Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Tímový projekt – RoboCup 3D

Dokumentácia k riadeniu projektu

Študijný odbor: Informačné systémy, Softvérové inžinierstvo
Predmet: Tímový projekt
Vedúci projektu: Ing. Marián Lekavý
Tím: RFC Megatroll
Členovia tímu: Vladimír Bošiak
Samuel Benkovič
Michal Petráš
Martin Adámik
Igor Homola
Michal Čerešnák
Matej Bádal

Obsah

1 Úvod

Tento dokument obsahuje zoznam kompetencií tímu RFC Megatroll, dlhodobé rozdelenie úloh. Taktiež obsahuje zoznam podielov autorstva na dokumentácií k riadeniu ale i k inžinierskemu dielu.

Prvá kapitola pojednáva o ponuke nášho tímu a schopnostiach tímu. Druhá kapitola slúži pre prehľad o podiele autorstva v dokumentácií. V ďalších kapitolách sú spísané nami používané metodiky, komunikačné prostriedky. Posledná kapitola obsahuje metodiky vytvorené jednotlivými členmi tímu.

2 Ponuka tímu

2.1 Predstavenie tímu

Spoločné Znalosti: HTML,CSS,Java,UML,XML

Adámik Martin (Absolvent FIIT STU)

- Technológie: JQuery, PHP, SQL
- Práca v tíme: školské projekty

Bádal Matej (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend 2, Nette základy), Android, SQL
- Práca v tíme: niekoľko mesiacov (Medium13, s.r.o.)

Benkovič Samuel (Absolvent FIIT STU)

- Technológie: ASP.NET, CMS Umbraco, SQL
- Práca v tíme: niekoľko mesiacov (Sféra, a.s.)

Bošiak Vladimír (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend 1,2), Xpath, SOAP, OWL, SQL
- Práca v tíme: viac ako rok (Medium 13, s.r.o.)

Čerešňák Michal (Absolvent VUT FIT)

- Technológie: C#, SQL
- Práca v tíme: školské projekty (5 členov)

Homola Igor (Absolvent FI PEVŠ)

- Technológie: Administrácia Linux, Windows
- Práca v tíme: viac ako rok

Petráš Michal (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend), Návrh a anlýza IS, SQL
- Práca v tíme: viac ako rok (Eby, s.r.o.)

2.2 Preferencia tém

2.2.1 Virtuálna FIIT na mobile

V poslednej dobe narastá dopyt po mobilných aplikáciách z dôvodu rozšírenosti mobilných zariadení pre bežných používateľov. Z týchto dôvodov si myslíme, že virtuálna FIIT na mobile poskytne množstvo informácií jednoduchým spôsobom pre používateľov smartphone-ov. Táto mobilná aplikácia hlavne umožní cieľovým používateľom zjednodušiť ich orientáciu v priestoroch našej fakulty.

2.2.1.1 Motivácia

Motiváciou je zdokonalenie sa vo vývoji aplikácií pre mobilné zariadenia implementovaných pomocou štandardných webových technológií (HTML5, CSS, JS) a zároveň sa naučiť, ako tieto technológie zobraziť na mobilnom zariadení.

Ďalší dôvod prečo sa zaujímame o túto tému je uľahčiť dostupnosť dôležitých informácií nielen pre nových študentov, ale aj pre stálych návštevníkov našej fakulty. S tým súvisí aj propagácia fakulty a jej zviditeľnenie a zatraktívnenie širšej verejnosti.

Veríme, že náš tím je dostatočne kvalifikovaný a spĺňa všetky kritéria pre úspešné uskutočnenie tohto projektu, keďže každý jeden člen má skúsenosti s webovými technológiami. Navyše jeden člen má základné skúsenosti s tvorbou aplikácií pre mobilnú platformu Android.

2.2.1.2 Námety

- Informácie o prebiehajúcich udalostiach v miestnostiach(Aj vďaka použiteľnosti QR kódov),
- denné menu v bufete,
- osobný rozvrh na "jeden klik",
- zlepšovať aplikáciu na základe feedback-ov od používateľov.

2.2.2 Analýza výsledkov výskumu

Pri písaní študentských a odborných prác sa autori odkazujú na zdroje z iných publikácií. Vzhľadom na veľký objem zdrojov a nejednoznačnosť v uvádzaní mien autorov, názvov publikácií prípadne kľúčových slov sa ľudia, ktorí v týchto dátach hľadajú, stretávajú s mnohými problémami. Vytvorenie produktu, ktorý by zefektívnil spracovanie, vyhľadávanie, prípadne vizualizáciu informácií by mohlo pomôcť pri celkovom spracovaní bibliografických odkazov.

2.2.2.1 Motivácia

Naša motivácia je práca nad reálnymi dátami z dôvodu efektívnosti pri overovaní výsledkov, ich grafickej reprezentácií vo webovom rozhraní a zapojení nových technológií (MongoDB / Sémantické technológie) do riešenia problému.

Všetci členovia majú skúsenosti s technológiami, ktoré sú dominantné na webe. Viacero členov tímu má skúsenosti s tvorbou webových aplikácií / služieb s použitím databáz. Tieto skúsenosti by sme vedeli naplno využiť pri práci na projekte.

2.2.2.2 Námety

Jedným z možných riešení by bolo vytvorenie hrubého klienta, ktorý by spolu s intuitívnym rozhraním poskytoval pre koncového používateľa jednoduchú prácu s bibliografickými odkazmi.

Využitie existujúcich knižníc napísaných v JS (napr. D3.js) pre zobrazovanie relevantných dát na frontend-e.

2.3.1 Webový komunitný systém otázok a odpovedí

Na internete je v dnešnej dobe dostupných mnoho informácií na rôznorodé témy. Problémom pre koncového používateľa je orientácia vo veľkom množstve výsledkov vyhľadávania. Je prirodzené, že rôzne skupiny používateľov majú problémy, na ktoré hľadajú odpoveď.

Jednou z takýchto skupín sú aj študenti, ktorí často musia riešiť podobné úlohy. Tým pádom hľadajú odpovede na otázky, ktoré už niekto zodpovedal.

V globálnom merítku vyhľadávanie odpovedí na danú tému uľahčujú tzv. Q&A portály.

2.3.1.1 Motivácia

Poskytnúť skupine študentov portál, kde by bolo možné nájsť v kategóriách usporiadané otázky a odpovede. Použitie overených procesov z populárneho portálu Stack Overflow, ktoré by boli prispôsobené potrebám študentov.

Rozšírenie existujúcich konceptov, tak aby bolo čo najjednoduchšie opísať daný problém (prikladanie mediálneho obsahu). V neposlednom rade nám ide o čo najlepší user-experience.

Myslíme si, že sme vhodní kandidáti, pretože každý člen tímu má skúsenosti s technológiami používanými na webe (HTML, CSS, Javascript, PHP). Taktiež viacerí členovia majú skúsenosti s programovaním v tíme a prácou na rozsiahlejších webových aplikáciách.

2.3.1.2 Námety

- hodnotenie prispievateľov
- zatraktívniť GUI
- zhodnotenie kvality príspevku
- implementovanie existujúceho procesu získavania odpovedí

3 Úlohy členov tímu

Táto kapitola obsahuje dlhodobé a krátkodobé úlohy jednotlivých členov tímu. Taktiež obsahuje i autorstvo k jednotlivým častiam dokumentácie.

3.1 Dlhodobé úlohy

V nasledovnej tabuľke (1) sú rozpísané úlohy jednotlivých členov tímu ku dňu 9.10.2013 .

Tabuľka 1- dlhodobé úlohy

Meno	Funkcia
Vladimír Bošiak	Vedúci tímu, Manažér komunikácie
Michal Čerešňák	Manažér rizík
Igor Homola	Manažér plánovania, Webmaster
Matej Bádal	Manažér kvality
Martin Adámik	Manažér monitorovania projektu
Samuel Benkovič	Manažér podpory vývoja
Michal Petráš	Manažér dokumentácie

3.2 Autorstvo v dokumentácii k riadeniu

Tabuľka číslo 2 zobrazuje podiel práce každého člena na tvorbe dokumentácie k riadeniu.

Tabuľka 2- autorstvo v dokumentácii k riadeniu

Autor	Kapitola	Podiel práce
Michal Petráš	Úvod	100%
	Ponuka tímu	14%
	Úlohy členov tímu	100%
	Zápisnica č.5	100%
	6.7	100%
	7.7	100%
Michal Čerešňák	Ponuka tímu	14%
	Zápisnica č.2	100%
	Zápisnica č.10	100%
	6.5	100%
	7.5	100%
Igor Homola	Ponuka tímu	14%
	Zápisnica č.3	100%
	Zápisnica č.9	100%
	6.1	100%
	7.1	100%

Martin Adámik	Ponuka tímu	14%
	6.6	100%
	7.6	100%
	Zápisnica č.7	100%
Matej Bádal	Ponuka tímu	16%
	Zápisnica č.1	100%
	Zápisnica č.8	100%
	6.4	100%
	7.4	100%
Vladimír Bošiak	Ponuka tímu	14%
	Zápisnica č.6	100%
	6.2	100%
	7.2	100%
Samuel Benkovič	Ponuka tímu	14%
	Zápisnica č.4	100%
	6.3	100%
	7.3	100%

3.3 Autorstvo v dokumentácii k inžinierskemu dielu

V tabuľke 3 je zobrazený podiel na vypracovaných častiach dokumentácie k inžinierskemu dielu.

Tabuľka 3 - autorstvo v dokumentácii

Autor	Kapitola	Podiel práce
Michal Petráš	2.1.1 2.1.2 2.1.3 2.1.4 2.1.7 2.2.1 2.2.5 2.2.6 2.2.8 2.2.9 2.3.1 2.3.2 2.4.3 2.5 3.2	14% 14% 14% 100% 16% 16% 25% 100% 100% 100% 33% 100% 20% 100% 100%
Michal Čerešňák	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.5 2.3.1	14% 14% 14% 14% 16% 25% 33%
Igor Homola	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1	14% 14% 14% 14% 16%

	2.2.5 3.1	25% 100%
Martin Adámik	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.5 2.2.7 2.3.1 3.4.1	14% 14% 14% 14% 16% 25% 50% 33% 100%
Matej Bádal	2.1.1 2.1.2 2.1.3 2.1.5 2.1.7 2.2.1 2.2.2 3.4.2	14% 14% 14% 100% 14% 16% 100% 100%
Vladimír Bošiak	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.7 2.4.5	14% 14% 14% 14% 16% 50% 100%
Samuel Benkovič	2.1.1 2.1.2 2.1.3 2.1.6 2.1.7 2.2.3 2.2.4 2.3.3 2.4.1 2.4.2 2.4.3 3.4.3	16% 16% 16% 100% 14% 100% 100% 100% 100% 100% 80% 100%

4 Podporné prostriedky

4.1 Komunikácia

Na komunikáciu používame skype, mobilné telefóny, taktiež máme vytvorenú facebook skupinu pre informovanie všetkých členov. Máme založený aj tímový email, ktorý automaticky rozosiela správy všetkým členom tímu, tak aby bol informovaní ihneď. V prípade potreby sa stretávame osobne a riešime problémy spoločne.

4.2 Manažment projektu

Pre manažment projektu sme si vybrali nástroj Jira. Tento nástroj umožňuje sledovať prácu na všetkých zadaných úlohách, podporuje agilný vývoj formou "Agile" pluginu. Taktiež umožňuje monitorovanie času a lepší odhad na dokončenie jednotlivých úloh, prípadne celého šprintu. Ku každému šprintu sú vytvárané úlohy, ktoré sa následne prideľujú jednotlivým členom tímu.

5 Používané metodiky

5.1 Metodika Scrumovania

ScrumMaster na každý šprint sa zvolí poďla abecedy. Vychádzame z abecedného zoradenia podľa priezviska. Teda v tomto poradí:

1.Adámik

2.Bádal

3.Benkovič

4.Bošiak

5.Čerešnák

6.Homola

7.Petráš

5.2 Metodika pre určenie zapisovateľa

Každý týždeň je zapisovateľom iný člen tímu. Poradie , v ktorom sa na pozícii zapisovateľa členovia tímu striedajú je nasledovné:

1.Bádal

2.Čerešňák

3.Homola

4.Benkovič

5.Petráš

6.Bošiak

7.Adámik

5.3 Metodika pre odovzdávanie dokumentácie používateľských príbehov

Každý člen tímu odovzdá požadované časti dokumentácie na DropBox do priečinka nazvaného podľa šprintu (teda napr. šprint3) a do podpriečinka so svojím menom. Časť dokumentácie nazve podľa názvu príbehu tak aby bolo jasné čoho sa dokument týka. Teda napríklad : OdstranenenieKodu-Adamik.doc

6 Manažment

6.1 Manažment plánovania

6.1.1 Pre koho je metodika určená

Metodika je určená pre členov tímového projektu, každý člen týmu si musí zaznamenávať svoju prácu na projekte. Primárne je metodika určená pre tímy 4 a 9.

6.1.2 Obsah metodiky

Metodika je určená pre prácu so systémom JIRA. Systém je určený na evidenciu, sledovanie a plánovanie úloh vo vývojárskych tímoch.

6.1.3 Zoznam nadväzujúcich metodík a dokumentov

Metodika pre tvorbu dokumentácie (Michal Petráš)

6.1.4 Definícia pojmov

Task – úloha
Story / Userstory –používateľská požiadavka
Log work – zaznamenávanie práce
Sub-task – pod úloha
Issue/issues–otázka, požiadavka, úloha, záznam(v slovenskom jazyku neexistuje vhodný preklad slova preto sa v dokumente sa používa označenie issue / issues pre množné číslo)
Backlog – zoznam všetkých issues
Query - dotaz

6.1.5 Role a zodpovednosti používateľov

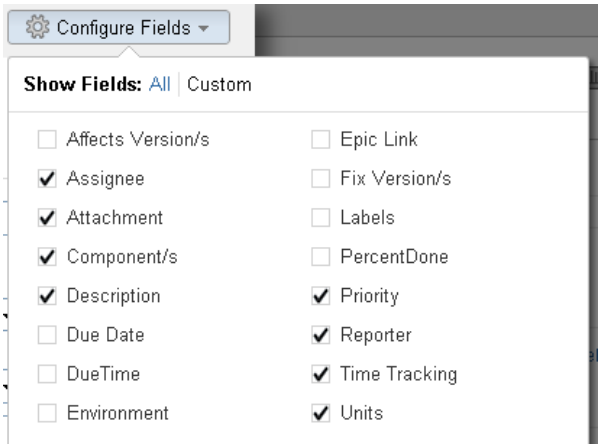
Správca systému JIRA za tímvytváranie a ukončenie šprintov
Scrummaster tvorba reportov, priraďovanie issues
Člen tímuzaznamenávanie času, tvorba issue,získavanie a ukončovanieissue
Vedúci tímu (Reporter)priraďovanie a uzatváranie issues

6.1.6 Práca s issues

6.1.6.1 Nastavenie dotazníka pre tvorbu novej issue

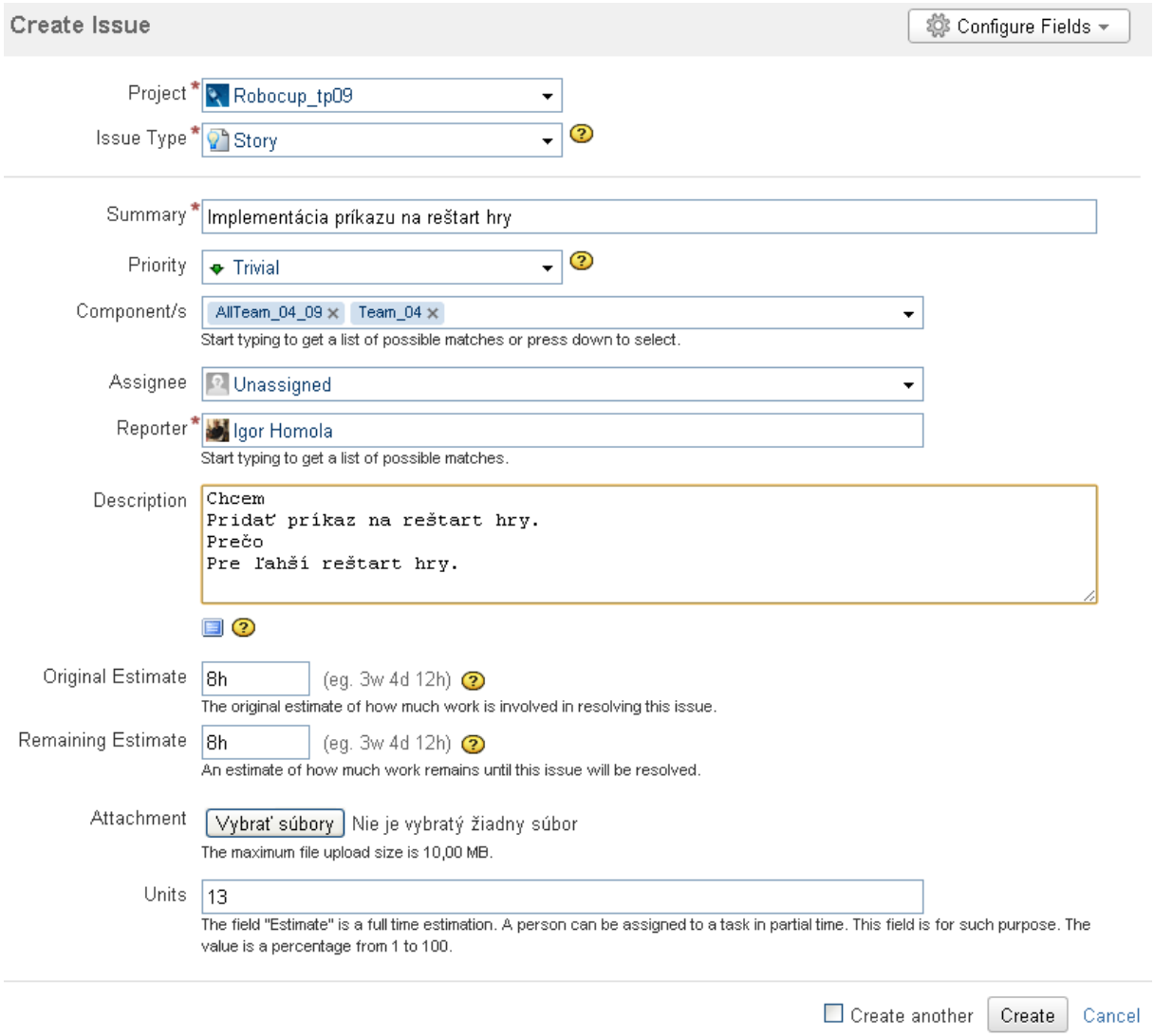
Pred vytváraním issue treba nastaviť v dotazníku vybrané polia. PO kliknutí na tlačidlo ConfigureFields treba vybrať možnosť Custom a zvoliť nasledovné polia pre dotazník podľa obrázku 5.

Obrázok 1 - Polia pre zobrazenie v dotazníku pre tvorbu issue



6.1.6.2 Vytváranie novej issue

Obrázok 2 - vytváranie novej issue



Obsah polí:
Project: Musí byt zvolený projekt Robocup_tp09
Issue Type: Story-základný typ issue vytváraný pri novej požiadavke na vyvíjaný systém

Task– používa sa pri požiadavkách, ktoré priamo nesúvisia s vyvíjaným systémom (napr. Tvorba dokumentácie, úprava webovej stránky ...)
Sub-task – issue využívaná pri potrebe tvorby menších pod úloh, keď je potreba rozdeliť Story prípadne Task na menšie časti.

Vzhľadom na rozsah a účel projektu nepoužívať ostatné tipy issues.
(Bug Epic Improvement New Feature)

Summary: Názov issue , názov musí byť dostatočne konkrétny aby sa z neho dal poznať účel a cieľ issue
Priority: Priorita issue, priorita sa určuje po vzájomnej konzultácii tímu, ak konzultácia neprebehla priorita sa nastavuje na hodnotu Trivial

Component/s: Nastavuje na ktorej tabuli sa issue bude zobrazovať, issue sa musí vždy zobrazovať na spoločnej tabuli AllTeam_04_09, a potom na jednej alebo oboch tabuliach tímu (Team_04 – tabuľa tímu 4, Team_09 – tabuľa tímu 9)
Assignee:Člen týmu ktorému je issue priradená, primárne sa Assignee nastavuje na hodnotu Unassigned (nepriradená)
Reporter: Člen týmu ktorý bude informovaný o zmenách v issue, primárne sa táto hodnota nastavuje na vedúceho tímu.
Description: Podrobnejší popis issue, v prípade typu issuetask a sub-task sa do tohto poľa píše podrobnejší popis úlohy/pod úlohy. V prípade Story sa do poľa vpisuje informácia podľa metodiky scrum pre tvorbu user-story (Chcem – popis požiadavky, Prečo – dôvod požiadavky)
OriginalEstimate: Predpokladaný čas dĺžky riešenia issue *
RemainingEstimate: Zostávajúci čas dĺžky riešenia issue
Attachment: Prílohy potrebné na riešenie issue
Units: Počet bodov pridelený issue (podľa metodiky scrum) po tímovej konzultácii, ak konzultácia neprebehla pole sa necháva nevyplnené

6.1.7 Práca so šprintmi

6.1.7.1 Vytváranie šprintu

1. V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť Plan Work Report).
2. Následne dostávame možnosť vytvoriť nový šprint pomocou tlačidla createsprint.
3. Metodoudrag and drop pridávame issues do šprintu zo zoznamu úloh - Backlogu
4. Začať šprint kliknutím na odkaz StartSprint
5. Zadať názov šprintu, začiatočný a konečný čas šprintu
6. Potvrdiť stlačením tlačidla Start

6.1.7.2 Pridávanie issues počas priebehu šprintu

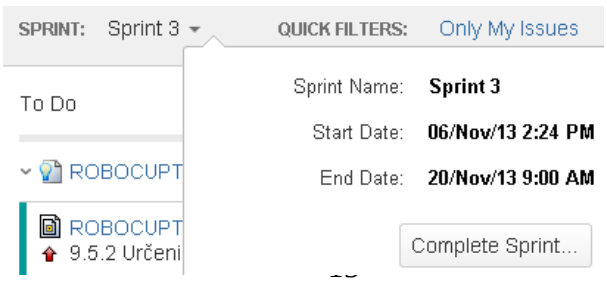
Pridávanie issues je možne v priebehu celého šprintu

1.V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť Plan Work Report).
2.Metodou drag and drop pridávame issues do šprintu.

6.1.7.3 Ukončenie šprintu

1.V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť W Plan Work Report).
2.Kliknutím na prebiehajúci šprint získavame možnosť ukončiť šprint (CompleteSprint)

Obrázok 3 - ukončenie šprintu



6.1.8 Priraďovanie, zaznamenávanie a ukončenie práce

*Formát zadávania času: 1w 3d 4h 5m (w-týždne, d-dni, h-hodiny, m-

Priradiť prácu používateľovi možno dvoma spôsobmi.
1. Nájdením záznamu v záložke Issues
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile
V oboch prípadoch sú možnosti Assign (priradiť) a Assign To Me (priradiť mne).
Každý člen tímu si priraďuje issues sám (Assign To Me (priradiť mne)).
Priraďovať prácu inému členovi tímu môže vedúci tímu, scrummaster, alebo správca systému iba po vzájomnej dohode s daným členom tímu.

6.1.8.2 Zaznamenávanie práce (Log Work)

Zaznamenať prácu možno dvoma spôsobmi.
1. Nájdením záznamu v záložke Issues kliknutím na menu More Actions
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile
Pri zaznamenaní práce je nutné zadať:
TimeSpent: Odpracovanú dobu
DateStarted: Začiatok práce
WorkDescription: Popis práce ktorá bola vykonaná.

6.1.8.3 Ukončenie práce (zatvorenie issue)

Pred ukončením práce je nutné pripojiť k issue súbor s dokumentáciou (Metodika pre tvorbu dokumentácie)
Ukončiť prácu možno dvoma spôsobmi
1.Nájdením záznamu v záložke Issues prácu ukončujeme kliknutím na tlačidlo ResolveIssue.
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile a prenesením issue do záložky Done na polohu ResolveIssue
Pri ukončení práce je nutné zadať:
Resolution: Done (Vzhľadom na rozsah a účel projektu nepoužívať iné možnosti)
TimeSpent: Poslednú odpracovanú dobu v formáte: 1w 3d 4h 5m
DateStarted: Začiatok poslednej práce
Comment: Popis práce ktorá bola vykonaná.

Ukončenú úlohu následne musí akceptovať a zavrieť Reporter (vedúci tímu) pomocou tlačidla CloseIssue.

6.1.9 Práca s tabuľou

Pri práci s JIRA sa využíva tabulaAllTeam_04_09 pre prehľad, a tabuľa konkrétneho tímu (Team_04 ,Team_09) pre vykonávanie potrebných operácii s issues. Tieto tabule sú typu scrumboard a zobrazujú sa na nej iba issues z konkrétneho šprintu.

Issue pridelené konkrétnemu členovi tímu, si daný člen sám prenáša do záložiek na tabuli tímu, podľa toho v ktorej fáze práce sa konkrétna issue nachádza To Do (práca sa nezačala), In Progress(na issue sa pracuje) Done(práca skončená).

Zoznam všetkých issues(Backlog) možno získať v záložke Issues použitím vhodného Query dotazu.

6.2 Manažment podpory vývoja

6.2.1 Úvod

Perconik je nástroj na pridávanie rôznych značiek do zdrojového kódu programu, ktorými dokáže zabezpečiť väčšiu prehľadnosť o tom ktorú časť kódu treba modifikovať, pridať či odstrániť.

6.2.2 Pojmy

Eclipse - je vývojárske prostredie ktoré slúži najmä na programovanie v jazyku java a taktiež v ňom používame plugin PerConIK

Značka - komentár špecifického charakteru ktorý označuje určitú časť kódu.

6.2.3 Proces pridávania značiek

Pre správne vytvorenie značky je potrebné aby ste si najprv označili kód ktorý chcete označiť, následne pravým kliknutím zobrazíme lištu z ktorej vyberieme PerConIK tool -> Insert Tag.

Zo zoznamu si vyberieme požadovanú značku ktorú si následne upravíme do potrebného formátu. Opis použitia značiek sa nachádza v podkapitolách 3.1 až 3.6.

Po kliknutí na uloženie sa nám značka z validuje a označí na ľavej lište modrou farbou.

6.2.4 Pridanie značky Todo

Každá nedokončená časť kódu musí byť označená značkou Todo. Značka obsahuje atribút Task do ktorého je vhodné zadať meno úlohy. **Značka musí obsahovať atribútu Priority** ktorá nadobúda hodnoty Low, Normal, High, Urgent, Immediate podľa toho o akú dôležitú úlohu sa jedná. Pomocou atribúty Solver je možné prideliť úlohu jednému riešiteľovi. **Todo sa musí používať ako párová značka.**

Príklad (Správne)

```
....
//Todo #Task(Implementation) #Solver(sppred@gmail.com) #Priority(low)  |
sppred@gmail.com 2013-11-17T11:07:21.4660000Z
...
//@<Todo | sppred@gmail.com 2013-11-17T11:07:21.4660000Z
```

Príklad (Nesprávne)

```
....
//Todo #Task(Implementation) #Solver(sppred@gmail.com) | sppred@gmail.com 2013-11-
17T11:07:21.4660000Z
...
```

Príklad (Nesprávne) - jedná sa o smell (viď nižšie)

```
//Todo #Priority(low) this is only temporary | sppred@gmail.com 2013-11-
17T10:50:28.3930000Z
...
//@<Todo | sppred@gmail.com 2013-11-17T10:50:28.3930000Z
```

6.2.5 Pridanie značky Bug

Označuje nejakú časť kódu v ktorej sa nachádza chyba. Obsahuje rovnaké atribúty ako značka Todo. **Musí obsahovať atribútu Priority**. **Musí sa používať ako párová značka.** Špeciálny prípad značky Todo.

Príklad (Správne)

```
//Bug #Task(returns the wrong value) #Solver(sppred@gmail.com) #Priority(High) |
sppred@gmail.com 2013-11-17T11:40:20.1280000Z
....
//@<Bug | sppred@gmail.com 2013-11-17T11:40:20.1280000Z
```

Príklad (Nesprávne)

```
...
//Bug #Task(returns the wrong value) #Solver(sppred@gmail.com) #Priority(High) |
sppred@gmail.com 2013-11-17T11:40:20.1280000Z
```

6.2.6 Pridanie značky Ranking

Značka ktorá slúži na ohodnotenie zdrojového kódu čomu zodpovedajú aj atribúty tejto značky ktoré nadobúdajú hodnoty Low, Normal, High. Atribút Comprehensibility hodnotí ako ľahko, ťažko sa dá kód pochopiť. Na druhej strane atribút Testability hodnotí kód z pohľadu testovateľnosti. **Každá značka Ranking musí obsahovať aspoň jeden atribút a musí byť použitá ako párová značka.**

Príklad(Správne)

```
//Ranking #Comprehensibility(High) #Testability(High) | sppred@gmail.com 2013-11-
17T12:17:14.1690000Z
...
//@<Ranking | sppred@gmail.com 2013-11-17T12:17:14.1690000Z
```

Príklad(Nesprávne)

```
//Ranking | sppred@gmail.com 2013-11-17T12:17:14.1690000Z
...
```

6.2.7 Pridanie značky CodeReview

Označený kód značkou CodeReview neprešiel posudzovaním pretože porušil nejakú z definovaných konvencií**.** Porušená konvencia sa zadáva do atribútu Violation. **Značka musí obsahovať aspoň jeden atribút Vilation. CodeReview musí byť použitý ako párová značka.**

Príklad(Správny) - Variable Naming Conventions

```
//CodeReview #Violation(VNC) | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
...
//@< | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
```

Príklad (Nesprávny)

```
//CodeReview | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
```

```
...
//@<CodeReview | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
```

6.2.8 Pridanie značky Recommendation

Značka Recommondation predstavuje odporúčania ktoré by boli vhodné aplikovať na časť kódu. Jediným atribútom tejto značky je Type. **Každá takáto značka musí obsahovať aspoň jeden takýto atribút. Za každým atribútom sa musí nachádzať text ktorý bližšie špecifikuje daný problém.** Atribút Type nadobúda hodnoty :

- Efficiency - Odporúčanie na zvýšenie efektívnosti kódu.
- Technology - Odporúčanie na použitie technológie ktorá sa ma použiť v danom kóde.
- Reuse - Odporúčanie na to aby bola metóda znovu používala v kóde.
- Other - Ostatné odporúčania.

Príklad (správny)

```
//Recommendation #Type(Reuse) You can call it in method methodName|
sppred@gmail.com 2013-11-17T15:40:56.2240000Z
...
//@<Recommendation | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
```

Príklad (nesprávny)

```
//Recommendation #Type(Reuse) | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
...
//@<Recommendation | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
```

6.2.9 Pridanie značky Smell

Každá časť kódu označená touto značkou obsahuje takzvaných pach. **Každá značka musí obsahovať atribút SmellType** za ktorým sa môže ale nemusí nachádzať text ktorý ho bližšie špecifikuje. Dôležitá je tiež nasledujúca Tabuľka hodnôt[1] ktoré môže nadobúdať atribút SmellType:

Hodnota	Popis	Hodnota	Popis
DupCode	duplikovaný kód	LazyClass	Trieda s malou funkcionalitou
LongMethod	dlhá metóda	SpecGener	Špekulatívna generalizácia
LargeClass	veľká trieda	TempField	Dočasné pole
LongParamList	dlhý zoznam parametrov metódy	MesgChains	Reťazenie volaní
DiverChange	Časté zmeny v kóde	MiddleMan	Nevhodné použitie sprostredkovateľa
ShotSurgery	Veľa malých zmien z jednej požiadavky	InappropIntim	Nadmerné využívanie privátnych polí inej triedy
FeatureEnvy	veľa volaní metódy inej triedy	AlterClasses	Exitencia triedy s alternatívnou funkcionalitou, ale rozdielnym rozhraním

DataClumps	Zhluky dát	IncomLibClass	Nekompletná trieda knižnice
PrimitObses	Nevhodné používanie základných typov	DataClass	Dátová trieda
SwitchStat	Nevhodné použitie vetvenia	RefBequest	Dedenie nepotrebných dátových polí
ParInherHier	Paralelne hierarchie dedenia	Comments	Používanie komentárov namiesto samovysvetľujúceho kódu.

Príklad(správne)

```
//Smell #SmellType(DupCode) | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
...
//@< | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
```

Príklad(Nesprávne)

```
//Smell | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
...
//@<Smell | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
```

6.3 Manažment podpory vývoja

6.3.1 Úvod

Táto metodika popisuje prácu s distribuovaným verizonovacím systémom GIT s cieľom zabezpečiť jednoduchší vývoj a uchovávanie zdrojového kódu. Metodikou by sa mali riadiť programátori.

6.3.2 Súvisiace dokumenty

- Pro Git

  http://git-scm.com/book
- Sémantické verzionovanie

  http://semver.org/

6.3.3 Slovník pojmov

Branch - vývojová vetva

Merge - spojenie vývojových vetiev

6.3.4 Postupy

Vytváranie commit-u

Scenár:

- Ukončenie vývoja na časti funkcionality
- Zdieľanie kódu s iným vývojárom

Postup:

1. Zistenie zmien v aktuálnej lokálnej branch-i.

   Zmeny, ktoré sme vykonali v aktuálnej branch-i a nie sú v stage-i zistíme pomocou príkazu:

```
git status
```

2. Pridanie súborov do stage-u
   a. určitý súbor alebo všetky súbory v danom adresári

```
git add <cesta k súboru/adresáru>
```

   b. všetky súbory

```
git add .
```

3. Vytvorenie commit-u
   a. Veľká zmena: použijeme príkaz

```
git commit
```

Popis commit-u vztvárame pomocou textového editoru.

```
1.    <task z Jira / časť projektu> - <jednoduchý popis>
2.
3-x.  <podrobný popis commit-u>
```

b. Malá zmena (dá sa popísať kratšou správou)

```
git commit –m "<task z Jira / časť projektu> - <jednoduchý
popis>"
```

4. Odoslanie commit-u na server

   Stiahneme aktuálne zmeny

```
git pull
```

Ak sú konflikty, vyriešime konflikty.
Odošleme commit do vzdialeného repozitára

```
git push origin <názov branch-e>
```

Zmena aktuálneho branch-u

Scenár:

- Práca na inej feature.
- Bugfix
- Vykonanie zmien v zlej branch-i

Postup:

1. Uvedenie aktuálnej branch-e do konzistentného stavu
   a. Commit-neme aktuálne zmeny podľa postupu 1
   b. Revert-neme zmeny

```
git reset --hard HEAD
```

   c. Zmeny uložíme do stash-u (ak chceme zmeny preniesť)

```
git stash
```

2. Zmena branch-e

   Aktualizácia lokálneho repozitára

```
git pull
```

Zmena aktuálnej branch-e

```
git checkout <názov branch-e>
```

3. Aplikovanie zmien z predošlého repozitára

   (prípad: vykonanie zmien v zlej branch-i)

```
git stash pop
```

Vytváranie branch-ov

Scenár:

- Vytvárame novú feature

  (nový branch vytvárame z development branch-e)
- Rozsiahlejší bugfix

  (nový brnach vytvárame z master branch-e)

Postup:

1. Príprava východzej branch-e

   Zmeníme branch na master / development podľa postupu 2.
2. Vytvorenie novej branch-e

   Názov branch-e bude vo formáte:

```
<Jira-story/task>-<meno zodpovedného>-<popis feature/bug>
Príklad:
TASK-1-Mrkvicka-vylepsenie_kopu_do_lopty
```

Vytváranie tagov

Vytváranie tagov nám umožňuje rozlišovať rôzne verzie aplikácie.

Používa sa sémantické verzionovanie.

Verzia sa zapisuje: <*verzia API*>.<*feature*>.<*fix*>.

Scenár:

- Označenie verzie aplikácie

Postup:

1. Vytvorenie tagu
   a. Aktuálny commit

```
git tag -a <verzia> -m '<správa>'
```

   b. Určitý commit

```
git tag -a <verzia> -m '<správa>' <číslo commit-u>
```

Merge-ovanie branch-í

Scenár:

- Ukončenie vývoja určitej feature
- Ukončenie bugfix-u

Postup:

1. Uvedenie aktuálnej branch-e do konzistentného stavu (postup 2 krok 1)
2. Merge nadradenej branch-e do feature branch-e

   Dôvod je, že v prípade konfliktu riešime konflikt v branch-i s nižšou prioritou.

   ```
   git merge origin/<nadradená branch>
   ```
3. Odoslanie aktuálnej branch-e do vzdialeného repozitára

   ```
   git push origin <feature branch>
   ```
4. Zmeníme aktuálnu branch na nadradenú branch

   ```
   git checkout <nadradená branch>
   ```
5. Merge lokálnej feature branch-e do nadradenej branch-e

   ```
   git merge <feature branch>
   ```
6. Odoslanie aktuálnej branch-e do vzdialeného repozitára

   ```
   git push origin <nadradená branch>
   ```

6.4 Manažment kvality

6.4.1 Úvod

Nasledujúci dokument obsahuje metodiky pre písanie zdrojových kódov s dôrazom na prehľadnosť, čitateľnosť a zrozumiteľnosť. Ďalej obsahuje metodiky pre údržbu kódu a jeho vhodnú dokumentáciu.

6.4.2 Pre koho je metodika určená

Metodika je určená primárne pre členov, tímu, ktorí upravujú kód, teda pre programátorov. Taktiež je určená pre manažéra kvality, ktorý je zodpovedný za celkový stav kódu.

6.4.3 Slovník pojmov

CamelCase – štýl písania, kedy sa slovné spojenia píšu bez medzery, pričom každé nové slovo začína veľkým písmenom. Úvodné písmeno môže byť bu veľké, alebo malé.

6.4.4 Obsah metodiky

Ako už bolo vyššie spomenuté, metodika pozostáva z viacerých častí. Prehľad celých názvov jednotlivých častí je uvedený v tabuľke.

Tabuľka 4 - názvy častí

Názov postupu
Pomenovávanie tried, metód a premenných
Používanie zátvoriek
Zalamovanie kódu
Dokumentovanie pomocou JavaDocu

6.4.5 Postupy pri písaní kódu

6.4.5.1 Pomenovávanie tried, metód a premenných

6.4.5.1.1 Triedy

Všetky triedy musia by pomenované špecifickým spôsobom. Pri písaní názvu triedy je nutné použiť CamelCase. V prípade tried je vhodné, aby aj úvodné písmenko bolo veľké. Názov triedy musí byť v anglickom jazyku a musí vyjadrovať podstatné meno, najlepšie nejakú entitu z reálneho života, ktorú trieda reprezentuje.

Príklad:

```
 public class FastCar {

       /*

        * implementácia triedy

        */

}
```

Nesprávne použitie:

```
public class Fast_Car {

       /*

        * implementácia triedy

        */

}


public class fastCar {

       /*

        * implementácia triedy

        */

}
```

6.4.5.1.2 Rozhrania

Rovnako, ako pri názvoch tried, aj pri názvoch rozhraní sa musí použiť CamelCase s veľkým začiatočným písmenom. Narozdiel od názvu triedy, názov rozhrania musí opisovať vlastnosť, ktorú dané rozhranie vyjadruje. Názov rozhrania teda bude odvodený od anglického názvu pre vlastnosť, ktorá je v trpnom príčastí. Pre ilustráciu je vhodné uviesť rozhranie **Runnable,** ktoré poskytuje možnosť spustenia triedy, ktorá ho implementuje, v

samostatnom vlákne. V prípade, že by sa nedal jednoznačne vymedziť výraz pre konkrétnu vlastnosť, rozhrania by malo byť pomenované s prefixom „I“, za ktorým bude nasledovať podstatné meno entity z reálneho života.

Príklad:

```
public interface  Movable {

	/*

	 * implementácia rozhrania

	 */

}


public interface ICar {

	/*

	 * implementácia rozhrania

	 */

}
```

Nesprávne použitie:

```
public interface CarInterface {

	/*

	 * implementácia rozhrania

	 */

}
```

6.4.5.1.3 Metódy

Opäť, pri názve metódy musí byť požitý CamelCase. Oproti názvom tried a rozhraní však názov metódy bude začínať malým písmenom. Ako názov metódy bude použitý anglický výraz pre sloveso, prípadne viacero slovies, ktoré opisujú, čo daná metóda robí.

Príklad:

```
public int countTotalTravelTime() {

	/*

	 * logika metódy

	 */

}


public boolean isMoving() {

	/*

	 * logika metódy

	 */

}
```

Nesprávne použitie:

```
public int travelTime() {

	/*

	 * logika metódy

	 */

}
```

6.4.5.1.4 Premenné

Premenné sa dajú rozdeliť na dve skupiny – statické premenné a inštančné premenné. Obe skupiny sa musia nazýva podľa toho, na čo sa používajú. Takisto musia byť pomenované v anglickom jazyku. Rozdiel pri pomenovávaní je, že inštančné premenné musia využívať CamelCase a mená statických premenných sa musia písať veľkými písmenami, pričom viacslovné premenné majú jednotlivé slova oddelené podtŕžnikom.

Príklad:

```
public class FastCar {

        //konštanta

        public static final int NUMBER_OF_SEATS;



        //inštančná premenná

        private String carModel;

}
```

Nesprávne použitie:

```
public class FastCar {

       //konštanta

        public static final int number_of_seats;



        //instančná premenná

        private String CarModel;

}
```

6.4.5.2 Používanie zátvoriek

Každý separátna čas kódu (trieda, metóda, riadiace cykly, podmienky) musí byť správnym spôsobom ozátvorkovaná. Otváracia zátvorka sa musí nachádzať na rovnakom riadku, na akom je napísaný daný výraz, ktorý jej predchádza. Táto notácia je zvolená kvôli väčšej prehľadnosti a taktiež úspore miesta, kedy je ušetrený jeden riadok, oproti notácii kde sa otváracia zátvorka píše až na nový riadok.

Príklad:

```
public class Car {

     public static void main(String[] args) {

           int  i = 0;

           if (i == 0) {

                  //logika

           } else {

                 //logika

           }

     }

}
```

Nesprávne použitie:

```
public class Car

{

     public static void main(Strring[] args)

    {

            int  i = 0;

            if (i == 0)

            {

                   //logika

            }

            else

            {

                  //logika

            }

     }

}
```

6.4.5.3 Zalamovanie kódu

Vytvorený kód musí byť správne zalomený do jednotlivých riadkov. Je potrebné rozlišovať dva prípady – metóda nám vráti rovnaký objekt, alebo nám metóda vráti iný objekt.

6.4.5.3.1 Rovnaký objekt

Pri tomto type objektu musí byť každá metóda, ktorá sa pri zreťazení metód volá, zalomená vždy v novom riadku. Takýmto spôsobom je jednoduché rozlíšiť, ktorý objekt sa danými volaniami metód mení. Dobrým príkladom na tento stav sú triedy, ktoré mapujú reálne entity a teda neobsahujú žiadnu výpočtovú logiku, ale len zapúzdrujú dáta.

Príklad:

```
CarEntity car = new CarEntity();

car.setModel( “Volvo “)

   .setMaxSpeed(250)

   .setNumberOfSeats(5);
```

Nesprávne použitie:

```
CarEntity car = new CarEntity();

car.setModel( “Volvo “).setMaxSpeed(250).setNumberOfSeats(5);
```

6.4.5.3.2 Iný objekt

V tomto prípade je nutné, aby bola pri zreťazení metód nasledujúca metóda zalomená do nového riadku len v prípade, že by dĺžka riadku presiahla maximálnu dĺžku riadku.

Príklad:

```
CarRepository repository = new CarRepository();

String model = repository.findCarByMaxSpeed(150).getModel();
```

6.4.5.4 Dokumentovanie pomocou JavaDoc-u

Je nutné, aby bol kód prehľadne a dobre zdokumentovaný. Prostriedkom na to sú komentáre, pričom programovací jazyk Java nám poskytuje podporu dokumentácie pomocou vytvárania komentárov so štandardnou štruktúrou – JavaDoc.

Zdrojový kód musí obsahovať dva typy komentárov – komentáre pre triedy a komentáre pre jednotlivé metódy. Všetky komentáre musia by napísané v anglickom jazyku.

Štandardná hlavička komentáru pre triedu musí obsahovať opis danej triedy, v ktorom bude zhrnuté, na čo daná trieda slúži. Keďže Robocup je projekt, na ktorom do tejto doby pracovala viacero tímov, pričom každý z týchto tímov strávil na projekte určitý čas. Z tohto dôvodu je povinným poľom v štandardnej hlavičke pole @author, kde sa bude nachádza informácia o autorovi zdrojového kódu triedy, respektíve informácia o členovi tímu, ktorý je aktuálne zodpovedný za daný zdrojový kód. Jednotlivé mená sa budú písať za sebou do riadku, pričom budú oddelené čiarkou. Jednotlivé mená budú vo formáte meno, priezvisko oddelené medzerou. Je zakázané zasahovať do mien, ktoré sú už uvedené v tomto poli.

Príklad:

```
/**
 *
 * Class, that represents car and encapsulates data
 *
 * @author Jožko Mrkvička, Matej Bádal
 */
public class Car {
    //implementácia triedy
}
```

Štandardná hlavička pre metódu je pomerne odlišná. Hlavička taktiež musí obsahovať krátky opis. Ďalej musí obsahovať pole @param, kde sa bude nachádza opis konkrétneho vstupu pre metódu. Samozrejme za predpokladu, že metóda nejaký vstup vyžaduje. V neposlednom rade musí hlavička obsahovať pole @return, kde bude opísaný typ návratovej hodnoty, ktorú metóda vracia.

Príklad:

```
public class Travel {

	private long timeOfStart;

	private long timeOfArrival;


	/**
	 *
	 * returns remaining time to arrival to destination
	 * @return remaining time in miliseconds
	 */
	 public long getRemainingTravelTime() {
		return this.timeOfArrival - this.timeOfStart;
	 }
}
```

6.5 Manažment riadenia

6.5.1 Úvod dokumentu

Tento dokument popisuje riadenie a priebeh tímových stretnutí. Opísaný postup je odvodený od scrum modelu pre riadenie vývoju softvérového projektu.

6.5.2 Slovník pojmov

Scrum

Inkrementálna a iteratívna metodika pre agilný vývoj softvérového projektu

Šprint

Základná časová jednotka metodiky scrum. Šprint je časovo ohraničené obdobie, vopredstanovenej dĺžky, v ktorom sú vypracovávané úlohy. Dĺžka šprintu je typicky 2 týždne.

ScrumMaster

Člen tímu zodpovedný za správne fungovanie tímu

ProductOwner

Reprezentuje zákazníka, definuje požiadavky na produkt

Retrospektíva

Obhliadnutie sa za práve dokončeným šprintom. Členovia tímu sa vyjadrujú k tomu čo bolo v danom šprinte dobré a čo treba zlepšiť

BurndownChart

Graf, ktorý vyjadruje čas práce strávený na projekte a čas práce, ktorá ostáva na dokončenie v rámci

šprintu

Backlog

Zoznam úloh, ktoré je nutné spraviť pre dokončenie projektu

Review

Prehľad dokončenej a nedokončenej práce v rámci šprintu

6.5.3 Tímové stretnutia

Tímové stretnutia delíme na formálne a neformálne. Neformálne stretnutia sa konajú výnimočne ak sa členovia tímu potrebujú iba na niečom dohodnúť. Tieto stretnutia majú voľný priebeh kde nie je nutnosť zaznamenávať žiadne dokumenty o priebehu stretnutia. Táto metodika sa ďalej venuje už iba formálnym stretnutiam, ktorých priebeh ma presné postupy. V rámci jedného šprintu sa konajú tri formálne stretnutia:

- Tímové stretnutie na začiatku šprintu
- Tímové stretnutia v priebehu šprintu
- Tímové stretnutie na konci šprintu

Spoločným prvkom všetkých formálnych stretnutí je tvorba zápisu. Zápis tvorí vždy jeden člen týmu, ktorého určí vedúci tímu. V nasledujúcich kapitolách sú jednotlivo popísané stretnutia. Všetky veci, ktoré sú na stretnutiach stanú alebo dohodnú musia byť evidované v zápise, preto ďalej zápis už nie je spomínaný pri jednotlivých procesoch.

6.5.3.1 Tímové stretnutie na začiatku šprintu

6.5.3.1.1 Popis stretnutia

Tento typ stretnutia sa koná na začiatku každého šprintu. Tento proces definuje spôsob, akým sú plánované úlohy na ďalší šprint. Zaoberá sa výberom úloh, hodnotením náročnosti a rozdeľovaním úloh medzi členov týmu.

	Krok	Kapitola
1.	Vyberanie úloh z backlog-u	3.2
2.	Hodnotenie náročnosti úloh – Planning poker cards	3.3
3.	Rozdelenie úloh medzi členov týmu	3.4

6.5.3.1.2 Vyberanie úloh z backlog-u

Predpokladom pre tento proces je mať v backlog-u zoznam všetkých úloh, ktoré je potrebné spraviť pre dokončenie projektu. Výber úloh má na starosti ProductOwner.

Proces:

1. ProductOwner vyberie úlohy pre aktuálny šprint

2. ProductOwner určí priority úloh:

- Minor
- Major
- Critical

6.5.3.1.3 Hodnotenie náročnosti úloh –PlanningPoker Cards

V tomto procese členovia tímu určujú náročnosť úloh, ktoré vybral ProductOwner, pomocou Planning Poker Cards. Hodnoty týchto úloh sa potom pripisujú ako body členom tímu, ktorí danú úlohu vypracujú.

Proces:

1. Každý člen si pripraví karty (buď papierové alebo môže použiť mobilnú aplikáciu)

2. ScrumMaster určí, ktorá úloha sa ide hodnotiť

3. Každý člen podľa seba ohodnotí kartou náročnosť úlohy (čím vyššie číslo tým náročnejšia úloha)

4. Na výzvu ScrumMaster-a všetci ukážu hodnotu svojich kariet

5. Členovia tímu, ktorí majú najnižšie a najvyššie číslo musia svoje hodnotenie zdôvodniť

6. Opakujú sa kroky 3,4 a 5 až kým sa všetci členovia tímu nezhodnú na rovnakom čísle

6.5.3.1.4 Rozdelenie úloh medzi členov tímu

Riadenie rozdeľovania úloh medzi členov tímu má na starosti ScrumMaster. Konkrétne úlohy, na ktorých chcú jednotliví členovia pracovať, si však vyberajú sami.

Proces:

1. ScrumMaster určí ktorá úloha sa ide prideľovať

2. Členovia tímu hovoria jeden po druhom či majú o úlohu záujem alebo nie

3. Prideľovanie danej úlohy končí keď jeden z členov úlohu príjme

4. Ak o úlohu nemá záujem nikto, vedúci tímu určí člena, ktorý túto úlohu vypracuje

5. Kroky 1,2,3 a 4 sa opakujú až pokým nie sú všetky úlohy pridelené

Dodatok: Každý člen tímu musí mať pridelenú aspoň jednu úlohu na každý šprint aby nenastala situácia, že mu na konci šprintu bude udelených 0 bodov, čo by znamenalo opustenie tímu a nespravenie predmetu Tímový projekt I.

6.5.3.2 Tímové stretnutie v priebehu šprintu

6.5.3.2.1 4.1 Popis stretnutia

Tento typ stretnutia sa koná vždy v polovici šprintu teda týždeň po začiatočnom stretnutí. Hlavnou náplňou stretnutia je sledovanie pokroku a zistenie čo robí najväčšie problémy pri riešení.

	Krok	Kapitola
1.	Sledovanie progresu prác	4.2
2.	Problémy pri práci	4.3
3.	Preloženie úlohy na iného člena tímu	4.4

6.5.3.2.2 Sledovanie progresu prác

Cieľom sledovania progresu prác je aby ProductOwner zistil ako sa vývoj produktu posunul. Tu sa tiež dozvie kto na akej úlohe pracuje a s akou úspešnosťou.

Proces:

1. ScrumMaster odovzdá vytlačený BurndownChart spolu so zoznamom úloh na daný šprint ProductOwner-ovi

2. Každý člen tímu prezentuje slovne na čom týždeň pracoval a aký má plán na ďalší týždeň

3. Na základe prezentovaných výsledkov ProductOwner môže spresniť zadanie úlohy

6.5.3.2.3 Problémy pri práci

Pri prezentácii doposiaľ odvedenej práce sa venuje čas aj na rozobratie problémov pri riešení. Úlohou tohto sedenia je predísť situácii, že na konci šprintu nebudú splnené všetky úlohy. Tu nastáva aj možnosť preloženia úlohy z jedného člena na druhého.

Proces:

1. ScrumMaster vyzýva člena tímu na diskusiu o problémoch, ktoré nastali pri vývoji

2. Člen tímu prezentuje problémy

3. ProductOwner spresní konkrétne požiadavky

4. Kroky 1,2 a 3 sa opakujú pre všetkých členov, kým nie sú odstránené všetky problémy

6.5.3.2.4 Preloženie úlohy na iného člena tímu

Počas šprintu môže nastať situácia kedy člen tímu nie je schopný danú úlohu vypracovať. V tomto prípade je možné preložiť úlohu z jedného člena tímu na iného.

Proces:

1. Člen tímu uvedie dôvod prečo nie je schopný úlohu dokončiť

2. Vedúci tímu vyzýva ponúkne úlohu ostaným členom týmu

3. Ak o úlohu nie je záujem pridelí ho členovi, ktorý pracuje na úlohách s najnižším ohodnotením

4. Body za prenášanú úlohu sa presunú z člena, ktorý sa vzdal úlohy na člena, ktorý ju prevzal

6.5.3.3 Tímové stretnutie na konci šprintu

6.5.3.3.1 Popis stretnutia

Toto stretnutie prebieha vždy na konci šprintu čiže po dvoch týždňoch. V tomto type tímového stretnutia sa vyhodnocuje, či sa úlohy podarilo dokončiť, prideľujú sa body členom týmu a hodnotí sa spätne priebeh celého šprintu.

	Krok	Kapitola
1.	Review - BurndownChart	5.2
3.	Prideľovanie bodov členom tímu	5.3
4.	Review a Retrospektíva	5.4

6.5.3.3.2 Review – BurndownChart

Proces kontroly splnenia úloh na konci šprintu pomocou BurndownChart-u je podobný ako v priebehu. Tu už by však všetky úlohy mali byť splnené a kždý člen týmu musí prezentovať svoju dosiahnutú robotu.

Proces:

1. ScrumMaster odovzdá vytlačený BurndownChart spolu so zoznamom úloh na daný šprint ProductOwner-ovi

2. Každý člen tímu prezentuje výsledok svojej práce

3. Typy prác a požiadavky pre úspešné splnenie:

- Analýza

    - musí odovzdať dokument s vypracovanou analýzou
    - prezentovať dôležité prvky použiteľné pre náš projekt
- Kódovanie
    - musí prezentovať kód s popisom
    - ukázať funkcionalitu
    - odovzdať dokumentáciu
- Ostatné úlohy
    - každá úloha musí byť popísaná v dokumente
    - odovzdanie dokumentu s popisom

6.5.3.3.3 Prideľovanie bodov členom týmu

Na základe predvedených výsledkov prideľuje ProductOwner body jednotlivým členom. Za splnenú úlohu získava počet bodov, ktorými bola úloha ohodnotená pomocou Planning Poker Cards. Ak na úlohe pracovalo viac členov, body sa rovnomerne rozdelia.

Proces:

1. ProductOwner pridelí body každému členovi týmu

2. ProductOwner zapíše body do systému kde sa každému členovi kumulujú

6.5.3.3.4 Retrospektíva

V rámci retrospektívy sa riešia otázky ohľadom riadenia prebehnutého šprintu.

Proces:

1. ScrumMaster vyzýva členov týmu aby prezentovali svoje postrehy k riadeniu

2. Každý člen prezentuje čo v šprinte prebiehalo dobre, čo treba zlepšiť a čo by zrušil

6.6 Manažment testovania

6.6.1 Úvod

Predmetom metodiky dolnej úrovne je spôsob vytvorenia a spustenia testovacej triedy za pomoci frameworku JUnit v prostredí Eclipse a programovacom jazyku Java.

6.6.2 Dedikácia metodiky

Metodika je primárne určená pre tím č. 4 na predmete Tímový Projekt I. a II. v akademickom roku 2013/2014, ale taktiež aj pre programátorov (testerov), využívajúcich programovací jazyk Java a vývojové prostredie Eclipse.

6.6.3 Použité pojmy

Tabuľka 5 - použité pojmy a ich vysvetlenie

#	Pojem	Vysvetlenie
1	jednotkový test	nízkoúrovňový test, zameraný na testovanie funkčných jednotiek (zväčša tried) zdrojového kódu
2	JUnit	framework pre testovanie zdrojových kódov písaných v jazyku Java
3	framework	programový balík nástrojov pre riešenie komplexných úloh
4	Eclipse	vývojové prostredie v ktorom využijeme framework Junit.
5	Testovacia trieda	trieda vykonávajúca testy nad testovanou triedou
6	Testovaná trieda	trieda, ktorá je testovaná testovacou triedou
7	TestCase	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda vykonávajúca test(y) nad testovanou triedou
8	TestSuite	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda zoskupujúca množinu testovacích tried

6.6.4 Vytvorenie jednotkového testu v prostredí JUnit

V tejto časti metodiky sú pokryté procesy, ktoré sú znázornené v tabuľke č. 5.

Tabuľka 6 - pokryté procesy

#	Názov procesu	Kapitola
1	Proces prípravy testu	5
2	Proces spustenia a vyhodnotenia testu	6

6.6.5 Proces prípravy testu

Tabuľka 7 - jednotlivé kroky v procese prípravy testu

#	Krok	Kapitola
1	Vytvorenie zdrojového priečinka a testovacieho balíka	5.1
2	Vytvorenie testovacej triedy	5.2
3	Zápis anotácie v testovacej triede	5.3
4	Vytvorenie testovacej metódy	5.4
5	Vytvorenie vhodných komentárov	5.5

6.6.6 Vytvorenie zdrojového priečinka a testovacieho balíka

Kvôli väčšej prehľadnosti testovacích tried, ktoré testujú správnosť zdrojového kódu, je nutné vytvoriť nový zdrojový priečinok a v ňom „testovací“ balík, kde sa budú triedy zoskupovať.

6.6.6.1 Vytvorenie zdrojového priečinka

V okne *Package Explorer* kliknúť pravým tlačidlom myši na projekt, v ktorom sa má testovať a v sekcii *New* zvoliť možnosť *Source Folder*. V novootvorenom okne do textového poľa *Folder Name* vložiť názov *test*.

6.6.6.2 Vytvorenie testovacieho balíka

Na novovytvorený *Source Folder* kliknúť pravým tlačidlom myši a v sekcii *New* zvoliť možnosť *Package*. V novootvorenom okne do textového poľa *Name* vložiť názov taký istý ako má balík, ktorý je testovaný.

6.6.7 Vytvorenie testovacej triedy

Každej testovanej triede prislúcha práve jedna testovacia trieda, ktorá overuje jej správnosť.

6.6.7.1 Určenie rozsahu testovania

Samotný rozsah testu je definovaný podľa scenárov testovania a tým pádom sa testuje buď trieda alebo cely balík.

6.6.8 Testovanie triedy

Kliknúť pravým tlačidlom myši na balík, v ktorom sa nachádza trieda na otestovanie a v sekcii *New* zvoliť možnosť *JUnit Test Case*. V novootvorenom okne do textového poľa *Name* vložiť názov testovacej triedy vo formáte *<názov testovanej triedy>Test* . Pokiaľ je vložený názov testovacej triedy v správnom formáte, je nutné stlačiť na tlačidlo *Finish*, po ktorom sa vytvorí testovacia trieda (dedí od triedy TestCase).

6.6.9 Testovanie balíka

Kliknúť pravým tlačidlom myši na balík, ktorý má byť otestovaný a v sekcii *New* zvoliť možnosť *JUnit Test Suite*. V novootvorenom okne do textového poľa *Name* vložiť názov testovacej triedy vo formáte *PackageTest*. Pokiaľ je vložený názov testovacej triedy v správnom formáte, je nutné stlačiť na tlačidlo *Finish*, po ktorom sa vytvorí súhrn testovacích tried (dedí od triedy TestSuite).

6.6.10 Zápis anotácie v testovacej triede

V testovacej triede slúži anotácia na popis autora, verzie a informácií o teste. Ešte pred deklaráciou samotnej testovacej triedy sa musí vytvoriť anotačný komentár.

6.6.10.1 Meno autora a názov tímu

Meno autora testu spolu s názvom tímu, v ktorom pôsobí sa do zdrojového kódu vloží vo formáte:

@author: <meno autora testu>/<názov tímu v ktorom autor pôsobí>

6.6.10.2 Verzia kódu

Verziovanie testovacích tried je určené troma číslicami oddelených bodkou. Do zdrojového kódu sa zapisuje vo formáte:

@version: <prvé číslo>.<druhé číslo>.<tretie číslo>

- prvé číslo – sa inkrementuje, keď sa spravia nekompatibilné API zmeny
- druhé číslo – sa inkrementuje, keď sa pridá funkcia, ktorá je spätne kompatibilná
- tretie číslo – sa inkrementuje pri opravení chýb, ktoré sú spätne kompatibilné

6.6.10.3 Opis testu

V priebehu písania tela testu sa opierať o konvenciu komentovania zdrojového kódu [1].

6.6.11 Vytvorenie testovacej metódy

Každá testovacia metóda prislúcha práve jednej testovanej metóde a overuje jej správnosť na viacerých odlišných vstupoch.

- Nad deklaráciou novej metódy vložiť anotáčný komentár podľa zásad JUnit [2].
- Vytvoriť testovaciu metódu s deklaráciou public void *Test<názov testovanej metódy>*
- Napísať telo testovacej metódy podľa konvencie zápisu zdrojového kódu [3].
- Použiť metódy *Assert<typ>* na porovnanie reálneho a požadovaného výstupu. Namiesto *<typ>* je nutné dosadiť funkciu, ktorá je dostupná vo frameworku JUnit [4].

6.6.12 Proces testovania triedy

Tabuľka 8 - jednotlivé kroky v procese testovania triedy

#	Krok	Kapitola
1	Spustenie testu	6.1
2	Vyhodnotenie testu	6.2
3	Zápis výsledkov	6.3

6.6.13 Spustenie testu

V zdrojovom priečinku pre testy, v testovacom balíku kliknúť pravým tlačidlom myši na testovaciu triedu a v sekcii *Run as* zvoliť možnosť *JUnit Test*.

6.6.14 Vyhodnotenie testu

Po spustení JUnit testu sa v okne, v ktorom bol doteraz zobrazený *Package Explorer*, otvorí tab s názvom *JUnit*, v ktorom sa nachádzajú všetky informácie o vykonaných testoch.

Konkrétne sú to informácie o čase vykonania, počte vykonaných testov, počte chybných testov a chýb v zdrojovom kóde. Od predošlých informácií závisí sfarbenie lišty, ktoré môže byť:

- Zelené – všetky testy zbehli bez chýb.
- Bordové – testovacia metóda dostala na výstup rozdielnu hodnotu ako očakávala.
- Červené – testovacia metóda nebola správne naimplementovaná.

Pod lištou sa nachádza zoznam testov, ktoré je možné expandovať a prezerať si jednotlivé testovacie metódy. V prípade, že lišta nie je zelená, sa pri teste a následne pri metódach, ktoré sú chybné zobrazí ikonka „x“. Kliknutím na konkrétnu chybnú metódu sa v okne *Failure Trace* zobrazí výpis o chybe. Pokiaľ sa objaví chyba alebo sa požadovaná hodnota liši od reálnej, je nutné navrhnúť spôsob riešenia týchto problémov.

6.6.15 Zápis výsledkov

Po vyhodnotení testov musí zapisovateľ uskutočniť zápis nasledujúcich poznatkov do zápisnice:

- Chybné testy spolu s informáciami, ktoré sú zobrazené v okne *Failure Trace* (viď. kapitola 6.2 Vyhodnotenie testu).
- Odlišnosť medzi požadovanými a reálnymi hodnotami spolu s návrhom spôsobu riešenia tohto problému a jeho postup.
- Vytvoriť a zapísať štatistiku testovaných modulov alebo balíčkov. Porovnať počet správnych a chybných testov s celkovým počtom testov.

6.6.16 Zoznam nadväzujúcich metodík dokumentov

[1] Konvencia komentovania zdrojového kódu

[3] Konvencia zápisu zdrojového kódu

6.7 Manažment dokumentovania

6.7.1 Dokumentovanie používateľského príbehu

Tento dokument opisuje metodiku pre tvorbu dokumentácie používateľského šprintu. Metodika obsahuje 2 procesy :

- Proces 1 – Tvorba písomnej dokumentácie
- Proces 2 – Dokumentácia pre RoboCup Wiki

6.7.1.1 Pojmy

- Dokumentácia šprintu – dokument, ktorý obsahuje všetky informácie o danom šprinte
- Wiki – portál založený na softvéri MediaWiki

6.7.1.2 Skratky

- PP – používateľský príbeh
- PpID – identifikačné číslo používateľského príbehu

6.7.1.3 Tvorba písomnej dokumentácie

Táto časť obsahuje konvencie pre tvorbu písomnej dokumentácie – konkrétne dokumentácie používateľského príbehu.

6.7.1.3.1 Nástroj na tvorbu písomnej dokumentácie

Na tvorbu dokumentácie sa použije program Microsoft Word 2010 alebo 2013.

6.7.1.3.2 Konvencie

Pre zjednodušenie spájania dokumentácie každého šprintu do jedného konzistentného dokumentu bola vypracovaná šablóna, ktorá obsahuje základné konvencie pre písanie dokumentu.

6.7.1.3.3 Názov dokumentu

Pri odovzdávaní dokumentu na spájanie je potrebné aby bol názov dokumentu nasledovný:

Autor_SprintCislo_ppID_datum

Autor: Meno autora

SprintCislo : Číslo aktuálneho šprintu

ppID : Identifikačné číslo používateľského príbehu

datum: Dátum odovzdania dokumentu

6.7.1.3.4 Štýly

Hlavná požiadavka je dodržanie zadefinovaných štýlov, ktoré sú uvedené v tabuľke 17.

Tabuľka 9 - zoznam zadefinovaných štýlov

Názov štýlu	Použitie
Normálny	Bežný text
Bez riadkovania	Tabuľka
Nadpis1	Nadpis úrovne 1
Nadpis2	Nadpis úrovne 2
Nadpis3	Nadpis úrovne 3
Nadpis4	Nadpis úrovne 4
Nadpis5	Nadpis úrovne 5
Bold	Bežný text – dôležité vyznačenie

6.7.1.3.5 Vkladanie obrázkov

Obrázky vkladať v nasledovných prípadoch:

- Zmena štruktúry projektu – diagramy
- Návrh nových súčastí systému – diagramy
- Analýza – získané obrázky z iných zdrojov
- Špecifické prípady, ktoré si vyžadujú obrázok

6.7.1.3.6 Vkladanie tabuliek

Tabuľky vkladať v nasledovných prípadoch:

- Štruktúrovaný popis nových funkcií
- Prehľad tried vytvorených v používateľskom príbehu
- Zoznam vyskytnutých problémov a návrh ich riešenia
- Špecifické prípady, ktoré si vyžadujú tabuľku

6.7.1.3.7 Štruktúra dokumentu

6.7.1.3.7.1 Úvod

Obsahuje:

- Krátky opis používateľského príbehu
- Zoznam používateľských príbehov, s ktorými je previazaný

6.7.1.3.7.2 Analýza
Obsahuje:

- Podrobná analýza pre daný používateľský príbeh

6.7.1.3.7.3 Návrh
Obsahuje:

- Návrh riešenia – diagramy, obrázky, zmeny
- Opis riešenia – podrobný opis navrhovaného riešenia

Možnosť vynechania: v prípade príbehu, ktorý sa týka iba analýzy

6.7.1.3.7.4 Implementácia
Obsahuje:

- Opis implementácie – podrobný opis implementovaného riešenia
- Tabuľka zmien – zoznam ovplyvnených tried, balíkov

Možnosť vynechania: v prípade príbehu, ktorý sa týka iba analýzy alebo návrhu

6.7.1.3.7.5 Zhodnotenie - výstup
Obsahuje:

- Krátke zhodnotenie používateľského príbehu
- Tabuľka výsledkov – celkový zoznam vykonaných zmien
- Tabuľka so zoznamov nových problémov – opísane objavené problémy

6.7.1.4 RoboCup Wiki

6.7.1.4.1 Konvencie
Pre potreby dokumentovania aktuálneho stavu na jednom mieste bol vypracovaný postup pre pridávanie relevantných častí dokumentácie na RoboCup Wiki.

6.7.1.4.2 Formátovanie

6.7.1.4.2.1 Text
- Písanie do nového riadku – vynechať jeden prázdny riadok

- Vypnutie formátovania wiki – tagy <nowiki>, </nowiki >
  Používať iba v špeciálnych prípadoch, kedy nevyhovuje formátovanie Wiki.

- Zdrojový kód – tagy <code>, </code>
  Používať pre všetky zdrojové kódy vkladané do Wiki.

6.7.1.4.2.2 Úrovne nadpisov

- Úroveň 1 : =Názov úrovne 1=
  Názov kapitoly
- Úroveň 2: ==Názov úrovne 2==
  Názov podkapitoly
- Úroveň 3: ===Názov úrovne 3===
  Názov dôležitej časti podkapitoly

6.7.1.4.2.3 Odkazy

- Externý odkaz – [http://www.ukazka.com názov odkazu]
  Používať pre všetky externé odkazy, ktoré sú zahrnuté v dokumentácii

- Interný odkaz – [[názov podstránky na wiki]]
  Používať na miestach, kde je potrebné sa odkázať na inú podstránku

- Odkaz na súbor - [[Súbor: obrázok.jpg]]

6.7.1.5 Práca s Wiki

6.7.1.5.1 Prihlásenie

- Kliknúť na systémové prihlásenie
- Zadať pridelené prihlasovacie údaje

6.7.1.5.2 Rozhodnutie

- Existujúca dokumentácia a jej úpravy – kapitoly 1.4.3.3 , 1.4.3.4
- Nová dokumentácia – kapitola 1.4.3.5

6.7.1.5.3 Výber kapitoly

- Vybrať správnu kapitolu prislúchajúcu k PP
- Vybrať podstránku pre konkrétny problém

6.7.1.5.4 Editácia stránky

- Kliknúť Upraviť pri konkrétnej časti stránky
- Pridať dokumentované časti
- Editovať pole zhrnutie úprav podľa aktuálneho stavu
- Uložiť

6.7.1.5.5 Vytvorenie novej podstránky

- Zadať novú url - wiki/index.php?title=NazovPodstranky
  Názov podstránky – musí prislúchať k dokumentovaniu používateľského príbehu
- Vybrať z ponuky hornej lišty možnosť vytvoriť
- Vykonať potrebné úpravy na stránke – pridať dokumentáciu
- Vyplniť zhrnutie úprav
- Uložiť

7 Manažment – horné metodiky

7.1 Manažment plánovania

7.1.1 Obsah metodiky

Metodika je určená pre manažment úloh, šprintov a projektov pomocou systému Jira. Systém slúži na evidenciu, sledovanie a plánovanie úloh, šprintov a projektov vo vývojárskych tímoch riadiacich sa agilnou metodikou scrum.

7.1.2 Dedikácia metodiky

Metodika je určená pre programátorské tímy, ktoré na zaznamenávanie a plánovanie práce využívajú systém Jira. A administrátorov udržujúcich systém Jira v prevádzke.

7.1.3 Nadväzujúce metodiky a dokumenty

Metodika práce so systémom Jira, Metodika dokumentácie, Metodika k riadeniu a priebehu tímových stretnutí, Metodika riadenie tímu , Metodika Scrum, Metodika administrácie systému Jira

7.1.4 Vymedzenie pojmov

Scrum– agilný spôsob vývoja v tímoch (viac v Metodike Srum)

Úloha - požiadavky zákazníka, implementačné a technické požiadavky ktoré majú byť riešené v projekte

Issue– reprezentácia úlohy v systéme Jira

Vlastník produktu – -Productowner (podľa metodikyScrum)

Zoznam úloh – zoznam všetkých zaznamenaných úloh -backlog (podľa metodiky Scrum)

7.1.5 Roly

Scrummaster - tvorba reportov, priraďovanie úloh, vytváranie a ukončovanie šprintov

Systémový administrátor – inštalácia systému, zálohovanie systému, správa verzií systému

Jira administrátor – manažment používateľských kont, riešenie používateľských problémov, udržovanie konzistentnosti obsahu systému

Vlastník produktu – pridávanie a úprava zoznamu úloh, plánovanie šprintu, prioritizácia úloh

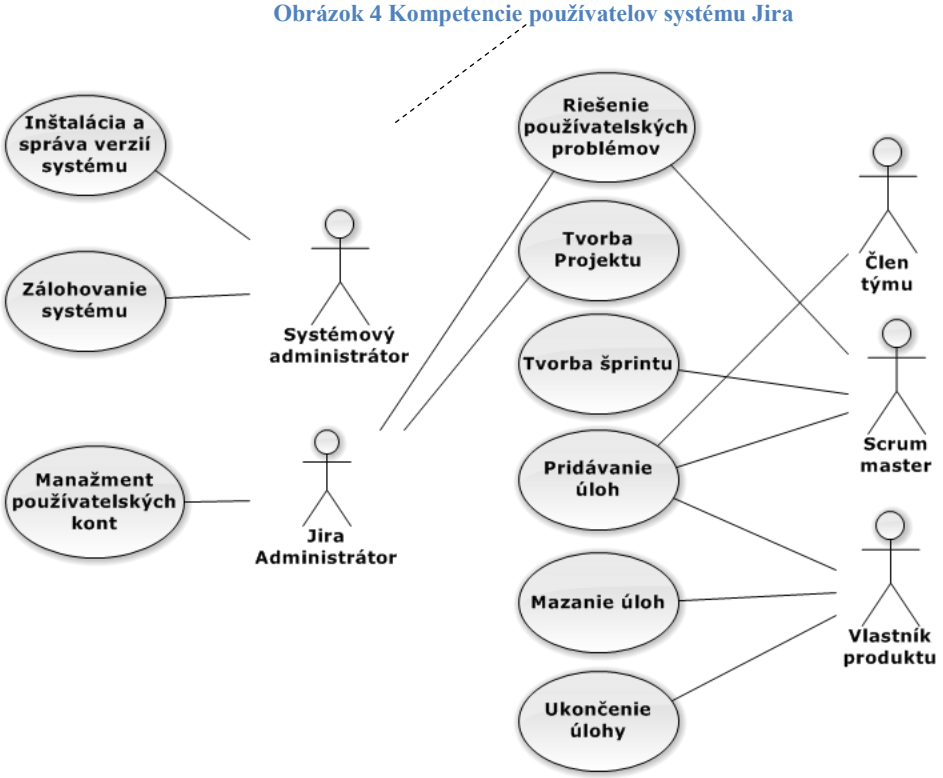
Člen týmu – práca s úlohami, zaznamenávanie času a komentárov

7.1.6 Procesy

7.1.6.1 Administrácia systému Jira

Proces administrácie systému Jira zachytáva inštaláciu, udržiavanie systému, správu verzií, používateľských kont a pokyny pre pravidelnú archiváciu databáz.

Obrázok 4 Kompetencie používatelov systému Jira



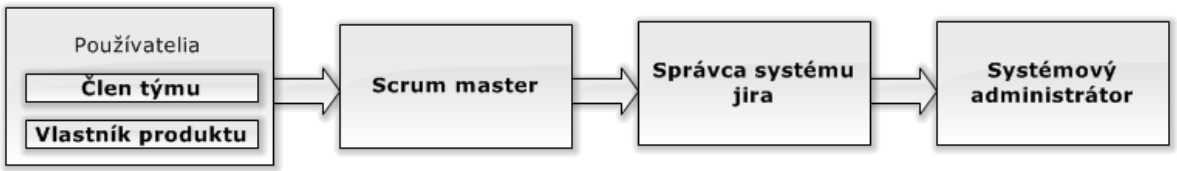
Inštaláciaa správa verziísystému – všetky inštalácie a úpravy verzii, doplnkov systému má na starosti systémový administrátor. O nastávajúcich zmenách systému je Systémový administrátor povinný informovať Jira administrátora.

Zálohovanie systému (databáz) – vykonáva Systémový administrátor v pravidelných intervaloch (Intervaly zálohovania sú stanovené v Metodike administrácie systému Jira)

Manažment používateľských kont –zmeny privilégií a tvorbu kont vykonáva Jira administrátor na základe oficiálnych požiadaviek ktoré prešlischvaľovacím procesom

Riešenie používateľských problémov -postup riešenia vzniknutého problému (ak nemá daný používateľ kompetencie riešiť problém postupuje problém ďalej)

Obrázok 5 - Postup riešenia používateľských problémov



7.1.6.2 Priebeh projektu

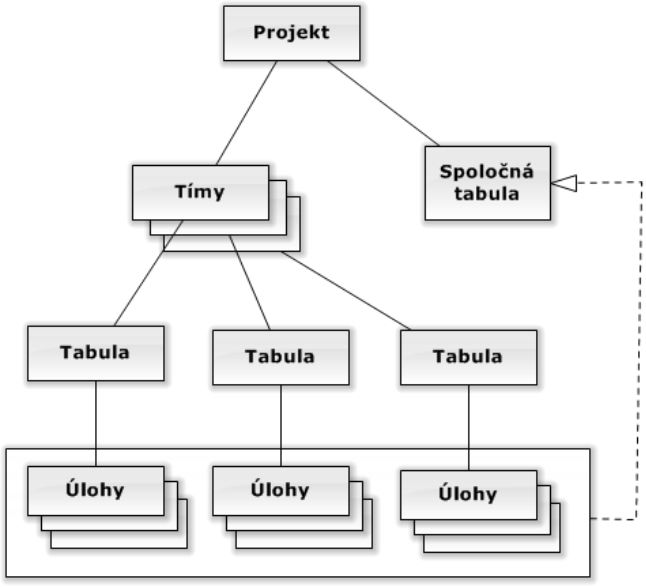
Proces popisuje priebeh projektu a kompetencie používateľov pri jeho tvorbe a ukončení v systéme Jira.

Vytvorenie projektu, tímov a tabúl- je v kompetencii Jira administrátora na základe podkladov od vlastníka produktu

Tvorba úloh - je v kompetencii členov tímu a vlastníka produktu

Ukončenie projektu –projekt ukončuje Jira administrátor po súhlase vlastníka produktu, ktorému bola dodná potrebná dokumentácie. Dáta z projektu zálohuje Jira administrátor podľa Metodiky administrácie systému Jira.

Obrázok 6 štruktúra projektu v systéme Jira



7.1.6.3 Šprint

Proces popisuje priebeh šprintu a kompetencie používateľov pri jeho tvorbe a ukončení v systéme Jira.

Vytváranie šprintu - je v kompetencii scrummastra na základe podkladov od vlastníka produktu

Pridávanie úloh do šprintu –úlohy pridáva do šprintu scrummaster po vzájomnej do členov tímu a vlastníka produktu

Ukončenie šprintu - šprint ukončuje scrummaster po súhlase vlastníka produktu, ktorému bola dodná potrebná dokumentácie. Dáta zo šprintu zálohuje Jira administrátor podľa Metodiky administrácie systému Jira.

7.1.6.4 Priebeh úlohy

Proces priebehu úlohy zachytáva všetky stavy úlohy v ktorých sa úloha môže nachádzať. Rozhodnutie o zaradení úlohy do procesu má vlastník produktu, sledovanie napredovania a riešenia úloh má na starosti scrummaster. O akceptovaní riešenia rozhoduje vlastník produktu.

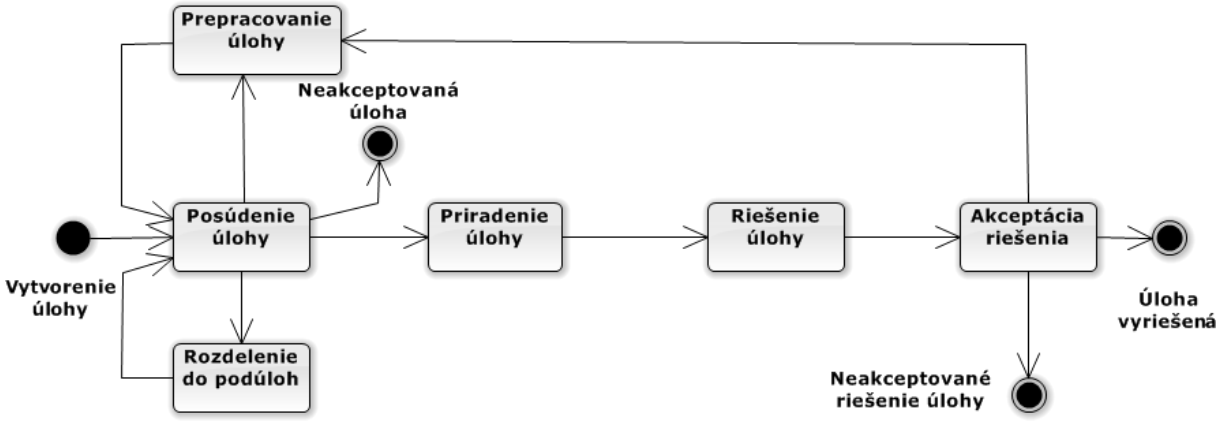
7.1.6.5 Vstupy

-požiadavky zákazníka
- implementačné požiadavky
- technické požiadavky

7.1.6.6 Výstup

- vyriešená / nevyriešená úloha

Obrázok 7 - možné stavy úloh



Vytvorenie úlohy – úlohy sa vytvárajú na základe požiadaviek a potrieb zákazníka, vlastníka produktu alebo členov týmu, tieto požiadavky a potreby sa zadávajú do systému formou issue do zoznamu úloh.

Posúdenie úlohy– úlohy posudzuje výhradne vlastník produktu, ktorý úlohám následne prideľuje prioritu pre ich riešenie.

Prepracovanie úlohy – neakceptovateľné požiadavky pre riešenie úlohy dáva vlastník produktu na prepracovanie členom týmu, prípadne sám konzultuje zmeny so zadávateľom úlohy.

Rozdelenie do podúloh – úlohy ktoré nemožno vyriešiť v jednej iterácií šprintu prípadne sú príliš zložité by mali byť rozdelené na menšie a prehľadnejšie podúlohy

Priradenie úlohy–členovia tímu si úlohy výhradne prideľujú samy podľa priorít stanovených vlastníkom produktu, vo výnimočných prípadoch môže úlohy priradiť vlastník produktu

Riešenie úlohy – scrummaster má dohlaď nad stavom a napredovaním úlohy

Akceptácia riešenia – vlastník produktu po vyriešení úlohy a dodaní všetkých potrebných dokumentov vyhodnotí stav riešenia a následne úlohu akceptuje, dá na prepracovanie, alebo zamietne

7.2 Manažment podpory vývoja (verziovanie)

7.2.1 Obsah metodiky

Metodika je určená pre manažment verzií vo verzionovacom systéme GIT. Systém GIT slúži na ukladanie (a zálohovanie) verzií projektov, dokumentov, zdrojových kódov.

7.2.2 Dedikácia metodiky

Metodika je určená pre tímy vývojárov, ktorí vytvárajú rozsiahle systémy, moduly a.i., ktoré je nutné verzionovať pre zaistenie archivácie a zálohovania kódu.

7.2.3 Slovník pojmov

Pojem	Význam
Branch	vývojová vetva
Merge	spojenie vývojových vetiev
Repozitár	úložisko zdrojového kódu
Feature	Menšia (menej zložitá) funkcionalita
Master branch	hlavná vývojová vetva, obsahuje jednotlivé verzie produktu
Development branch	vývojová vetva, ktorá obsahuje vývoj zložitejšej funkcionality
Feature branch	vývojová vetva, ktorá obsahuje menej zložitú funkcionalitu (väčšinou podúloha, fix, hotfix)
Fix	Oprava chybnej funkcionality
Hotfix	Oprava chybnej funkcionality na produkčnej verzií

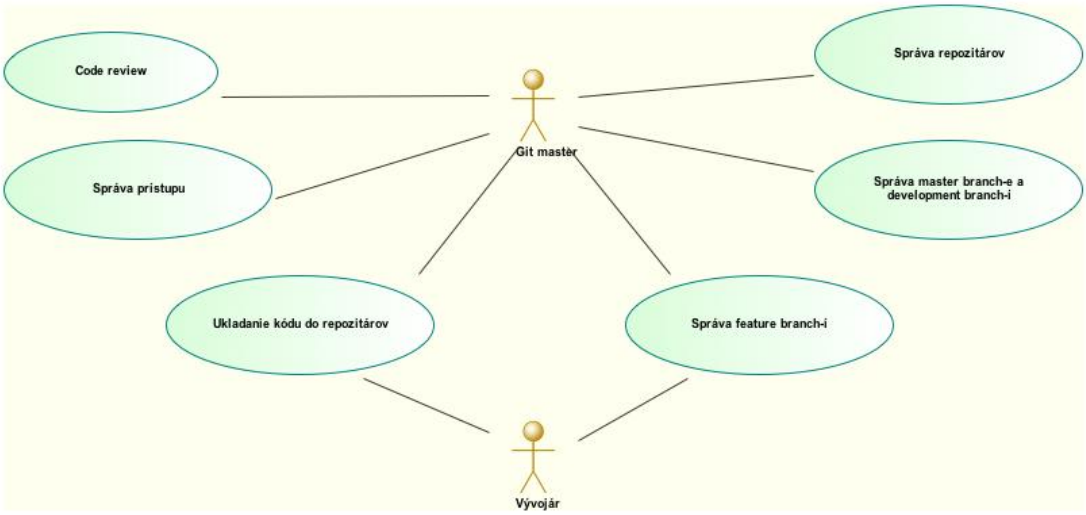
7.2.4 Roly

Roly, ktoré sme definovali sú popísané v tabuľke 4 a graficky znázornené v diagrame na obrázku 8.

Tabuľka 10 - Popis úloh pre definované roly

Rola	Úlohy
Git Master	• Správa repozitárov • Správa master branch-e a development branch-í • Review kódu (Podľa metodiky) • Označovanie branch-í a commit-ov • Úlohy vývojára viď. nižšie • Správa prístupu k repozitárom
Vývojár	• Ukladanie kódu do repozitárov (vývoj, hotfix, fix) • Správa feature branch-í

Obrázok 8 - Use case diagram pre definované roly



7.2.5 Procesy

7.2.5.1 Príchod nového vývojára

Pri príchode nového vývojára do vývojového tímu je potrebné aby si nainštaloval verzionovací systém GIT a poslal Git master-ovi prístupoví SSH kľúč.

Git master vytvorí prístup do repozitárov, na ktorých bude nový vývojár pracovať.

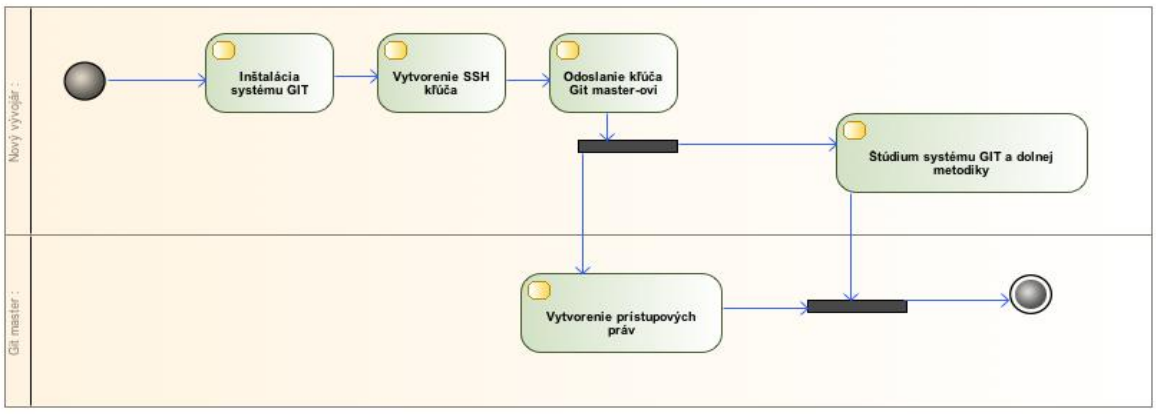
Vývojár má povinnosť oboznámiť sa s prácou s verzionovacím systémom GIT a to formou návodov, dokumentácie systému a dolnej metodiky práce so systémom GIT.

Vstup: nový vývojár.

Výstup: vývojár oboznámený s verzionovacím systémom GIT a prístupom do repozitáru

Zodpovedný: Git master

Obrázok 9 - činností pri príchode nového vývojára



7.2.5.2 Vytvorenie projektového repozitára

Pri vývoji na novom projekte alebo module je potrebné vytvoriť repozitár. Git master vytvorí vo verzionovacom systéme GIT nový repozitár, ktorému správne nastaví prístupové práva (Tabuľka 5).

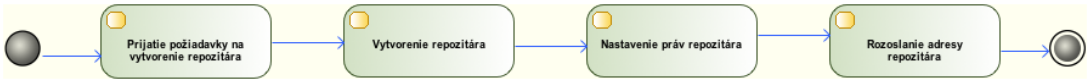
Po vykonaní predchádzajúcich úkonov odošle Git master vývojárom adresu repozitára.

Vstup: Požiadavka na vytvorenie repozitára.

Výstup: Vytvorený repozitár s nastavenými prístupovými právami.

Zodpovedný: GIT master

Obrázok 10 - Diagram činností pri vytváraní nového repozitára



Tabuľka 11 - Práva pre zápis a odvodenie

Rola	Branch					
	Master		Development		Feature	
	Zápis	Odvodenie	Zápis	Odvodenie	Zápis	Odvodenie
Git master	Áno	Áno	Áno	Áno	Áno	Áno
Vývojár	Iba hotfix	Nie	Áno	Áno	Áno	Áno

7.2.5.3 Ukončenie vývoja a deployment projektu

Merge feature branch-í do development branch-e

Vývojári ukončia vývoj a zapíšu zmeny do feature branch-e. Danú feature branch merge-nú do development branch-e, kde svoju funkcionalitu otestujú.

Git master vykoná review kódu (priebežne ju musí vykonávať aby na konci vývoja nemusel robiť rozsiahlejšiu review). V prípade, že odhalí chybu obráti sa na vývojára, ktorý vykoná fix priamo v development branch-i.

Merge development branch-e do master branch-e

Git master po záverečnom otestovaní a vykonaní review kódu, merge-ne danú development branch-u do master branch-e.

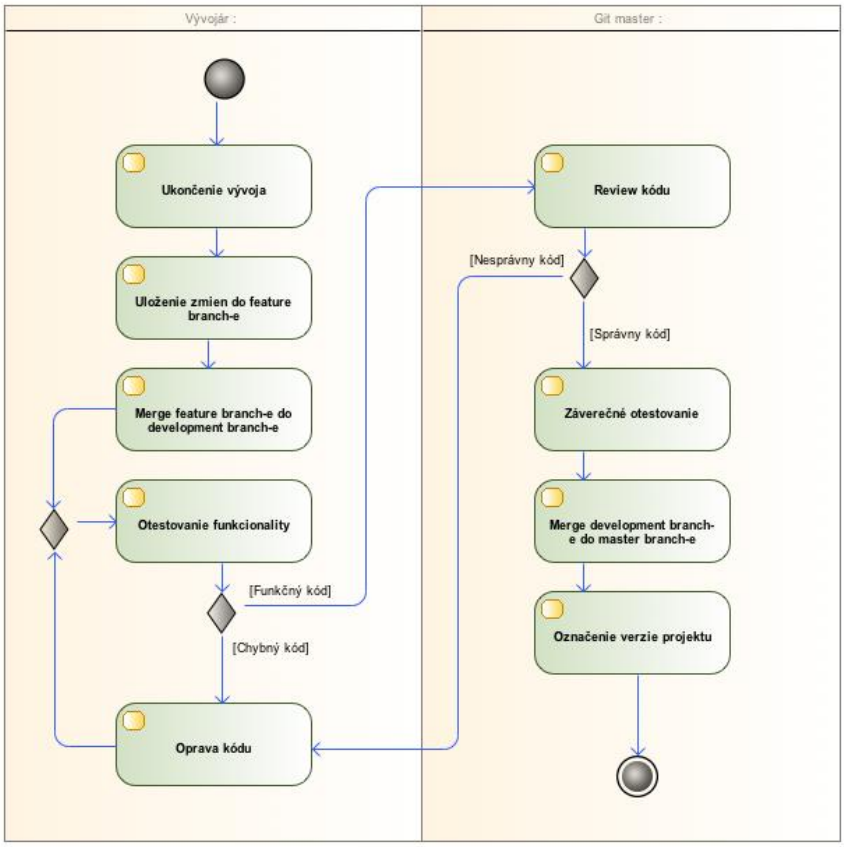
Po merge-i vytvorí tag s príslušnou verziou podľa spodnej metodiky.

Vstup: Požiadavka pre ukončenie vývoja alebo deployment.

Výstup: Funkčné zmeny uložené v master branch-i.

Zodpovedný: GIT master, vývojári

Obrázok 11 - Diagram činností pri ukončovaní vývoja



7.2.5.4 Vytvorenie development branch-e

Pri vývoji na novej (zložitejšej) funkcionalite projektu je potrebné vytvoriť novú development branch-u.

Git master vytvorí vo verzionovacom systéme GIT v príslušnom repozitári novú branch-u, ktorá bude obsahovať názov funkcionality (change request-u / content update-u).

Vstup: Požiadavka pre vytvorenie novej development branch-e.

Výstup: Nová branch pre vývoj novej funkcionality.

Zodpovedný: GIT master

7.2.5.5 Vytvorenie feature branch-e

Pri vývoji na novej (jednoduchšej) funkcionalite, ktorá môže ovplyvniť funkcionalitu inej časti projektu je potrebné vytvoriť novú feature branch-u.

Vývojár vytvorí svoju feature branch-u z development branch-e ktorá zahŕňa danú pod-funkcionalitu.

Vstup: Potreba oddelenia jednoduchšej funkcionality (napr.: kvôli funkčnosti iných komponentov).

Výstup: Nová branch pre vývoj novej (feature) funkcionality.

Zodpovedný: Vývojár

7.3 Manažment podpory vývoja (perconik)

7.3.1 Zavedenie štábnej kultúry

Vstup : *Zdrojový kód programu s chybami ktorý ešte nebol kontrolovaný.*

Výstup: *Zdrojový kód ktorý bude mať identifikované časti ktoré treba nejakým spôsobom doplniť, upraviť alebo len označiť za správne implementované.*

7.3.2 Dedikácia metodiky

Metodika sa zaoberá procesmi ohľadne správneho prechádzania zdrojového kódu. Obsahuje hlavné procesy ako sú :

#	Krok	Kapitola
1	Proces inštalácia PerConIK	7.3.3.1
2	Proces pridania funkcionality.	7.3.3.2
3	Proces vyriešenia chyby v kóde.	7.3.3.4
4	Proces CodeReview	7.3.3.5

7.3.3 Role

Člen týmu - Programátor ktorý pridáva do programu nové časti kódu spolu zo značkami.

Manažér podpory vývoja - Člen týmu zodpovedný za správny vývoj softvéru, snaží sa identifikáciu chýb v kóde, neimplementovaných častí kódu.

Manažér kontrola kvality - Člen týmu zodpovedný za kvalitu kódu, realizuje review kódu.

7.3.3.1 Proces inštalácie PerConIK

Na zlepšenie vývoja softvéru používame nástroj PerConIK ktorý nám umožňuje pridávať značky do zdrojového kódu programu. Preto pri pridávaní nového člena do týmu je potrebné zabezpečiť aby si dobre nainštaloval program PerConIK.

Vstup : *Eclipse bez pluginu PerConIK*

Výstup : *Funkčný PerConIK s validáciu značiek v Eclipse.*

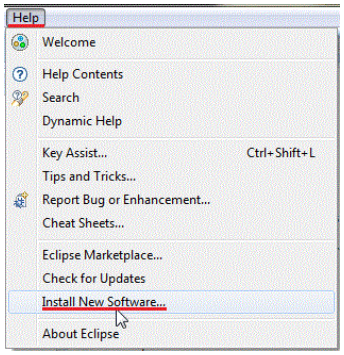
Zodpovedný: *Manažér podpory vývoja.*

#	Krok	Kapitola
1	Proces inštalácie PerConIK-u do Eclipsu	4.1
2	Nastavenie PerConIK-u	4.2
3	Proces zapnutia validácie značiek.	4.3

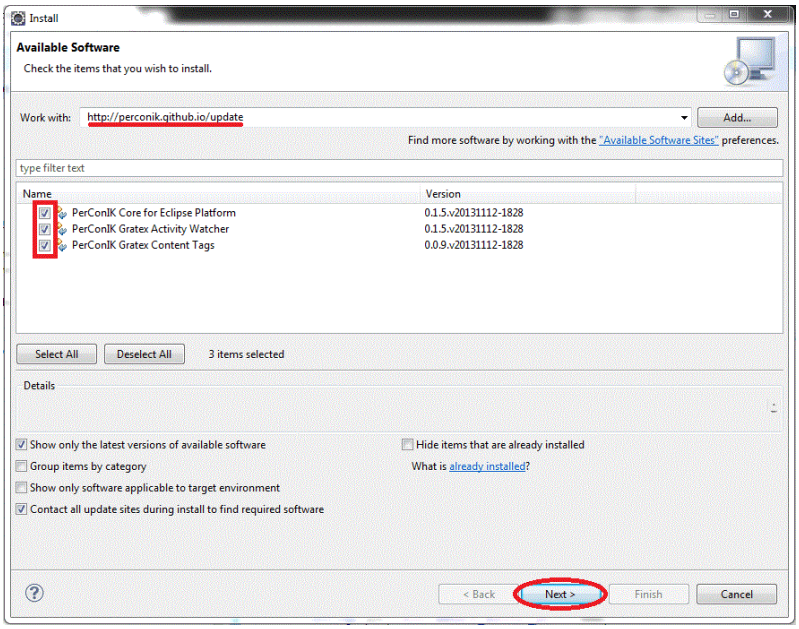
7.3.3.2 Proces inštalácie PerConIK-u do Eclipsu

Pre správnu inštaláciu treba zvoliť v hornej lište Eclipsu položku *Help → Install New Software....* tak ako je vidieť na obrázku 12 . Zobrazí sa okno. Do položky *Work With* zadajte adresu http://perconik.github.io/update. Zobrazí sa zoznam s ktorého vyberiete všetky položky a kliknete na next. Toto okno je zobrazená aj na obrázku 13 (v pravo). Následne postupujte podľa inštrukcií.

Obrázok 12 - Inštalácia PerConIK-u.



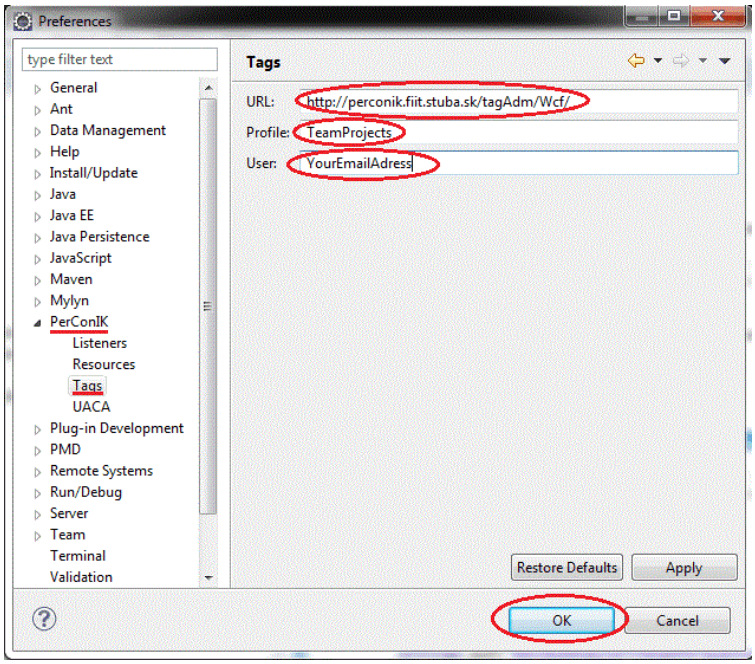
Obrázok 13 - Inštalácia PerConIK-u.



7.3.3.3 Nastavenie PerConIK-u

Pre správne fungovanie programu je potrebné nastaviť prihlasovacie údaje do repozitára PerConIK-u podľa obrázka 14. Toto nastavenie nájdete v menu pod *Window → Preferences.*

Obrázok 14 - Nastavenie údajov do repozitára.



7.3.3.4 Proces zapnutia validácie značiek.

Pravým kliknutím na projekt v ktorom chcete značky validovať zobrazíte lištu s ktorej vyberiete *PerConIK Tools → Enable Tags Nature*. V kóde sa vám označia nesprávne napísané značky.

7.3.4 Proces pridania funkcionality.

Dôležitý proces ktorý vykonáva každý člen týmu ktorý určitú funkcionalitu pri implementácií vynechá, nedokončí. Tento proces sa týka priamo značky TODO a môžeme ho rozložiť na nasledujúce pod-procesy:

#	Krok	Kapitola
1	Identifikovanie neimplementovanie funkcionality.	7.3.4.1
2	Pridelenie solver-a.	7.3.4.2
3	Implementácia funkcionality.	7.3.4.3
4	Posun funkcionality na review	7.3.4.4

Vstup : *Program bez implementovanej funkcionality.*

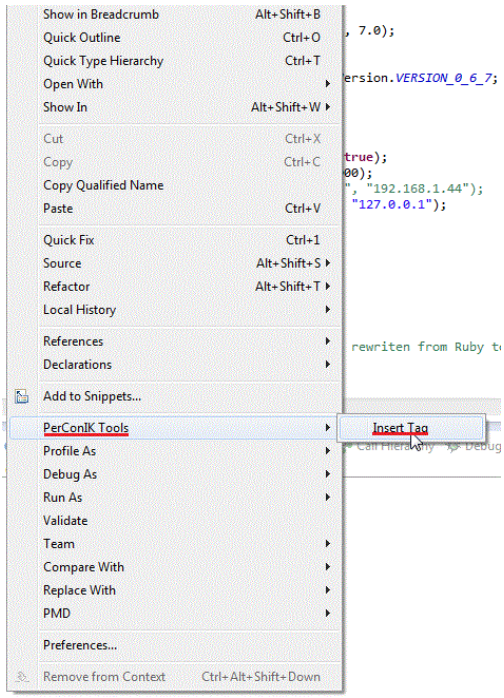
Výstup : *Program s implementovanou funkcionalitou.*

Zodpovedný: *Celý tím.*

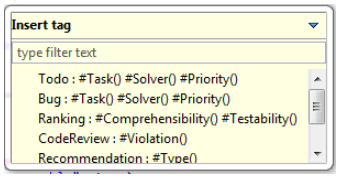
7.3.4.1 Identifikovanie neimplementovanie funkcionality.

Proces pri ktorom sa do zdrojovom kódu pridá značka TODO. Označíte časť kódu kde chceme implementovať funkcionalitu. Pravým kliknutím na vyznačenú oblasť sa nám zobrazí zoznam ktorý je vidno na obrázku číslo 15 zvolíme *PerConIK Tools → Insert Tag*. Zobrazí sa nám výber všetkých značiek, z ktorých zvolíme značku TODO. Jej Opis je v metodike: "Metodika na pridávanie značiek pomocou nástroja PerConIK".

Obrázok 15 - Pridanie požadovanej značky značky.



Obrázok 16 - Pridelenie solver-a.



Jedná sa o jednoduchý proces kedy člen tímu ktorý sa práve chystá doplniť funkcionalitu. Doplní do značky údaj #Solver(x) kde x bude jeho email. Dôraz sa kladie nato aby každý člen tímu si zobral sám funkcionalitu keďže tým je riadený pomocou agilnej metódy scrum.

7.3.4.2 Implementácia funkcionality.

V tomto procese sa snaží programátor implementovať danú úlohu. Dôležité je poznamenať, že nato aby mohol začať implementovať danú funkcionalitu musí pridať značku #Solver() proces 5.2, aby nedochádzalo k situáciám, kedy dvaja programátori riešia jednu úlohu. Za správne implementované sa považuje taká úloha ktorá je otestovaná a funkčná.

7.3.4.3 Posun funkcionality na review.

Po implementovaní a otestovaní členom tímu bude potrebné vymazať značku TODO a vytvoriť novú značku Recommendation z údajom #Type(other) a textom za značkou CodeReview aby iný člen tímu vedel o tom, že tejto časti kódu treba spraviť review. Taktiež je potrebné aby koniec párovej značky TODO upravil tak aby obsahovala všetky implementované časti.

Obrázok 17 - Príklad posunu funkcionality na review.

```
//Recommendation #Type(Other) CodeReview | sppred@gmail.com 2013-12-08T14:43:07.6920000Z
    void implementovaná_funkcionalita(){
    }
//@< | sppred@gmail.com 2013-12-08T14:43:07.6920000Z
```

7.3.5 Proces vyriešenia chyby v kóde.

Proces pri ktorom je identifikovaná, spracovaná a vyriešená chyba v zdrojovom kóde programu.

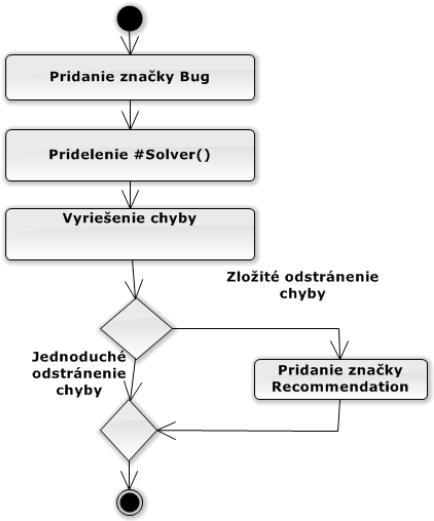
#	Krok	Kapitola
1	Identifikovanie Chyby.	7.3.5.1
2	Pridelenie solver-a.	7.3.5.2
3	Označenie chyby za vyriešenú.	7.3.5.3

Vstup : *Program s chybou .*

Výstup : *Program bez chyby.*

Zodpovedný: *Manažér podpory vývoja.*

Obrázok 18 - diagram Odstraňovania chýb v kóde



7.3.5.1 Identifikovanie Chyby.

Je potrebné aby chyba v programe bola označená a bola jej vhodne určená priorita. Pre tento účel sa používa značka z PerConIK-u Bug. Jej Opis je v metodike: "Metodika na pridávanie značiek pomocou nástroja PerConIK". Značka sa pridáva podobne ako značka TODO ktorá je opísaná v kapitole 5.1.

7.3.5.2 Označenie chyby za vyriešenú.

Ak sa jednalo o jednoduchú chybu ktorá nemenila veľkú časť kódu jednoducho vymaže značka bug.

7.3.6 Proces CodeReview.

Proces pri ktorom sa snažíme identifikovať porušenie konvencií pri vytváraní tohto kódu. Tento proces je podrobnejšie špecifikovaný v dokumente : "Horná metodika pre tvorbu, údržbu a dokumentáciu kódu" od Mateja Bádala. Z pohľadu životné procesu značiek CodeReview obsahuje nasledujúce pod-procesy:

#	Krok	Kapitola
1	Proces odporúčaného CodeReview	7.3.6.1
2	Proces pravidelného CodeReview	7.3.6.2

Vstup : *Program z nekvalitným kódom.*

Výstup : *Program z kvalitnejším kódom.*

Zodpovedný: *Manažér kontrola kvality.*

7.3.6.1 Proces odporúčaného CodeReview

Proces ktorý sa vykonáva vždy ak manažér kvality nájde v kóde značku Recommendation z textom CodeReview. V skratke sa dá povedať že vymaže túto značku a nahradí ju značkou CodeReview. Člen tímu ktorý danú časť kódu vyvíjal následne opraví nedostatky a odstráni značku CodeReview.

7.3.6.2 Proces pravidelného CodeReview

Podobný proces ako proces odporúčaného code review s tým že manažér sám prechádza kód a hľadá v ňom nedostatky.

7.4 Manažment kvality

7.4.1 Úvod

Nasledujúci dokument obsahuje metodiku manažérskej úrovne, pre tvorbu, údržbu a dokumentáciu zdrojových kódov s dôrazom na prehľadnosť a zrozumiteľnosť.

7.4.2 Dedikácia metodiky

Metodika je určená pre všetkých členov tímu. Taktiež je určená pre manažéra kvality a manažéra plánovania.

7.4.3 Obsah metodiky

Metodika postupne opisuje jednotlivé procesy z pohľadu manažérskej úrovne. Pri každom procese je uvedené, kto je zodpovedný za daný proces. Nasledujúca tabuľka obsahuje zoznam všetkých procesov, ktoré sú v metodike opísané.

Tabuľka 12 - procesy

#	Názov procesu
1	Príchod nového člena
2	Práca na projektoch
3	Code review
4	Dohodnutie meetingu
5	Meeting
6	Pridelenie úlohy prepracovania kódu
7	Prepracovanie kódu
8	Overenie zmien

7.4.4 Nadväzujúce dokumenty

S dokumentom súvisia ďalšie dokumenty, ktorých účel je buď doplniť túto metodiku, alebo ju ďalej rozviesť a bližšie opísať procesy, ktoré prechádzajú aj do iných oblastí.

- **[1] - Metodika pre tvorbu, údržbu a dokumentáciu kódu** – Dolná metodika, ktorá opisuje konkrétne postupy pri tvorbe údržbe a dokumentácii kódu. Dolná metodika by mala byť podkladom, z ktorého sa vychádza pri procesoch opísaných v nasledujúcej hornej metodike.

7.4.5 Slovník pojmov

Code review – Proces, pri ktorom sa skúma kvalita vytvoreného kódu z celkového pohľadu, ale aj z pohľadu zavedených interných konvencií.

Code review meeting – stretnutie programátorov a ľudí zodpovedných za kvalitu kódu, na toto stretnutí sú prebrané chyby a spôsoby ich odstránenia

7.4.6 Opis rôl

Manažér kvality – Osoba zodpovedná za kvalitu a finálny stav kódu, jeho úlohou je takisto code review, vedie meeting, kde sa rozoberajú jednotlivé chyby.

Manažér plánovania – Osoba zodpovedná za naplánovanie a rozdelenie jednotlivých úloh v tíme.

Člen tímu – Programátor, ktorý prispieva svojimi schopnosťami pri vývoji softvéru, musí sa riadiť konvenciami vymedzenými v dolnej metodike

7.4.7 Metodika

7.4.7.1 Príchod nového člena tímu

Pri príchode nového člena tímu je nutné, aby sa nový člen čo najskôr oboznámil s konvenciami, ktoré sa v danom tíme používajú pri tvorbe zdrojového kódu. Na tento účel slúži dolná metodika [1], ktorú tím musí mať vo vytlačenej forme. Nový člen po vybavení formálnych záležitostí dostane dolnú metodiku, ktorú si musí naštudovať. Obdržanie tejto metodiky má na starosti manažér kvality. Nový člen bude riešiť prípadne otázky, alebo nezrovnalosti takisto s manažérom kvality.

Vstup – nový člen tímu

Výstup – zaučený člen tímu

Zodpovedný – manažér kvality

7.4.7.2 Práca na projektoch

Člen tímu pracuje na pridelených úlohách. Pri vytváraní zdrojového kódu sa musí riadiť konvenciami zapísanými v dolnej metodike [1], ako aj svojím úsudkom, tak aby pri tejto tvorbe nevytváral anti-vzory. Vytvorený kód musí správne zdokumentovať v súlade so zavedenými konvenciami.

Vstup – požiadavka na vypracovanie úlohu

Výstup – vytvorený zdrojový kód

Zodpovedný – člen tímu

7.4.7.3 Code review

Manažér kvality musí v pravidelných intervaloch vykonávať code review. Dĺžka týchto intervalov závisí na internej dohode tímu. V prípade šprintu, ktorý trvá dva týždne, sa code review musí vykonávať minimálne raz počas behu tohto šprintu, a taktiež na konci šprintu.

Code review spočíva v tom, že manažér kvality preskúma vytvorený kód a snaží a hľadá v ňom časti, ktoré sú v rozpore s dolnou metodikou [1], ale aj anti-vzory vo všeobecnosti.

Vstup – vytvorený zdrojový kód

Výstup – overený zdrojový kód

Zodpovedný – manažér kvality

7.4.7.4 Dohodnutie meetingu

Ak je pri procese code review odhalených viacero chýb, prípadne chyba, ktorá sa viackrát opakuje, je potrebné stretnutie členov tímu, na ktorom sa tieto chyby rozoberú a ukážu sa aj možné riešenia. Dohodnutie tohto meetingu má na starosti manažér plánovania, ktorý určí termín, kedy sa takýto meeting bude konať. Na meetingu sa musí zúčastni väčšina členov tímu, ktorí vytváranú zdrojové kódy, tomu sa tento termín musí prispôsobiť.

Vstup – požiadavka na dohodnutie meetingu

Výstup – naplánovaný meeting

Zodpovedný – manažér plánovania

7.4.7.5 Meeting

Manažér kvality si pripraví prezentáciu, v ktorej poukáže na chyby nájdené počas code review. Ku každej chybe naznačí možné riešenie, alebo sa odkáže na dolnú metodiku [1]. Identifikované chyby spíše aj s ich autormi a poskytne ich manažérovi plánovania.

Vstup – požiadavka na uskutočnenie meetingu

Výstup – uskutočnený meeting

Zodpovedný – manažér kvality

7.4.7.6 Pridelenie úlohy prepracovania kódu

Manažér plánovania jednotlivým osobám zo zoznamu s chybami naplánuje úlohu na prerobenie týchto chýb.

Vstup – zoznam chýb s ich autormi

Výstup – pridelená úloha na prepracovanie kódu

Zodpovedný – manažér plánovania

7.4.7.7 Prepracovanie kódu

Člen tímu prepracuje kód, kde sa nachádza chyba. Ako podklad pri tom použije dokument s dolnou metodikou [1]. V prípade, že ide o väčšiu zmenu, konzultuje riešenie s manažérom kvality. Po oprave kódu informuje o tejto skutočnosti manažéra kvality.

Vstup – požiadavka na prepracovanie kódu

Výstup – prepracovaný kód

Zodpovedný – člen tímu

7.4.7.8 Overenie zmien

Manažér kvality dostane informáciu o tom, že kód s chybami bol prerobený. Opäť vykoná

code review, ale len na kóde, ktorý bol prepracovávaný. V prípade, že identifikuje ďalšie problémy, znovu spíše ich zoznam s autorom a poskytne ho manažérovi plánovania na prerobenie. Vysvetlenie chýb a návrh riešenia v tomto prípade rieši individuálne s členom tímu.

Vstup – notifikácia o prepracovaný kód

Výstup – overený kód

Zodpovedný – manažér kvality

7.5 Manažment riadenia

7.5.1 Úvod dokumentu

Tento dokument opisuje manažment riadenia vývoja softvéru pomocou agilnej metodológie scrum. Dokument je vhodný pre menšie tými pracujúce na nie veľmi rozsiahlych softvérových projektoch.

7.5.2 Slovník pojmov

Scrum

Inkrementálna a iteratívna metodika pre agilný vývoj softvérového produktu

Šprint

Základná časová jednotka metodiky scrum. Šprint je časovo ohraničené obdobie, vopred stanovenej dĺžky, v ktorom sú vypracovávané úlohy.

UserStory

Zachytáva požiadavky zákazníka na produkt vo forme Čo, Kto a Prečo – Čo je funkcionalitou, Kto s tým bude pracovať a Prečo je to treba implementovať

7.5.3 Roly

Jednotlivé role a popis účastníkov, ktorí vystupujú v manažmente riadenia metodiky Scrum sú opísané v tabuľke 8.

Tabuľka 13 - Role a popis účastníkov Scrum-u

Rola	Popis
ProductOwner	• Reprezentuje zákazníka • Zadáva a upravuje požiadavky • Udáva prioritu požiadavkám • Hodnotí splnenie/nesplnenie úloh
ScrumMaster	• Člen tímu • Zabezpečuje správny priebeh šprintu

	• Moderuje tímové stretnutia
Vývojový tím	• Je tvorený študentmi • Je zodpovedný za vývoj softvéru • Hodnotí náročnosť úloh
Člen týmu	• Študent • Vyberá si úlohu, na ktorej chce pracovať • Je zodpovedný za splnenie úlohy

7.5.4 Elementy

V Tabuľke 9 sú popísané základné elementy, ktoré sú využívané pri riadení projektu pomocou metodiky Scrum.

Tabuľka 14 - Elementy Scrum Metodiky

Element	**Popis**
ProductBacklog	• Zoznam požiadaviek zoradený podľa priority • Reprezentuje čo treba spraviť a v akom poradí • Správcom je ProductOwner • ProductOwner určuje priority
Požiadavka	• Predstavuje funkcionalitu alebo vlastnosť produktu, ktorú treba vytvoriť • Je písaná ako “ userstory“ • Vytvára a upravuje ProductOwner • Je pomerne abstraktná • Typicky sa delí na viacero úloh
SprintBacklog	• Zoznam úloh pre jeden šprint • Vzniká odoberaním požiadaviek z ProductBacklog-u a rozdelením na jednotlivé úlohy • Požiadavky si vyberá vývojový tím
Úloha	• Má konkrétne znenie • Úlohu si vyberá aj vypracováva člen týmu • Vzťahuje sa ku konkrétnej požiadavke
Tímové stretnutia	• Konajú sa raz za týždeň • 3 typy podľa toho v ktorej fáze šprintu sa uskutočňujú
ProductIncrement	• Predstavuje súbor všetkých dokončených požiadaviek v šprinte

7.5.5 Riadenie vývoja softvéru

Na sledujúce podkapitoly opisujú procesy vykonávané v rámci metodiky Scrum. Každý proces má svoj vstup a výstup a tiež roly účastníkov ktoré sa daného procesu účastina.

7.5.6 ProductBacklog

7.5.6.1 Pridanie požiadavky

Role:ProductOwner

Vstup: Prázdny alebo z časti zaplnený ProductBacklog

Výstup:ProductBacklog rozšírený o novú požiadavku

Na základe identifikovaných vlastností, ktoré by mal produkt spĺňať, pridá ProductOwner novú požiadavku do ProductBacklog-u. Pridávanie požiadaviek je možné v ktorejkoľvek fáze vývoja.

7.5.6.2 Definícia požiadavky

Role:ProductOwner

Vstup:ProductBacklog

Výstup: ProductBacklog s presne definovanými požiadavkami

Každá požiadavka musí mať jednoznačne určený názov, popis, prioritu a podmienky splnenia. Všetky vlastnosti požiadavky určuje ProductOwner. Názov jednoznačne určuje požiadavku. Popis má formu "userstory". Priorita je určená na základe dôležitosti pre výsledný produkt. Podmienky splnenia definujú kedy sa úloha považuje za splnenú (napr.: novovytvorený kód musí prejsť unit testami).

7.5.6.3 Zmena požiadavky

Role: ProductOwner

Vstup: ProductBacklog

Výstup: ProductBacklog s upravenou požiadavkou

Zmenu požiadavky má v kompetencii ProductOwner. Zmeniť môže všetky vlastnosti danej požiadavky (názov, popis, prioritu, podmienky splnenia), ak požiadavka ešte nie je vykonávaná.

7.5.6.4 Odstránenie požiadavky

Role:ProductOwner

Vstup:ProductBacklog

Výstup:ProductBacklog bez odstránenej požiadavky

ProductOwner môže odstrániť požiadavku z ProductBacklog-u ak bola upravená špecifikácia produktu a daná požiadavka už v novej špecifikácii nehrá rolu.

7.5.7 Šprint Backlog

7.5.7.1 Výber požiadaviek

Role:ProductOwner

Vstup:ProductBacklog

Výstup: Šprint Backlog

Na začiatku každého šprintu vyberá ProductOwner požiadavky z ProductBacklog-u a tým vytvorí Šprint Backlog. Požiadavky sú vyberané na základe priority.

7.5.7.2 Rozdelenie úloh

Role: Členovia tímu

Vstup. Šprint Backlog

Výstup: Rozdelené úlohy medzi členov tímu

Členovia tímu rozdelia každú požiadavku zo Šprint Backlog-u na menšie podúlohy. Potom si každý člen tímu vyberie úlohu na ktorej chce pracovať.

7.5.7.3 Vloženie úloh do plánovacieho systému

Role: Člen tímu – manažér plánovania

Vstup: Zoznam úloh s riešiteľmi

Výstup:Úlohy vložené do plánovacieho systému

Manažér plánovania tvorí zoznam úloh s riešiteľmi počas delenia úloh. Následne vkladá plánované úlohy do systému kde každý člen tímu vidí aké úlohy má pridelené a do kedy ich treba spraviť.

7.5.7.4 Reportovanie úloh

Role: Členovia tímu

Vstup: Úloha v plánovacom systéme

Výstup: Úloha v plánovacom systému s reportom

Každý člen tímu musí reportovať postup práce na úlohách v plánovacom systéme. Reporty je nutné podávať hneď po ukončení práce na úlohe aj keď ešte úloha nie je hotová. Reportovaním sa sleduje čas strávený na úlohe a percentuálny odhad dokončenia úlohy.

7.6 Manažment testovania

7.6.1 Úvod

Predmetom tejto metodiky je určenie procesov prebiehajúcich počas celej fázy testovania. Taktiež definuje zodpovedné osoby spolu so vstupným a výstupným výsledkom každého procesu. Táto metodika pokrýva manažment kvality a testovania.

7.6.2 Dedikácia metodiky

Metodika je primárne určená pre tím č. 4 na predmete Tímový Projekt I. a II. v akademickom roku 2013/2014, ale taktiež aj pre programátorov (testerov), využívajúcich programovací jazyk Java a vývojové prostredie Eclipse.

7.6.3 Použité pojmy

Tabuľka 15 - použité pojmy a ich vysvetlenie

#	Pojem	Vysvetlenie
1	jednotkový test	nízkoúrovňový test, zameraný na testovanie funkčných jednotiek (zväčša tried) zdrojového kódu
2	JUnit	framework pre testovanie zdrojových kódov písaných v jazyku Java
3	framework	programový balík nástrojov pre riešenie komplexných úloh
4	Eclipse	vývojové prostredie v ktorom využijeme framework Junit.
5	Testovacia trieda	trieda vykonávajúca testy nad testovanou triedou
6	Testovaná trieda	trieda, ktorá je testovaná testovacou triedou
7	TestCase	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda vykonávajúca test(y) nad testovanou triedou
8	TestSuite	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda zoskupujúca množinu testovacích tried

7.6.4 Manažment testovania

Software je nutné testovať v každej fáze vývoja. Táto kapitola obsahuje celý proces testovania v rámci vývojového prostredia Eclipse, jazyku java a jednotkového testovania za pomoci frameworku JUnit. Taktiež sú tu obsiahnuté roly a zodpovednosti jednotlivých účastníkov.

7.6.5 Roly zahrnuté do procesov testovania

Tabuľka 16 - roly obsiahnuté v procese testovania a ich zodpovednosti

#	Rola	Zodpovednosti
1	Vedúcitestovania	Koordinácia testovacieho tímu Vedenieceléhoprocesutestovania Rozdeľovanie a prideľovanie jednotlivých úloh Vyhodnocovanie celého procesu testovania
2	Vývojár	Vypracovanie testov pre triedy, ktoré boli určené na testovanie Navrhovanie riešení pri odhalení chýb pri procese testovania
3	Tester	Vypracovanie testov pre triedy, ktoré boli určené na testovanie Spúšťanie testov Identifikácia odhalených skutočností Reportovanie odhalených chýb Navrhovanie riešení pri odhalení chýb pri procese testovania

4	Zapisovateľ	Zapisovanie zistených skutočností pri celom procese testovania

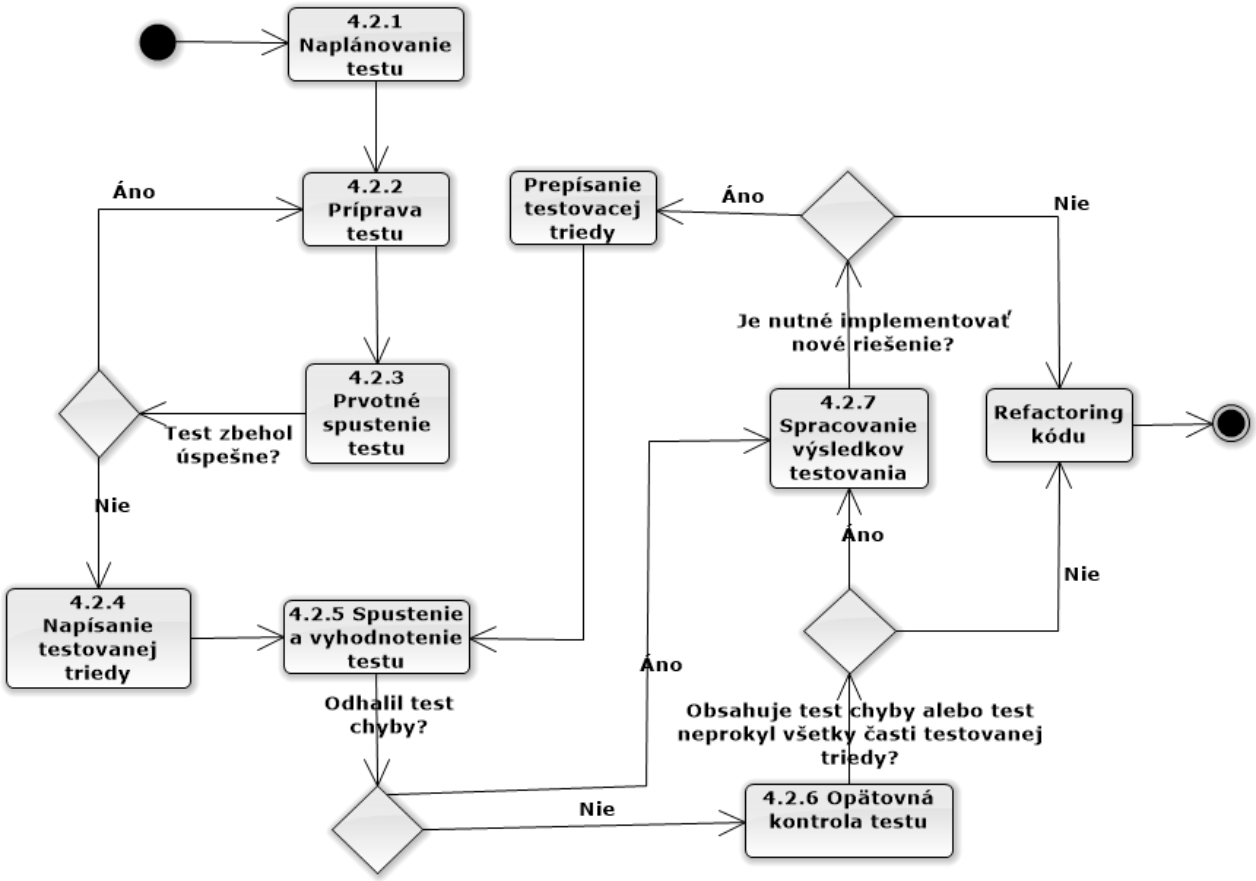
7.6.6 Procesy definované v oblasti manažmentu testovania

Tabuľka 17 - procesy definované v rámci tejto metodiky

#	Názov	Kapitola
1	Proces naplánovaniatestovania	4.2.1
2	Proces prípravy testu	4.2.2
3	Proces prvotného spustenia testu	4.2.3
4	Proces napísania testovanej triedy	4.2.4
5	Proces spustenia a vyhodnotenia testu	4.2.5
6	Proces opätovnej kontroly testu	4.2.6
7	Proces spracovania výsledkov testovania	4.2.7

Na obrázku č. 19 je znázornená postupnosť horeuvedených procesov.

Obrázok 19 - diagram aktivít v procese testovania



7.6.7 Proces naplánovania testovania

Vstup: požiadavka na otestovanie softvérového projektu
Výstup: pridelenie úloh, dokumentácia testovania
Zodpovedný: vedúci testovania

Vedúci testovania poverí rolou každého zo zainteresovaných ľudí v rámci procesu testovania. V prípade, že nie je vytvorená dokumentácia testovania, v ktorej sa zaznamenáva celý priebeh jeho procesu, vytvorí sa. Následne vedúci pridelí úlohy všetkým zainteresovaným, ktoré je potrebné vykonať ešte pred samotným začatím procesu testovania. Základom je spustenie IDE Eclipse spolu s konfiguráciou frameworku Junit (vo väčšine prípadov je framework Junit už predkonfigurovaný, ale môže nastať prípad, kedy je nutná konfigurácia).

7.6.8 Proces prípravy testu

Vstup: dokumentácia testovania s požiadavkami na testovanie
Výstup: zdrojový kód testovacej triedy
Zodpovedný: vývojár

Vývojár vytvorí v prostredí Eclipse prvotný test definujúci požiadavku na funkcionalitu kódu, ktorá sa od neho očakáva. Taktiež sa týmto uistí, že presne chápe funkcionalitu a požiadavky na testovaný komponent, čo zaručí, že samotný kód sa neodchýli od pôvodného zámeru a cieľa definovaného pri plánovaní testu.

7.6.9 Proces prvotného spustenia testu

Vstup: zdrojový kód testovacej triedy
Výstup: funkčný test
Zodpovedný: vývojár

Vývojár musí všetky novonapísané testy spustiť ešte pred napísaním samotného kódu a je nutné, aby skončili neúspechom, pretože pokiaľ neexistuje kód, ktorý by umožnil úspešné ukončenie, nemôže ani test skončiť úspechom. V prípade, že by nejaký test v tomto kroku skončil úspešne , čo značí chybu, je nutné, aby vývojár navrhol a implementoval vhodné riešenie tohto problému.

7.6.10 Proces napísania testovanej triedy

Vstup: funkčný test
Výstup: zdrojový kód testovanej triedy
Zodpovedný: vývojár

Pokiaľ budú pripravené testovacie triedy je potrebné aby vývojár napísal zdrojový kód testovanej triedy, ktorý má test pokryť. Tento proces sa priamo netýka tejto metodiky, ale je dôležitou súčasťou pri procese testovania.

7.6.11 Proces spustenia a vyhodnotenia testu

Vstup: zdrojový kód testovacej triedy spolu s testovanými triedami, dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: tester, zapisovateľ

Tester spustí jednotkový test a zapisovateľ zaznamenáva výsledky testu. V prípade, že sa reálny výstup testu líši od očakávaného, autor zdrojového kódu, prípadne iný vývojár alebo tester, navrhne vhodné riešenie. Všetky návrhy zapisovateľ zaznamená do dokumentácie testovania.

7.6.12 Proces opätovnej kontroly testu

Vstup: dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: tester, zapisovateľ

V prípade, že test neodhalil žiadne chyby, tester prekontroluje testovaciu triedu, či je správne implementovaná. Pokiaľ sa nájdu chyby v implementácii, je nutné, aby navrhol vhodné riešenie, ktoré zapisovateľ zaznamená do dokumentácie testovania.

7.6.13 Proces spracovania výsledkov testovania

Vstup: dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: vedúci testovania

Vedúci testovania po obdržaní dokumentácie testovania vyhodnotí testovanie z hľadiska chýb a návrhov riešení a určí ako sa bude postupovať. V prípade, že sú v dokumentácií testovania uvedené návrhy riešení testovacích tried, poverí vývojára, aby uskutočnil ich implementáciu. V opačnom prípade vykoná refaktoring kódu.

7.6.14 Zoznam nadväzujúcich metodík a dokumentov

Konvencia zápisu zdrojového kódu

Konvencia komentovania zdrojového kódu

Konvencia refaktoringu zdrojového kódu

7.7 Manažment dokumentovania

7.7.1 Pojmy

Používateľský príbeh – úloha, vykonávaná v jednom šprinte

Šprint – dvojtýždňový cyklus

Centrálna dokumentácia – aktuálna celková dokumentácia

7.7.2 Roly

Manažér dokumentácie – človek, ktorý je poverený správou dokumentácie

Zákazník – zadávateľ projektu

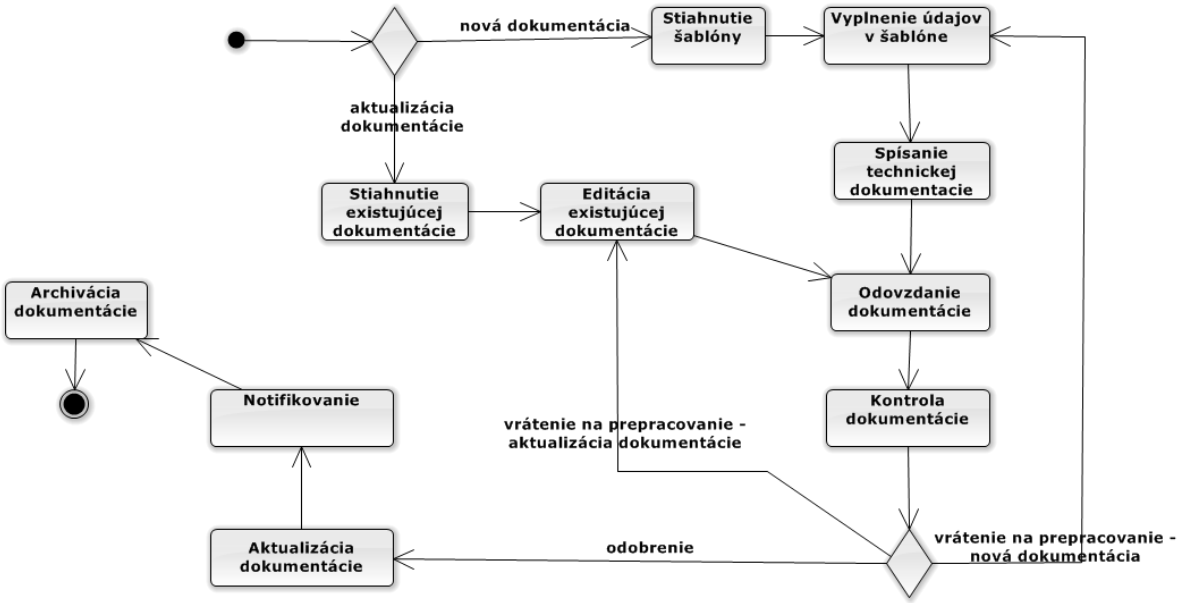
Vývojár – človek, ktorý sa podieľa na realizácii používateľského príbehu

Manažér kvality – poverený kontrolou dokumentácie

7.7.3 Popis metodiky

Metodika popisuje spôsob tvorby technickej dokumentácie. Proces tvorby je znázornený na obrázku č. 20. Proces tvorby začína rozhodnutím, či ide o tvorbu novej dokumentácie alebo aktualizáciu už existujúcej. Ak ide o tvorbu novej dokumentácie, nasleduje stiahnutie a vyplnenie údajov v šablóne, spísanie technickej dokumentácie. Ak ide o aktualizáciu dokumentácie, je potrebné dokumentáciu stiahnuť, editovať ju. Po vykonaní týchto procesov nasleduje proces odovzdania dokumentácie, jej kontrola. Po kontrole je potrebné rozhodnúť o tom či dokumentácia spĺňa požiadavky. Ak nespĺňa, podľa druhu sa dokumentácia vracia na prepracovanie. Ak požiadavky spĺňa, nasleduje proces aktualizácie dokumentácie, proces notifikácie a pred ukončením proces archivácie dokumentácie.

Obrázok 20- aktivity diagram



7.7.4 Popis procesov

7.7.4.1 Proces 1 – Stiahnutie šablóny

Vstup: Odkaz na šablónu pre dokumentáciu

Výstup: Stiahnutá šablóna dokumentácie

Zodpovedný: Vývojár

Opis procesu: Vývojár sťahuje šablónu z úložiska na svoj pracovný stroj.

7.7.4.2 Proces 2 – Vyplnenie údajov v šablóne

Vstup: Stiahnutá šablóna dokumentácie

Výstup: Vyplnená šablóna dokumentácie (základné údaje, názov, autor, čas)

Zodpovedný: Vývojár

Opis procesu: Vývojár vypĺňa základné údaje ako názov, meno autora, dátum a čas, krátky popis, nadväznosť na iné časti.

7.7.4.3 Proces 3 – Spísanie technickej dokumentácie

Vstup: Používateľský príbeh

Výstup: Dokument s opísaným používateľským príbehom, návrhom, implementáciou, testovaním

Zodpovedný: Vývojár

Opis procesu: Vývojár dokumentuje používateľský príbeh, jeho návrh, implementáciu a testovanie.

7.7.4.4 Proces 4 – Odovzdanie dokumentácie

Vstup: Dokument s opísaným používateľským príbehom

Výstup: Dokument uložený v úložisku dát

Zodpovedný: Vývojár

Opis procesu: Vývojár odovzdáva vytvorený dokument do spoločného úložiska dát.

7.7.4.5 Proces 5 – Kontrola dokumentácie

Vstup: Dokument uložený v úložisku

Výstup: Rozhodnutie o odobrení alebo vrátení dokumentácie

Zodpovedný: Manažér dokumentácie, Manažér kvality

Opis procesu: Manažér dokumentácie a manažér kvality kontroluje stav odovzdanej dokumentácie. Ak je stav vhodný, dokumentácia je odobrená a posunutá ďalej. Ak je stav nevyhovujúci, dokumentácia sa vracia na prepracovanie.

7.7.4.6 Proces 6 – Stiahnutie existujúcej dokumentácie

Vstup: Odkaz na existujúcu dokumentáciu

Výstup: Stiahnutá existujúca dokumentácia

Zodpovedný: Vývojár, Manažér dokumentácie

Opis procesu: Vývojár sťahuje na svoj pracovný stroj existujúcu dokumentáciu podľa odkazu, ktorý mu poskytne manažér dokumentácie.

7.7.4.7 Proces 7 – Editácie existujúcej dokumentácie

Vstup: Stiahnutá existujúca dokumentácia

Výstup: Upravená existujúca dokumentácia

Zodpovedný: Vývojár

Opis procesu: Vývojár upravuje už existujúcu dokumentáciu, aktualizuje potrebné časti.

7.7.4.8 Proces 8 – Aktualizácie dokumentácie

Vstup: Odovzdaná dokumentácia

Výstup: Aktualizovaná centrálna dokumentácia

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácia aktualizuje centrálnu dokumentáciu o nové časti z odovzdanej dokumentácie.

7.7.4.9 Proces 9 – Notifikovanie

Vstup: Aktualizovaná centrálna dokumentácia

Výstup: Notifikovanie všetkých zainteresovaných strán

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácie rozosiela notifikácie o aktualizovaní centrálne dokumentácie všetkým zainteresovaným stranám – zákazníkovi, vývojárom, manažérovi kvality, prípadne ďalším.

7.7.4.10 Proces 10 – Archivácia dokumentácie

Vstup: Centrálna dokumentácia

Výstup: Archívna dokumentácia

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácie archivuje do úložiska dát centrálnu dokumentáciu

8 Zápisnice zo stretnutí

V tejto kapitole sú uvedené všetky zápisy zo stretnutí počas semestra.

8.1 Zápis 1. Stretnutia tímu č.4

Dátum: 2.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík, Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Samuel Benkovič

Michal Petráš

Martin Adámik

Vladimír Bošiak

Michal Čerešňák

Igor Homola

Téma:

Vypracoval: Matej Bádal

Opis stretnutia:

Úvod do predmetu

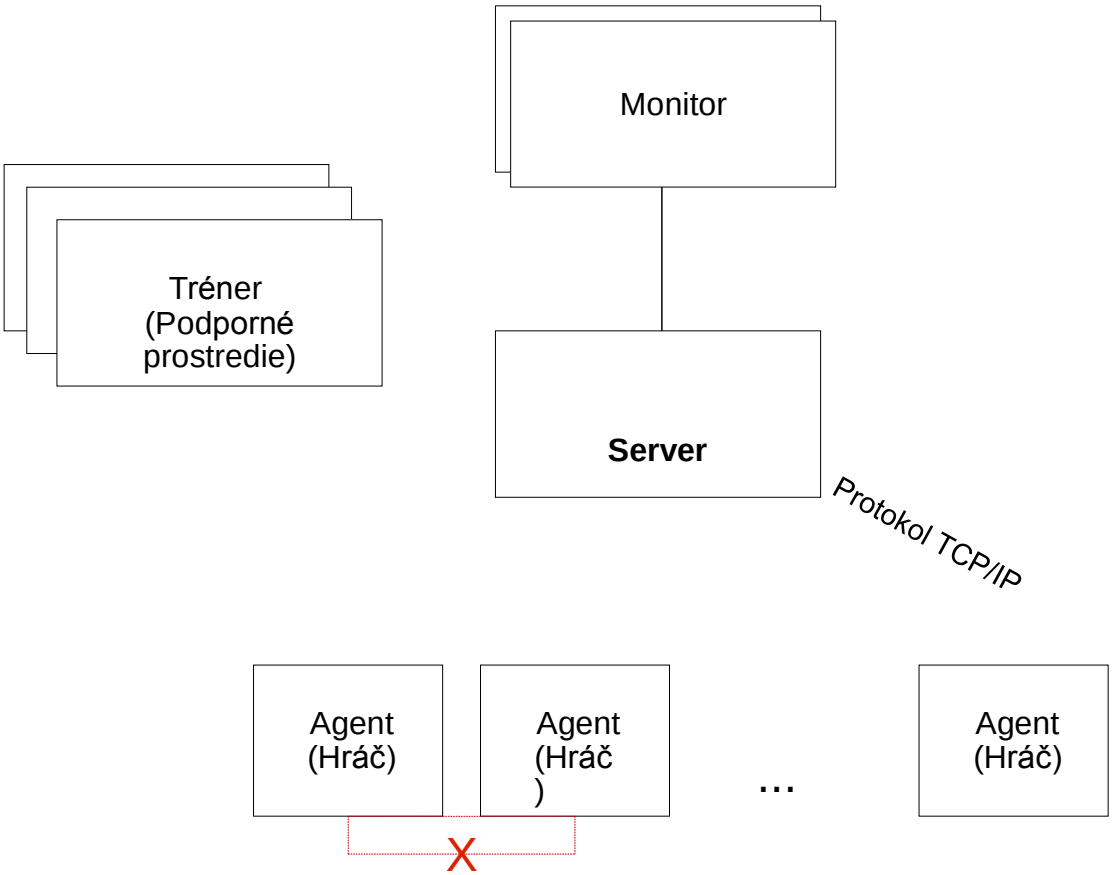
- rozprava o dokumentácii – finálne musí byť dokumentácia odovzdaná
- pozor na dodržiavanie termínov
- stretnutia primárne vedú tímy

Úvod do problematiky

- opis fungovania pohybov robota
- opis SERVERA
  - SERVER simuluje väčšinu vecí (ihrisko, loptu)
  - SERVER ododiela informácie o stave sveta a jednotlivých robotov
  - SERVER pracuje v krokoch, každý krok ma 20 ms
- robot Nao (Aldebaran Robotics)
  - low skill opisy pohybov
  - high skill fázy pohybov
  - riadenie kĺbov
  - pohyby kĺbov v časovom úseku (stabilizované a

nestabilizované)
- skladanie vyšších pohybov (dostaň sa k lopte, postav sa...)
- rozhodovací strom

- dôležitý komponent – monitor -> pripája sa na server a zobrazuje simuláciu
- podporné prostredie slúži (hlavne) na testovanie
  - niekedy je napojené priamo na hráča
- komunikácia medzi agentami navzájom je **ZAKÁZANÁ**
- bližší opis hráča
  - pozícia hráča je vždy 0,0 pre neho samého
  - hráč si ráta svoju pozíciu na ihrisku sám
  - každý agent má zorný uhol – 120 stupňov
- návrhy na zlepšenia
  - highskilly sú spracované dosť nepríjemne
  - bolo by treba zlepši rozhodovanie vrámci tímu
- priblíženie SCRUM-u a predmetu
  - buď je úloha hotová, alebo nie
  - kto za šprint nemá nič hotové, nemal by dostať zápočet



Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
1	-nezadané	Rozdelenie úloh v tíme
2	-nezadané	Skúsiť si spustenie nejakého hráča
3	-nezadané	Určenie SCRUM mastera
4	-nezadané	Založenie dokumentácie
5	-nezadané	Premyslieť, čo by sme chceli (konkrétne) robiť
6	-nezadané	Dohodnúť si technológie pre komunikáciu a dokumentáciu

8.2 Zápis 2. Stretnutia tímu č.4

Dátum: 9.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Začiatok prvého šprintu

Vypracoval: Michal Čerešňák

Opis stretnutia:

- Prebehla kontrola úloh z minulého týždňa
- Kontrola ako sa komu podarilo nainštalovať hráča

- Martin Adámik – Linux 12.04: server, monitor, agent – OK, Jim – chyby
- Michal Čerešňák – Windows XP: server, monitor, agent – OK, hráč sa nepohybuje
- Matej Bádal – Linux Mint: server, monitor – OK, Jim a RoboCupLibrary – chyby
- Michal Petráš – Windows 7: server, monitor, agent – OK
- Igor Homola – Windows 8: server – chyba
  Linux: server, monitor – OK, Eclipse – chyba

- Do budúceho týždňa odstrániť problémy s inštaláciou hráča
- K inštalácii hráča musí každý člen týmu vypracovať dokument obsahujúci
  - Problémy pri inštalácii
  - Riešenie problému
  - Použité nástroje a ich verzie (Java, Eclipse, ...) + typ operačného systému

- Ing. Marián Lekavý nám dodal nové zdrojové kódy
- Zápisy zo stretnutí treba dodávať na web do 24 hodín od stretnutia
- Pre ohodnocovanie náročnosti úlohy sme použili Planning poker cards
- Žiaden člen týmu nesmie mať na konci šprintu 0 bodov
- Úloha pre SCRUM Mastera
  - Na stretnutia mať vytlačený zoznam úloh s percentom splnenia + burndownchart graf

- Úloha 1. Šprintu
  - Dôkladne sa oboznámiť s hráčom
  - Naštudovať dokumentáciu a kód
  - Rozdeliť analýzu medzi členov týmu
  - Zistiť kde sa nachádzajú nastaviteľné časti hráča
  - Každý člen týmu musí vedieť zmeniť nejakú triviálnu vec v správaní hráča (napr. mávanie rukou)
  - Každý člen týmu musí napísať unittest na nejakú vybranú metódu
  - Usporiadať tímový brainstorming – vymyslieť nápady na druhý šprint a usporiadať ich do backlogu
  - Každá úloha musí byť zdokumentovaná

- Doplnková úloha 1. Šprintu
  - Oboznámiť sa s architektúrou svetových tímov
  - Analyzovať minimálne 7 tímov

- Výsledok 1. Šprintu
  - Vedieť čo máme k dispozícii a na čom ideme ďalej stavať

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
1	-nezadané Dokončené	Rozdelenie úloh v tíme	
2	-nezadané	Skúsiť si spustenie hráča	Prebieha
3	-nezadané	Určenie SCRUM Mastera	Dokončené
4	-nezadané	Založenie dokumentácie	Prebieha
5	-nezadané	Premyslieť čo by sme chceli (konkrétne) robiť	Prebieha

6	-nezadané	Určiť technológie pre komunikáciu a dokumentáciu	Dokončené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy	Stav
7	Bádal, Homola	Spojazdniť tímový web	
8	všetci	Zaradenie úloh do backlogu	
9	všetci	Spojazdnenie hráča	
10	Benkovič	Vedenie analýzy kódu	
11	Bádal, Bošiak	Analýza kódu – Jim	
12	Čerešňák, Homola	Analýza kódu – RoboCupLibrary	
13	Petráš, Adámik	Analýza kódu – TestFramework	
14	všetci	Unit Test	
15	všetci	Zmena správania hráča	
16	všetci	Analýza svetových tímov	

8.3 Zápis 3. Stretnutia tímu č.4

Dátum: 16.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Priebeh prvého šprintu

Vypracoval: Igor Homola

Opis stretnutia:

- Prebehla kontrola backlogu, burndown chartu a úloh z minulého týždňa
- Brainstorming ohľadne úloh do backlogu
- Informácia k unit testom
    - Unit test treba vytvoriť k časti kodu ktora sa nebude menit
    - Možnostiunit testu - pohľad na svet, parser správ zo servera, knižnice
- Návrh úloh do backlogu
- Kontrola ako sa komu podarilo nainštalovať hráča
    - všetci členovia tímu spustili hráča a testframework

- Úloha 1. Šprintu
    - Dôkladne sa oboznámiť s hráčom
    - Naštudovať dokumentáciu a kód
    - Analýza prostredia - Samuel Benkovič
        - Jim
            - Matej Bádal
            - Vladimír Bošiak
        - TestFramework
            - Martin Adámik
            - Michal Petráš
        - RoboCupLibrary
            - Michal Čerešňák
            - Igor Homola
    - Zistiť kde sa nachádzajú nastaviteľné časti hráča
    - Každý člen týmu musí vedieť zmeniť nejakú triviálnu vec v správaní hráča (napr. mávanie rukou)
    - Každý člen týmu musí napísať unittest na nejakú vybranú metódu
    - Každá úloha musí byť zdokumentovaná

- Výsledok 1. Šprintu
    - Vedieť čo máme k dispozícii a na čom ideme ďalej stavať
    - Vytvorenie backlogu

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
1	Bádal, Homola	Spojazdniť tímový web	Dokončené
2	všetci	Zaradenie úloh do backlogu	Prebieha
3	všetci	Spojazdnenie hráča	Dokončené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy	Stav
4	všetci	Zaradenie úloh do backlogu	Prebieha
5	Benkovič	Vedenie analýzy kódu	
6	Bádal, Bošiak	Analýza kódu – Jim	
7	Čerešňák, Homola	Analýza kódu – RoboCupLibrary	

8	Petráš, Adámik	Analýza kódu – TestFramework
9	všetci	Unit Test
10	všetci	Zmena správania hráča
11	všetci	Analýza svetových tímov

8.4 Zápis 4. Stretnutia tímu č.4

Dátum: 23.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Priebeh štvrtého stretnutia

Vypracoval: Samuel Benkovič

Opis stretnutia:

- Vedúcemu projektu boli prezentované výsledky úloh s prvého šprintu.
  - Unit Test
    - Samuel Benkovič dostal prerobiť unit test, problém riešenia bol ten, že testované metódy boli závislé od hry.
  - Správanie hráča
    - Všetci členovia tými splnili túto úlohu.
  - Analýza projektu
    - pri analýze sa mali dorábať aj komentáre.
    - mali sme spisovať veci ktoré sa dajú odstrániť.
    - Jim
      - potrebný refaktoring.
      - Stav kódu : Zlý.

        - TestFramework
            - prijateľná štruktúra.
            - Stav kódu : Primeraný.
        - RoboCupLibrary
            - nevyužívané metódy.
            - Stav kódu : Primeraný.
    - BackLog
        - definovať presnejšie úlohy - napríklad pri kope úlohe stabilizovať kop do lopty určiť kedy je táto úloha splnená napríklad z 10 pokusov spadne len 1.
        - máme spojiť backlog s druhým tímom.

- Boli identifikované nasledovné stories :
    - 9.1 Verzia servera (critical)
    - 9.2 Hĺbková analýza zahraničných tímov (critical)
    - 9.3 Zjednodušiť časť ruby (critical)
        - Odhadovaná redundancia 20-30%
    - 4.2 Odstrániť ruby (critical)

    Najprv zjednodušiť (9.3) potom odstrániť (4.2)

    - 9.4.1 Spojazdniť automatický anotátor (minor)
        - Dorobiť anotácie (minor)
    - 9.4.2 Vytvoriť architektúru highskills (major)
    - 9.4.3 Návrh strategickej vrstvy (major)
    - Implementácia 9.4 (major-)
    - 9.5.1,2 Odstránenie nepoužívaného kódu (major)
    - 9.5.3 Presun sekcií do RoboCupLibrary (minor)
    - 9.5.4 Premenovať Lowskill (trivial)
    - 9.6 Analýza DP - dorobiť do 9.4.2 a 9.4.3 (critical)
    - 9.6.2 Integrácia DP - implementácia (viac stories) (minor)
    - 9.7.1 Test framework - doplniť UI (minor+)
    - 9.7.2 upraviť framework podľa zmien v architektúre (minor+)
    - 9.8 Zmena taktiky na základe výkrikov (minor)
    - 4.1 Vylepšenie stability hráča po kopnutí do lopty (minor)
        - Vylepšiť evolučný prístup, alebo diplomový projekt
        - dlhý kop je jedno či hráč spadne pri krátkom nesmie spadnúť.
        - kopov je veľký počet za zápas
        - doplniť do stories.
    - Parametrický kop do lopty
      vzdialenosť a uhol kam chceme loptu kopnúť (minor)
    - 4.3 Rozbehanie GIT-u (!blocker)
    - Automatický build-systém
      minimálne 1x denne (integrácia s GIT) (critical)

- 4.4 Aktualizovať návody na inštaláciu na wiki (critical)
- 4.5 Editor pohybov => export do XML (trivial)
- 4.6 Vylepšiť príchod k lopte (minor)
  - Tak aby bol dostatočne rýchli a dostatočne presný
- 4.9.1 Nastavenie sa k lopte (minor)
- 4.9.2 Rozhodnutie čo s loptou (minor)
- 4.12. Vytvorenie rozhodovanie pri strele hráčov (trivial)
- 4.13. Hľadanie lopty (minor)
  - Otáčať viac hlavou
- 4.14. Zlepšiť driblovanie (minor)
- 4.15 Auto-reštart hry (trivial)

- Vždy je treba pretestovať viac krát. overenie ako finálne 100 pokusov.

- V architektúre treba zohľadniť :
  - Strategická vrstva
    - zadá, že hráči majú vykonať určitú stratégiu
    - Príklad : útok po pravom krídle
  - Taktická vrstva
    - posúdi situáciu a rozhodne čo ktorý hráč ma vykonávať
    - Príklad : Jeden ide po loptu, druhý mu nabieha na prihrávku.
  - Highskill
    - Highskill danú akciu ktorú ma vykonať a rozkúskuje ju na menšie časti lowskilly
    - Príklad : Choď po loptu rozkúskuje na natoč sa, kráčaj , natoč sa ...
  - Lowskill
    - realizuje pohyb hráča po ihrisku.
    - Príklad: Kráčaj (hráč ohýba svoje kĺby a kráča)
- Určili sme si čo sa bude vykonávať v tomto šprinte :
  - 9.1 Verzia servera.
  - 9.3 , 4.2 Zjednodušiť a odstrániť ruby
    - rozdeliť triedy, highskilly,
    - rozdeliť ruby do balíčkov.
    - preprogramovať do javy.
      - otestovať a až potom zmeniť ruby referencie na tie java.
  - 9.5.1,2 Odstránenie nepoužitého kódu.
    - Opatrne aby ste nevymazali len nedokončenú ale stále potrenú funkcionalitu.

8.5 Zápis 5. Stretnutia tímu č.4

Dátum: 30.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samule Benkovič

Téma: Šprint 2 – progres

Vypracoval: Michal Petráš

Opis stretnutia:

- Začiatok stretnutia
  Martin začal stretnutie, informoval Mariána o jednotlivých úlohách a ich stave.
  Marián sa pýtal na kritickú úlohu "spoločný backlog".
  Spoločný backlog bol založený v Jire.

- Informovanie o stave úloh jednotlivých členov
  - Mišo Čerešnák spojazdnil GIT, začal analyzovať zahraničný tím a začal analyzovať diplomovku
  - Martin Adámik má hotovú analýzu zahraničného tímu, začal analyzovať diplomovku, taktiež spojazdnil GIT
  - Matej Bádal spojazdnil GIT, vybral si tím na analýzu. Snažil sa spojazdniť SSH na tímovom serveri.
  - Vlado Bošiak spojazdnil GIT, vybral si tím na analýzu, taktiež pripravuje návody na analýzu pre inštaláciu na MACu a Linuxe.
    - Martin Adámik sa pýtal na zdokumentovanie inštalácie, s ohľadom na zmeny v ruby kóde. Marián odpovedal, že návod má byť písaný s ohľadom na túto skutočnosť.

- Michal Petráš spojazdnil GIT, analyzoval zahraničný tím a zároveň začal analyzovať diplomovku. Taktiež spisuje dokumentáciu z prvého šprintu.
    - Mal otázku ohľadne dokumentácie – ako spisovať zmenu správania a podobné malé podúlohy, špecificky uvádzať autorov ? Marián odpovedal, že je vhodné spisovať to do odstavcov, prípadne uviesť autorov.
- Samo Benkovič spojazdnil GIT, poskytol analýzu zahraničného tímu z jeho bakalárske práce. Plánuje sa pustiť do ruby kódu.
    - Marián upozorňuje na skutočnosť, že to nie je vec, ktorú dokončím a hneď pôjde.
- Igor Homola spojazdnil GIT, analyzoval zahraničný tím a začal analyzovať diplomovku

- Diskusia
    - Ruby – Marián upozorňuje na ruby kód a pýta sa ako sme si to podelili. Náš tím si zobral na starosti plánovač.
    - Interface – Marián odporúča vytvoriť a dohodnúť si interface tak aby sme neboli od seba závislí a mohli pracovať súčasne
    - Upozornenie – Marián špecificky upozorňuje na to, že ruby kód je vec , ktorá ma potenciál byť nestihnutá!
    - Burn down chart – graf nie je veľmi dobrý – dôsledok veľkého počtu zadaní odovzdávaných počas víkendu
    - Igor Homola oznamuje , že bola podaná prihláška na TP CUP

8.6 Zápis 6. Stretnutia tímu č.4

Dátum: 6.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samule Benkovič

Téma: Zhodnotenie práce po druhom šprinte a definovanie cieľov pre tretí šprint

Vypracoval: Vladimír Bošiak

Opis stretnutia:

- Prezentácia výsledkov šprintu
  - Prezentácia analýzi zahraničných tímov
    - Hamburg Bit-Bots
    - Nao Team HTWK
    - FC Portugal
    - Austin Villa
    - Upennalize
    - MagmaOffenburg
  - Prezentácia analýzi diplomových projektov
    - Hudec J. – rýchla chôdza
    - Ďurčák – rozhodovacia logika hráča a riešenie kolizí
  - Prezentácia vytovrenej inštalačnej príručky pre MacOS a Linux
  - Prezentácia prepísaného kódu z Ruby do Java
    - Konzultácia zmien
    - Problémi s komunikáciou s druhým tímom a chyba integrácie
- Zlepšenia
  - pomenovanie taskov
  - úplne odstránenie Ruby
- Retrospektíva
  - Zlepšenie komunikácie s druhým tímom

- Zavedenie hlavičiek do súborov (autor, tím, rok)
- Riešenie znalosti GIT-u
- Riešenie konfliktov s druhým tímom
- Vytvorené tasky dávať vedúcemu na pridelenie priorít
- Keď chceme niečo vymazať, treba to označiť ako *TODO vymazať* ak to zostane aj po mesiaci v kóde môžeme vymazať danú časť
- Určenie vrstiev – strategická, taktická

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
RFCMTROLL-76	Adámik, Bošiak	Vytvorenie inštalačných príručiek pre MacOS a Linux	splnené
RFCMTROLL-3	Bádal, Bošiak, Homola, Adámik, Petráš, Čerešňák	Vytvorenie analýzi zahraničných tímov	splnené
RFCMTROLL-70	všetci	Prepísať plánovanie do jazyku Java	splnené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
ROBOCUPTP-30	Tím 9	V projekte Jim je potrebne prepisat vsetky konstanty zavisle od verzie servera
ROBOCUPTP-62	všetci	Určenie, či zakomentovaný kód funguje
ROBOCUPTP-61	všetci	Odstránenie nepoužívaných funkcií a premenných
ROBOCUPTP-39	všetci	Návrh a implementícia vrstiev (strategická, taktická vrstva)
ROBOCUPTP-38	všetci	Vytvoriť architektúru highskills

8.7 Zápis 7. Stretnutia tímu č.4

Dátum: 13.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Šprint 3 - progress

Vypracoval: Martin Adámik

Opis stretnutia:

- Začiatok stretnutia
  Marián začal stretnutie s požiadavkou o informovaní o stave úloh.
- Informovanie o stave úloh jednotlivých členov:
  - Samuel Benkovič odstránil ruby a riešil návrh architektúry.
  - Igor Homola vložil jednotlivé úlohy do systému Jira.
  - Michal Petráš analyzoval zakomentované časti kódu a spisoval veľkú dokumentáciu.
    - Michal Petráš sa taktiež spýtal na zdokumentovanie celkového pohľadu. Marián ozrejmil ako sa má zdokumentovať celkový pohľad na produkt.
  - Vladimír Bošiak navhroval architektúru a refaktoroval triedu agent model v balíku Jim.
  - Martin Adámik analyzoval zakomentované časti kódu.
  - Matej Bádal odstránil ruby a riešil návrh architektúry.

        - Matej Bádal taktiež ozrejmil Mariánovi, že ruby sa v doterajšej fáze nemôže ešte úplne odstrániť, pretože komunikuje s testframework-om, ale čo sa týka hráča bolo ruby už úplne odstránené.
    - Michal Čerešňák analyzoval zakomentované časti kódu.
- Diskusia
    - Mariánovi a Ivanovi bola predvedená doposiaľ navrhnutá architektúra. Následne sa začala diskusia, do ktorej sa zapájal každý člen tímu.
    - Marián s Ivanom navrhli, aby si taktiky vyhodnocovali svoju fitness, ktorá bude rozhodným prvkom pri výbere najvhodnejšej taktiky.
    - Ivan navrhol, aby bol vytvorený default high skill, čo znamená, že hráč bude stáť, ale bude mať celkový prehľad o tom, kde sa lopta nachádza.
    - Treba vytvoriť príkladové taktiky a následne 5 až 10 testovacích scenárov na testovanie jednotlivých taktík.

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
9.5.2	Adámik, Petráš, Čerešňák	Určiť, či zakomentovaný kód funguje alebo nie.	Prebieha
	Všetci	Návrh a implementácia taktickej vrstvy	Prebieha
9.4.3	Všetci	Navrh a Implementacia strategickej vrstvy	Prebieha
9.4.2	Všetci	Vytvoriť architektúru Highskills	Prebieha
	Petráš	Vytvoriť dokumentáciu	Prebieha
	Benkovič	Úplne odstrániť RUBY z kódu	Dokončené

Úlohy na ďalší týždeň:

Pokračovať v rozpracovaných úlohách.

8.8 Zápis č.8 stretnutia tímu č.4

Dátum: 20.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samul Benkovič

Téma: Ukončenie šprintu, návrh nových úloh na ďalší šprint

Vypracoval: Matej Bádal

Opis stretnutia:

- preriešenie webovej stránky
- prezentáciu práce vykonanej za posledný šprint
- diskusia ohľadom zakomentovaných častí kódu
- riešenie vytvorenej architektúry
  - vysvetlenie a opis jednotlivých aspektov architektúry vedúcim
  - navrhnutie možnosti konfigurácie pomocou XML
  - vysvetlenie podstaty šablónovej situácie
  - prediskutovanie aspektu architektúry – situácií
  - vedúcim sa príliš nepozdáva myšlienka so šablónovými situáciami
  - návrh, že situácia by mohla mať ohodnotené atribúty
- vedúci poukazujú na to, že niekto v budúcnosti bude možno chcieť iné delenie ihriska, ako na 4. kvadranty
- porovnávanie situácii bude prebiehať pomocou porovnávania stringov
- vyhodnocovanie situácii bude prebiehať pri výbere taktík
- stratégia bude mať zoznam taktík, ktoré sa dajú vykonať
- Ivan navrhuje, že jednotlivé taktiky musia ma istú zotrvačnosť – nesmú sa prerušiť ak dôjde k chvíľkovej zmene situácie

- vedúcich zaujíma stanovenie defaultnej situácie
- navrhnutá taktika „ZORIENTUJ SA“
- diskusia ohľadom tejto taktiky

- Taktika sa sama musí vyhodnotiť, či je hotová
- vyhodnocovanie fitness by malo by rozdielne pri výbere taktiky a iné pri zisťovaní, či taktika už skončila
- rozoberá sa pokračovanie taktiky pri zmene situácie
- padla otázka, či viac situácii sa môže naraz vyskytovať
  dohodli sme sa, že ihrisko bude rozdelené na 4 kvadranty
  rieši sa identifikácia jednotlivých situácii
  každá TAKTIKA bude mať inicializačnú a progresovú podmienku
  konkrétne situácie – taktiky majú vlastnú podmienku

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
1	Tomáš Nemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML
7	Benkovič/Linner	Papierový prototyp

8.9 Zápis č.9 stretnutia tímu č.4

Dátum: 27.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

SamuelBenkovič

Téma: Priebeh štvrtého šprintu, prezentácia papierových prototipov

Vypracoval: Igor Homola

Opis stretnutia:

- Prezentácia papierového prototypu - taktiky (detailný popis papierového prototypu sa nachádza na stránke timu9)
    - Prototyp útočníka v sektore 4L (je to oblasť napravo od protivníkovho brankára). Potreba do implementovať:
        - Počet voľných spoluhráčov (počítajú sa tí hráči, ktorých agent vidí)
        - Rýchlosť protihráča a čas, za ktorý je schopný pristúpiť k danému hráčovi
        - Či je daný hráč voľný. Tento parameter sa určuje na základe času, za ktorý k nemu môže prísť protihráč
        - Diskusia k prototypu :spoluhráči by mali komunikovať a spoločne dosahovať ciele. Taktika by nemala mať len situačné rozhodovanie.
        - Je potrebné riadiť sa pravidlami robotického futbalu - kontrola či hráč neprihráva do offside.
    - Prototyp útok s prihrávkou
        - Útok zľava s prihrávkou – hráč z pravej strany ihriska prihrá predsunutému ľavému spoluhráčovi a ten pokračuje v útoku.
        - Útok sprava s prihrávkou – zrkadlová situácia ako v útok zľava s prihrávkou
        - Diskusia: navrhnuté prototypy by mali problém realizovať zložitejšie správanie. Napr. pri preberaní prihrávky by hráč čakal dokým lopta nepríde na požadované miesto a až následne by pokračoval v pohybe.
        - Vytvárať názvy taktík ktoré by reprezentovali taktiku
    - prototyp brankára
        - Diskusia: potreba uvažovať s rôznymi druhmi ukončenia taktiky brankára
            - protihráč je príliš blízko na to, aby brankár stihol zareagovať
            - spoluhráč príde brankárovi na pomoc – získa loptu
            - protivník kopol loptu a brankár stihne zareagovať

- Priebeh 4. šprintu

Problémy na riešenie
- pomenovanie taktik
- príliš krátke taktiky (mali by byt dlhšie)
- problém nadväznosti taktik

Vyriešené
- framework pre taktiky
- Situácie - riešenie je schválené
- vyber situácii
- funkcionalita selektora

To Do
- implementovať (prerušeniehighskillu po zmenení taktiky) taktika musípodaťinformáciu o svojom dokončení
- Identifikovať taktiky
- doplniť do komentára (hlavičky) taktiky na čo je a čorobíkonkrétna taktika
- rozdelenie taktik do balíčkov na útočné a obranne
- Riešenie rol hráča bude riešené pomocou situácii (brankár)
- implementovať do selektora seter ktorý bude vedieť meniť aj samotná taktika
- implementovať factory - automatizovaný spôsob inicializácie objektov
- Observer

Predošlé úlohy:

ID	**Člen tímu**	**Popis úlohy**
1	TomášNemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML
7	Benkovič/Linner	Papierový prototyp

Úlohy na ďalší týždeň:

ID	**Člen tímu**	**Popis úlohy**
1	TomášNemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML

8.10Zápis č.10 stretnutia tímu č.4

Dátum: 4.12. 2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Koniec štvrtého šprintu

Vypracoval: Michal Čerešňák

Opis stretnutia:

- **Kontrola úloh**
  - Parser XML – implementovaný
  - Stratégie a taktiky – implementované
  - Observer – sleduje taktiky a stratégie
  - Selektor – bez zmeny oproti minulému týždňu

- **Úlohy do budúceho týždňa**
  - Vyriešiť podmienky v taktikách
  - Dokončiť integráciu stratégií taktík a situácií
  - Otestovať funkčnosť integrovaných častí
  - Doplniť dokumentáciu
  - Aktualizovať webovú stránku
  - Odovzdať dokumentáciu v elektronickej podobe
  - Odovzdať dokumentáciu na stretnutí vo vytlačenej podobe
  - Odovzdať zoznam častí, ktoré sme robili my a ktoré druhý tím
    - Zoznam je spoločný pre oba tímy

Preberací protokol
Tímový projekt I

Projekt: Robocup3D

Odovzdávajúci tím: RFC Megatroll

Preberajúci: Ing. Marián Lekavý, PhD.

Dátum odovzdania: 11.12.2013

Odovzdané dokumenty: 1.tlačená dokumentácia k inžinierskemu dielu (počet strán 76)
2.tlačená dokumentácia k riadeniu (počet strán 105)

_____ _____

Vedúci tímu Ing. Marián Lekavý, PhD.