,875	4	/TestFramework/src/sk/fiit/testframework/annotator...	Annotator
Nested Block Depth (avg/max per method)		1,497	1,037	8	/TestFramework/src/sk/fiit/testframework/annotator...	parse
Afferent Coupling (avg/max per packageFragn		8,19	8,606	27	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Efferent Coupling (avg/max per packageFragm		4	2,895	12	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Instability (avg/max per packageFragment)		0,488	0,333	1	/TestFramework/src/sk/fiit/testframework/beta/ui	
Abstractness (avg/max per packageFragment)		0,118	0,147	0,5	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Normalized Distance (avg/max per packageFra		0,416	0,273	0,96	/TestFramework/src/sk/fiit/testframework/parsing/m...	
Depth of Inheritance Tree (avg/max per type)		1,641	1,083	6	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Weighted methods per Class (avg/max per typ	1623	11,43	17,477	118	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Children (avg/max per type)	75	0,528	1,647	10	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Number of Overridden Methods (avg/max per	19	0,134	0,398	2	/TestFramework/src/sk/fiit/testframework/trainer/tes...	
Lack of Cohesion of Methods (avg/max per typ		0,318	0,384	1,2	/TestFramework/src/sk/fiit/testframework/init/Imple...	
Number of Attributes (avg/max per type)	697	4,908	15,061	158	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	
Number of Static Attributes (avg/max per type	75	0,528	1,437	14	/TestFramework/src/sk/fiit/testframework/init/C.java	
Number of Methods (avg/max per type)	718	5,056	6,375	40	/TestFramework/src/sk/fiit/testframework/beta/ui/M...	
Number of Static Methods (avg/max per type)	53	0,373	1,417	11	/TestFramework/src/sk/fiit/testframework/annotator...	
Specialization Index (avg/max per type)		0,08	0,378	3	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Classes (avg/max per packageFragr	142	6,762	3,676	14	/TestFramework/src/sk/fiit/testframework/monitor	
Number of Interfaces (avg/max per packageFra	5	0,238	0,426	1	/TestFramework/src/sk/fiit/testframework/communi...	
Number of Packages	21					
Total Lines of Code	11045					
Method Lines of Code (avg/max per method)	7382	9,575	52,474	1209	/TestFramework/src/sk/fiit/testframework/ui/MainFr...	createGUI

Obr. 60: Výstup z Code Metrics pre projekt TestFramework po 10. šprinte.

14 Metodika tvorby testov v projekte JIM

Posledná úprava	Peter Filípek
Platné od	8. december 2014
Poznámky	

Účelom tejto metodiky je definovať postup pri vytváraní testov nad implementovanými metódami v jazyku Java. Použijeme vývojové prostredie Eclipse a knižnicu JUnit. Metodika nepokrýva problematiku ako treba testy spúšťať, ako často testy spúšťať, ako reagovať v prípade že testy odhalia chybu v kóde.

Táto metodika je určená všetkým členom tímu ktorí:

- Vytvárajú nové metódy
- Refaktorujú aktuálne metódy
- Reimplementujú staré metódy
- Vytvárajú nové testy nad metódami ktoré nie sú pokryté testami
- Refaktorujú aktuálne testy

Všetky vytvorené triedy a testy musia dodržiavať konvenciu písania kódu uvedenú v dokumente Konvencie písania kódu v projekte Jim 13.2.

14.1 Použité pojmy

- Eclipse – vývojové prostredie, v ktorom využijeme framework JUnit.
- JUnit – knižnica slúžiaca na vytváranie, spúšťanie a vyhodnocovanie testov implementovaných tried.
- Testovacia trieda – trieda vykonávajúca test alebo skupinu testov.
- Testovaná trieda – trieda, ktorá je testovaná.
- Test – metóda nachádzajúca sa v testovacej triede ktorá testuje správanie metódy umiestnenej v testovanej triede.
- Tag – anotácia nachádzajúca sa pred metódou, ktorá sa využíva pri vykonávaní metódy.

14.2 Metódy pre ktoré vytvárajte testy

- Testy vytvárajte len pre metódy s prístupom public.
- Testy vytvárajte pre metódy pri ktorých často nastavajú chyby.
- Testy vytvárajte pre metódy ktoré často voláte v iných metódach.
- Testy vytvárajte pre metódy v ktorých voláte väčšie množstvo iných metód.

14.3 Umiestnenie testovacích tried

- Všetky testovacie triedy umiestnite do zdrojového priečinku, ktorý je oddelený od hlavného zdrojového priečinku projektu.
- Zdrojový priečinok vyhradený pre testovacie triedy pomenujte test.
- Do zdrojového priečinku vyhradeného pre testovacie triedy neumiestňujte žiadne iné triedy.
- Pri pridávaní testovacej triedy dodržujte rovnakú hierarchiu a názvy balíkov ako v hlavnom zdrojovom priečinku projektu.

Názov tried a testu píšte tak, aby bol v súlade s konvenciou uvedenou v dokumente Konvencie písania kódu v projekte Jim 13.2 kapitola Názvy. Testovaciu triedu pomenujte rovnako ako sa volá testovaná trieda s tým, že za názov pridajte príponu Test:

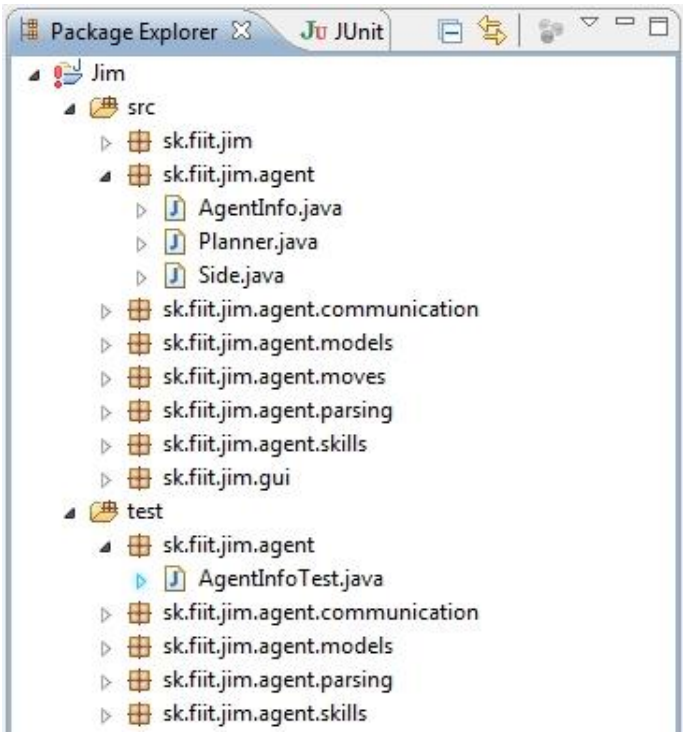
```
LowSkill (Testovana trieda)
LowSkillTest (Testovacia trieda)
```

Test pomenujte tak, aby jeho názov výstižne opisoval správanie ktoré testujete. Názov testu píšte v anglickom jazyku. Napríklad testujete či informácie o vzdialenosti boli vypočítané správne. Metódu pomenujte *calculateDistance()*.

14.4 Tagy

- @Test

  Tag @Test umiestňujte vždy pred každý test



Obr. 61: Príklad hierarchie balíkov a umiestnenia tried v projekte

- @Before

  Tag @Before umiestňujte vždy pred metódu ktorá sa ma vykonať pred každým testom

  Metódu pomenujte setUp()

  Metódu vytvorte s prístupom public

  V metóde inicializujte všetky údaje ktoré využívate vo viacerých testoch testovacej triedy

- @After

  Tag @After umiestňujte vždy pred metódu ktorá sa ma vykonať po každom teste

  Metódu pomenujte tearDown()

  Metódu vytvorte s prístupom public

Povolené sú všetky tágy nachádzajúce sa v dokumentácii k JUnit (`http://junit.org/apidocs/`), kde nájdete popis ich vlastností a využitia. Vyššie uvedené tagy sú najčastejšie používané.

Na testovanie používajte metódy triedy Assert. Nevyužívajte priamy prístup k metódam triedy Assert. Na prístup k metódam využívajte import static org.junit.Assert.*. Na testovanie výstupu testovanej metódy uprednostnite metódu assertEquals(). Metódy assertTrue() a assertFalse() použite len na testovanie výstupu ak je typu boolean. Metódy assertNull() a assertNotNull() použite len na testovanie či je alebo nie je objekt NULL.

Na testovanie sú povolené všetky metódy triedy Assert. Popis ich vlastností a využitia nájdete v dokumentácii k JUnit (`http://junit.org/apidocs/`). Vyššie uvedené sú najčastejšie používané.

Pre každé testovanie uvádzajte aj správu. Správy píšte v anglickom jazyku. Správy píšte tak, aby boli zmysluplné a nápomocné pri identifikácii problému.

```
assertEquals("The␣method␣should␣be␣invoked␣3␣times",
             3, invocationCount);
```

Ak má testovaná metóda chybový výstup, tak testujte aj tento výstup. Chybový výstup testujte v oddelenom teste v rovnakej testovacej triede.

Príklad: Ukážka testovacej triedy

```
import org.junit.*;
import static org.junit.Assert.*;
public class CalculatorTest {
        private Calculator calc;
        @Before
        public void setUp(){
        calc = new Calculator();
}

@Test
public void sum() {
        calc.sum(4.0);
        calc.sum(7.0);
        assertEquals("4+7␣should␣be␣11", 11.0,
           calc.getResult());
}
```

```
@Test
public void div() {
        calc.sum(5.0);
        calc.divide(2.0);
        assertEquals("5/2␣should␣be␣2.5", 2.5,
           calc.getResult());
}

@Test(expected=IllegalArgumentException.class)
public void divByZero() {
        calc.divide(0.0);
}
}
```

Pre každý vytvorený test vytvorte JavaDoc komentár podľa KONVENCIE PÍSANIA KÓDU V PROJEKTE JIM 15. V komentároch opíšte situáciu, ktorá sa testuje.

15 Konvencie písania zdrojového kódu

Posledná úprava	Juraj Šimek
Platné od	19. november 2014
Poznámky	Pridané príklady logovacích typov

Tento dokument opisuje to, aké konvencie je nutné dodržiavať pri dopĺňaní kódu projektu agenta hrajúceho robotický futbal a s ním súvisiacich projektov. Konvencia je určená pre všetkých členov tímu, ktorí zasahujú do zdrojového kódu projektu. Konvencie písania kódu uvedené v tomto dokumente vychádzajú najmä z dokumentu *Java Code Conventions* [4].

15.1 Základné konvencia písania kódu v jazyku Java

15.1.1 Názvy

Názvy *tried, rozhraní a výčtových typov* začínajte veľkým písmenom, pričom je dôležité, aby ste im vyberali výstižné, ale pokiaľ možno stručné názvy. Pri viacslovných pomenovaniach používajte štýl camel-case. V prípade rozhraní uveďte v rámci názvu predponu `i`.

```
class KickHighSkill { }
interface IHighSKill { }
```

Názvy *premenných a metód* začínajte malým písmenom. Pri viacslovných pomenovaniach znova použite camel-case. Opäť je dôležité dbať na to, aby bol názov zrozumiteľný, ale pritom stručný.

```
void getPrescribedSituations() { ... }
int currentHighSkill = 40;
```

Názvy *konštánt* uvádzajte veľkým písmom. Ich jednotlivé slová oddeľujte podtržníkom (_).

```
public final static int THIS_IS_CONSTANT = 30;
```

Názvy *balíkov* píšte malými písmenami.

```
sk.fiit.jim.agent
```

15.1.2 Formátovanie zdrojového kódu

Každý kód vo vnútri akéhokoľvek bloku odsaďte *tabulátorom*:

```
{
    int x = 4;
}
```

Otváraciu zátvorku označujúcu blok kódu píšte hneď *za* príkazom, ku ktorému prislúcha:

```
if (x == 3) {
    ...
}
```

Telo každého cyklu a podmieneného príkazu uvádzajte v zátvorkách `{` a `}` *aj v prípade, že predstavuje len jeden príkaz!* Toto je nesmierne dôležité kvôli predchádzaniu chybám pri rozširovaní kódu v budúcnosti.

```
if (x == 3) {
    return false;
} else {
    return true;
}
```

Každý príkaz píšte do osobitného riadku.

```
Xcoord++;
Ycoord++;
kick();
```

Každú *premennú* (aj lokálnu) vhodne definujte na novom riadku.

```
int myVariableA = 4;
int anotherVariable;
int thisVariableIsAlsoNeeded;
```

Dĺžka riadku by nemala presiahnuť 80 znakov. Ak je príkaz príliš dlhý rozdeľte ho, pričom rozdelenú časť vhodne odsaďte tak, aby sa operátory nachádzali na novom riadku. Bližšie informácie nájdete v kapitole *Wrapping Lines* dokumentu [4]. V tomto prípade treba pamätať hlavne na to, aby bolo pri pohľade na kód jasné, čo patrí spolu, a čo je už iný príkaz.

Medzi definíciami metód v triede uveďte jeden prázdny riadok.

```
public String toString() {
    ...
}

/**
 * Comment
 */
public int getPlaygroundSize() {
    ...
}
```

15.1.3 Serializácia

Ak trieda požaduje implementovanie rozhrania *Serializable* a v rámci toho definíciu *serialVersionUID*, neuvádzajte *@SupressWarning("serialization")*, ale do vnútra triedy uveďte definíciu, napríklad *static final long serialVersionUID = 42L;* Číslo *serialVersionUID* môže byť samozrejme ľubovoľné. Viac o jeho význame a o serializácii objektov je v [5] a [1].

15.1.4 Písanie komentárov v jazyku Java

Jednoriadkové komentáre píšte výlučne pomocou `//` a neuzatvárajte ich medzi `/*` a `*/`

```
// single line comment
```

Ak má komentár viac riadkov, napíšte ho medzi `/*` a `*/`, pričom je vhodné aby ste na každom riadku uviedli znak `*`. Medzi znakom `*` a prvým písmenom komentára uveďte jednu medzeru.

```
/*
* This is
* Multiline
* Comment
*/
```

15.2 Dokumentačné komentáre

Pred každou triedou a metódou, ktorá je verejná sa uvádzajte dokumentačný komentár medzi `/**` a `*/` generujúci JavaDoc. Viac informácií o písaní JavaDocu je v [3]. Nepoužívajte žiadne vymyslené tagy, ktoré nedefinuje JavaDoc. JavaDoc pripúšťa tieto tagy:

```
@author
@version
@param
@return
@exception
@see
@since
@serial
@deprecated
```

Význam jednotlivých tagov je uvedený v [3].

Každý *autor* uvedie do dokumentačného komentára kódu, ktorému prislúcha svoje meno v nasledovnom tvare:

```
@author <meno_autora> (<nazov_timu> - <rok>)
```

Pričom:
`<meno_autora>` - autorovo celé meno
`<nazov_timu>` - meno tímu, ak v tíme nepracuje, uvedie len rok
`<rok>` - rok v ktorom je kód upravovaný

Príklad:

```
@author Juraj Simek (Infinity - 2014)
```

Pred každou public metódou do komentára uveďte pomocou `@param` význam jednotlivých parametrov, pomocou `@exception` opíšte kedy vyhadzuje akú výnimku a pomocou `@return` napíšte, čo metóda vracia:

```
/**
 * Returns position of the goal according to player's side.
 *
 * @param side player's team side
 * @return absolute position of the goal
```

```
 */
private Vector3D getGoalAbsolutePosition(Side side) {
    ...
}
```

Dokumentačné komentáre musia byť stručné, ale predovšetkým zrozumiteľné, aby sa v nich vyznali aj tí, čo daný kód neprogramovali.

15.3 Výnimky

Ak zachytávate výnimku, je potrebné ju zalogovať. Nikdy možné výnimky neignorujte pomocou `catch(Exception e) {}` ! Vhodné je buď výnimku zachytiť a zalogovať a potom ošetriť výnimočný stav, alebo ju znova vyhodiť a ošetriť prípadný výnimočný stav v inej časti kódu, ak to v tejto časti nie je možné. Nasledovný kód by sa preto nemal vyskytovať často:

```
catch(NoSuchMethodException e) {
    LOG.error("Situation␣description", e);
    throw e;
}
```

Niekedy je však vhodné zalogovať, že výnimka niekde nastala a znova ju vyhodiť. Preto je vyššie uvedený kód povolený v situáciách, keď je to naozaj opodstatnené. Ak uznáte za vhodné zalogovať výnimku a opäť ju vyhodiť bez jej ošetrenia, uveďte komentár vysvetľujúci vaše rozhodnutie.

15.4 Logovanie

V metóde, kde sa loguje získate *loger* pomocou `JLog.getLogger()`. Ideálne je, aby ste vytvorili členskú premennú pre celú triedu, ktorá bude loger používať:

```
Private Logger LOG = JLog.getLogger();
```

Logujte potom pomocou štandardného API triedy Logger [2]. Pre úplnosť informácií v krátkosti uvádzame to, ako sa loguje pomocou tohto rozhrania. Štandardne loger vytvoríte ako privátnu statickú konštantu v triede, ktorá ho používa. Vďaka tomu budete môcť zachytávať, v ktorej triede vznikla udalosť logovania. Logeru dáte identifikátor, ktorý najčastejšie predstavuje meno danej triedy:

```
private final static Logger LOG =
   Logger.getLogger(LoggingExamples.class.getName());
```

Na logovanie v Jimovi však použite predkonfigurovaný globálny loger pomocou `JLog.getLogger()` tak, ako to bolo opísané vyššie.

Keď ste získali loger, v akejkoľvek metóde danej triedy potom možno tento loger použiť na logovanie rôznych udalostí. Java API predpisuje nasledovné najčastejšie používané úrovne logovania *Level.FINEST, Level.FINER, Level.FINE, Level.INFO, Level.WARNING a Level.SEVERE*, kde *FINEST – FINE* sú na najnižšej úrovne logovania a logujú sa nimi najmenej podstatné informácie. Nimi sa logujú najmä informácie o úspešnom vykonaní určitého bloku kódu. Level *FINEST* používajte na logovanie pomocných ladiacich výpisov v čase vývoja novej funkcionality. Pomocou *INFO* logujte rôzne informácie, ktoré chcete vložiť do logu, majú informačný charakter a neviete im priradiť vhodnejší level. *WARNING* slúži na logovanie varovaní (napríklad informácie o tom, že nastala výnimka a bola ošetrená) a *SEVERE* na logovanie závažných chýb, z ktorých sa program nevie zotaviť. Logovať môžete pomocou metódy `log()`:

```
LOG.log(Level.INFO, "Information");
```

Pre základné typy logovania definované Java API logerom môžete použiť aj logovanie pomocou predpísaných metód:

```
LOG.finest("Finest");
LOG.finer("Finer");
LOG.fine("Fine");
LOG.info("Information");
LOG.warning("Warning");
LOG.severe("Severe");
```

Loger JLog používaný v projekte Jim navyše, oproti úrovniam v triede Level, definuje tieto typy logov:

```
LogType.INIT
LogType.AGENT_MODEL
LogType.WORLD_MODEL
LogType.INCOMING_MESSAGE
LogType.OUTCOMING_MESSAGE
```

```
LogType.LOW_SKILL
LogType.HIGH_SKILL
LogType.GUI
LogType.TACTIC
```

Tieto typy umožňujú logovať informácie pre špeciálne udalosti. Pomocou týchto typov logujte len informácie do úrovne *Level.INFO* štandardného Java API logera, pričom ak môžete použiť jeden z týchto rozšírených typov, použite ho. Ak chcete logovať nejakú udalosť, ktorú LogType nepredpisuje, buď ju do LogType pridajte, alebo použite Level.INFO. Nový typ logu môžete pridať do triedy LogType pridaním nasledovného riadku v zozname definícií konštánt:

```
public static LogType <NEW_TYPE_NAME> =
  new LogType("<NEW_TYPE_NAME>",
    BASE_LEVEL_NUMBER + <NEXT_NUMBER>);
```

<NEW_TYPE_NAME> predstavuje názov nového logovacieho typu a píšte ho veľkými písmenami, nakoľko ide o konštantu. *<NEXT_NUMBER>* udáva poradové číslo nového levelu a zapíšte ho ako celé číslo o jednotku väčšie od najväčšieho čísla, ktoré používa nejaký typ logu definovaný v triede *LogType*.

Význam jednotlivých typov dodefinovaných v triede LogType je nasledovný:

- *INIT*: Inicializačné správy napríklad vo funkcii main(), nadviazanie spojenia so serverom, načítanie XML súborov a všetky akcie, ktoré konfigurujú agenta skôr, ako začne odosielať nejaké dáta na sever. Príklad: sk.fiit.jim.init.SkillsFromXmlLoader metóda laod()

  ```
  long xmlEnd = System.currentTimeMillis();
  double seconds = (xmlEnd - xmlStart) / 1000.0;
  LOG.log(LogType.INIT, "Moves␣loaded␣from␣files␣in␣"
              + seconds + "␣seconds");
  ```

- *AGENT_MODEL*: Logy týkajúce sa modelu agenta. Najčastejšie v balíku jim.agent. Tento level možno použiť všade, kde sa spracovávajú informácie o reprezentácii agenta, ako jeho poloha. Príklad: sk.fiit.jim.agent.AgentInfo metóda isWayFree()

  ```
  LOG.log(LogType.AGENT_MODEL,
  ```

```
        "opponentPosition = ["
        + opponentPlayerPosition.getX()
        + ","
        + opponentPlayerPosition.getY() + "]");
```

- *WORLD_MODEL*: Logy týkajúce sa modelu sveta. Používa sa napríklad vo funkciách, ktoré prepočítavajú polohu objektov na ihrisku. Príklad: sk.fiit.jim.agent.models.WorldModel metóda calculateBallPosition()

```
LOG.log(LogType.WORLD_MODEL,
        "Player IDs are not implemented yet!!!");
```

- *INCOMING_MESSAGE*: Logovanie správy prichádzajúcej zo servera alebo logovanie informácií z funkcií, ktoré takúto správu spracúvajú. Príkad: sk.fiit.jim.agent.communication.Communication metóda receive()

```
String incoming = new String(buffer, 0,
                             messageLength);
LOG.log(LogType.INCOMING_MESSAGE, incoming);
```

- *OUTCOMING_MESSAGE*: Logovanie správy odosielanej na server alebo logovanie informácií z funkcií, ktoré takúto správu spracúvajú a odosielajú. Príklad: sk.fiit.jim.agent.communication.Communication metóda transmit()

```
private void transmit(String what) throws
IOException{
   LOG.log(LogType.OUTCOMING_MESSAGE, what);
   ...
```

- *LOW_SKILL*: Informačné logy týkajúce sa low skillov. Príklad: sk.fiit.jim.agent.moves.LowSkill metóda step()

```
LOG.log(LogType.LOW_SKILL,
       "Currently active phase: "
       + activePhase.name);
```

- *HIGH_SKILL*: Informačné logy týkajúce sa high skillov. Príklad: sk.fiit.jim.agent.highskill.GoTo metóda pickLowSkill()

```
else if (this.tacticalInfo.isOnPosition()) {
   LOG.log(LogType.HIGH_SKILL, "close");
   return null;
}
```

- *GUI*: Informačné logy týkajúce sa grafického rozhrania robota. Príklad: sk.fiit.jim.gui.ReplanWindow metóda reloadScripts()

```
LOG.log(LogType.GUI,
       "Script␣reload␣request␣from␣GUI");
```

- *TACTIC*: Informačné logy týkajúce sa vrstvy taktík. Príklad: sk.fiit.jim.decision.tactic.MatchStarterTactic metóda runBeam()

```
this.beamExec
   .BeamAgent(Vector3D.cartesian(-4, 7, 0.1));
LOG.log(LogType.TACTIC, "BEAM␣-␣LEFT␣7");
```

Ak logujete iba pomocné výpisy (najmä pre seba), ktoré nie sú potrebné pre finálne logovanie, používajte *Level.FINEST*, ako už bolo spomenuté vyššie. Loger môžete potom v main() funkcii nastaviť tak, aby sa logovali len vyššie úrovne. Ak potrebujete vypisovať počas vývoja nejaké informácie pomocou *System.out.println()*, môžete tak urobiť, ale pri finalizovaní a nasadení vášho kódu nezabudnite tieto časti skutočne zmazať, nie len zakomentovať.

Loger môžete konfigurovať vo funkcii main() projektu. Pred jeho prvým spustením je potrebné, aby ste zavolali metódu *JLog.setup()*, ktorá prijíma 2 boolean hodnoty. Prvá určuje, či sa bude logovať na konzolu, druhá, či sa bude logovať do súboru. Súbor sa nachádza v koreňovom adresári projektu a má názov *jim_logs.html*. Logy sú uložené v prehľadnej HTML tabuľke.

Bez ďalších nastavení sa loger obalený triedou JLog správa ako štandardný Logger z Java API. Ak chcete logovať aj dodatočné typy logov menované vyššie, musíte v main() tieto typy pridať pomocou metódy addLogType():

```
JLog.addLogType(LogType.INCOMING_MESSAGE);
```

Ak chcete pridať všetky typy, ktoré definuje LogType, zavoláte metódu addAllLogTypes() triedy JLog. Potom možno logovať tieto nové typy pomocou metódy log triedy Logger:

```
LOG.log(LogType.INCOMING_MESSAGE, "message");
```

Logy by mali byť zrozumiteľné. Každý zápis do logu má mať opodstatnenie a má byť zmysluplný a formulovaný tak, aby sa dalo pochopiť, čo sa udialo a prečo. Logovať treba len pomocou tu opísaného API. Na logovanie sa nepoužívajú vlastné metódy a hlavne nie System.out.println(). Nedefinujte ďalšie logery ale používajte globálny loger, ktorý je nakonfigurovaný pomocou JLog. Vhodné je obmedziť sa len na používanie metód *log(), finest(), finer(), fine(), info(), warning(), severe()*.

15.5 Ďalšie konvencie

Metódy, ktoré vracajú kolekcie alebo polia nesmú vracať null. Namiesto null je vráťte prázdnu kolekciu alebo prázdne pole.

Pri pomenovávaní premenných a písaní komentárov sa vyhnite skratkám. Všeobecne známe skratky (napr. HTML, TCP/IP) sú v poriadku.

Ak potrebujete použiť nejakú funkcionalitu, najprv sa dôkladne pozrite do dokumentácie, či na to neexistuje trieda alebo nejaká metóda a až vtedy ak neexistuje vytvorte odpovedajúci kód. Nikdy neprogramujte triedy ako hešovacie tabuľky, ktoré má Java v knižniciach.

Používajte výhody generického programovania pri práci s kolekciami. Vždy uvádzajte do zátvoriek < a >, aké typy sa v danej kolekcii budú používať.

Metódy, ktoré vracajú kolekcie alebo polia nesmú vracať *null*. Namiesto *null* je potrebné vrátiť prázdnu kolekciu alebo prázdne pole.

Vyhnite sa skratkám. Všeobecne známe skratky sú v poriadku.

Ak chcete niečo vytvoriť, najprv sa pozrite do dokumentácie, či na to neexistuje trieda a až vtedy ak neexistuje ju vytvorte. Neprogramujte veci ako hešovacie tabuľky, ktoré má Java v knižniciach.

Používajte výhody generického programovania pri práci s kolekciami. Vždy uvádzajte do zátvoriek < a >, aké typy sa v danej kolekcii budú používať.

References

[1] Journal Dev. *Java Serialization Example Tutorial, Serializable, serialVersionUID*. http://www.journaldev.com/2452/java-serialization-example-tutorial-serializable-serialversionuid. [Online; accessed 19-October-2014]. 2013.

[2] Jakob Jenkov. *Java Logging*. http://tutorials.jenkov.com/java-logging/index.html. [Online; accessed 19-October-2014].

[3] Oracle. *How to Write Doc Comments for the Javadoc Tool*. http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html. [Online; accessed 19-October-2014].

[4] Oracle. *Java Code Conventions*. http://www.oracle.com/technetwork/java/codeconventions-150003.pdf. [Online; accessed 19-October-2014]. 1997.

[5] Java Practices. *Implementing Serializable*. http://www.javapractices.com/topic/TopicAction.do?Id=45. [Online; accessed 19-October-2014].