SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

TÍMOVÝ PROJEKT

Monitorovanie programátora v IDE

Autori:
Bc. Michal JURANYI
Bc. Ivan KOŠDY
Bc. Jozef MARCIN
Bc. Tomáš MARTINKOVIČ
Bc. Matej NOGA
Bc. Ján PODMAJERSKÝ
Bc. Juraj RABČAN

Pedagóg:
Doc. Mgr. Daniela CHUDÁ, PhD.

2013/2014

POĎAKOVANIE

Celý tím ďakuje Doc. Mgr. Daniele Chudej, PhD. za jej vedenie a cenné rady počas práce na tímovom projekte.

Obsah

1 Úvod **1**

1.1 Celkový pohľad na projekt 1

2 Šprint 1 **3**

2.1 Analýzy vykonané v 1. šprinte 3

2.1.1 Niektoré súvisiace diplomové a bakalárske práce vedené na FIIT STU 3

2.1.2 Metódy rozpoznávania používateľa 4

2.1.3 Logovací plugin projektu Perconik 6

2.2 Redmine 8

2.3 Webová stránka 8

2.4 Zhodnotenie šprintu 8

3 Šprint 2 **9**

3.1 Analýza logerov 9

3.1.1 Eclipse Metrics plugin 1.3.8 9

3.1.2 Rabbit 11

3.1.3 Fluorite 11

3.2 Porovnávanie vektorov 12

3.3 Inštalácia Gitu 12

3.4 Inštalácia databázy na server 13

3.5 Pristúpenie k dátam z externej databázy 13

3.6 Spring security a registrácia s prihlásením sa do aplikácie 13

3.7 Inštalácia GlassFishu na server a nasadenie aplikácie 14

4 Šprint 3 **15**

4.1 Identifikácia jednotlivých prvkov vektora 15

4.1.1 diGraphs (character, flightTime, latency) 15

4.1.2 graphs 15

4.1.3 leftToRightClickPart 15

4.1.4 numClicks 16

4.1.5 numLeftClicks 16

4.1.6 numMaxAppl 16
4.1.7 numMiddleClicks 16
4.1.8 numMinAppl 16
4.1.9 numMoves 16
4.1.10 numRightClicks 16
4.1.11 numScrolls 16
4.1.12 numNormAppl 16
4.1.13 trigraphs 17
4.2 Vytvorenie agregovaného vektora 17
4.3 Implementácia grafu na zobrazovanie výsledkov 18
4.4 Implementácia filtra a zobrazenie vyfiltrovaných dát 18
4.5 Preverenie získavania dát 19
4.6 Zhodnotenie stavu aplikácie po treťom šprinte 19
4.7 Súčasná architektúra systému 20
4.8 Ganttov graf 21

5 Šprint 4 **23**
5.1 Git repozitár pre Eclipse rozšírenie 23
5.2 Podpora správy tímov 23
5.3 Softvérové metriky pre sledovanie aktivity programátora 24
5.4 LOC metriky 24
5.4.1 Ako naprogramovať vlastný plugin 26
5.5 Agregácia vektorov 29
5.6 Rozšírenie pre riadiace premenné 30
5.7 Zhodnotenie šprintu 4 30
5.8 Ganttov graf 31
5.9 Burn-Down chart 34

6 Stanovenie globálnych cieľov projektu na letný semester **35**
6.1 Zhodnotenie po zimnom semestri 35

7 Šprint 5 a šprint 6 **37**
7.1 Spojenie pluginov 37
7.1.1 Úloha 37

7.1.2 Analýza . . . 37
7.1.3 Návrh . . . 37
7.1.4 Implementácia . . . 37
7.2 Vytvorenie prepojenia medzi projektami v Eclipse a vo webovej aplikácii . . . 38
7.2.1 Úloha . . . 38
7.2.2 Analýza . . . 38
7.2.3 Návrh . . . 38
7.2.4 Implementácia . . . 38
7.3 Vytvoriť základné grafy (bullet chart a spline chart) . . . 39
7.3.1 Úloha . . . 39
7.3.2 Analýza . . . 39
7.3.3 Návrh . . . 39
7.3.4 Implementácia . . . 40
7.4 Listener na Build, ktorý meria počet chýb . . . 40
7.4.1 Úloha . . . 40
7.4.2 Analýza . . . 40
7.4.3 Návrh . . . 41
7.4.4 Implementácia . . . 41
7.5 Zmena zrnitosti odosielania dát na Save súboru . . . 42
7.5.1 Úloha . . . 42
7.5.2 Analýza . . . 42
7.5.3 Návrh . . . 42
7.5.4 Implementácia . . . 42
7.6 Listener pre počet statických metód a atribútov cez AST . . . 43
7.6.1 Úloha . . . 43
7.6.2 Analýza . . . 43
7.7 Prezentácia . . . 43
7.7.1 Úloha . . . 43
7.7.2 Analýza . . . 43
7.8 Stránkovanie . . . 44
7.8.1 Úloha . . . 44
7.9 Odoberanie užívateľa . . . 44

7.9.1 Úloha 44
7.10 Dopĺňanie mena používateľa 44
7.10.1 Úloha 44
7.10.2 Analýza 44
7.10.3 Návrh 44
7.10.4 Implementácia 45
7.11 Opraviť zobrazovanie tabuliek pri nízkom rozlíšení 45
7.11.1 Úloha 45
7.11.2 Analýza 45
7.11.3 Návrh 45
7.11.4 Implementácia 46
7.12 Filtrovanie v administrátorskom móde 46
7.12.1 Úloha 46
7.12.2 Analýza 46
7.12.3 Implementácia 47
7.13 Web security anotácie 47
7.13.1 Úloha 47
7.13.2 Analýza 47
7.14 Triedenie administrátorských tabuliek 47
7.14.1 Úloha 47
7.14.2 Analýza 48
7.14.3 Implementácia 48
7.15 Úprava vzhľadu stránky 48
7.15.1 Úloha 48
7.15.2 Analýza 48
7.16 Návrh 48
7.16.1 Implementácia 49
7.17 Žiadosť o prístup ku projektu 49
7.17.1 Úloha 49
7.17.2 Analýza 49
7.18 Návrh 49
7.18.1 Implementácia 50

8 Šprint 7 **51**
8.1 Upraviť odosielanie dát do UACA 51
8.1.1 Úloha 51
8.1.2 Analýza 51
8.1.3 Návrh 51
8.1.4 Implementácia 51
8.2 Upraviť listenery na Save pre ukladanie viacej súborov naraz . . . 52
8.2.1 Úloha 52
8.2.2 Analýza 52
8.2.3 Návrh 52
8.2.4 Implementácia 52
8.3 Počet errorov pri Builde iba pre aktuálneho používateľa 52
8.3.1 Úloha 52
8.3.2 Analýza 53
8.3.3 Návrh 53
8.3.4 Implementácia 53
8.4 Odosielanie metrík pomocou JSON-u 53
8.4.1 Úloha 53
8.4.2 Analýza 54
8.4.3 Návrh 54
8.4.4 Implementácia 54
8.5 Vytvorenie koláčového grafu 55
8.5.1 Úloha 55
8.5.2 Analýza 55
8.5.3 Návrh 55
8.5.4 Implementácia 55
8.6 Zobrazenie údajov (serverové API pre grafy) 56
8.6.1 Úloha 56
8.6.2 Analýza 56
8.6.3 Návrh 56
8.6.4 Implementácia 56
8.7 Obmedzenie zobrazených údajov podľa práv používateľa 57
8.7.1 Úloha 57

8.7.2 Analýza . . . 57
8.7.3 Návrh . . . 57
8.7.4 Implementácia . . . 57
8.8 Úprava rátanie cyklomatickej zložitosti . . . 57
8.8.1 Úloha . . . 57
8.8.2 Analýza . . . 58
8.8.3 Návrh . . . 58
8.8.4 Implementácia . . . 58
8.9 Spojenie súčastí pluginu a update site . . . 59
8.9.1 Úloha . . . 59
8.9.2 Analýza . . . 59
8.9.3 Návrh . . . 59
8.9.4 Implementácia . . . 60
8.10 Oprava nemožnosti mazania tried v Eclipse . . . 60
8.10.1 Úloha . . . 60
8.10.2 Analýza . . . 60
8.10.3 Návrh . . . 61
8.10.4 Implementácia . . . 61
8.11 Odstraňovanie chýb v plugine . . . 61
8.11.1 Úloha . . . 61
8.11.2 Analýza . . . 61
8.11.3 Návrh . . . 61
8.11.4 Implementácia . . . 61
8.12 Listener na hĺbku dedenia . . . 62
8.12.1 Úloha . . . 62
8.12.2 Analýza . . . 62
8.12.3 Návrh . . . 62
8.12.4 Implementácia . . . 62
8.13 Upraviť meranie LOC metrík nad pridaným zdrojovým kódom . . . 63
8.13.1 Úloha . . . 63
8.13.2 Analýza . . . 63
8.13.3 Návrh . . . 63
8.13.4 Implementácia . . . 63

8.14 Parser na dáta 64
8.14.1 Úloha 64
8.14.2 Analýza 64

9 Šprint 8 **65**
9.1 Zmeniť logiku listenera chybovosti pri build-e 65
9.1.1 Úloha 65
9.1.2 Analýza 65
9.1.3 Návrh 65
9.1.4 Implementácia 65
9.2 Zmeniť získavanie ID projektu v plugine 66
9.2.1 Úloha 66
9.2.2 Analýza 66
9.2.3 Návrh 66
9.2.4 Implementácia 67
9.3 Duplicitný zdrojový kód na posielanie dát do UACA 67
9.3.1 Úloha 67
9.3.2 Analýza 67
9.3.3 Návrh 67
9.3.4 Implementácia 68
9.4 Listener čitateľnosti zdrojového kódu 68
9.4.1 Úloha 68
9.4.2 Analýza 68
9.4.3 Návrh 69
9.4.4 Implementácia 70
9.5 Zobrazovanie zaznamenaných metrík 70
9.5.1 Úloha 70
9.5.2 Analýza 70
9.5.3 Návrh 71
9.5.4 Implementácia 71
9.6 Modul porovnania podobnosti 71
9.6.1 Úloha 71
9.6.2 Analýza 71

9.6.3 Návrh . . . 72
9.6.4 Implementácia . . . 72
9.7 Verziovanie pluginu . . . 72
9.7.1 Úloha . . . 72
9.7.2 Analýza . . . 73
9.7.3 Návrh . . . 73
9.7.4 Implementácia . . . 73
9.8 Zmeniť logiku listenera chybovosti pri build-e . . . 73
9.8.1 Úloha . . . 73
9.8.2 Analýza . . . 74
9.8.3 Návrh . . . 74
9.9 Plagát . . . 74
9.9.1 Úloha . . . 74
9.9.2 Analýza . . . 74
9.9.3 Návrh . . . 74
9.10 Reset užívateľského hesla . . . 74
9.10.1 Úloha . . . 74
9.10.2 Implementácia . . . 75

10 Šprint 9 **76**
10.1 Testovanie inštalácie finálneho pluginu . . . 76
10.1.1 Úloha . . . 76
10.1.2 Analýza . . . 76
10.1.3 Návrh . . . 76
10.1.4 Implementácia . . . 77
10.2 Pridanie interaktívnosti spline chart grafu . . . 77
10.2.1 Úloha . . . 77
10.2.2 Analýza . . . 77
10.2.3 Návrh . . . 77
10.2.4 Implementácia . . . 78
10.3 Gui pre modul podobnosti . . . 78
10.3.1 Úloha . . . 78
10.3.2 Analýza . . . 78

10.3.3 Návrh 78
10.3.4 Implementácia 78

11 Šprint 10 **80**
11.1 Čitateľnosť kódu 80
11.1.1 Úloha 80
11.1.2 Analýza 80
11.1.3 Návrh 80
11.1.4 Implementácia 81
11.2 Finálna verzia pluginu za letný semester 81
11.2.1 Úloha 81
11.2.2 Analýza 81
11.2.3 Návrh 81
11.2.4 Implementácia 81
11.3 Zisťovanie autorstva kódu za pomoci knižnice JGit 82
11.3.1 Úloha 82
11.3.2 Analýza 82
11.3.3 Návrh 82
11.3.4 Implementácia 82

12 Celkový pohľad na projekt **83**
12.1 Analýza 83
12.2 Architektúra riešenia 83
12.3 Použité technológie 84
12.4 Opis výsledného produktu 84
12.4.1 Web Aplikácia 84
12.4.2 Eclipse plugin 88
12.5 Model databázy 91
12.6 Testovanie 92
12.7 Zhodnotenie 94

13 Používateľská príučka **96**

14 Prílohy **A-1**

14.1 Riadenie letný semester A-1

Zoznam obrázkov

1 5
2 Ukážka ako pracuje Perconik 7
3 Fyzický model externej databázy perconik 8
4 Ukážka webovej stránky 9
5 Ukážka grafu vo webovej aplikácii 18
6 Ukážka architektúry systému 20
7 Formulár pre správu projektov 24
8 Ukážka architektúry systému 36
9 Ukážka bullet chart diagramu 39
10 Ukážka spline chart diagramu 40
11 Ukážka dopĺňania 45
12 Ukážka tabuľky 46
13 Ukážka tabuľky 47
14 Ukážka triedenia 48
15 Ukážka zmeny GUI 49
16 Ukážka požiadávky o zaradenie do projektu 50
17 Ukážka priradeného projektu 50
18 Ukážka koláčového grafu 55
19 Ukážka implementovaných metód 59
20 Ukážka implementácie 60
21 Ukážka implementovaných metód 72
22 Ukážka implementovaných metód 73
23 Ukážka implementovaných metód 79
24 Ukážka kontroly podobnosti 79
25 Ukážka architektúry 83
26 Formulár potrebný na prepojenie s UACA 85
27 Formulár potrebný na prihlásenie 85
28 Formulár potrebný na vytvorenie projektu 86
29 Formulár potrebný na požiadanie o práva 86
30 Ukážka formuláru na pridanie používateľa a aj vymazanie 86
31 Ukážka grafov 87

32 Ukážka grafov 87
33 Ukážka kódu 90
34 Ukážka pluginu 90
35 Ukážka databázového modelu 92

1 Úvod

Dokument predstavuje technickú dokumentáciu k projektu Monitorovanie programátora v IDE. Projekt prebieha na Slovenskej technickej univerzite v Bratislave na Fakulte informatiky a informačných technológii v Bratislave. V tomto dokumente čítateľ nájde celkový pohľad na projekt, dokumentáciu k prebehnutým šprintom a aj ciele projektu.

1.1 Celkový pohľad na projekt

Našim hlavný cieľom je vytvorenie webovej aplikácie, ktorá analyzuje zozbierané dáta z monitorovania programátora v IDE Eclipse. Z dát je schopná vytvoriť kvalitatívny rebríček, ohodnotiť kvalitu a efektívnosť programátora. Tieto analýzy sa dajú využiť manažérmi v softvérových projektoch vyvíjaných v IDE Eclipse. Manažéri môžu sledovať efektivitu a výkonnosť programátorov v rámci tímu. Poskytne im to oveľa lepší pohľad na tím, ktorý môžu využiť pri osobnom ohodnotení programátora. Bez tejto aplikácie mohlo byť ohodnotenie založené iba na intuitívnosti a nerelevantnej metriky merania celkového času práce programátora. Prvý semester sa venujeme vytvoreniu prototypu webovej aplikácie, ktorá umožňuje zobrazovanie analýzy nalogovaných dát pomocou projektu Perconik. Poskytovaná webová služba v tomto projekte umožňuje jednoduché získanie týchto dát. Naša webová aplikácia ich sprístupňuje registrovaným používateľom. Monitorovaní programátori musia používať logovač PerConIK User Activity a plugin v Eclipse. Programátor pracujúci v IDE vykonáva množstvo rôznych operácií a akcií, ktoré sú pre neho charakteristické ako napríklad orientácia v systéme, štruktúra zdrojového kódu, pomenovávanie premenných, metód, tried, balíkov, využívanie akcií . Z týchto charakteristík sa dá vytvoriť model konkrétneho používateľa. Naša webová aplikácie z týchto údajov vytvára vektor predstavujúci model používateľa, ktorý obsahuje jeho atribúty práce s klávesnicou a myšou. Tieto vektory sme sa vytvorili na to, aby sme dokázali programátorov porovnávať. Na toto porovnanie sme sa rozhodli použiť známe algoritmy porovnávania vektorov (kosínusová vzdialenosť , manhatanská vzdialenosť, euklidova vzdialenosť). Na vyhodnotenie efektívnosti programátora používame softvérové metriky. Nami naprogramované softvérové

metriky ukladáme do pôvodného modelu externej databázy, pomocou mapovania na metriku selekcie textu. Zobrazovanie dát v aplikácii je intuitívne, poskytujeme aj grafy(kvôli často sa meniacemu vektoru je funkcia momentálne pozastavená) a delenie na tímy. Ku každému členovi tímu, ktorý vytvorí manažér sa dá vypísať vektor, umožňujeme porovnávať dvojice programátorov. V tomto semestri sme získavali nalogované dáta iba od členov tímu. V letnom semestri plánujeme pridávať ďalšie softvérové metriky a ukladať ich do zmeneného modelu v gratex databáze. Taktiež experimentovanie so zozbieranými nalogovanými dátami a ich podrobnú analýzu. To bude smerovať k určeniu tresholdov, tak by sme sa mali dostať ku korektnému porovnaniu programátorov. Plánujeme poskytnúť, resp. zozbierať viac dát aj od iných programátorov v IDE Eclipse, najmä od skúsených programátorov.

2 Šprint 1

2.1 Analýzy vykonané v 1. šprinte

V prvom šprinte boli vykonané prevažne analýzy. Ich prehľad sa nachádza v nasledujúcich kapitolách.

2.1.1 Niektoré súvisiace diplomové a bakalárske práce vedené na FIIT STU

V tejto časti sa nachádza stručný prehľad niektorých diplomových a bakalársky prác vedených na FIIT STU, ktoré sa venujú problematike identifikácie používateľa. Témou identifikácie používateľa sa čiastočne venujeme aj v našom tímovom projekte Monitor programátora v IDE.

Model používateľa charakterizovaný dynamikou písania na klávesnici

V tejto diplomovej práci sa autor Rastislav Kršák zameriava na identifikáciu používateľa. Túto identifikáciu používateľa vyhodnocuje počas celej práce s počítačom. Priebežne sa vyhodnocuje, či v danom okamihu sedí za počítačom prihlásená osoba. Práca poskytuje prehľad o realizovaných výskumoch v predmetnej oblasti a ich výsledkoch. Popisuje výskum v oblasti dynamiky písania na klávesnici a práce s myšou. Vysvetľuje základné pojmy a metódy súvisiace s touto problematikou. Modelovanie používateľa vykonáva pomocou klávesnice a polohovacieho zariadenia. Výsledkom práce je implementovaný systém, ktorý dokáže priebežne sledovať činnosť používateľa s počítačom a na základe toho overiť alebo určiť jeho identitu. To zahŕňa vytvorenie modelu používateľa založeného na vybraných charakteristikách, ktoré boli získané extrakciou z údajov získaných pri práci používateľa s klávesnicou a polohovacím zariadením.

Model používateľa pre jeho identifikáciu

V tejto diplomovej práci sa autor Robert Godány venuje tvorbe modelu používateľa pre jeho identifikáciu na základe práca s myšou a klávesnicou. Výsledkom je vytvorenie modulu do webových aplikácií, ktorý bol integrovaný do Moodle. Identifikácia používateľa prebiehala pri písaní textu. V práci sa zameriaval na statické

a dynamické charakteristiky.Extrahovali časy všetkých n-grafov, čo predstavuje čas od stlačenia prvej klávesy po stlačenie n-tej klávesy na klávesnici.

Podobnosti v slovenských textoch a v programových kódoch

V tejto diplomovej práci sa autor Marián Hraško zameriava súčasne na plagiátorstvo v slovenských textoch a v zdrojových kódoch. Výsledkom sú navrhnuté transformácie programových štruktúr v jazyku C++ na jednotné tvary pre jednoduchšie nájdenie podobnosti v zdrojových kódoch.

Vplyv biometrických charakteristík na model používateľa pre identifikáciu

V tejto bakalárskej práci sa autor Peter Krajník zameriava na analýzu niekoľkých prác venujúcich sa biometrickým systémom. Tieto systémy sú zamerané na dynamiku klávesových úderov a pohybov myši.

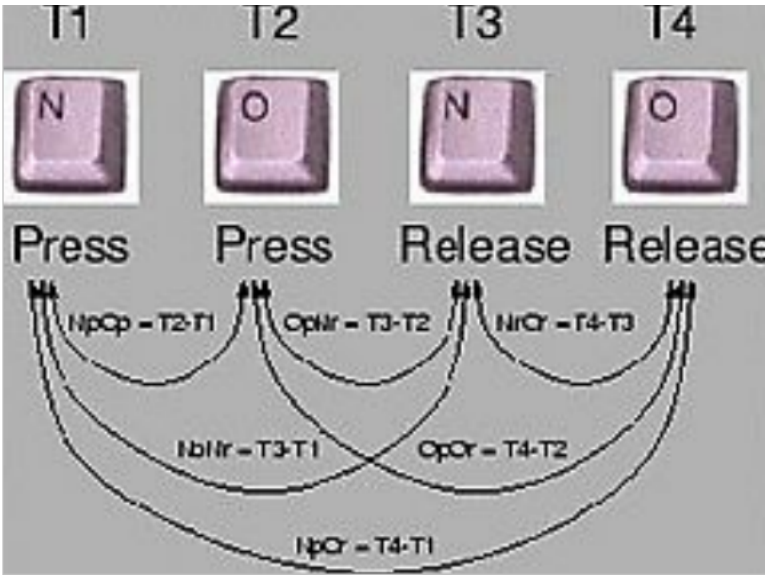
2.1.2 Metódy rozpoznávania používateľa

Pri vypracovávaní tejto úlohy sme vychádzali z vedeckých a diplomových prác. Výsledkom vyhľadávania údajov relevantných pre náš projekt bola najmä metóda Keystroke Dynamics. Táto metóda bola prijatá ako de facto štandard pri vyhodnocovaní identifikácie používateľa, ktorý v daný čas používa klávesnicu.

Keystroke Dynamics

- Proces analyzovania spôsobu akým používateľ píše, pričom sa jeho aktivita sleduje v milisekundách. Ide o zaužívané rytmické patterny, ktoré svojím písaním vytvára.
- Veľa prác sa venovalo tejto metóde, ktorá sa prijala ako nosná a výskumníci ju považujú za jednu z najlepších spôsobov určenia identity.
- Každý používateľ si vytvára špecifický podpis svojím písaním.
- Nespomína vytvorenie používateľského modelu, ale modeluje ho na základe jeho rytmiky písania.

- Sleduje sa:
  - čas oneskorenia – čas medzi stlačením klávesy dole a jej pustením
  - čas letu – čas medzi stlačením párov kláves. Napríklad čas od stlačenia dvoch kláves po uvoľnenie ďalších dvoch kláves



Obr. 1

  - čas stlačenia špecifických párov, trojíc, štvoríc(napr. e-a, a-b-c, d-a-n-o)
  - rôzne špecifiká
    - Či používateľ používa ľavý alebo pravý shift na napísanie L, alebo dokonca capslock
    - Štatistický výskyt chýb(kláves delete/backspace)
- Spôsoby vyhodnocovania :
- T – Test – Porovnáva sa štatistika dvoch sedení, nie veľmi presná(55%)
- Euclidova vzdialenosť – založený na porovnaní vzdialenosti medzi vektormi
  - $R = [r_1, r_2, ..., r_n]$ - prvý vektor
  - $R = [u_1, u_2, ..., u_n]$ - druhý vektor

  - $D(R,U) = [\sum_{i=1}^{N}(r_i - u_i)^2]^{\frac{1}{2}}$ - ak je nulová, zhoda je úplná

- Manhattanská vzdialenosť
- Cosínusová podobnosť
- Vážená vzdialenosť
  - Niektoré prvky sú viacej hodnoverné ako ostatné pretože pozostávajú z dlhších vstupov(nejakej vety napríklad).

Vzhľadom k vyššie prezentovaným skutočnostiam sme sa rozhodli na identifikáciu používateľa použiť metódu Keystroke Dynamics, kde budeme sledovať časy stláčania rôznych trojíc a dvojíc písmen. Vytvoríme vektor, ktorý bude okrem iného obsahovať aj tieto údaje. Tento vektor popíšeme v nasledujúcich kapitolách.

2.1.3 Logovací plugin projektu Perconik

Perconik je projekt poskytujúci logovací nástroj, ktorý používame na monitorovanie programátora. Cieľom tohto nástroja je zaznamenanie aktivity používateľa za účelom zistenia ako sa používateľ správa. Nástroj o programátorovi zachytáva rôzne informácie ako klikanie myšou či text, ktorý programátor skopíroval do IDE a veľa ďalších vecí. Aktivity, ktoré používateľ vykonáva sú zachytávané pomocou aplikácie *User Activity Client Application* (UACA), ktorá beží ako nezávislý proces. Zachytené aktivity si UACA ukladá do svojej internej databázy, odkiaľ sa postupne prenášajú do externej databázy. UACA zachytáva udalosti pomocou komponentov *Activity Watcher*, ktoré o zachytenej udalosti informuje *Activity Managera*. *Activity manager* ukladá udalosti zásobníka *Event Stack*. Do zásobníka sa ukladajú udalosti pre práve logovanú aktivitu. Po ukončení logovania aktivity je stav zásobníka uložený do lokálnej databázy UACY. UACA ma nastavený časový interval po, ktorom sa databáza vyprázdňuje a logy sa odošlú do externej databázy.Externá databáza sprístupňuje svoje dáta pomocou externej služby opísanej vo WSDL, ktorej sa pošle XML súbor a na jeho základe nám vráti požadované dáta.

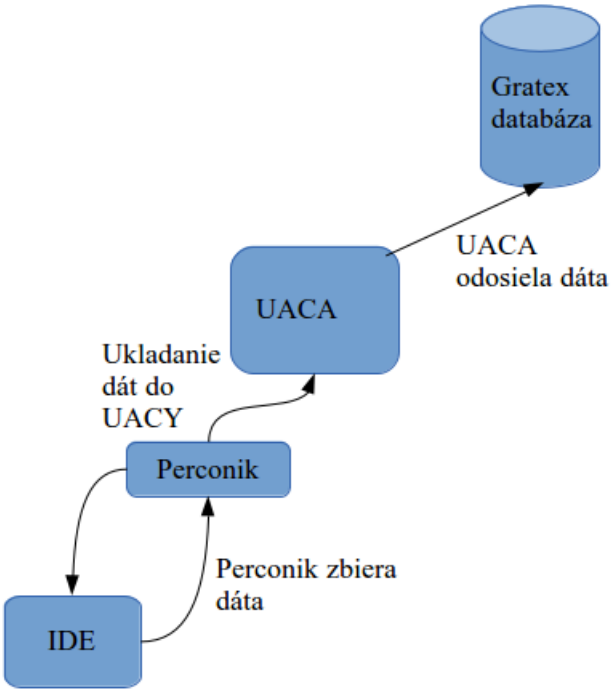
Perconik dokáže logovať používateľov v nasledujúcih IDE:

- Eclipse

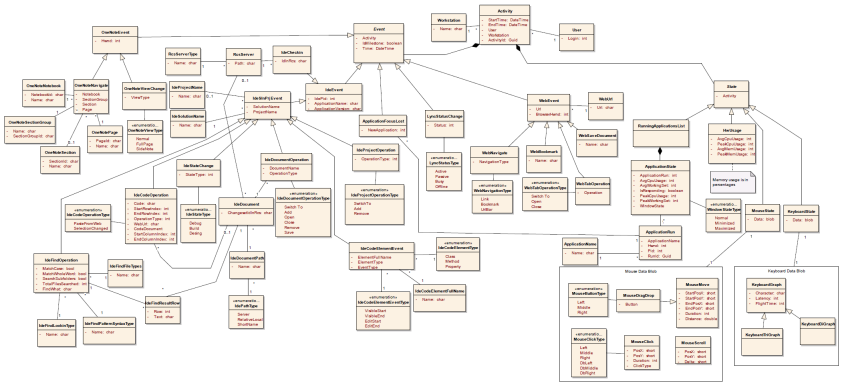
- MS Visual Studio od verzie 2010

Okrem toho dokáže logovať aj:

- MS Office OneNote od verzie 2010
- Stav spustených okien
- Stav spustených aplikácií
- MS Outlook a Lync od verzie 2010
- Klikanie myšou
- Aktivitu na klávesnici



Obr. 2: *Ukážka ako pracuje Perconik*



Obr. 3: *Fyzický model externej databázy perconik*

2.2 Redmine

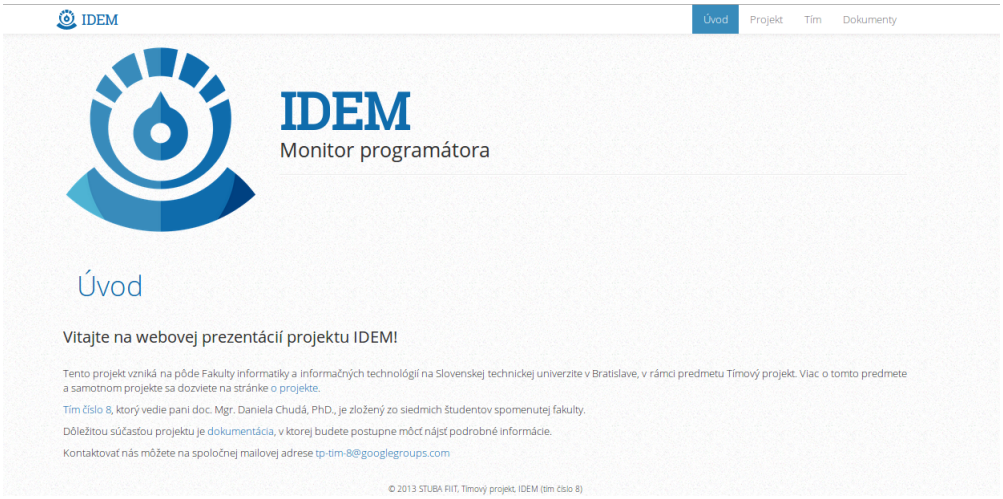
Monitorovanie práce a manažment úloh je pri tímovej práci veľmi dôležitý a preto sme museli začať používať softvér, ktorý by nám to uľahčil. Z veľkého množstva softvéru, ktorý bol pre tento účel vytvorený, sme sa rozhodli použiť Redmine. Redmine nám beží na školskom serveri a môžeme ho navštíviť na adrese: *http://team08-13.ucebne.fiit.stuba.sk:443/*

2.3 Webová stránka

Pre potrebu publikovania informácií o tíme bola vytvorená webová stránka. Stránka sa dá navštíviť na adrese: *http://labss2.fiit.stuba.sk/TeamProject/2013/team08is-si/index.html*. Na stránke sa dajú nájsť informácie o členoch tímu, dokumenty a informácie o projekte. Stránka je vytvorená staticky a obsahuje len HTML kód a kaskádové štýly.

2.4 Zhodnotenie šprintu

Tento šprint sa niesol v duchu robenia analýz potrebných pre vypracovanie projektu. Vybrali sme nástroj pre správu úloh a to Redmine. Bola vytvorená stránka tímu a tiež sme si stanovili nové úlohy do ďalšieho šprintu.



Obr. 4: *Ukážka webovej stránky*

3 Šprint 2

3.1 Analýza logerov

V tejto kapitole analyzujeme jednotlivé pluginy do vývojového prostredia Eclipse, ktoré zabezpečujú logovanie používateľovej aktivity. Cieľom je vybrať plugin, ktorý sleduje softvérové metriky a spôsob, ako ho do nášho projektu zakomponovať.

3.1.1 Eclipse Metrics plugin 1.3.8

- Podpora pre Eclipse 3.5 a vyššie
- Sledujú sa programátorske metriky
  - *McCabe Cyclomatic Complexity* Sleduje počet riadiacich premenných (else, if, switch, while, ternarne operatory atď.). Robí priemer pre všetky metódy nachádzajúce sa v projekte. Vracia konkrétne číslo pre jednu metódu a to je maximálne číslo z existujúcich metód.
  - *Number of Parameters* Počet parametrov metód v jednotlivých metódach, priemerný počet, maximálny počet.

  - *Nested Block Depth* Koľko krát sa v metóde volá konkrétna iná metóda. Tak isto obsahuje aj priemerný počet a maximálny počet
  - *Afferent Coupling(CA)* Počet tried mimo balíčka, ktoré závisia na triedach vnútri balíčka
  - *Efferent Coupling(CE)* Počet tried v balíčku, ktoré závisia na triedach v inom balíčku
  - *Abstractness(RMA)* Počet abstraktných tried (a interfejsov) delený celkovým počtom typov v balíčku
  - *Normalized Distance*. Predstavuje hodnotenie dizajnu balíčkovania. Čím je číslo menšie, tým lepšie. Jej hodnotu vypočítame $|RMA + RMI - 1|$
  - *Total Lines of Code* Celkový počet neprázdnych riadkov, nezaratáva komentáre.
  - *Number of Interfaces* Počet interfejsov.
  - *Number of classes* Počet tried.
  - *Number of Static Methods* Počet statických metód.
  - *Number of Static Methods* Počet metód v triede.
- Tento plugin nesleduje
  - počet znakov za daný čas
  - počet klikov myši
  - celkový čas strávený tvorbou programu
  - metriky vývojového prostredia(kto kopíroval odkiaľ kam, aké príkazy používateľ používa najčastejšie atď.)
- Metriky sa nezaznamenávajú periodicky, ale ak sa niečo uloží, alebo ak sa prepne myšou medzi triedami, tak až potom sa zaznamenávajú. Metriky sa dajú exportovať do XML súboru, avšak nie je to automatizované.
- Ak chceme XML vytvoriť, stačí kliknúť na tlačidlo XML aplugin XML vytvorí a vyexportuje, kde používateľ chce. Avšak, metriky je možné pozerať

si v samostatnom view, ktorý plugin do Eclipse vytvorí. V rámci tohto view si používateľ môže jednotlivé hodnoty prezerať v reálnom čase.

- Plugin nie je možné voľne upravovať.

3.1.2 Rabbit

Plugin je podporovaný v Eclipse 3.4 až 3.6. Tento plugin sa už aktívne nevyvíja. Zobrazenie údajov je možné len v eclipse, pretože sa neukladajú do XML súbora. Sledujú sa metriky časového použitia Eclipse a to:

- čas strávený kompilovaním
- čas strávený používaním jednotlivých perspektív
- čas práce na úlohách
- tvorbou nejakej metódy alebo triedy
- Používané príkazy(cut, copy, paste)

3.1.3 Fluorite

Plugin loguje character typing:

- Posúvanie kurzora
- Zmeny označeného textu
- Loguje príkazy a aj parametre príkazov
- Kopírovanie textu s uvedenými hodnotami odkiaľ, kam a dokonca aj čo
- Loguje zmeny dokumentu

Dáta sa logujú do XML, pričom tieto XML súbory sú strašne veľké. Za 5 minút skúšania sa vygeneroval XML súbor, ktorý mal viac ako 700 riadkov. Nepodarilo sa nájsť dokumentáciu o tom ako sa XML súbor má správne rozparsovať. Prístup k logom je možný až po ukončení Eclipse. Logujú sa aj skratky používané v Eclipse, napríklad Ctrl + Space. Logujú sa aj eventy ako otvorenie triedy v prehliadači

súborov. Zaloguje to, ako presne trieda vyzerá, importy balíčkov, tried atď. **Zhodnotenie analýzy logerov** Vyššie spomenuté pluginy síce sledujú potrebné aktivity používateľa, ktoré dokážu vypovedať o identite používateľa, kvalite jeho práce, prípadne sledovať aj iné činnosti ako kopírovanie údajov, avšak ich prepojenie so sledovačom systému PerCoNik je nemožné, preto ich nebudeme v ďalšej práci uvažovať. Používať teda budeme iba sledovač systému PerCoNik, v rámci ktorého si vytvoríme vlastný sledovač softvérových metrík.

3.2 Porovnávanie vektorov

Porovnanie vektorov je dôležité z hľadiska zistenia podobnosti dvoch vektorov. Naše vektory budú obsahovať rôzne údaje, ktoré určitým spôsobom charakterizujú používateľa. Používateľský model vektora bude vysvetlený v ďalších kapitolách. V tejto úlohe sa budeme zaoberať najpoužívanejšími metódami porovnania podobnosti dvoch vektorov a vytvoríme funkcionalitu, ktorá zabezpečí rátanie podobnosti.

V tomto šprinte bola vytvorená trieda, ktorá obsahuje metódy na porovnávanie vektorov. Nech P a Q sú vektory, $p_1...p_n$ sú prvky vektora P a $q_1...q_2$ sú prvky vektora Q.

- *Euclidová vzdialenosť*: $d(Q,P) = \sqrt[2]{|q_1 - p_1| + |q_2 - p_2| + ... + |q_n - p_n|}$
- *Manhattanská vzdialenosť* $D(Q,P) = |q_1 - p_1|^2 + |q_2 - p_2|^2 + ... + |q_n - p_n|^2$
- *Manhattanská vzdialenosť*$(\dot{Q}P) = \frac{\sum_{i=1}^{n} \mathbf{q}_i \mathbf{p}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{q}_i)^2} \sqrt{\sum_{i=1}^{n} (p_i)^2}}$

Vytvorením metód sme vytvorili počiatočnú funkcionalitu porovnávania vektorov a vyhodnocovania podobnosti dvoch vektorov, čo v konečnom dôsledku znamená zistenie podobnosti dvoch používateľov. Táto časť funkcionality ešte bude doplňovaná podľa potreby. Keďže najpresnejšia sa nám vidí cosínusová podobnosť, budeme používať predovšetkým práve ju.

3.3 Inštalácia Gitu

V tomto šprinte bolo načase vybrať si jeden z mnoha nástrojov, ktorý by nám umožnil správu zdrojového kódu. Nástroj sme museli vybrať preto, lebo sme

potrebovali rozšíriť workspace medzi všetkých členov tímu. Rozhodnutie padlo na necentralizovaný a distribuovaný systém riadenia revízií GIT. GIT bol nainštalovaný na server a každý člen tímu si ho nainštaloval lokálne. Ku Gitu dostali členovia tímu, ktorý snim nikdy nerobili školenie od členov, ktorý skúsenosti s nástrojom už mali. Pôvodne boli vytvorené dve vetvy: *Master* a *develop*. Neskôr si každý člen tímu vytvoril svoju vetvu. Inštaláciou Gitu sme docielili rozšírenie aktuálnej verzie workspace ku všetkým členom tímu a tiež každý člen tímu odteraz mohol prispieť do zdrojového kódu aplikácie.

3.4 Inštalácia databázy na server

V tomto šprinte sme sa rozhodovali aku databázu budeme používať. Po dlhej diskusii sme sa rozhodli pre MongoDB. MongoDB je NoSql databáza s dobrou podporou vo frameworku Spring, ktorý používame pri vývoji aplikácie. Databázu si každý musel nainštalovať aj lokálne, pretože aktuálna verzia aplikácia vyžaduje databázu na stroji, kde aplikácia beží.

3.5 Pristúpenie k dátam z externej databázy

Prostredníctvom webovej služby vieme dostať údaje z databázy Gratexu. Problém je, že tieto údaje sú dosť neprehľadné a ťažko sa v nich orientuje. Preto je potrebná analýza týchto údajov.

3.6 Spring security a registrácia s prihlásením sa do aplikácie

Počas šprintu došlo k nastaveniu Security Springu, čiže používateľ, ktorý má obmedzené práva môže stránku vidieť inakšie ako používateľ s plnými právami. Bol vytvirený registračný formulár, ktorý umožní používateľovi registrovať sa do aplikácie a bol vytvorený aj druhý formulár, ktorý používateľovi umožní prihlásenie do aplikácie.

3.7 Inštalácia GlassFishu na server a nasadenie aplikácie

Pretože aplikácia je vyvíjaná ako webová aplikácia, tak musela byť umiestnená na web, aby bola prístupná zvonka. Aby sme aplikáciu mohli umiestniť na server bolo potrebné nainštalovať jeden z mnoha aplikačných serverov. Nakoľko sme už lokálne používali server GlassFish, ktorý nám lokálne fungoval dobre, tak rozhodnutie bolo rýchle. Na tímový server bol nainštalovaný GlassFish najnovšej verzie. Po jeho nainštalovaní aplikácia nasadená na tímový server. Tým sa aplikácia stala prístupná zvonka a môže byť využitá kýmkoľvek kto má url adresu stránky cez, ktorú sa ovláda aplikácia.

4 Šprint 3

4.1 Identifikácia jednotlivých prvkov vektora

V úlohe bolo potrebné vytvoriť vektor zodpovedajúci modelu používateľa. Tento vektor je vytvorený najmä z metrík, ktoré sledujú klávesnicu a myš. Softvérové metriky budú zakomponované neskôr. K realizácií tejto úlohy bolo potrebné:

- Analyzovanie dát potrebných k vytvoreniu vektora
- Výber dát a ich následné spracovanie do vektora

4.1.1 diGraphs (character, flightTime, latency)

1. character – Dvojica písmen napr. „AB“, „RC“, atď.
2. flightTime – čas, milisekundy
3. latency – čas, milisekundy

Keďže prakticky každý model má viacero dvojíc, niekedy spoločných, niekedy dokonca úplne rozdielnych, vybrať reprezentatívnu zložku nebude práve najlepšie. Preto vyberieme priemer hodnôt.

Položka vektora V1 sa teda vyráta takto: $V_1 = 1/n \sum_(i = 0)^n P_i$ Kde – n je počet nenulových položiek v liste diGraphs. A P_i je číslo vyrátané z hodnôt:
$P_i = (flightTime + latency)/2$

4.1.2 graphs

podobne ako 1., rovnaká trojica, avšak character sa skladá len z 1 písmena. Ráta sa teda rovnako ako 1. Položka vektora V2 - je teda double číslo vyrátane rovnako avšak $P_i = (flightTime + latency)/1$

4.1.3 leftToRightClickPart

V3 = double číslo skopírované z tejto položky.

4.1.4 numClicks

Počet klikov, V4 = skopírované číslo z tejto položky

4.1.5 numLeftClicks

Počet klikov ľavým tlačidlom, V5 = skopírované číslo z tejto položky

4.1.6 numMaxAppl

len skopírujeme toto číslo do nášho vektora, konkrétne do položky V6

4.1.7 numMiddleClicks

Počet klikov stredným tlačidlom – iba skopírujeme do položky V7

4.1.8 numMinAppl

Počet minimalizovaných aplikácií, iba toto číslo skopírujeme do položky V8

4.1.9 numMoves

Iba skopírujeme do položky vektora V9

4.1.10 numRightClicks

Iba skopírujeme do položky vektora V10

4.1.11 numScrolls

Skopírujeme do položky vektora V11

4.1.12 numNormAppl

Počet spustených aplikácií, skopírujeme do položky vektora V12

4.1.13 trigraphs

Podobne ako bod 1., avšak vzorce sú: $P_i = (flightTime + latency)/3\ V12 = 1/n \sum_(i = 0)^n P_i$

Výsledkom tejto úlohy je vytvorenie štandardu pre formu vektora, ktorý charakterizuje používateľa v rámci metrík z klávesnice a myši. Ak to bude potrebné v rámci ďalšieho vývoja projektu, tento vektor ešte môže byť doplnený, prípadne modifikovaný iným spôsobom.

4.2 Vytvorenie agregovaného vektora

Úloha je zameraná na vytvorenie dátovej štruktúry, v ktorej budeme uchovávať vektory aktivít používateľov na ich ďalšie spracovávanie a vyhodnocovanie. Pre splnenie tejto úlohy bolo potrebné vytvorenie triedy reprezentujúcej dátovú štruktúru ukladanú do databázy, pričom trieda obsahuje:

- ID používateľa PerCoNik
- Čas začiatku a konca aktivity
- Vektor reprezentujúci nalogované dáta v rámci aktivity

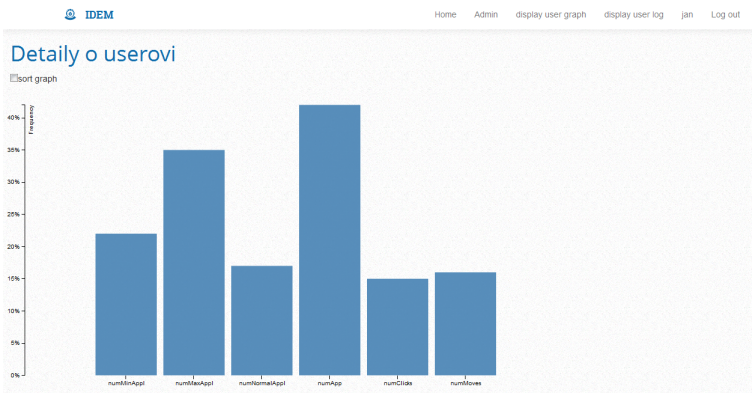
Ďalší dôležitý krok bol vytvorenie triedy agregovaného vektora, táto trieda sa skladá zo všetkých vektorov jednotlivého používateľa. Agregovaný vektor je štruktúra obsahujúca

- ID používateľa PerCoNik
- Čas začiatku a konca aktivity
- Vektor reprezentujúci nalogované dáta v rámci aktivity

Výsledkom úlohy je trieda reprezentujúca dátovú štruktúru ukladaných údajov o aktivite používateľa vo forme vektoru a metadát o aktivite. Túto triedu budeme využívať na ukladanie vytvorených vektorov do databázy.

4.3 Implementácia grafu na zobrazovanie výsledkov

Vytvorili sme triedu na grafickú reprezentáciu vektora modelu používateľa. Táto trieda zobrazí všetky hodnoty vektora na grafe. Graf zobrazuje hodnoty dynamicky, pri zobrazení sa hodnoty najprv zobrazí a v štandardnom poradí, ale po pár sekundách sa tieto hodnoty utriedia od najväčšej po najmenšej, tak poskytnú kompletný prehľad.



Obr. 5: *Ukážka grafu vo webovej aplikácii*

Tento graf je naprogramovaný v javascripte v d3 frameworku. Hodnoty na zobrazenie sú poslané do metódy, ktorá ich jednoducho spracuje a zobrazí na grafe.

4.4 Implementácia filtra a zobrazenie vyfiltrovaných dát

Potom ako sme mali dostatok údajov mohli sme vizualizovať dáta. Preto bol implementovaný filter, ktorý umožní vybrať agregované vektory používateľov, pričom sa môže filtrovať podľa dátumu, maximálnej dĺžky logu, minimálnej dĺžky logu a ešte aj podľa používateľského mena v Perconiku. Podľa toho aký používateľ zadá filter sa vytvorí najskôr query do MongoDB databázy a vytiahne všetky agregované vektory pre používateľa, ktorý spĺňa kritéria filtra. Vyťahuje sa tak, aby bol najmladší vektor na prvom mieste listu, ktorý vráti databáza. Ďalší krok je výpočet podobnosti medzi vektormi. Potom ako sa tento proces vykoná, pošle sa do JSP stánky atribút pomocou anotácie, ktorá je na to určená. V JSP sa pomocou iterátora zobrazí obsah vrátených vektorov aj s vypočítanou podobnosťou do tabuľky.

4.5 Preverenie získavania dát

V predchádzajúcom šprinte sme mali problém, lebo aj napriek tomu, že sme logovali dáta z IDE, tak nám ich volaná webová služba nikdy nevrátila. Vrátila len dáta mimo IDE. Po analýze a konzultácii s tvorcami Perconika a UACY sme zistili, že služba dáta vracia, ale my ku ním nepristupujeme. Po miernej úprave aplikácie sa už k dátam vieme dostať.

4.6 Zhodnotenie stavu aplikácie po treťom šprinte

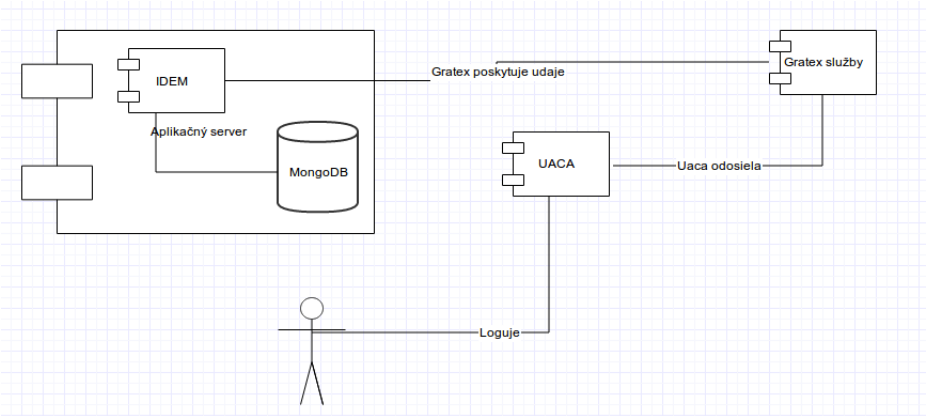
Aplikácia je vyvíjaná v Jave 1.7 a frameworku Spring MVC. Vzhľadom na technológie a webový charakter aplikácie sme sa ju rozhodli nasadiť na aplikačný server GlassFish. Tento server už bol nainštalovaný aj na náš tímový server. Aplikácia pracuje s NoSql databázou MongoDB.

- Naša aplikácia má momentálne implementované prepojenie s externou databázou. Vieme z nej prebrať údaje aj poslať ďalšie údaje, ktoré sa do nej zapíšu.
- Na logovanie programátora používame logovací systém projektu Perconik.
- Do aplikácie boli implementované algoritmy, ktoré slúžia na porovnávanie vektorov a to euklidova vzdialenosť, manhattanská vzdialenosť a kosínusová vzdialenosť.
- Podľa údajov, ktoré dostávame z externej databázy sme identifikovali model používateľa. Podľa tohto modela bola vytvorená trieda, ktorej úlohou je vytvorenie agregovaného vektora z modela používateľa.
- Do aplikácie sa môže používateľ registrovať a prihlásiť. Bolo nastavené Spring security a autentifikácia.
- Sú vytvorené dve okná slúžiace najmä pre debugovanie. Jedno okno zobrazuje surové dáta z externej databázy a druhé zobrazuje modely používateľa podľa zadaného mena pričom dáta berie tiež z externej databázy.

- V aplikácii je okno v ktorom je filter a po jeho vyplnení sa zobrazia agregované vekory, ktoré prislúchajú konkrétnemu používateľovi. Pri vypĺňaní filtra sa môže vybrať aký druh podobnosti sa bude s vektorov počítať.

4.7 Súčasná architektúra systému

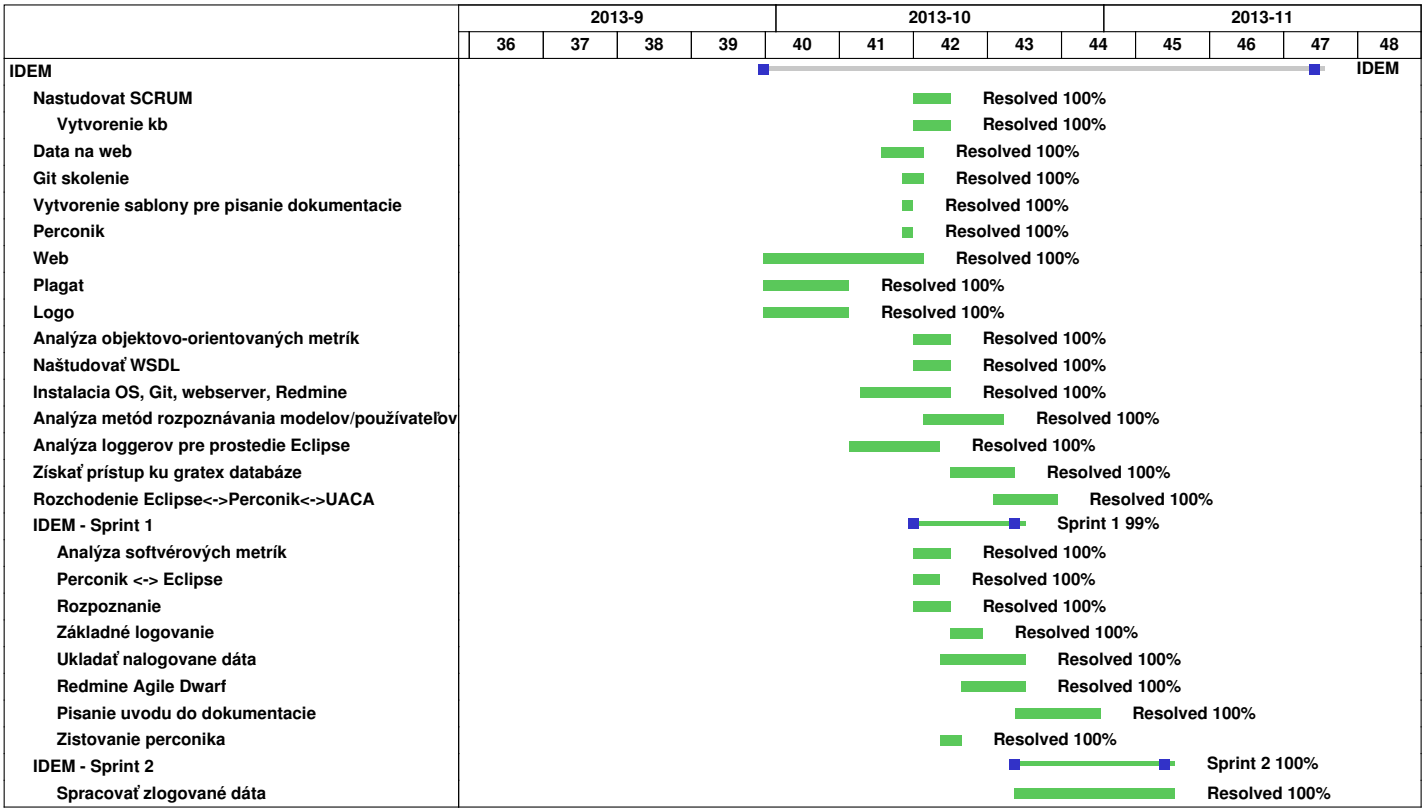
Súčastná architektúre systému je vyjadrená nasledujúcim obrázkom:

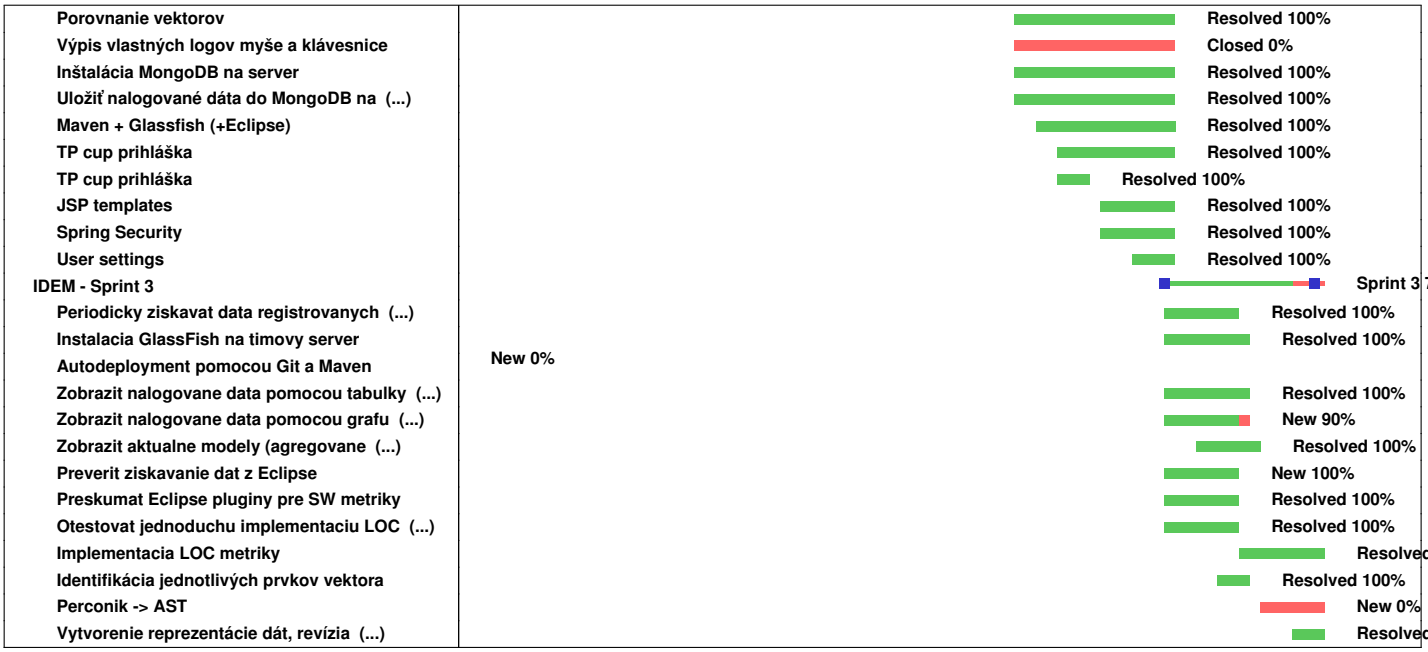


Obr. 6: *Ukážka architektúry systému*

4.8 Ganttov graf

IDEM




Porovnanie vektorov
Resolved 100%
Výpis vlastných logov myše a klávesnice
Closed 0%
Inštalácia MongoDB na server
Resolved 100%
Uložiť nalogované dáta do MongoDB na (...)
Resolved 100%
Maven + Glassfish (+Eclipse)
Resolved 100%
TP cup prihláška
Resolved 100%
TP cup prihláška
Resolved 100%
JSP templates
Resolved 100%
Spring Security
Resolved 100%
User settings
Resolved 100%
IDEM - Sprint 3
Sprint 3 79%
Periodicky ziskavat data registrovanych (...)
Resolved 100%
Instalacia GlassFish na timovy server
Resolved 100%
Autodeployment pomocou Git a Maven
New 0%
Zobrazit nalogovane data pomocou tabulky (...)
Resolved 100%
Zobrazit nalogovane data pomocou grafu (...)
New 90%
Zobrazit aktualne modely (agregovane (...)
Resolved 100%
Preverit ziskavanie dat z Eclipse
New 100%
Preskumat Eclipse pluginy pre SW metriky
Resolved 100%
Otestovat jednoduchu implementaciu LOC (...)
Resolved 100%
Implementacia LOC metriky
Resolved 100%
Identifikácia jednotlivých prvkov vektora
Resolved 100%
Perconik -> AST
New 0%
Vytvorenie reprezentácie dát, revízia (...)
Resolved 100%

5 Šprint 4

V tomto šprinte sa pracovalo na nových metrikách pre logovací systém projektu Perconik. Toto si vyžiadalo aj vytvorenie nového repozitára. Ďalej sa pridala správa projektov, ku ktorej bola vytvorená správa používateľov ku jednotlivým projektom. Opis vykonaných úloh nasleduje v nasledujúcich kapitolách.

5.1 Git repozitár pre Eclipse rozšírenie

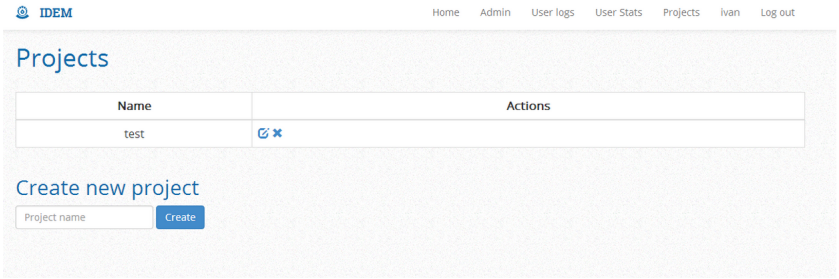
Systém monitorovania programátora pozostáva v podstate z dvoch dôležitých častí: klientskej aplikácie zaznamenávajúcej aktivitu programátora a webovej aplikácie zobrazujúcej získané a zanalyzované údaje. Pre rozšírenie možností klienskej aplikácie potrebujeme vytvoriť vlastné rozšírenie pre prostredie Eclipse, získavajúce požadované údaje.

Pre podporu vývoja tohto rozšírenia je potrebný samostatný Git repozitár, aby bolo možné využívať výhody systému pre správu verzií a aby bolo klientské rozšírenie oddelené od webovej aplikácie. Bol preto vytvorený nový repozitár, do ktorého boli zároveň nahrané knižnice projektu Perconik, slúžiace na vývoj rozšírenia. Takto je možné aj jednoducho aplikovať aktualizácie na používané podporné knižnice.

5.2 Podpora správy tímov

Podpora správy tímov Keďže náš projekt nemá monitorovať iba individuálnych programátorov a poskytovať informácie im samým, bolo potrebne implementovať do systému funkcionalitu správy projektových tímov, ktorá umožní prehľadne zobraziť informácie o skupine programátorov a porovnávať ich. Táto funkcionalita zahŕňa vytváranie projektov, ktoré sú logickým prepojením skupiny programátorov a správu ich členov. Po vytvorení projektu a pridaní príslušných programátorov do tohto projektu, je možné zobraziť štatistiky tímu v podobe zoznamu členov a ich modelov. Plánované je reprezentovanie vektorov pomocou čiarových, či stĺpcových diagramov a možnosť zobraziť vzájomnú podobnosť jednotlivých používateľov pomocou porovnávania vektorov.

Správa projektových tímov tak umožní vedúcim projektov získať prehľadné informácie o členoch tímov z hľadiska efektivity práce a softvérových metrík vytváraného softvéru.



Obr. 7: *Formulár pre správu projektov*

5.3 Softvérové metriky pre sledovanie aktivity programátora

Pre vytvorenie správneho modelu programátora potrebujeme zistiť ako kvalitne programuje, nestačí len vedieť aké okná má pootvárané, aké stránky navštevuje, aké klávesy stláča, alebo ako hýbe myšou. Tieto metriky sú všeobecné pre všetkých používateľov počítača. My ale chceme vytvoriť monitorovanie programátora, preto potrebujeme o používateľovi zistiť ďalšie atribúty, softvérové metriky, avšak tie PerConik neposkytuje. Keďže zaznamenávame aktivitu programátora cez Perconik plugin v IDE Eclipse, potrebujeme vytvoriť vlastné perconik pluginy.

5.4 LOC metriky

Na vyhodnocovanie efektívnosti programátora je potrebné sa zaoberať softvérovými metrikami, a preto sme sa rozhodli na stretnutí na základe dohody na začiatku implementovať jednoduché softvérové metriky.

Prvotným nápadom bolo spraviť softvérové metriky LOC (lines of code), ktoré vyhodnocujú projekt na základe jeho veľkosti. Pôvodný nápad sme rozvinuli tak, že budeme vyhodnocovať softvérové metriky LOC nad daným dokumentom v Eclipse a to tak, že sa najskôr vyhodnotia LOC nad otvoreným nezmeneným dokumentom v našej aplikácií a následne sa pri uložení tohto dokumentu opäť vyhodnotia. Na základe tohto by bolo možné vyhodnotiť LOC metriky len pre tú časť zdrojového kódu, ktorá sa zmenila pri práci programátora nad týmto dokumentom. Problémom

pri tomto návrhu bolo, že by bolo potrebné odosielať celý zdrojový kód z daného dokumentu do databázy pri jeho otvorení a uložení. Potom až v našej aplikácii porovnať dokumenty, čím by sme zistili pridané, zmenené a odstránené časti zo zdrojového kódu a následne vyhodnocovať LOC, pričom tento spôsob by mohol odrádzať používateľa používať našu aplikáciu v dôsledku posielania celého zdrojového kódu z každého otvoreného dokumentu. Taktiež pri prvotnom preskúmaní pluginu Perconik v Eclipse sme narazili na to, že nepodporuje listener v Eclipse pre uloženie daného dokumentu. Následne sme návrh LOC metrík upravili tak, že ich meriame nad dokumentom, v ktorom programátor práve upravuje zdrojový kód. LOC metriky sa vyhodnocujú priamo pri písaní zdrojového kódu s nízkou zrnitosťou predstavujúcu pridanie nového znaku do dokumentu. Vyhodnocovanie LOC metrík sme implementovali ako rozšírenie pluginu Perconik o náš vlastný zaregistrovaný listener. Vytvorili sme tieto LOC metriky:

- celkový počet riadkov daného dokumentu
- celkový počet zmenených riadkov
- celkový počet riadkov komentáru (//, /**/)
- celkový počet riadkov bez komentáru
- celkový počet prázdnych riadkov

Na základe celkového počtu riadkov daného dokumentu môžeme získať informáciu o programátorovi, či viacej upravuje zdrojový kód, čím ho môže zjednodušovať, alebo pridáva častejšie nový zdrojový kód a komentáre. Vyhodnotením celkového počtu zmenených riadkov môžeme získať informáciu, či sa programátor viacej venuje vlastnému písaniu zdrojového kódu, respektíve maximálnym kopírovaním jedného riadku, alebo častejšie kopíruje väčšie bloky zdrojových kódov. Na základe celkového počtu riadkov komentáru môžeme zistiť, či programátor viacej píše komentáre a na základe celkového počtu riadkov bez komentáru, či programátor viacej píše zdrojový kód. Rozdielom týchto dvoch posledných LOC metrík môže získať informáciu o pomere komentáru ku zdrojovému kódu, kde nás môže zaujímať či nie sú komentáre príliš dlhé. Následne celkový počet prázdnych riadkov môže poodhaliť informáciu zbytočného rozširovania zdrojového kódu o

prázdne riadky. Z týchto všetkých LOC metrík sa dajú ich kombináciami získať ďalšie rôzne zaujímavé informácie o efektívnosti programátora.

5.4.1 Ako naprogramovať vlastný plugin

Pre vyhodnocovanie efektívnosti programátora je potrebné merať softvérové metriky nad zdrojovým kódom, ktorý pridáva, odstraňuje a upravuje. Rozhodli sme sa preto použiť projekt Perconik, ktorý nám umožňuje zachytávať aktivity programátora v IDE Eclipsu a odosielať tieto dáta do databázy. Avšak plugin PerConIK Eclipse Integration je framework, ktorý neposkytuje zachytávanie všetkých aktivít v IDE. Preto je potrebné ho rozšíriť o zaregistrovanie požadovaného listeneru. Na jeho registrovanie je potrebné vytvoriť plugin projekt a nastaviť dependencies na všetky balíky z projektu PerConIK Eclipse Integration a pridať požadované balíky z org.eclipse, ktoré si vyžaduje vlastný listener. V extensions je potrebné vytvoriť nový provider, kde je potrebné takto zaregistrovať vlastný listener:

```
import sk.stuba.fiit.perconik.core.services.listeners.ListenerProvider;
import sk.stuba.fiit.perconik.core.services.listeners.ListenerProviderFactory;
import sk.stuba.fiit.perconik.core.services.listeners.ListenerProviders;

  public class MyProvider implements ListenerProviderFactory {
    public myprovider () {
    // TODO Auto-generated constructor stub
  }
  @Override
  public ListenerProvider create(ListenerProvider parent) {
    return ListenerProviders.builder()
    .add(MyListener.class)
    .parent(parent)
    .build();
  }
}
```

Ukážka 1: *Ukážka vygenerovaného listenera*

Následne je potrebné vytvoriť vlastnú triedu, v ktorej sa bude nachádzať vlastný listener rozširujúci ľubovoľne zvolený adaptér z PerConIK Eclipse Integration. Potrebné je override zvolenej ponúkanej metódy z adaptéra a následne vytvoriť vlastnú implementáciu metódy na základe dát, ktoré chceme v tejto metódy z IDE Eclipsu zachytávať. Ukážka príkladu vlastného listeneru:

Tento prístup nám umožní jednoducho vytvárať ľubovoľné vlastné listenery, v ktorých môžeme zachytávať rôzne aktivity z IDE Eclipsu.

```
import org.eclipse.jface.text.DocumentEvent;
import sk.stuba.fiit.perconik.core.adapters.DocumentAdapter;

public class MyListener extends DocumentAdapter {
  @Override
    public void documentChanged(DocumentEvent event) {
      int lineofdoc = event.getDocument().getNumberOfLines();
      System.out.println("Total number of lines: " + lineofdoc);
    }
  }
}
```

Ukážka 2: *Ukážka prikladu vlastného listeneru*

Mapovanie posielania dát z vlastných pluginov

Keďže databáza, kde uaca posiela všetky získané údaje o používateľovi je pre nás blackbox, nemáme k nej priami prístup, a požiadavky na zmenu dátového modelu by ľudia v gratexe mohli riešiť dlho, rozhodli sme sa naše zozbierané softvérové metriky posielať v existujúcom modely. Namapujeme ich na metriku označeného textu. Táto metrika je príliš nebezpečná, preto sme sa rozhodli prepisovať ju. Programátori, ktorí by nepoužívali náš produkt s vedomím, že ukladáme všetky ich označené texty. Stačí ak si kopírujú heslo do databázy a už je uložené aj v databáze v gratexe. Z toho dôvodu táto metrika nebude chýbať k analýze programátorov.

Na ukladanie našich získaných údajov sme sa rozhodli používať gratex databázu aj preto, že na spätné získavanie dát a ich analýzu sa dá aj naďalej používať rovnaká webová služba ako doteraz, bez zmeny. V budúcnosti plánujeme ukladať naše metriky do upraveného modelu gratex databázy. Teraz ale požadujeme rýchlu funkčnosť prototypu.

Ukážka štruktúry posielaných údaje z metriky označovania textu v uace:

My v $< code >< /code >$ posielame údaje, ktoré získajú naše pluginy, a to štýlom $< code >$ "&idemProjekt&"{json, ktory cheme poslat}$< /code >$. V tomto jsone je atribút metriky, ktorá údaje posiela, teda metriky statických metód posielajú takéto xml:

Celú túto funkčnosť môžeme dosiahnuť vďaka použitiu tried z perocnik projektu. Na odosielanie potrebujeme triedy z balíku com.gratex.perconik.activity, konkrétne UnderlyingDocument a IdeDataTransferObjects. Metóda performWatcherServiceOperation umožňuje poslať dáta do vytvoreného xml modelu. Celý

```
    <Code>Oznaceny text</Code>
    <Document>
      <ChangesetIdInRcs>aa02ef4a9c6b619f120ae03966a4e47f82ba7a48</»
         » ChangesetIdInRcs>
      <RcsServer>
        <Path>git@team08-13.ucebne.fiit.stuba.sk:repos/idem.git</Path>
        <Type>git</Type>
      </RcsServer>
      <Path>idem/src/main/java/sk/stuba/fiit/idem/controller/Browser.java</Path>
      <PathType>RelativeLocal</PathType>
      <BranchName>origin/feature/graph</BranchName>
    </Document>
    <StartColumnIndex>2</StartColumnIndex>
    <EndColumnIndex>39</EndColumnIndex>
  </IdeCodeOperationDto>
```

Ukážka 3: *Ukážka XML súbora*

```
    <Code>&amp;amp;idemProjekt&amp;amp;{'pocet_statickych_metod':3}</Code>
    <Document>
      <Path>ui2/src/sk/stuba/fiit/podmajerskyj/ui2/Player.java</Path>
      <PathType>RelativeLocal</PathType>
    </Document>
    <StartColumnIndex>1</StartColumnIndex>
    <EndColumnIndex>3</EndColumnIndex>
  </IdeCodeOperationDto>
```

Ukážka 4: *Ukážka XML súbora*

tento prístup využíva zverejnené zdrojové kódy k perconik projektu na programovanie eclipse pluginov.

Tento prístup sa v budúcnosti zmené s tým, že budeme posielať celý nový objekt.

Rozšírenie o statické atribúty a metódy

Statické atribúty, môžu veľa napovedať o kvalite kódu v objektovom princípe. Viac statických metód znižuje abstrakciu a využitie potenciálu objektovo orientovaného programovania. Java je objektovo orientovaný jazyk, ktorý poskytuje veľa pokročilých možností, avšak sú náročné na implementáciu. Cieľom tejto metriky je odlíšiť neskúsených programátorov od profesionálov.

Písaním veľkého množstva statických metód sa dospeje k písaniu procedurálneho kódu, v Jave to nie je žiadúce. Samozrejme, určitý počet statických metód je prípustný, niekedy aj vhodný, avšak nikdy nemôže byť viac statických metód ako klasických. Preto má táto metrika zmysel v kotexte projektu s celkovým po-

čtom tried. Keď vytvoríme pomer statických metód k celkovému počtu metód získame relevantné výsledky porovnania programátorov, čiže pohľad na kvalitu programovania rôznych ľudí.

Metrika zisťovania počtu statických metód je vytvorená vďaka DocumentAdapter, od ktorého listener tohto pluginu dedí. Údaje z tohto pluginu sa posielajú keď je zmenený daný dokument. Rozhodli sme sa preto tak, lebo dokument sa často krát mení, a my chceme aj tie zmeny odsledovať. Ako programátor zmýšľa, opreto chceme mať prehľad o tom ako sa kód vyvýjal. Na určenie počtu statických, využijeme to, že static je kľučové slovo, tak jednoducho spočítame koľko krát sa v texte toto slovo vyskytuje.

Táto metrika by sa v budúcnosti dala ešte upraviť, že by odosielala štatistiky len keď sa uloží dokument. Viac premyslieť by sa dalo aj samotné rátanie, mohli by sme využiť to, že by sme počítali iba metódy, teraz sú zarátané aj statické premenné. Rátali by sme iba tie static kľúčové slová, keď sa na tom istom riadku nenachádza bodkočiarka.

5.5 Agregácia vektorov

Táto úloha je zameraná na vytvorenie metodiky, ktorá bude slúžiť na prácu s vektorom. Konkrétne na agregáciu vektorov do agregovaného vektora pre jedného používateľa, vplyv času na zmenu vektora, spriemerovanie hodnôt vektora atď.

Úloha bola riešená spôsobom vyhľadávania informácií ohľadom tejto problematiky. Všetky vedecké práce, ktoré sme našli pojednávali o tom, že jednotlivé operácie s vektormi sa definujú podľa špecifických požiadaviek problému. Obrátili sme sa teda na Bc. Hudáka, ktorý nám tento trend potvrdil.

Agregáciu vektorov teda budeme musieť riešiť tak, že postupnou analýzou vytvoríme vlastnú metodiku. Momentálne je najvhodnejšie vektory agregovať pomocou váženého priemeru, kde budeme zarátavať s vyššou váhou tie vektory, ktorých údaje obsahujú logy s väčším časovým horizontom a obsahujú čo najúplnejšie dáta.

V ďalšom priebehu riešenia nášho projektu bude metodika dotvorená a definované najvhodnejšie postupy práce s vektormi. V najbližšej dobe budeme experimentálne zisťovať, aké metódy sú v rámci našej problematiky najvhodnejšie.

5.6 Rozšírenie pre riadiace premenné

Cieľom tejto úlohy je vytvorenie pluginu do Eclipse, ktorý pomocou rozšírení vytvorených Bc. Pavlom Zbellom dokáže sledovať zmeny v kóde týkajúce sa softvérovej metriky počtu riadiacich premenných, konkrétne príkazy if, else, switch, while, for. Táto metrika sa nazýva McCabe Cyclomatic Comlpexity a je jednou z najpoužívanejších metrík sledujúcich kvalitu kódu.

Pri riešení úlohy sme postupovali podľa inštrukcií definovanými Bc. Tomášom Martinkovičom a Bc. Jánom Podmajerským. Vytvorili sme plugin zaregistrovaním listenera. Tento plugin pracuje nasledovne:

- Listener sleduje zmeny v kóde v reálnom čase.
- Ak používateľ napíše kľúčové slová if, else, while, for, switch, priráta sa k jednotlivým počítadlám definovaným pre každé kľúčové slovo samostatne jednotka.
- Tieto údaje sa pomocou mapovania definovaným Bc. Jánom Podmajerským logujú cez UACA do databázy.
- Jednotlivé údaje sú súčasťou aktivity, v ktorej je presne definovaný používateľ, ktorý dané riadky upravoval.

Úlohou bolo vytvoriť plugin, ktorý loguje riadiace premenné, čo je splnené. Je však treba podotknúť, že keďže logujeme sami seba, je potrebné analyzovaním dát nastaviť potrebné prahy vyhodnocovania. Myslíme tým vytvorenie vyhodnocovača, ktorý na základe definovaných prahov bude vyhodnocovať kvalitu kódu z pohľadu McCabe Cyclomatic Comlexity.

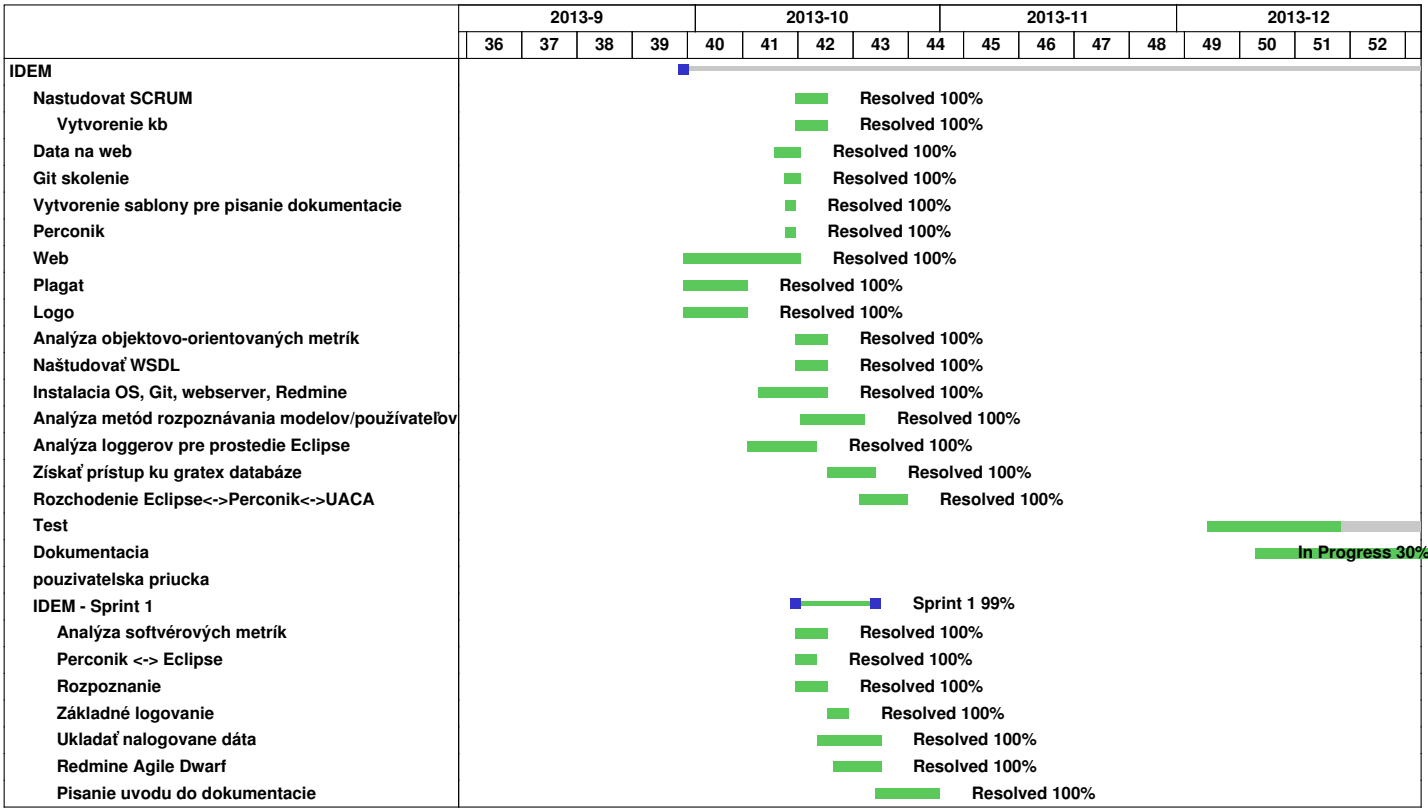
V ďalšej práci bude potrebné editovať tento plugin tak, aby dokázal vyrátať hodnoty McCabe Cyclomatic Complexity v rámci kódu, ktorý vytvoril jeden používateľ samostatne a tak určiť mieru kvality programovania daného používateľa.

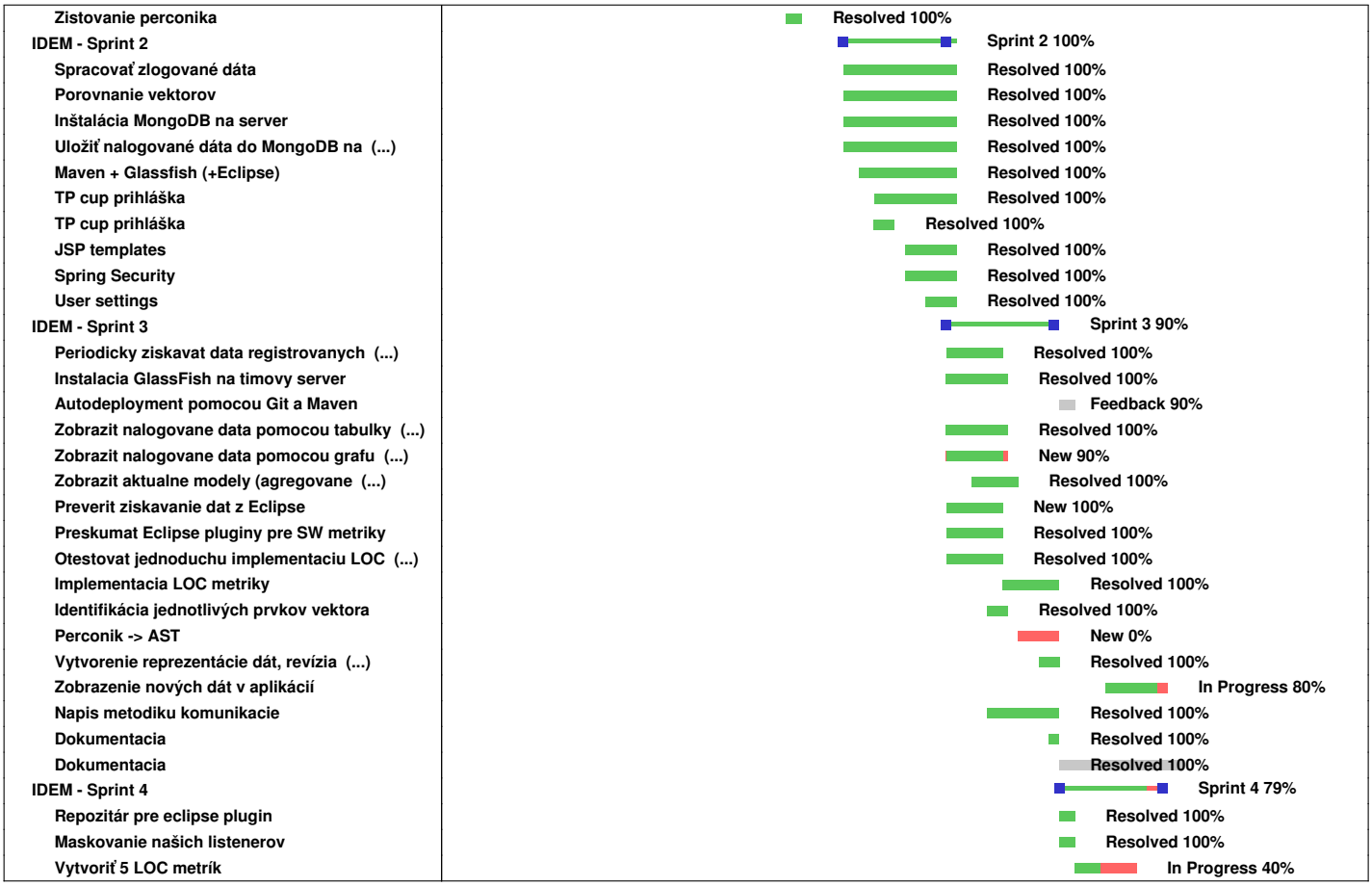
5.7 Zhodnotenie šprintu 4

Po uzavretí 4. šprintu máme funkčnú správu projektov, ku ktorým vieme priradiť používateľov. Rozšírili sme agregovaný vektor o nové atribúty a tiež sme rozšírili logovací systém o nové aktivity.

5.8 Ganttov graf

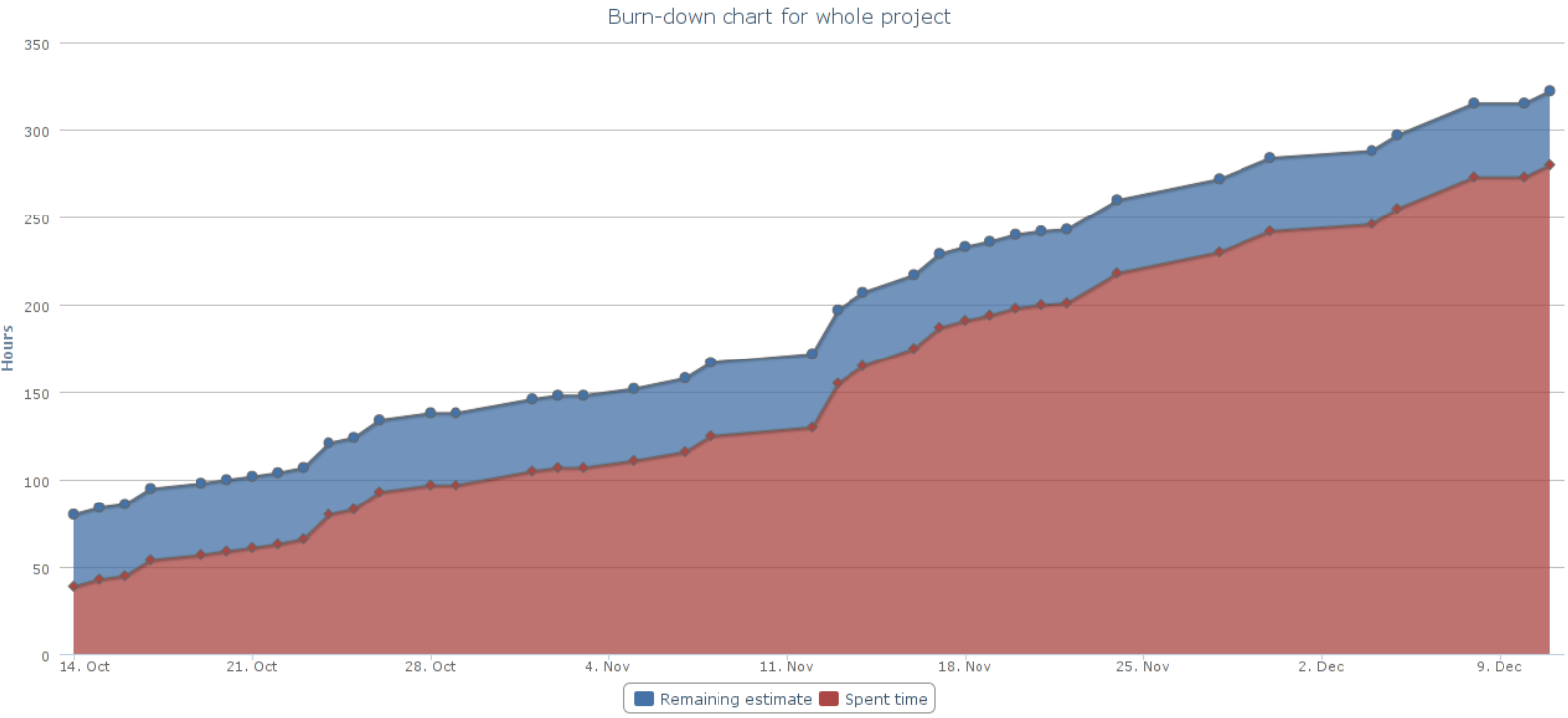
IDEM




Zistovanie perconika
Resolved 100%
IDEM - Sprint 2
Sprint 2 100%
Spracovať zlogované dáta
Resolved 100%
Porovnanie vektorov
Resolved 100%
Inštalácia MongoDB na server
Resolved 100%
Uložiť nalogované dáta do MongoDB na (...)
Resolved 100%
Maven + Glassfish (+Eclipse)
Resolved 100%
TP cup prihláška
Resolved 100%
TP cup prihláška
Resolved 100%
JSP templates
Resolved 100%
Spring Security
Resolved 100%
User settings
Resolved 100%
IDEM - Sprint 3
Sprint 3 90%
Periodicky ziskavat data registrovanych (...)
Resolved 100%
Instalacia GlassFish na timovy server
Resolved 100%
Autodeployment pomocou Git a Maven
Feedback 90%
Zobrazit nalogovane data pomocou tabulky (...)
Resolved 100%
Zobrazit nalogovane data pomocou grafu (...)
New 90%
Zobrazit aktualne modely (agregovane (...)
Resolved 100%
Preverit ziskavanie dat z Eclipse
New 100%
Preskumat Eclipse pluginy pre SW metriky
Resolved 100%
Otestovat jednoduchu implementaciu LOC (...)
Resolved 100%
Implementacia LOC metriky
Resolved 100%
Identifikácia jednotlivých prvkov vektora
Resolved 100%
Perconik -> AST
New 0%
Vytvorenie reprezentácie dát, revízia (...)
Resolved 100%
Zobrazenie nových dát v aplikácií
In Progress 80%
Napis metodiku komunikacie
Resolved 100%
Dokumentacia
Resolved 100%
Dokumentacia
Resolved 100%
IDEM - Sprint 4
Sprint 4 79%
Repozitár pre eclipse plugin
Resolved 100%
Maskovanie našich listenerov
Resolved 100%
Vytvoriť 5 LOC metrík
In Progress 40%

Vytvoriť rozšírenie na statické atribúty/metódy	**Resolved 100%**
Vytvoriť rozšírenie na riadiace premenné	**New 90%**
Získanie dát z nových metrík a ich uloženie (...)	**In Progress 0%**
Manažovanie skupín užívateľov	**Resolved 100%**
Agregacia vektorov	**New 100%**
Funkcionality pre tímy	**New 70%**
Anonymný rebríček	**Resolved 100%**
Chyba pri rebase	**Resolved 100%**

5.9 Burn-Down chart



6 Stanovenie globálnych cieľov projektu na letný semester

Hlavným cieľom projektu na letný semester je vytvorenie webovej aplikácie spolu s pluginom do Eclipse IDE. Plugin nám umožní monitorovanie programátora pomocou zvolených metrík vo vývojovom prostredí Eclipse. Získané dáta na základe nami zvolených metrík budú vyhodnocované vo webovej aplikácií a v rozumnej forme podané programátorovi, respektíve jeho manažérovi. Zobrazenie dát bude odzrkadlovať jeho vývoj v čase spolu s jeho zvyklosťami používania Eclipse. Medzi hlavné ciele projektu patrí:

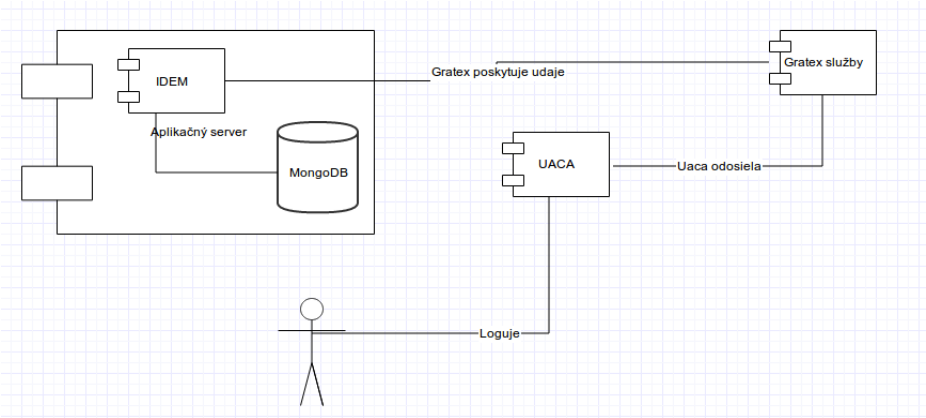
- Získavanie dát pomocou metrík v Eclipse plugine
- Zaradenie používateľa do tímu, resp. skupiny
- Meranie metrík iba nad zdrojovým kódom prislúchajúceho programátorovi
- Zobrazenie metrík vo webovej aplikácií formou grafov a tabuliek

6.1 Zhodnotenie po zimnom semestri

- Vytvorili sme web stránku tímu, GIT repozitáre a nastavili sme RedMine pre správu úloh.
- Naša aplikácia má momentálne implementované prepojenie s externou databázou. Vieme z nej prebrať údaje aj poslať ďalšie údaje, ktoré sa do nej zapíšu.
- Na logovanie programátora používame logovací systém projektu Perconik.
- Do aplikácie boli implementované algoritmy, ktoré slúžia na porovnávanie vektorov a to euklidova vzdialenosť, manhattanská vzdialenosť a kosínusová vzdialenosť.
- Podľa údajov, ktoré dostávame z externej databázy sme identifikovali model používateľa. Podľa tohto modela bola vytvorená trieda, ktorej úlohou je vytvorenie agregovaného vektora z modela používateľa.

- Do aplikácie sa môže používateľ registrovať a prihlásiť. Bolo nastavené Spring security a autentifikácia.
- Sú vytvorené dve okná slúžiace najmä pre debugovanie. Jedno okno zobrazuje surové dáta z externej databázy a druhé zobrazuje modely používateľa podľa zadaného mena pričom dáta berie tiež z externej databázy.

Architektúra systému po zimnom semestri je nasledovná:



Obr. 8: *Ukážka architektúry systému*

7 Šprint 5 a šprint 6

Šprint 5 trval len jeden týždeň. Bol to posledný šprint v letnom semestri, kde sa najmä dopisovala dokumentácia a všetko odovzdávalo preto je v dokumentácii tento šprint zlúčený so šprintom 6.

7.1 Spojenie pluginov

7.1.1 Úloha

Spojiť existujúce pluginy do jedného pluginu, vytvoriť repozitár pre plugin a prepracovať plugin na riadiace premenné pre prácu nad projektom, metódou a triedou.

7.1.2 Analýza

Jednotlivé pluginy, ktoré sme vytvorili ešte v zimnom semestri nám neposkytovali možnosť dotvorenia funkcionality najmä z dôvodu, že neboli poskytnuté v git repozitári z ktorého by boli voľne dostupné k úprave. Zanalyzovali sme možnosti zdieľaných repozitárov a vybrali si pre túto časť projektu .git. Následne sme analyzovali možnosti tvorby pluginov pre vývojové prostredie Eclipse. Ďalšou časťou analýzy je zistenie, ako presne funguje a ako implementovať plugin na riadiace premenné. Zistili sme, že najvhodnejšou formou bude vytvoriť plugin pre rátanie cyklomatickej zložitosti (angl. Cyclomatic Complexity).

7.1.3 Návrh

Vytvoríme nový Eclipse plugin project, kde nakopírujeme vytvorené pluginy. Následne odstránime chyby v API baseline, ktorým sa sprvoti nedá vyhnúť. Následne s využitím AST vytvoríme plugin pre cyklomatickú zložitosť.

7.1.4 Implementácia

Spojenie pluginov prebiehalo v programovacom jazyku Java s použitím XML súborov pre zadefinovanie závislostí. Následne sme spojené pluginy exportovali

pomocou .git do nášho zdieľaného repozitára.

7.2 Vytvorenie prepojenia medzi projektami v Eclipse a vo webovej aplikácii

7.2.1 Úloha

Vytvoriť logické prepojenie medzi projektami v Eclipse IDE a vo webovej aplikácii, aby sme mohli používateľom poskytovať údaje relevantné ku konkrétnemu projektu.

7.2.2 Analýza

Základnú identifikáciu samotných používateľov zabezpečuje nástroj UACA, v ktorom má používateľ definované svoje ID. Toto zadá používateľ aj vo svojom profile vo webovej aplikácii IDEM Monitora. UACA potom v rámci záznamov odosiela aj ID používateľa, ktorému dané záznamy patria a je jednoduché pri spracovaní záznamov priradiť ich konkrétnemu používateľovi. V prípade projektov je situácia horšia, pretože na to nemá UACA priamo dosah. Rozhodli sme sa preto zaviesť samostatný identifikátor projektov, ktorý sa bude odosielať spolu so záznamami z Eclipse pluginu.

7.2.3 Návrh

Jednotlivé projekty v MongoDB, ktorú používame, majú priradený jedinečný identifikátor. Tento identifikátor zadá používateľ aj do Eclipse, aby sa k odosielaným záznamom mohol podľa zvoleného projektu tento identifikátor pripojiť.

7.2.4 Implementácia

Pri implementácii sme využili možnosť poskytovanú prostredím Eclipse, ktorá umožňila pridať vlastnú záložku do okna vlastností projektu a ukladať uložené údaje spolu s projektom.

7.3 Vytvoriť základné grafy (bullet chart a spline chart)

7.3.1 Úloha

Vytvoriť grafy a diagramy pre zobrazenie metrik, ktoré zároveň umožňujú porovnanie používateľov v kontexte projektu.

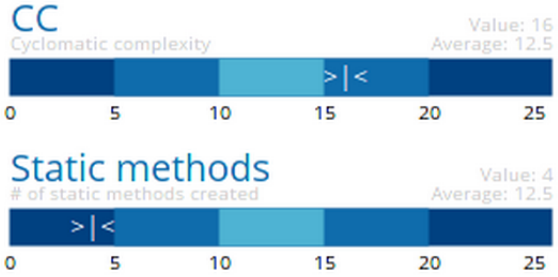
7.3.2 Analýza

Textové zobrazenie metrík sme nepovažovali za zaujímavé a prehľadné pre používateľa a navyše takéto zobrazenie sme pozorovali aj pri iných pluginoch na meranie metrík. Snažili sme sa preto nájsť spôsob ako prehľadne zobraziť namerané hodnoty a umožniť používateľovi jednoduché vizuálne porovnanie používateľov v rámci projektu.

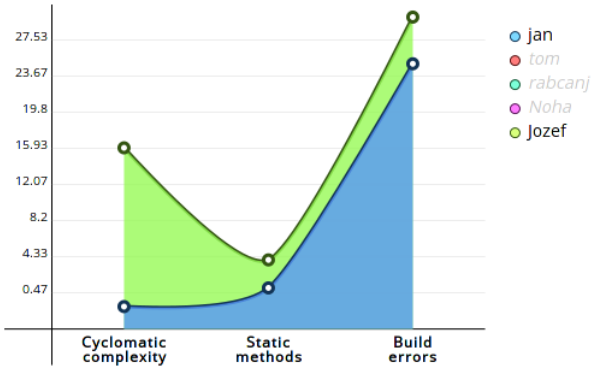
7.3.3 Návrh

Pre tvorbu grafov sme sa po preskúmaní niekoľkých možností rozhodli použiť javascriptovú knižnicu D3.js. Nevýhodou tohto riešenia je zložitejšia tvorba grafov, bezkonkurenčnou výhodou oproti iným riešeniam je ale možnosť vytvoriť ľubovoľný graf, či diagram, takže niesme obmedzení množstvom dostupných typov diagramov.

V prvej fáze sme sa rozhodli pre dva grafy - bullet chart (obr. 9) a spline chart (obr. 10). Bullet chart je určený pre zobrazenie aktuálnej hodnoty metriky pre daného používateľa a jeho vizuálne porovnanie s priemernou hodnotou v rámci projektu. Spline chart slúži na zobrazenie metrík viacerých používateľov v rámci projektu a možnosť ich vizuálneho porovnania.



Obr. 9: *Ukážka bullet chart diagramu*



Obr. 10: *Ukážka spline chart diagramu*

7.3.4 Implementácia

Grafy boli vytvorené s použitím javascriptovej vizualizačnej knižnice D3.js a knižnice d3.chart, ktorá sprehľadňuje implementáciu a zlepšuje znovupoužiteľnosť. Grafický výstup je v podobe SVG obrázkov, čo spája výhody vektorovej grafiky a možnosti prípadnej interakcie používateľa s diagramom.

7.4 Listener na Build, ktorý meria počet chýb

7.4.1 Úloha

Vytvoriť nový listener v plugine do Eclipsu, ktorý bude počúvať na akciu Build. Ak používateľ spustí build v Eclispe, spustí sa meranie metrík. Merané metriky sú počet vzniknutých chýb, respektíve markerov.

7.4.2 Analýza

Počet vzniknutých chýb pri písaní zdrojového kódu je zaujímavá metrika, ktorá môže poukázať na kvalitne a pozornosť programátorov. Na vierohodné meranie tohto ukazovateľa nemôžeme použiť listener na Save, alebo na zmenu kódu, ako boli doteraz na všetky listenery používané. Tieto listenery pri meraní chybovosti neobstoja, pretože každý, dokonca aj profesionálny a veľmi kvalitný programátor občas spraví nejakú drobnú chybu. My nechceme zaznamenať kto spraví koľko drobných, nepodstatných chýb, ktoré sú spôsobené preklepmi a ktoré si ich autor

ani častokrát nevšimne. My hľadáme počet úmyselných chýb, ktoré niekto spravil s vedomím, že sa nachádzajú v kóde, ale aj napriek tomu chce kód spúšťať. Hľadáme tvorcov chýb, preto nemôžeme počuvať na zmenu kódu, alebo na uloženie, potrebujeme listener na build projektu. Ak niekto spúšťa kód aj s chybami, je si vedomí, že tieto chyby funkčnosť programu neovplyvnia, mal by ich aj tak odstrániť a vyčistiť kód od zbytočností. My sa orientujeme na takéto detaily, ktoré odlišujú kvalitných programátorov od dobrých.

7.4.3 Návrh

Na zistenie ako fungujú jednotlivé akcie v eclipse sme museli prečítať mnoho dokumentácie k java org.eclipse. Nakoniec sme sa rozhodli inšpirovať perconik pluginom od Pavla Zbella, ktoré je aj na gite, a tak sme použili sme LaunchListener.

Na meranie počtu chýb sme využili Markers, ktoré poskytuje eclipse ku každému projektu. Ku každému projektu sa dá zistiť počet varovaní a počet chýb, my riešime iba chyby.

Návrh fungovania pluginu aj s týmto listenerom:

- používateľ napíše časť kódu
- spustí projekt
- listener zachytí akciu
- listener prejde všetky jeho projekty
- vyhľadá všetky markery a sčíta chybové
- listener odošle počet chýb pomocou UACI na server

7.4.4 Implementácia

Naprogramovanie tohto listenera prebiehalo v Jave v projekte so všetkými ostatnými listenermi. Postupovali sme štandartne ako pri vývoji ostatných listenerov.

Odosielanie jednotlivých metrík je zabezpečené rovnako ako pri iných listeneroch, cez statickú metódu, ktorá odošle namerané počty chýb vo formáte JSON. Tento JSON je odoslaný ako zamaskované zmeny zdrojového kódu. V tejto fáze posielame iba jedno číslo - počet chýb pre používateľa.

7.5 Zmena zrnitosti odosielania dát na Save súboru

7.5.1 Úloha

Upraviť implementované listenery v Eclipse plugine tak, aby ich meranie nad zdrojovým kódom prebiehalo iba pri ukladaný daného súboru.

7.5.2 Analýza

Doteraz fungovali všetky listenery so zrnitosťou jednej akcie klávesnice alebo myši. Pri každej akcii sa opätovne prepočítavali naimplementované metriky iba nad malou zmenou v zdrojom súbore a následne sa posielali do Gratex databázy. V databáze sa nachádzalo množstovo rovnakých dát, ktoré pri vyhodnocovaní kvality programátora neboli potrebné. Rozhodli sme sa zmeniť zrnitosť posielania dát na akciu ukladania .java súboru. Takáto zrnitosť nám zredukuje redundantnosť dát v databáze v blízkom časovom slede, ale zabezpečí nám odosielanie dát z metrík, ktoré vo väčšine prípadov sa budú merať nad ucelenou časťou zdrojového kódu. Vychádzali sme z predpokladu, že programátor neukladá súbor po každom znaku, ale priebežne po naprogramovaní niejakého bloku zdrojového kódu, respektíve ucelenejšej časti a nepredpokladáme, že programátor ukladá zdrojový kód po celých triedach.

7.5.3 Návrh

Na implemetovanie analýzy využije triedu *FileBufferAdapter* z PerConIK-u, ktorá nám umožní override verejnej metódy *dirtyStateChanged(IFileBuffer buffer, boolean dirty)*. Táto metóda nám zabezpečí, že časť jej tela sa vykoná v podmienke *if(!dirty)* iba v tom prípade, že dôjde k uloženiu súboru v Eclipse.

7.5.4 Implementácia

Na základe návrhu sme naimplementovali požadovanú metódu do každého listenera okrem listenera chybovosti pri builde. V tele podmienky if (!dirty) sme nechali meranie konkrétnych metrík.

7.6 Listener pre počet statických metód a atribútov cez AST

7.6.1 Úloha

Upraviť listener pre počet statických metód a atribútov tak, aby sa nevychádzalo pri ich meraní zo zdrojového kódu triedy, ale z informácií AST poskytujúceho Eclipsom.

7.6.2 Analýza

Pri analýze sme sa zamerali na AST (abstraktný syntaktický strom) v Eclipse, ktorý poskytuje balík org.eclipse.jdt.core.dom.AST. Jeho analyzovaním sme prišli k záveru, že z AST sa nedá získať informácia, či je daná metóda alebo atribút statický. Implementovanie tejto úlohy cez AST by bolo zbytočné, a preto sme sa rozhodli ponechať funkcionalitu predchádzajúceho listeneru na zachytávanie počtu statických metód a atribútov v danej triede, ktorý vychádzal zo zdrojového kódu.

7.7 Prezentácia

7.7.1 Úloha

Vypracovať prezentáciu na progress report za zimný semester. Prezentácia mala obsahovať popis projektu, opis súčasného stavu a plán na letný semester.

7.7.2 Analýza

Prezentácia mala obsahovať informácie o postupe za zimný semester. Vychádzali sme zo zadania tímového projektu, dokumentácie odovzdanej za zimný semester a súčasného stavu.

- Cieľ
- Identifikácia
- Hodnotenie kvality
- Architektúra

- Súčasný stav
- Plán na letný semester

7.8 Stránkovanie

7.8.1 Úloha

V admin rozhraní treba opraviť stránkovanie (stránku/počet stránok) po odobratí užívateľa/projektu.

7.9 Odoberanie užívateľa

7.9.1 Úloha

Odoberanie užívateľa vymaže len objekt v kolekcií užívateľov, ale v ostatných kolekciách ostávajú referencie.

7.10 Dopĺňanie mena používateľa

7.10.1 Úloha

Autocomplete pre meno užívateľa, pre zjednodušenie UX

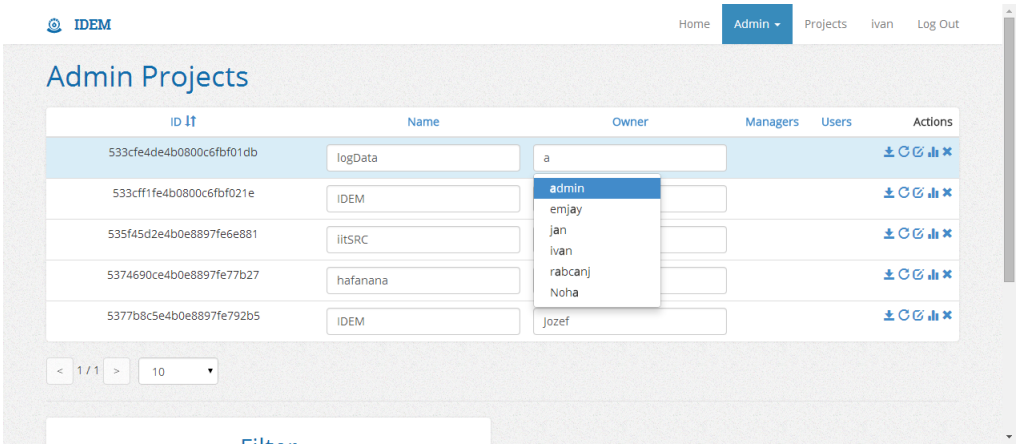
7.10.2 Analýza

Pre implementáciu sa využil modul pre angularjs typeahead a jednoduché rest api

7.10.3 Návrh

Bol navrhnutý jednoduchý REST API controller, ktorý poskytuje zoznam mien podľa dopytu.

7.10.4 Implementácia



Obr. 11: *Ukážka doplňania*

7.11 Opraviť zobrazovanie tabuliek pri nízkom rozlíšení

7.11.1 Úloha

Tabuľky sa nezobrazovali správne pri nízkom rozlíšení.

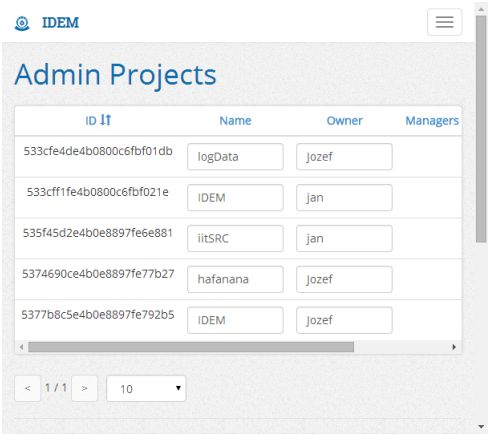
7.11.2 Analýza

Pre implementáciu sa využil modul pre angularjs typeahead a jednoduché rest api

7.11.3 Návrh

Podľa pokynov frameworku Bootstrap treba tabuľky vložiť do DIV-u s triedou table-responsive

7.11.4 Implementácia



Obr. 12: *Ukážka tabuľky*

7.12 Filtrovanie v administrátorskom móde

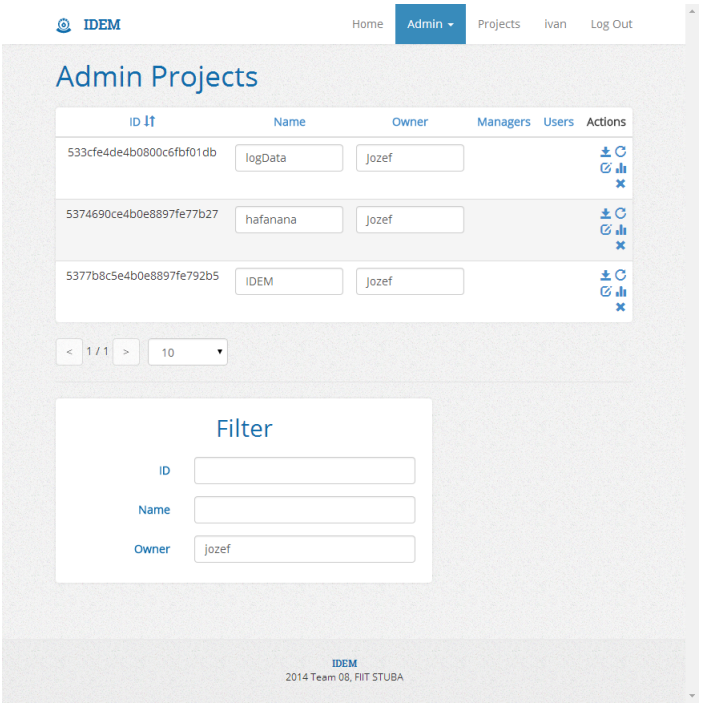
7.12.1 Úloha

Umožnenie administrátorovi prehľadne zobrazovať/filtrovať záznamy.

7.12.2 Analýza

Pre túto funkcionalitu je potrebné upraviť filtrovanie angularjs ng-repeat v tabuľkách

7.12.3 Implementácia



Obr. 13: *Ukážka tabuľky*

7.13 Web security anotácie

7.13.1 Úloha

Nastavenie/úprava prístupových rolí v controlleroch.

7.13.2 Analýza

Pre nastavenie rolí bolo potrebné nastavenie scurity anotácií frameworku Spring.

7.14 Triedenie administrátorských tabuliek

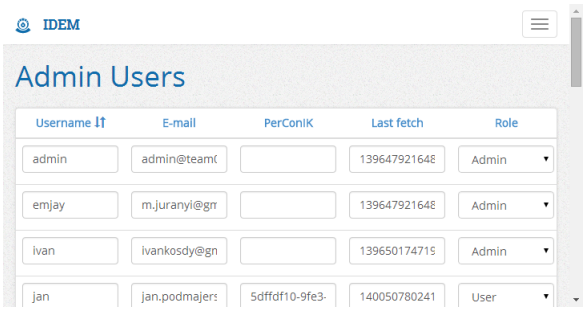
7.14.1 Úloha

Umožnenie zoraďovania dát v tabuľkách administrátorského rozhrania.

7.14.2 Analýza

Pre túto funkcionalitu je potrebné upraviť filtrovanie angularjs ng-repeat v tabuľkách

7.14.3 Implementácia



Obr. 14: *Ukážka triedenia*

7.15 Úprava vzhľadu stránky

7.15.1 Úloha

Táto úloha sa skladá z troch podúloh a to: zobrazovanie upozornení nesprávne vyplnených formulároch, zobrazenie textu titulky stránky, zvýraznenie vybratej položky v hlavnom menu.

7.15.2 Analýza

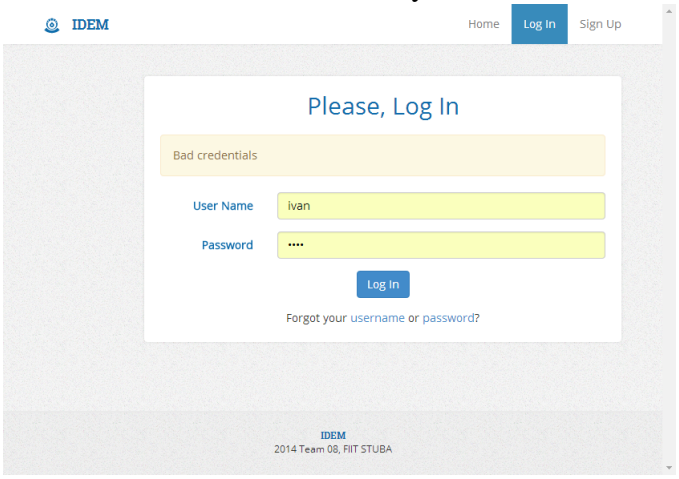
Zobrazovanie chybových/informačných hlásení formulárov bolo potrebné presunúť do Bootstrapového alert DIV-u. Rozšírenie šablóny o css triedu active pre jednotlivé zvolené položky. Oprava a validácia kódu pomocou validátora.

7.16 Návrh

Rozšírenie šablóny o titulku. Pridanie titulky do controllerov a opravenie chýb indikovaných nástrojom HTML W3 čím dosiahneme zlepšenie zobrazovania stránky a nakoniec vytvoríme responzívny design.

7.16.1 Implementácia

Zobrazovanie chybových/informačných hlásení formulárov bolo potrebné presunúť do Bootstrapového alert DIV-u. Rozšírenie šablóny o titulku.



Obr. 15: *Ukážka zmeny GUI*

7.17 Žiadosť o prístup ku projektu

7.17.1 Úloha

Pridať možnosť žiadosti o prístup ku projektu.

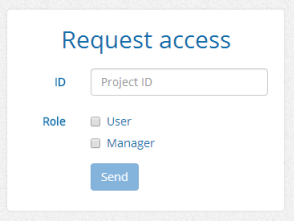
7.17.2 Analýza

Rozšírenie databázového modelu o kolekciu žiadostí a prístupov.

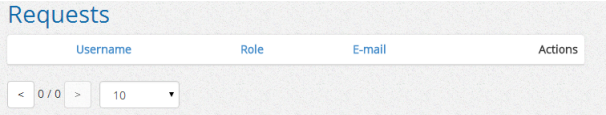
7.18 Návrh

Rozšírenie GUI o možnosť pridávania žiadostí, ich schvaľovanie. Rozšírenie modelu DB o žiadosti

7.18.1 Implementácia



Obr. 16: *Ukážka požiadávky o zaradenie do projektu*



Obr. 17: *Ukážka priradeného projektu*

8 Šprint 7

8.1 Upraviť odosielanie dát do UACA

8.1.1 Úloha

Upraviť chybné mapovanie našich vytvorených listenerov do IdeCodeOperationDto objektu v Gratex databáze, do ktorého UACA posiela naše namerané metriky.

8.1.2 Analýza

Pri získavaní dát na strane našej webovej aplikácie pomocou webovej služby do Gratex databázy sme zistili, že sa dáta z naších listeneroch nenachádzajú v IdeCodeOperationDto objektoch. Nalogované dáta zostávali v UACA neodoslané a pri bližšej analýze triedy IdeCodeListener.java v PerConIK-u, z ktorej sme vychádzali pri mapovaní, sme zistili, že je nová verzia PerConIK. V tejto verzii je zmenené odosielanie dát do UACA.

8.1.3 Návrh

Na základe zmenených metód upravíme odosielanie v každom listenery spolu s dodatočnou informáciou ohľadom mena a verzie aplikácie Eclipse.

8.1.4 Implementácia

Požadovanú zmenu odosielania dát do UACA sme naimplementovali pre každý náš listener. Následne sme otestovali správnosť odosielania dát z pluginu do UACA a potom do Gratex databázy. Na strane webovej aplikácie sme cez webovú službu vytiahli požadované dáta.

8.2 Upraviť listenery na Save pre ukladanie viacej súborov naraz

8.2.1 Úloha

Upraviť chybné mapovanie našich vytvorených listenerov do IdeCodeOperationDto objektu v Gratex databáze, do ktorého UACA posiela naše namerané metriky.

8.2.2 Analýza

Pri získavaní dát na strane našej webovej aplikácie pomocou webovej služby do Gratex databázy sme zistili, že sa dáta z naších listeneroch nenachádzajú v IdeCodeOperationDto objektoch. Nalogované dáta zostávali v UACA neodoslané a pri bližšej analýze triedy IdeCodeListener.java v PerConIK-u, z ktorej sme vychádzali pri mapovaní, sme zistili, že je nová verzia PerConIK. V tejto verzii je zmenené odosielanie dát do UACA.

8.2.3 Návrh

Na základe zmenených metód upravíme odosielanie v každom listenery spolu s dodatočnou informáciou ohľadom mena a verzie aplikácie Eclipse.

8.2.4 Implementácia

Požadovanú zmenu odosielania dát do UACA sme naimplementovali pre každý náš listener. Následne sme otestovali správnosť odosielania dát z pluginu do UACA a potom do Gratex databázy. Na strane webovej aplikácie sme cez webovú službu vytiahli požadované dáta.

8.3 Počet errorov pri Builde iba pre aktuálneho používateľa

8.3.1 Úloha

Upraviť funkčnosť našich listenerov tak, aby sa pri ukladaní viacej .java súborov v Eclipse IDE merali naše metriky nad všetkými ukladanými súbormi samostatne.

8.3.2 Analýza

Pri testovaní odosielania dát v 6.5 sa objavila chyba v logoch nachádzajúcich sa v UACA. Tieto logy boli duplicitné a ich počet bol rovnaký ako počet ukladaných súborov naraz. Pri podrobnej analýze sme zistili, že listenery pri ukladaní naraz pomocou klávesovej skratky Ctrl+Shift+S bežia len nad aktívnym dokumentom v Eclipse IDE. Nad týmto dokumentom zbehnú metriky presne toľkokrát, koľko je ukladaných súboro naraz a presne toľko logov sa objaví v UACA. Nad ostanými ukladanými dokumentami v Eclipse nezbehnú požadované metriky. Na základe tohto sme sa pozreli na riešenie v PerConIK, ktorý tento problém rieši tak, že požadovaný zdrojový kód tela metódy dirtyStateChanged sa vykonáva v threade.

8.3.3 Návrh

V každom listenery na Save obalíme telo metódy dirtyStateChanged do threadu.

8.3.4 Implementácia

Vykonané zmeny boli vykonané v Java ako plugin development. Avšak pri testovaní sme zistili, že programátori nebuildujú chybné projekty. Eclipse pred každým buildom upozorní a spýta sa, či má projekt buildnúť aj napriek chybám. Mnohí ich najprv opravia a až potom spustia projekt. Kvôli tomu sme častokrát mali počet chýb pri builde 0. Preto sme sa rozhodli, že bude potrebné danú metriku zmeniť na save listener.

8.4 Odosielanie metrík pomocou JSON-u

8.4.1 Úloha

Upraviť odosielaný string, ktorý sa mapuje do IdeCodeOperationDto objektu v Gratex databáze, v naších listeneroch tak, aby sa posielal validný JSON za použitia externej knižnice.

8.4.2 Analýza

Existujú rôzne open source knižnice, ktoré implementujú a podporujú jednoduchú prácu s JSON objektmi v jazyku Java. Vhodné knižnice sme našli na stránke *www.json.org*. Na základe ich analýzy sme vybrali knižnicu *org.json*.

8.4.3 Návrh

Vybranú externú open source knižnicu stiahneme a pridáne do repozitára medzi externé knižnice, nastavíme dependencies a následne naimplementujeme vytvorenie JSON objektu pri odosielaní metrík v každom listenery s požadovanými dátami, ktoré sa majú odosielať. Z takto vytvoreného JSON objektu vytvoríme validný JSON text, ktorý sa odošle.

8.4.4 Implementácia

Externú knižnicu sa podarilo pridať do projektu aj s nastavením požadovaných dependencies. Naimplementovali sme pre každý listener funkcionalitu pre odosielanie dát, kde sa z JSON objektu vytvoril validný JSON string. Pri testovaní funkčnosti sme narazili na chybu java.lang.NoClassDefFoundError pri volaní potrebnej triedy z externej knižnice org.json. Túto chybu sme sa pokúsili odstrániť vyskúšaním ďalších externých knižníc, ktorých výsledok bol s rovnakou chybou. Framework PerConIK nám poskytol dodatočnú informáciu z debug výpisu, že nastali problémy v triede org.eclipse.core.filebuffers. Na základe tohto bolo potrebné pridať dependencies na externú knižnicu do PerConIK. Túto požiadavku sme neadresovali na PerConIK, ale sme sa rozhodli odosielať v každom listenery string vo validnej notácií JSON, ktorý bude vo webovej aplikácií ľahko spracovateľný. Validný JSON pre metriku cyklomatickej zložitosti je:

```
{"Version":"1.0.1.2","IdProject":"533cff1fe4b0800c6fbf021e","Path":"/Pr1/src/»
   » pack1/class1.java","CycloCount":2}
```

Ukážka 5: *Ukážka JSONu*

8.5 Vytvorenie koláčového grafu

8.5.1 Úloha

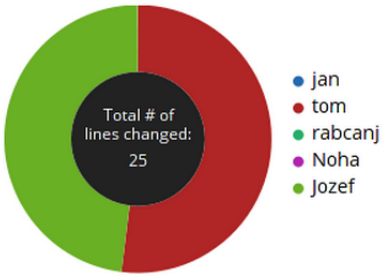
Vytvoriť graf pre porovnanie podielu práce jednotlivých používateľov na projekte.

8.5.2 Analýza

Pre porovnanie podielov zo 100% je najlepším a najpoužívanejším grafom koláčový graf. Takýto graf je použiteľný aj na zobrazenie iných pomerov ako je cieľom tejto úlohy. Napríklad porovnanie počtu riadkov vykonateľného kódu a riadkov s komentármi.

8.5.3 Návrh

Na vytvorenie grafu sme použili už predtým použité knižnice d3.js a d3.charts. Farby použité v grafe porovnávajúcom podiely práce na projekte korešpondujú s farbami použitými v spline chart grafe porovnávajúcom metriky používateľov v rámci projektu. Ukážka vytvoreného grafu je na obr. 18.



Obr. 18: *Ukážka koláčového grafu*

8.5.4 Implementácia

Implementácia prebehla podobne ako v prípade predchádzajúcich grafov (obr. 9 a obr. 10).

8.6 Zobrazenie údajov (serverové API pre grafy)

8.6.1 Úloha

Vytvoriť v serverovej časti webovej aplikácie rozhranie pre poskytovanie údajov pre grafy.

8.6.2 Analýza

Dynamicky generovaný obsah webových aplikácií na strane klienta často umožňuje klientovi interakciu, napríklad v podobe filtrovania údajov. Pri tom je však nežiadúce aby sa znovu načítala celá stránka. Takéto situácie sa riešia pomocou technológie AJAX, ktorá umožňuje komunikáciu so serverom na pozadí už načítanej stránky. Táto technológia sa tiež využíva pri prvotnom načítaní údajov aj v prípade, že k následnej interakcii už nedochádza. Využitie technológie AJAX však predpokladá, že server poskytuje rozhranie, ktoré reaguje na AJAX požiadavky a odpovedá na ne požadovanými údajmi.

8.6.3 Návrh

API pozostáva z niekoľkých metód určených pre získanie špecifických údajov pre jednotlivé grafy. Hlavnou úlohou každej metódy je získanie agregovaných dát z databázy (implementované v neskoršej fáze), ich preštruktúrovanie pre použitie v grafoch a ich odoslanie v podobne JSON odpovede na zadanú požiadavku.

8.6.4 Implementácia

V prípade našej webovej aplikácie sme pre vytvorenie API využili funkcionalitu poskytnutú použitým frameworkom Spring, čo okrem iného umožňuje jednoducho zabezpečiť prístup k API iba prihláseným používateľom. API odpovedá JSON objektami, ktoré v projekte využívame ako hlavný spôsob komunikácie medzi jednotlivými súčasťami systému.

8.7 Obmedzenie zobrazených údajov podľa práv používateľa

8.7.1 Úloha

Obmedziť zobrazenie nameraných údajov používateľom podľa ich oprávnení.

8.7.2 Analýza

Zbieranie údajov o používateľoch je citlivou záležitosťou a obzvlášť pokiaľ ide o neanonymné údaje. Najmä v pracovnom prostredí by mohlo dôjsť k nevôli pracovníkov používať tento nástroj, pokiaľ by vedeli, že údaje o ich pracovnom výkone niekto sleduje a to dokonca všetci kolegovia. Preto je dôležité umožniť sledovať získané údaje iba kompetentným ľuďom.

8.7.3 Návrh

Pri obmedzení práv na zobrazenie údajov bolo treba dospieť k určitému kompromisu, aby nebol týmto zásahom narušený zmysel projektov. Došlo tak skôr k anonymizácii údajov tak, že manažéri projektov vidia všetky dostupné informácie a bežní členovia vidia iba vlastné údaje a priemerné hodnoty za projekt pre porovnanie (výnimkou je zatiaľ porovnanie počtu zmenených riadkov).

8.7.4 Implementácia

Riešenie bolo implementované v podobe niekoľkých podmienok v šablóne stránky tak, aby sa pri nedostatočných oprávaneniach používateľa negenerovali niektoré časti stránky.

8.8 Úprava rátanie cyklomatickej zložitosti

8.8.1 Úloha

Úlohou je editovať rátanie cyklomatickej zložitosti tak, aby sa nezarátavala za projekt/ metódu / triedu, ale rátať ju v rámci zmeny kódu, ktorú programátor vyvinul od posledne uloženej zmeny (tzv. DIFF).

8.8.2 Analýza

Cyklomatická zložitosť je počet riadiacich premenných v určitej čiastke kódu. Medzi riadiace premenné patria kľúčové slová jazyka ako: „if“, „else“, „for“, „while“, „switch“. DIFF je rozdiel v kóde používateľa medzi posledným a aktuálny uložením. Preto bude potrebné túto metriku spúšťať a vyhodnocovať nad zmenenou čiastkou kódu. Je pre to nutné vytvoriť listener, ktorý sleduje zmeny v kóde od posledného uloženia.

8.8.3 Návrh

Vytvoríme listener, ktorý sleduje zmeny v otvorenom súbore. Použijeme triedu EditorAdapter, od ktorej budeme dediť a vytvoríme novú vlastnú triedu. Následne implementujeme rozhranie FileBufferListener, ktoré nám poskytne možnosť získania písaného textu. Následne si vytvoríme globálnu premennú typu Hashtable, kde kľúčom bude relatívna cesta k súboru a hodnotou bude časť kódu medzi jednotlivými uloženiami. Ďalšou časťou práce je vytvorenie parsera, ktorý bude sledovať pomocou regexu pridanie kľúčových slov a rátať frekvenciu ich výskytu. Následne pri každom uložení vytiahneme z mapy všetky údaje o neuložených súboroch, vyrátame cyklomatickú zložitosť nad celým daným súborom bez momentálneho DIFF-u a nad týmto DIFF-om a z ich rozdielu určíme prírastok, alebo úbytok cyklomatickej zložitosti. Následne ak DIFF je rozdielny od 0, tieto údaje pošleme do UACA-y, ktorá ich následne odošle do databázy.

8.8.4 Implementácia

Jednotlivé novo vytvorené triedy boli implementované v programovacom jazyku JAVA za pomoci abstraktnej triedy EditorAdapter a rozhrania FileBufferListener z balíčka projektu PerCoNik. Keďže posielanie údajov do Gratex databázy riešime pomocou maskovania údajov za údaje, ktoré nepotrebujeme, nebolo nutné zasahovať do databázového modelu. Správnosť vytvorenej metódy sme potvrdili testovaním, ktoré dopadlo podľa našich očakávaní. Na obr. 26 môžeme vidieť príklad implementovaných metód.

```
public static int getStaticsCount(String str){

		int count=0;
		Pattern patternIf = Pattern.compile("\\bstatic\\b");
		Matcher matcherIf = patternIf.matcher(str);
		for(count = 0; matcherIf.find(); count++);

		return count;
	}

public class ListenerCyclo extends FileBufferAdapter {

	private static final Executor executor = Executors.newCachedThreadPool();
	public final void dirtyStateChanged(final IFileBuffer buffer,
		final boolean dirty) {
			if (!dirty) {
				String first = GlobalVariables.source.get(key);
				if (!(first == null)) {
				String changed = TextFromFile.getTextFromFile(buffer);
				int primary = StringCounter.getCyclo(first);
				int secondary = StringCounter.getCyclo(changed);

				int cyclo = secondary - primary;

					if (cyclo != 0) {
					GlobalVariables.source.remove(key);
					GlobalVariables.source.put(key, changed);
		// odoslanie pre zjednodusenie neuvadzame parametre funkcie SendData.process()
					SendData.process();

}
```

Obr. 19: *Ukážka implementovaných metód*

8.9 Spojenie súčastí pluginu a update site

8.9.1 Úloha

Spojiť všetky doteraz vytvorené súčasti pluginu(listenery) a vytvoriť prvú celistvú verziu pluginu a vytvoriť update site.

8.9.2 Analýza

Vytvorenie update site pozostáva z vytvorenia nového Java Features projektu, v ktorom je potrebné namapovať náš plugin a všetky potrebné dependecie a pluginy, na ktorých je náš plugin závislý. Následne je potrebné tento *Features* projekt exportovať pomocou *Java Deployable Features*, pričom sa vytvorí priečinok, ktorý bude obsahovať všetky potrebné inštalačné súbory.

8.9.3 Návrh

Vytvoríme nový *Features* projekt *IDEMplugin.feature* v ktorom nastavíme potrebné dependencie a pridáme pluginy z balíčka *PerCoNik*. Následne exportujeme tento projekt a pomocou *WinSCP* pridáme celý priečinok s inštalačnými súbormi na náš server.

8.9.4 Implementácia

Implementácia tejto úlohy prebehla v programovacom jazyku JAVA a pomocou XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<feature
      id="IDEMplugin.feature"
      label="IDEMplugin"
      version="1.0.0.qualifier">
   <description url="http://www.example.com/description">
      [Enter Feature Description here.]
   </description>
   <copyright url="http://www.example.com/copyright">
      [Enter Copyright Description here.]
   </copyright>
   <license url="http://www.example.com/license">
      [Enter License Description here.]
   </license>
   <plugin
         id="IDEMplugin"
         download-size="0"
         install-size="0"
         version="1.0.1.2"
         unpack="false"/>
```

Obr. 20: *Ukážka implementácie*

8.10 Oprava nemožnosti mazania tried v Eclipse

8.10.1 Úloha

Zistiť dôvod nemožnosti vymazávania tried v Eclipse počas používania nášho pluginu.

8.10.2 Analýza

Počas používania nášho pluginu v prostredí Eclipse dochádzalo k chybe a pádu prostredia Eclipse, keď sa používateľ snažil vymazať nejakú triedu. Analýzou problému sme zistili, že je to tým ako vytvárame snapshot súboru, aby sme mohli pracovať a vyhodnocovať kód pomocou metrík nad DIFF-om. Je preto potrebné vytvárať DIFF-y iným spôsobom. Pôvodne sme pomocou listenera vytvárali nový súbor, do ktorého sme si uložili momentálne neupravený dokument. Tým sme implicitne otvorili *ImputStream*, ktorý sa však nedal uzavrieť. Tým pádom sme si zamkli súbor a nemohli sme ho vymazať.

8.10.3 Návrh

DIFF-y budeme teda vytvárať pomocou Hashtable, kde kľúčom bude relatívna cesta k súboru a hodnotou bude String, ktorý bude obsahovať momentálny stav súboru.

8.10.4 Implementácia

Opravu tohto problému sme implementovali v programovacom jazyku JAVA. Odstránenie chyby sme otestovali a dospeli k záveru, že chyba bola úspešne odstránená.

8.11 Odstraňovanie chýb v plugine

8.11.1 Úloha

Úlohou je odstrániť chyby, ktoré v plugine nastali. Chyby sa týkajú najmä zlého počítania jednotlivých metrík, ale aj padania pluginu pri preklikávaní nad jednotlivými triedami v projekte nad ktorým metriky počítame.

8.11.2 Analýza

V tejto úlohe bolo nutné analyzovať najmä prípady použitia, v ktorých nastala chyba a zistiť, prečo daná chyba nastala.

8.11.3 Návrh

Zistili sme, že chyby sa týkali najmä nemožnosti rátania metrík nad viacerými ešte neuloženými súbormi, a padanie pluginu pri spustení rátania metrík. Rátanie metrík sme teda prerobili na akciu OnSave, ktorá ráta metriky nad súborom pri ukladaní súboru.

8.11.4 Implementácia

Implementácia bola vykonávaná v programovacom jazyku JAVA. Na výpočet metrík nad súborom sme použili triedu *EditorAdapter* z ktorej sme odvádzali

nové triedy a implementovali rozhranie *FileBufferListener*. Trieda *EditorAdapter* a rozhranie *FileBufferListener* pochádzajú z balíčka projektu *PerCoNik*.

8.12 Listener na hĺbku dedenia

8.12.1 Úloha

Úlohou bolo vytvoriť listener, ktorý zistí hĺbku dedenia tried, ktoré má programátor aktuálne otvorené a pracuje s nimi.

8.12.2 Analýza

Táto metrika je dôležitá z hľadiska z hľadiska toho, že trieda ktorá má väčšiu hĺbku dedenia má väčší potenciál pre znovupoužitie implementovaných metód. Čím väčšia je hĺbka dedenia tak tým je návrh projektu zložitejší. Taktiež cím je hĺbka dedenia väčšia tým je zložitejšie predvídať správanie sa metód. Z uvedených dôvodov vyplýva, že hĺbku dedenia je dôležité sledovať.

8.12.3 Návrh

Budeme sledovať otvorené triedy a vrátime ku ním všetkých predkov.

8.12.4 Implementácia

Metóda na sledovanie hĺbky dedenia využíva listener reagujúci na uloženie projektu. Pri uložení projektu sa zistí aktuálne aktívne otvorené okno v Eclipse z ktorého sa vrátia všetky triedy. Následne ku každej triede vytiahneme jej rodičov a spočítame koľko ich je. Výsledok uložíme do JSONu a odošleme pomocou webovej služby na server.

8.13 Upraviť meranie LOC metrík nad pridaným zdrojovým kódom

8.13.1 Úloha

Úlohou bolo vytvoriť listener, ktorý zistí hĺbku dedenia tried, ktoré má programátor aktuálne otvorené a pracuje s nimi.

8.13.2 Analýza

Pri vyhodnocovaní kvality programátora potrebuje merať požadované LOC metriky iba nad zdrojovým kódom, ktorý napísal daný programátor, respektíve ho zmazal. Na väčšine projektoch sa pracuje v tíme a výsledný zdrojový kód je spoločné autorstvo. Nad takýmto zdrojovým kódom nemôžme merať LOC metriky pre jedného programátora a na základe nich vyhodnocovať jeho kvalitu a zručnosti. Zistiť vložený a odstránený zdrojový kód programátorom môžeme tak, že si najskôr uložíme pôvodný súbor a práve uložený súbor. Nad týmito dvom súbormi by bolo vhodné vykonať diff a následne merať požadované LOC metriky iba nad diffmi.

8.13.3 Návrh

Pri otvorení .java súboru v Eclipse sa uloží do hashtable celý jeho zdrojový kód ako string, kde kľúčom bude daná relatívna cesta k súboru. Po pridaní zdrojového kódu programátorom a následnom uložení súboru sa vyberie zdrojový kód ako string, ktorý sa potrebnou metódou v knižnici *google-diff-match-patch* porovná na diff s pôvodným prislúchajúcim súborom v hashtable. Následne sa získané diffy poskytnú požadovaným LOC metrikám na spracovanie.

8.13.4 Implementácia

Na získanie diffov sme použili knižnicu google-diff-match-patch, ktorá nám vráti LinkedList diffov, ktorým následne prechádzame a meriame tieto LOC metriky:

- InsertLinesAll - počet vložených riadkov aj prázdnych
- InsertCodeLines - počet vložených riadkov kódu bez prázdnych

- InsertCommentLines - počet vložených riadkov komentáru bez prázdnych
- DeleteLinesAll - počet odstránených riadkov aj prázdnych
- DeleteCodeLines - počet odstránených riadkov kódu bez prázdnych
- DeleteCommentLines - počet odstránených riadkov komentáru bez prázdnych. K tým metrikám pripájame ešte LOC metriku nad celým uloženým zdrojovým kódom.
- DocLines - počet riadkov celého uloženého dokumentu bez prázdnych

8.14 Parser na dáta

8.14.1 Úloha

Dokončiť parser na dáta a ich ukladanie do databázy.

8.14.2 Analýza

Úprava plánovanej úlohy sťahovania dát. JSON knižnica pre parsovanie dát. Rozšírenie konfigurácie o *threadpool*.

9 Šprint 8

9.1 Zmeniť logiku listenera chybovosti pri build-e

9.1.1 Úloha

Zmeniť listener pre počet error pri builde tak, že sa errors budú monitorovať pri každom ukladaní a odošlú sa iba pri builde alebo zavretí celej aplikácie Eclipse. K tejto metrike treba monitorovať aj warnings.

9.1.2 Analýza

Predchádzajúci spôsob merania chybovosti nebol vhodný, pretože veľa programátorov nespúšťa program s chybami. Ak chceme merať chybovosť programátora, tak je vhodné ju merať nad celým projektov vo väčšom časom úseku. Tento časový úsek predstavuje úsek medzi buidami programu, respektíve nemusí dôjsť k buildu. V tomto prípade je to celý beh aplikácie Eclipse až po jej ukončenie.

9.1.3 Návrh

Pri otvorení .java súboru sa zaznamená jeho počet chýb a upozornení do globálnej hash tabuľky na základe relatívnej cesty k súboru. Následne sa pri každom ukladaní súborov spraví rozdiel počtov chýb a upozornení medzi pôvodným a uloženým súborom. Tieto rozdiely sa uložia inkrementovaním počítadiel jednotlivých projektov v hash tabuľkách pre chyby a upozornenia. Meraná metrika sa odošle do UACA iba pri builde alebo vypnutí Eclipse.

9.1.4 Implementácia

Tento listener na chyby a upozornenia sme naimplementovali podľa návrhu s využitím abstraktnej triedy WorkbenchAdapter z PerConIK, ktorou sme zabezpečili odosielanie dát pri vypnutí Eclipse overridnutím metódy preShutdown(IWorkbench workbench, boolean forced). Overridnutím metódy launchChanged(ILaunch launch) z PerConIK LaunchListener sme zabezpečili odosielanie dát pri builde. V tomto listenery meriame nasledovné metriky:

- *ErrorsCreate* - celkový počet vytvorených nových chýb v celom projekte
- *ErrorsFix* - celkový počet odstánených chýb v celom projekte
- *WarningsCreate* - celkový počet vytvorených nových upozornení v celom projekte
- *WarningsFix* - celkový počet odstránených upozornení v celom projekte

9.2 Zmeniť získavanie ID projektu v plugine

9.2.1 Úloha

Zmeniť získavanie príslušného ID projektu pri ukladaní súborov.

9.2.2 Analýza

Predchádzajúce získavanie ID projektu vychádzalo z predpokladu, že má programátor aktívny jeden dokument a viacej neaktívnych dokumentov len z jedného konkrétneho projektu. V tomto prípade sa ID projektu hľadá na základe aktívneho IFile objektu nad daným dokumentom v Eclipse, ktorý prislúcha k IProject objektu, cez ktorý je možné získať ID projektu. Lenže vo workspace môže byť viacej projektov a aktívny dokument v Eclipse nemusí byť pri ukladaní z práve projektu, na ktorom sa pracovalo. Taktiež sa môže pracovať na viacerých projektoch naraz. Týmto dochádza k priradeniu nameraných metrík iba k jednému projektu ako by sa malo reálne priradiť.

9.2.3 Návrh

Naše listenery sú nastavené so zrnitosťou na ukladanie súborov. Pri ukladaní súborov zistíme relatívne cesty k daným súborom, ktoré využijeme na vytvorenie IFile objektov. Cez tieto objekty následne pristúpime k relevantným IProject objektom a získame požadované ID projektov.

9.2.4 Implementácia

Na základe návrhu sme naimplementovali metódu, ktorá sa zavolá zvlášť nad každým ukladaným súborom ešte pred odoslaním nameranej metriky na základe zaregistrovaných listenerov. Táto metóda je implementovaná nasledovne:

```
public String getProjectId(String relativePath){
try {
IPath path = new Path(relativePath);
      IFile file = (IFile) ResourcesPlugin.getWorkspace().getRoot().getFile(path»
        » );
      IProject project = file.getProject();
      String id = project.getPersistentProperty(new QualifiedName("", "»
        » IDEMPROJID"));
      return id;
    } catch (CoreException e) {
      return null;
    }
}
```

Ukážka 6: *Ukážka implementácie*

9.3 Duplicitný zdrojový kód na posielanie dát do UACA

9.3.1 Úloha

Odstrániť duplicitný kód zabezpečujúci funkcionalitu odosielania našich nameraných metrík do UACA.

9.3.2 Analýza

Kód potrebný na odosielanie nameraných metrík do UACA sa nachádza v každom našom naimplementovanom listenery. Tieto listenery sú na základe tohto kódu zbytočne neprehľadné, a preto je potrebné túto duplicitu odstrániť.

9.3.3 Návrh

Vytvoríme si novú triedu SendData.java, v ktorej sa budú nachádzať všetky metódy potrebné na odosielanie našich metrík. Odosielanie sa inicializuje zavolaním statickej metódy process v každom listenery nad danou triedou.

9.3.4 Implementácia

Vytvorili sme potrebnú triedu spolu s požadovanými metódami a následne sme v každom listenery zavolali odosielanie nasledovne:

```
SendData.process(time, part,"&IDEMplugin&{\"Version\":\"" + Activator.getVersion»
  » ()
        + "\",\"IdProject\":\"" + projectId
        + ... + "}");
}
```

Ukážka 7: *Ukážka implementácie*

9.4 Listener čitateľnosti zdrojového kódu

9.4.1 Úloha

Vytvoriť nový listener, ktorý bude poskytovať informáciu o tom ako čitateľný kód jednotlivý používatelia tvoria. Poskytne jednotlivé metriky, ako dĺžka riadka, atď.. ktoré sú relevantné na určenie kvality kódu.

9.4.2 Analýza

Na jednoznačné určenie kvality kódu asi nikto zatiaľ žiadnu metriku nevymyslel. Určenie čitateľnosti je veľmi náročná úloha, ktorá si vyžaduje hlbokú analýzu.

V článku [1] sa na čitateľnosť pozreli z rôznych aspektov a veľmi nás inšpirovali. Veľmi nás prekvapilo, ktoré metriky nepovažovali za dôležité. Veľmi málo zaujímavá metrika je podľa nich dĺžka premenných, ktorá síce s narastajúcou dĺžkou opticky môže znižovať čitateľnosť, ale zas vďaka dlhým názvom premenných, ktoré sami vysvetľujú čo uchovávajú, to prispieva k čitateľnosti. Samozrejme, že nie vždy to platí a preto sme sa rozhodli túto metriku vynechať. Podobne by sa dali chápať aj komentáre v zdrojovom kóde. Ak niekto napíše k nejakému bloku krátky opis, môže to pomôcť túto časť kódu lepšie pochopiť. Avšak cieľom každého programátora by malo byť písanie tak jednoduchých a zrozumiteľných blokov, že

[1] RAYMOND P.L. BUSE, WESTLEY R. WEIMER, A metric for software readability, ISSTA 08' Proceedings of the 2008 international synopsium on Software testing and analysis, 2008, s. 121-130

komentáre nie sú vôbec potrebné. Z tohto pohľadu by sa dalo povedať, že všetky komentáre znižujú čitateľnosť zdrojových kódov, avšak nič sa nedá generalizovať a k nejakým pokročilým algoritmom, ktoré sa nedajú veľmi zjednodušiť sa vždy hodia komentáre. Preto sme rozhodli aj túto metriku vynechať. Ďalej v článku sú opísané metriky, ktoré významne ovplyvňujú čitateľnosť sú najmä:

- *dĺžka riadka* opticky zlé, ak je dlhý
- *početintifikátorov na riadok* stráca sa prehľadnosť
- *počet príkazov na riadok* písať if na jeden riadok, veľmi kazí čitateľnosť a stráca sa prehľadnosť

9.4.3 Návrh

Na meranie jednotlivých metrík čitateľnosti kódu sme sa rozhodli použiť listener na uloženie dokumentu. Zaujímajú nás jednotlivé zmeny, ktoré vykonal aktuálny používateľ a akú má táto novo pridaná časť čitateľnosť. Preto používame snapshot, ktorý získame ako rozdiel súborov pri otvorení a následnom uložení. Návrh fungovania pluginu s novým listenerom:

1. používateľ otvorí zdrojový kód, alebo vytvorí nový
2. uloži sa snapshot
3. používateľ uloží súbor
4. listener zachytí akciu
5. listener porovná stav pôvodného a novo uloženého súboru
6. vyhľadá všetky zmeny a nad nimi vypočíta jednotlivé metriky
7. listener odošle dáta na server

9.4.4 Implementácia

Naprogramovanie tohto pluginu prebiehalo v Jave v projekte so všetkými ostatnými listenermi, kvôli tomu, aby sa dal celý plugin nainštalovať naraz. Jednotlivé metriky rátame pomocou reťazcových metód v Jave.

Odosielanie jednotlivých metrík je zabezpečené rovnako ako pri iných listeneroch, cez statickú metódu, ktorá odošle namerané počty chýb vo formáte JSON zamaskované v zmene zdrojového kódu. V tejto fáze posielame dokopy 6 čísel:

- priemerná dĺžka riadka
- maximálna dĺžka riadka
- číslo riadka, ktorý je najdlhší
- priemerný počet identifikátorov na riadok
- maximálny počet identifikátorov na riadok
- priemerný počet príkazov na riadok
- maximálny počet príkazov na riadok

9.5 Zobrazovanie zaznamenaných metrík

9.5.1 Úloha

Získať z databázy agregované hodnoty meraných metrík pre zobrazenie používateľom.

9.5.2 Analýza

Do databázy ukladáme všetky získané záznamy spolu s časom ich zachytenia. To umožňuje neskôr agregovať záznamy do skupín napríklad podľa používateľa, projektu, názvu metriky alebo ich kombináciou. Takto je možné spätne získať hodnoty k určitému času a zobraziť tak historické hodnoty.

9.5.3 Návrh

Aktuálnej verzii nášho projektu agregujeme všetky záznamy k aktuálnemu dátumu, čím získavame aktuálne hodnoty (napríklad hodnôt z jednotlivých záznamov o pridaní a odstránení riadkov získame objem doterajšieho príspevku používateľa k projektu).

9.5.4 Implementácia

Pri implementácii boli použité funkcie poskytnuté Spring frameworkom pre prácu s MongoDB (obr. 8.6.1). Po pochopení poskytnutej funkcionality sa z nej stáva silný nástroj pre rýchle vytváranie agregácií.

9.6 Modul porovnania podobnosti

9.6.1 Úloha

Vytvoriť modul porovnania podobnosti ako bod rozšírenia pre plugin. Tento modul porovnania podobnosti bude slúžiť na zistenie podobných častí kódu nad projektom, v ktorom sa momentálne realizuje jeho tvorba.

9.6.2 Analýza

Keďže vytvárame nový bod rozšírenia do nášho Eclipse pluginu, je nutné zistiť, ako funguje vytváranie nového pridaného menu pre Eclipse, ktorý bude tento modul poskytovať. Zistili sme, že sa táto funkcionalita sa dá pomerne jednoducho implementovať pomocou pridania nových tried, konkrétne *NewAction*, *SampleHandler* a *SampleAction*, ktoré sme však nepremenovávali. Ďalej bude potrebné vytvoriť XML súbor, ktorý bude tieto triedy na seba vhodne mapovať a vytvoriť v ňom príslušné operácie. Následne bolo potrebné zistiť ake API by bolo najvhodnejšie na túto funkcionalitu. Analyzovali sme jednotlivé API pre vyhodnocovanie podobnosti kódu a následne vybrali JCCD API, ktoré ako jediné poskytovalo možnosť zapnúť, alebo vypnúť si jednotlivé detektory. Zvolili sme ho taktiež aj z dôvodu, že je to API určené priamo pre programovací jazyk JAVA.

9.6.3 Návrh

Vytvoríme nové menu pre Eclipse ktoré sa bude zobrazovať po nainštalovaní nášho pluginu. Toto menu vytvoríme pomocou implementácie tried *NewAction*, *SampleHandler* a *SampleAction*, ktoré budú obsahovať konkrétnu funkcionalitu potrebnú k zobrazeniu podobnosti. Následne vytvoríme XML súbor, ktorý bude jednotlivé príkazy mapovať na prostredie Eclipse. Následne vytvoríme triedu *CodeSimilarityOperations*, ktorá bude obsahovať funkcionalitu pre určenie podobnosti jednotlivých zdrojových súborov.

9.6.4 Implementácia

Implementácia prebiehala pomocou programovacieho jazyka JAVA a XML. Do prostredia Eclipse sme importovali potrebné triedy API JCCD, následne vytvorili funkcionalitu v triede CodeSimilarityOperations, kde sme pomocou tried z JCCD API vytvorili potrebné detektory. Tieto detektory mali na starosti generalizáciu mien premenných, generalizáciu deklarácií metód, generalizáciu argumentov metód a kompletizáciu kódu do bloku. Pomocou týchto detektorov sme boli schopní odhaliť podobnosť v zdrojovom kóde.

```
APipeline detector = new ASTDetector();
detector.setSourceFiles(files);
detector.addOperator(new GeneralizeMethodDeclarationNames());
detector.addOperator(new GeneralizeVariableNames());
detector.addOperator(new CompleteToBlock());
detector.addOperator(new GeneralizeMethodArgumentTypes());
//    vytvorenie detektorov
```

Obr. 21: *Ukážka implementovaných metód*

9.7 Verziovanie pluginu

9.7.1 Úloha

Vytvoriť a posielať verziu pluginu spolu s nalogovanými dátami do našej databázy z dôvodu možnosti zahodenia, respektíve filtrácie a prípadnej úpravy dát.

9.7.2 Analýza

Verzia pluginu sa ukladá v súbore *MANIFEST.MF*, kde je by default nastavená na *1.0.0.qualifier*, čo znamená, že verzia sa vytvára automatizovane. Bude preto potrebné zmeniť nastavenia tohto súboru a pridať funkciu na selekciu tejto verzie a verziu pridať do údajov, ktoré sa odosielajú.

9.7.3 Návrh

Vytvoríme funkciu *getVersion()*, ktorá nám vráti *String*, v ktorom bude konkrétna verzia pluginu, s ktorou používateľ momentálne pracuje. Ešte predtým však zmeníme súbor *MANIFEST.MF* tak, aby sme verziu mohli nastavovať ručne pri každej zásadnej zmene pluginu.

9.7.4 Implementácia

V tejto úlohe sme implementovali funkciu getVersion() v rámci triedy Activator, ktorá uchováva potrebné údaje o práve spustenom plugine. Zmeny sme implementovali v programovacom jazyku JAVA. Následne sme implementovali miernu zmenu odosielania, kde sa okrem metriky odosiela aj verzia pluginu, v ktorej bola metrika rátaná.

```
SendData.process(time, part,"&IDEMplugin&{\"Version\":\"
+ Activator.getVersion() + "\",\"IdProject\":\" // odoslanie verzie
+ projectId + "\",\"Path\":\""
+ key + "\",\"CycloCount\":"
+ cyclo + "}");
```

Obr. 22: *Ukážka implementovaných metód*

9.8 Zmeniť logiku listenera chybovosti pri build-e

9.8.1 Úloha

Zmeniť listener pre počet error pri builde tak, že sa errors budú monitorovať pri každom ukladaní a odošlú sa iba pri builde alebo zavretí celej aplikácie Eclipse. K tejto metrike treba monitorovať aj warnings.

9.8.2 Analýza

Predchádzajúci spôsob merania chybovosti nebol vhodný, pretože veľa programátorov nespúšťa program s chybami. Ak chceme merať chybovosť programátora, tak je vhodné ju merať nad celým projektov vo väčšom časom úseku. Tento časový úsek predstavuje úsek medzi buidami programu, respektíve nemusí dôjsť k buildu. V tomto prípade je to celý beh aplikácie Eclipse až po jej ukončenie.

9.8.3 Návrh

Pri otvorení .java súboru sa zaznamená jeho počet chýb a upozornení do globálnej hash tabuľky na základe relatívnej cesty k súboru. Následne sa pri každom ukladaní súborov spraví rozdiel počtov chýb a upozornení medzi pôvodným a uloženým súborom. Tieto rozdiely sa uložia inkrementovaním počítadiel jednotlivých projektov v hash tabuľkách pre chyby a upozornenia. Meraná metrika sa odošle do UACA iba pri builde alebo vypnutí Eclipse.

9.9 Plagát

9.9.1 Úloha

Vytvorenie plagátu na konferenciu IITSRC

9.9.2 Analýza

Zhromaždenie vhodných informácií na plagát. Analýza predošlých plagátov predstavených na IITSRC

9.9.3 Návrh

Vytvorenie pracovných verzií plagátov. Úpravy na spoločných stretnutiach.

9.10 Reset užívateľského hesla

9.10.1 Úloha

Umožnenie resetu hesla používateľov z administračného rozhrania.

9.10.2 Implementácia

Rozšírenie GUI. Zásah do Bean objektu používateľa.

10 Šprint 9

10.1 Testovanie inštalácie finálneho pluginu

10.1.1 Úloha

Otestovať rôzne spôsoby inštalácie nášho pluginu do Eclipse spolu s registrovaním a odregistrovaním našich listenerov a následným posielaním namonitorovaných dát.

10.1.2 Analýza

Pred testovaním inštalácie pluginu a jeho následným otestovaním funkčnosti sme zanalyzovali spôsoby inštalácie. Plugin sa môže nainštalovať dvoma spôsobmi a to v časti Help/Install New Software.. v Eclipse. Tieto spôsoby zahŕňajú inštaláciu z update site alebo lokálne zo súboru. Registrácia listenerov sa nachádza v časti Window/Preferences/PerConIK/Listeners, kde je možné registrovať listenery a korektné odosielanie metrík do Gratex databázy je možné sledovať v UACA v časti Manage Activity Cache.

10.1.3 Návrh

Testovanie bude prebiehať nainštalovaním nášho pluginu z lokálneho priečinku a z update site do Eclipse Standard 4.3.2 a Eclipse IDE for Java EE Developers v týchto variantoch:

- čistá inštalácia Eclipse bez PerConIK pluginu
- čistá inštalácia Eclipse spolu s inštaláciou PerConIK pluginu
- čistá inštalácia Eclipse spolu s odinštalovaným PerConIK pluginom
- používaný Eclipse bez PerConIK pluginu
- používaný Eclipse spolu s inštaláciou PerConIK pluginu
- používaný Eclipse spolu s odinštalovaným PerConIK pluginom

Následne sa v každej inštalácií skontroluje registrácia všetkých našich listenerov. Naprogramovaním vhodného zdrojového kódu namonitorujeme naše metriky a následne ich skontrolujeme v UACA.

10.1.4 Implementácia

Testovanie prebehlo v Eclipse Standard 4.3.2 aj v Eclipse IDE for Java EE Developers na základe návrhu vo všetkých požadovaných variantoch. Nájdené nedostatky pri testovaní boli odstránené pri vytváraní novej verzie pluginu.

10.2 Pridanie interaktívnosti spline chart grafu

10.2.1 Úloha

Umožniť používateľovi interagovať so spline chart grafom pre sprehľadnenie vizualizovaných dát.

10.2.2 Analýza

Vo východzom stave sa v grafe zobrazia údaje všetkých členov daného projektu, čo je však pre porovnávanie často nežiadúce. Preto je potrebné umožniť používateľovi filtrovať členov, ktorých údaje chce mať zobrazené a tiež meniť poradie vrstiev.

10.2.3 Návrh

Filtrovanie je možné riešiť pridaním ovládacích prvkov, pomocou ktorých by používateľ nastavoval parametre zobrazenia, to by však spôsobilo zvýšenie počtu ovládacích prvkov na obrazovke, čo by mohlo znížiť prehľadnosť stránky. Rozhodli sme sa preto pokiaľ možno vyriešiť filtrovanie priamo interakciou s grafom. Klikaním na mená používateľov v legende sa udaje o týchto používateľoch skrývajú, resp. zobrazujú a klikaním na jednotlivé vrstvy grafu, či prechádzaním ponad mená v legende sa príslušná vrstva presunie na vrch.

10.2.4 Implementácia

Keďže grafy sú vygenerované vo formáte SVG, umožňujú interakciu s používateľom, môžu byť ovládané pomocou javascriptov a tiež je na ne do určitej miery možné aplikovať CSS štýly. Nevýhodou tohto riešenia však môže byť nevhodnosť takéhoto ovládania na mobilných zariadeniach a slabšia/žiadna podpora SVG v starších internetových prehliadačoch.

10.3 Gui pre modul podobnosti

10.3.1 Úloha

Úlohou bolo vytvoriť prehľadnejšie grafické rozhranie pre modul porovnania podobnosti.

10.3.2 Analýza

JCCD API nedovoľuje vytvoriť si vlastnú metódu výpisu. Výpis sa dal zobrazovať iba do konzoly, čo nám neprišlo ako najvhodnejšie, preto je potrebné tento výpis presmerovať do vhodného formátu.

10.3.3 Návrh

Vytvoríme novú triedu *SimilarityDialog*, ktorá rozširuje triedu Jdialog. Následne v nej vytvoríme potrebnú funkcionalitu, konkrétne presmerovanie výpisu do Stringu, ktorý budeme následne posielať do vytvorenej *textArey*, ktorú implementujeme v triede *SimilarityDialog*.

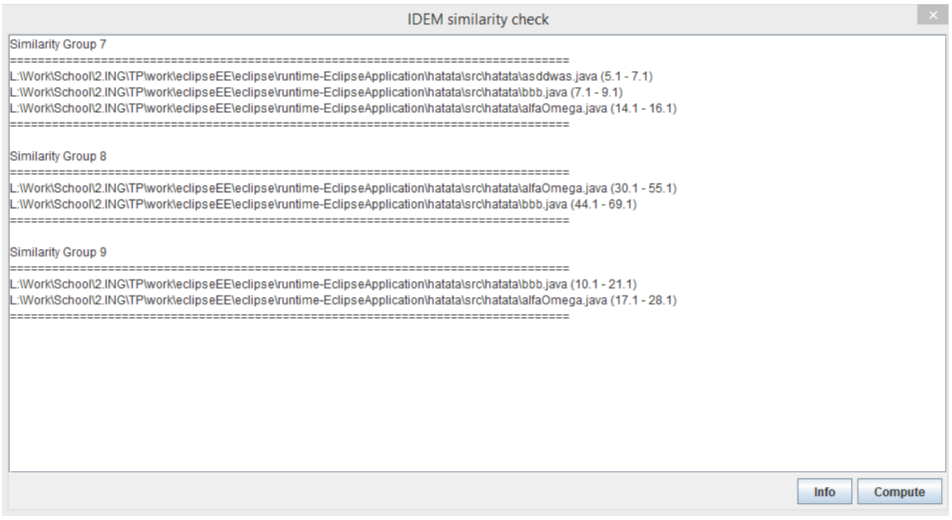
10.3.4 Implementácia

Vytvorili sme triedu SimilarityDialog, ktorá obsahuje GUI rozhranie používateľa a v prehľadným spôsobom mu zobrazuje podobnosť v danom súbore s ostatnými súbormi. Pridaná bola taktiež funkcionalita pre zobrazenie informácie, čo jednotlivý záznam znamená. Celá funkcionalita bola implementovaná pomocou programovacieho jazyka JAVA.

```
public void actionPerformed(ActionEvent arg0) {
textArea.setText("");
Exception Er = null;
      try{
      CodeSimilarityOperations CSO = new CodeSimilarityOperations();
      JCCDFile[] files = CSO.createJCCDfileArray(getProject(),  false);
      String output = null;
      output = CSO.compute(files);
      }catch(Exception e){
            Er = e;
            textArea.setText("CANT PERFORM OPERATION
            BECOUSEOF>>>"+"\n"+e.getMessage());
      }
      if(GlobalVariables.selectedString != null){

      textArea.setText(GlobalVariables.selectedString)
      }
      else{
      JOptionPane.showMessageDialog(contentPanel, "Cant perfom operation
due to some errors in your code! " + "\n" + "** " +Er.getMessage() +" **" ,
"CAN NOT PERFORM OPERATION", 2);
      }
```

Obr. 23: *Ukážka implementovaných metód*



Obr. 24: *Ukážka kontroly podobnosti*

11 Šprint 10

11.1 Čitateľnosť kódu

11.1.1 Úloha

Definovať vzorec, ktorým budeme vedieť odhadnúť čitateľnosť kódu

11.1.2 Analýza

Na vyhodnocovanie čitateľnosti kódu zatiaľ neexistujú nástroje, ktoré by sme mohli použiť. Preto bolo potrebné vytvoriť súčasť pluginu, ktorá zbiera potrebné metriky pre vyhodnocovanie tohto problému. Opis tvorby tohto pluginu nie je predmetom tejto úlohy. Na základe vedeckej práce[2], sme zistili tieto skutočnosti:

- AvgIdentifier – priemerný počet identifikátorov v riadku(napríklad názov metódy, triedy, premennej), ktorý má váhu 1.
- MaxIdentifier – maximálny počet identifikátorov na riadok – váha 0,7
- AvgCommand – priemerná počet príkazov na riadok – váha 0,6
- MaxCommand – maximálny počet identifikátorov v riadku – váha 0,4
- MaxLineLength – maximálna dĺžka najdlhšieho riadku – váha 0,8
- AvgLineLength – priemerná dĺžka riadku – váha 0,9

Pomocou vhodného vzorca sa vďaka týmto metrikám môžeme dopracovať k číselnej reprezentácií čitateľnosti kódu, kde by určité číselné intervaly mohli odhaliť mieru čitateľnosti kódu.

11.1.3 Návrh

$$Cit = \frac{AvgL*1+MaxI*0.7+AvgC*0.6+MaxC*0.4+Abs(MaxL*0.8-AvgL*0.9)}{AvgL*0.9}$$

[2]RAYMOND P.L. BUSE, WESTLEY R. WEIMER, A metric for software readability, ISSTA 08' Proceedings of the 2008 international synopsium on Software testing and analysis, 2008, s. 121-130

11.1.4 Implementácia

Daný vzorec sme na základe analýzy vyprodukovali a testovali na vzorke 20 kódov, kde sme odhalili, že najlepšiu mieru čitateľnosti majú kódy s hodnotou približne $Cit = 0.5$ a priemerne čitateľné kódy $Cit = 2.5$. Je však nutné poznamenať, že tento prístup je len prototypom a preto bude zrejme nutné tento prístup pozmeniť. Na to však potrebujeme väčší počet dát.

11.2 Finálna verzia pluginu za letný semester

11.2.1 Úloha

Vytvoriť finálnu verziu pluginu v letnom semestri. Odstrániť prípadné chyby a finálnu verziu nasadiť na náš update site.

11.2.2 Analýza

V analýze sme zisťovali, či finálny plugin neobsahuje nejaké chyby. Podrobným testovaním sa zistili, že chyby sa v plugine nevyskytujú.

11.2.3 Návrh

V našom *Feature* projekte *IDEMPlugin.feature* aktualizujeme verziu pluginu a následne ho exportujeme, čím vytvoríme priečinok obsahujúci potrebné inštalačné súbory.

11.2.4 Implementácia

V rámci implementácie tejto úlohy sme pracovali s programovacím jazykom JAVA, XML a pracovali sme v prostredí Eclipse. Následne sme vytvorený priečinok s inštalačnými súbormi pomocou WinSCP nakopírovali na server. Potom sme finálnu verziu otestovali tak, že sme sa pokúsili o aktualizáciu pluginu nainštalovaného v Eclipse. Pri inštalácií k chybám nedošlo a neboli odhalené ani počas používania novej verzie pluginu.

11.3 Zisťovanie autorstva kódu za pomoci knižnice JGit

11.3.1 Úloha

Úlohou bolo vytvoriť funkcionalitu, ktorá by zistila autora konkrétnej časti kódu v zdrojovom kóde za využitia funkcionality nástroja GIT.

11.3.2 Analýza

Na použitie nami navrhnutých metrík je potrebné vedieť, ktorú časť kto naprogramoval, aby danému používateľovi v metrikách zohľadňovali iba jeho vlastné časti kódu, ktoré pridal alebo upravil. Vďaka tomuto nie sú výsledky použitých metrík ovplyvnené druhými používateľmi, ale sú počítané iba z jeho vlastnej práce.

11.3.3 Návrh

Pomocou knižníc prepojíme projekt v Eclipse s verziovacím nástrojom GIT. Následne dokážeme identifikovať autora každého riadku kódu na základe komitov.

11.3.4 Implementácia

Na zistenie autora sme použili knižnicu JGit, ktorá je súčasťou vývojového prostredia Eclipse. S knižnicou sa pracovalo veľmi ťažko, pretože sa k nej nachádza veľmi málo príkladov. Nakoniec sa nám podarilo zistiť ako s danou knižnicou pracovať. Použili sme funkciu, ktorá nám z posledného commitu zistí ku každému riadku kódu autora a aj čas pridania alebo úpravy daného riadku kódu. Navrhnutá funkcia má ako parameter relatívnu cestu daného java súboru, o ktorom chceme informácie. Následne funkcia vráti zoznam autorov aj s časmi, kde poradie v zozname je zhodné s poradím riadkov v kóde, teda napríklad piaty záznam v zozname prislúcha k piatemu riadku v kóde. Takto je možné z každého java súboru zistiť presné informácie o autorovi daného riadku kódu.

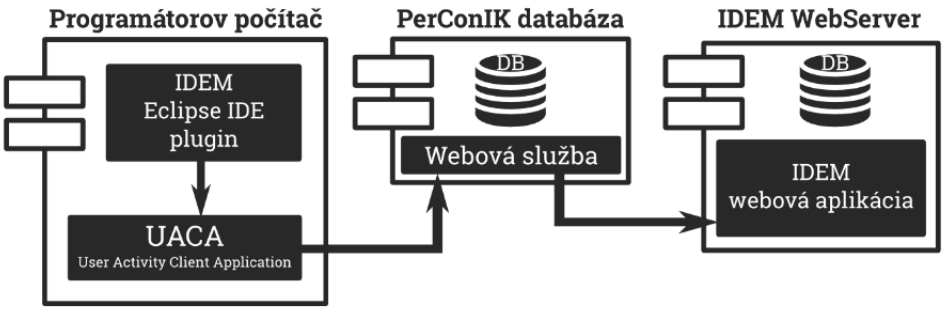
12 Celkový pohľad na projekt

V tejto časti je uvedená sumarizácia práce na projekte IDEM. Jednotlivé kapitoly popisujú analýzu a celkový pohľad na náš produkt.

12.1 Analýza

Základným prvkom analýzy v tomto semestri boli určenie kvalitatívnych metrík programového kódu, pomocou ktorých sa snažíme používateľovi ukázať, v akom stave sa jeho kód nachádza v zmysle porovnania týchto kvalít s ostatnými členmi vývojového tímu. Na základe výsledkov analýzy sme v jazyku JAVA pomocou nami vytvoreného pluginu postupne implementovali metriky: Cyclomatic Complexity, Lines of Code (viacero druhov), Errors (viacero druhov), počet statických atribútov a Depth of Inheritance. Následne sme sa v analýze zaoberali aj možnosťou porovnania podobnosti programového kódu a pomocou JCCD API sme implementovali aj túto funkcionalitu. Ďalšou časťou analýzy bolo určenie čitateľnosti programového kódu, kde sme na základe vedeckých článkov[1], implementovali ďalšiu metriku Code Readability (viacero druhov), pomocou ktorej sa snažíme určiť mieru čitateľnosti programového kódu. Analýzu tvorilo celkovo viacero častí nášho projektu – ako vytvoriť potrebné grafy k zobrazeniu metrík, ako implementovať jednotlivé metriky a plugin ako celok.

12.2 Architektúra riešenia



Obr. 25: *Ukážka architektúry*

Architektúra nášho systému pozostáva z troch hlavných častí:

- Programátorov počítač – IDEMplugin – nami vytvorený plugin, ktorý ohodnocuje programový kód počas jeho tvorby a následne do služby UACA odosiela tieto údaje maskované za event IdeCodeOperationDto. UACA následne v špecificky definovaných cykloch tieto údaje odosiela do PerConIK databázy.
- PercConIK databáza – Databáza ku ktorej nemáme administrátorský prístup, avšak ani ho vďaka maskovaniu dát nepotrebujeme. Pomocou našej automatizovanej metódy fetch sa potrebné údaje z eventu IdeCodeOperationDto a ostatných eventov s biografickými dátami ukladajú v definovanom intervale do našej databázy.
- IDEM WebServer – Obsahuje databázu (MongoDB) v ktorej sú pomocou Json uložené potrebné údaje ktoré v našej webovej aplikácií IDEM pomocou grafov prehľadne zobrazujeme

12.3 Použité technológie

Pri procese implementácie tohto projektu sme používali prevažne programovací jazyk JAVA spolu s databázou MongoDB a technológiami Jspring a Maven. Grafy boli vytvárané pomocou javascriptovej vizualizačnej knižnice D3.js.

12.4 Opis výsledného produktu

Výsledný produkt sa skladá z dvoch hlavných častí: webová aplikácia a Eclipse plugin. Webová aplikácia slúži najmä k zobrazovaniu štatistík kvality kódu programátora. Eclipse plugin slúži k zachytávaniu a rátaniu metrík nad práve vytváraným programovým kódom.

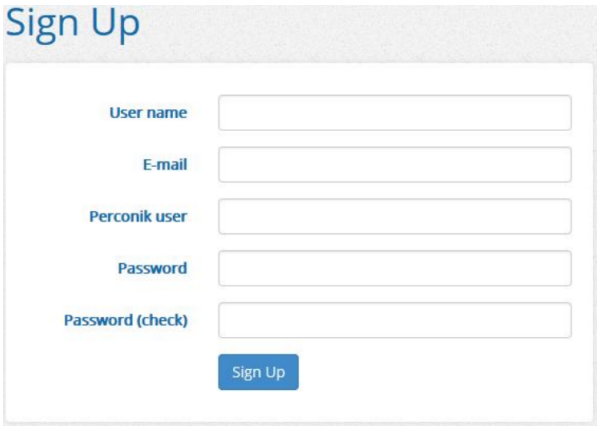
12.4.1 Web Aplikácia

Ako bolo spomenuté vyššie naša webová aplikácia slúži primárne na prezeranie si štatistík o kvalite programového kódu. Poskytuje možnosť tvorby nových používateľov a manažment projektov. Konkrétne tvorbu nového projektu, zapisovanie členov do projektu a prideľovanie práv

Implementácia

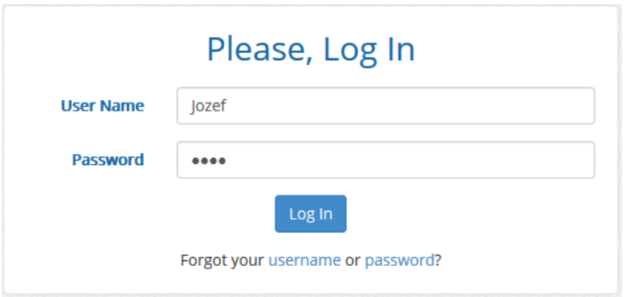
Implementácia našej webovej aplikácie prebiehala pomocou programovacieho jazyka JAVA a pomocou frameworku Jspring. Postupne na nej bola pridávaná funkcionalita. Momentálny stav zahŕňa túto funkcionalitu (spolu s obrázkami zo zaujímavých častí funkcionality):

- Vytvorenie nového používateľa – vyplnenie potrebných údajov a prepojenie novovzniknutého účtu s loginom v UACA.



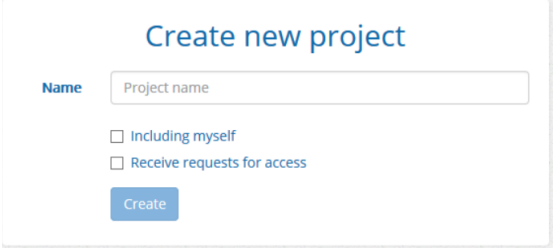
Obr. 26: *Formulár potrebný na prepojenie s UACA*

- Login – prihlásenie existujúceho používateľa do systému, prípadné preposielanie zabudnutého hesla, alebo mena.



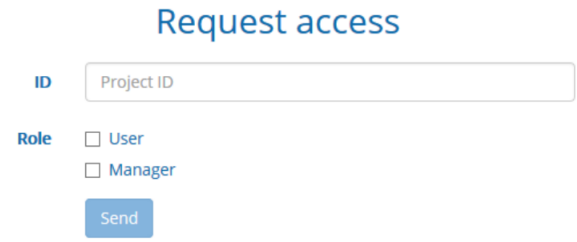
Obr. 27: *Formulár potrebný na prihlásenie*

- Login – v našom systéme si používateľ vytvorí nový projekt,ktorému vytvorí názov a môže sa doň hneď prida.



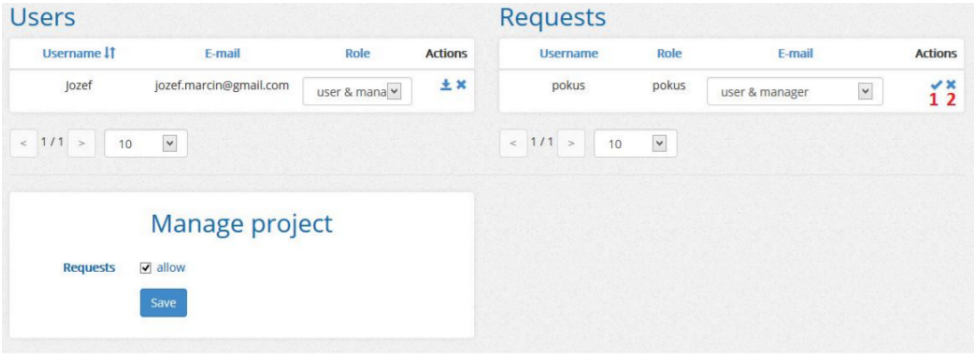
Obr. 28: *Formulár potrebný na vytvorenie projektu*

- Vyžiadanie prístupu k existujúcemu projektu – používateľ môže odoslať žiadosť o pridanie do projektu s dvoma typmi rolí – manažér a používateľ.



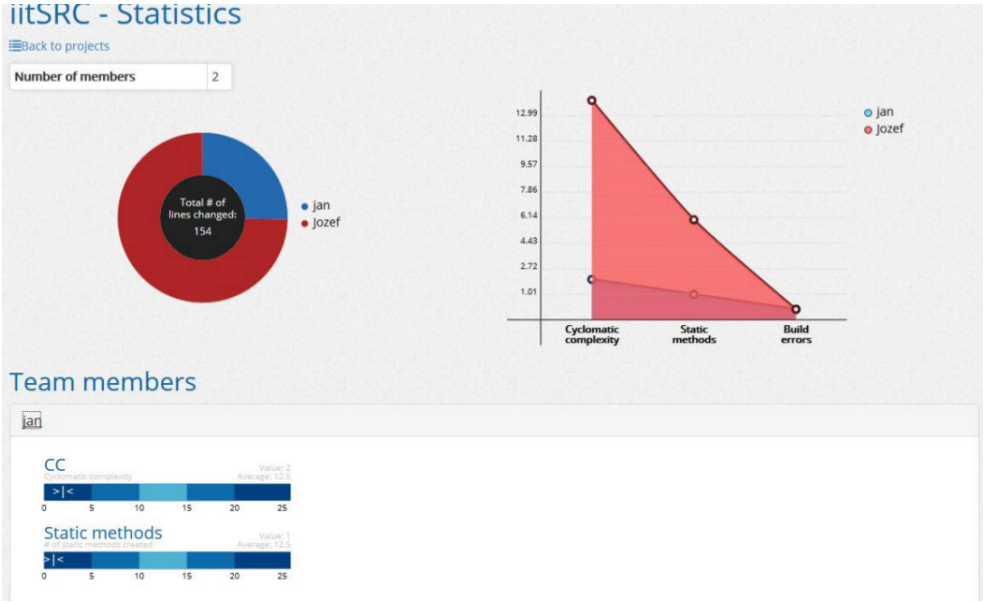
Obr. 29: *Formulár potrebný na požiadanie o práva*

- Následne môže manažér projektu používateľa pridať pomocou akcie pridaj(1) alebo odobrať pomocou akcie vymaž(2)



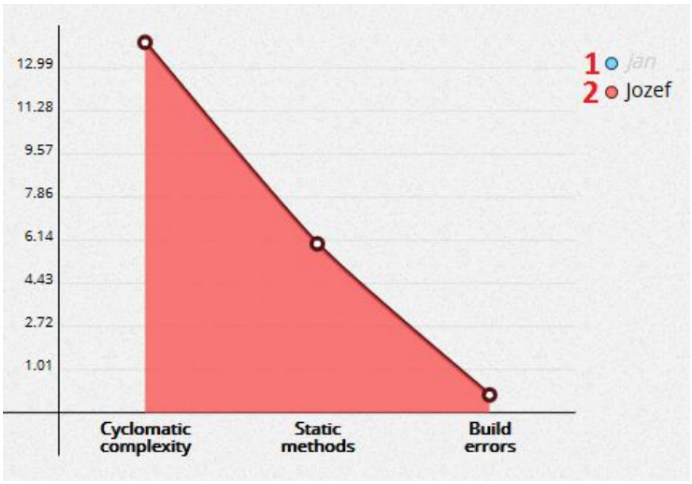
Obr. 30: *Ukážka formuláru na pridanie používateľa a aj vymazanie*

- Následne si používateľ v roli manažér môže prezerať všetky údaje od všetkých používateľov(pre zjednodušenie obrázok niektorých z nich)



Obr. 31: *Ukážka grafov*

- Jednotlivé grafy je možné editovať – vybrať iba určité osoby (1 – používateľa „jan“ sme vypli zo zobrazovania v grafe, 2– v grafe sme nechali iba používateľa „Jozef“)



Obr. 32: *Ukážka grafov*

Ďalšia funkcionalita:

- Zobrazenie grafov pre metriku cyklomatickej zložitosti
- Zobrazenie grafov pre metriku počtu statických atribútov

- Zobrazenie grafov pre metriku počtu chýb
- Zobrazenie celkového pomeru vytvorených riadkov kódu v rámci tímu / jednotlivca
- Zobrazenie úrovne čitateľnosti kódu (počas tvorby dokumentu ešte nebola vo web aplikácií spustená)
- Možnosť mazania vytvorených projektov
- Možnosť prezretia si všetkých svojich nalogovaných dát
- Editácia používateľov pomocou admin módu

12.4.2 Eclipse plugin

Náš Eclipse plugin slúži na rátanie metrík nad programovým kódom a taktiež obsahuje modul porovnania podobnosti programového kódu. Metriky sa následne odosielajú pomocou služby UACA do PerCoNIK databázy, odkiaľ sú pomocou automatizovanej funkcie dolované do našej databázy.

Implementácia:
Plugin sme vyvíjali v programovacom jazyku JAVA. Autorstvo programového kódu, nad ktorým metriky rátame, sme určovali pomocou metódy DIFF. To znamená, že sme metriky rátali nad poslednou zmenou v programovom kóde priamo počas jeho tvorby. Postupne sme k pluginu pripájali nasledovnú funkcionalitu:

- Rátanie metrík počtu riadkov kódu (LOC)
  - Počet všetkých riadkov kódu v celom dokumente
  - Počet všetkých posledne pridaných riadky
  - Počet všetkých posledne pridaných riadkov kódu
  - Počet posledne pridaných riadkov komentáru
  - Počet všetkých posledne zmazaných riadkov
  - Počet všetkých posledne zmazaných riadkov kódu

  - Počet všetkých posledne zmazaných riadkov komentáru

- Cyklomatická zložitosť
  - Počet posledne pridaných (kladné číslo), alebo odobraných (záporné číslo) riadiacich premenných

- Hĺbka dedenia
  - Číslo reprezentujúce počet tried, od ktorých je daná trieda odvodená

- Počet statických atribútov
  - Číslo reprezentujúce počet statických atribútov (metód, premenných, tried)

- Počet chýb
  - Počet vytvorených chýb
  - Počet opravených chýb
  - Počet vytvorených upozornení (angl. warnings)
  - Počet odstránených upozornení

- Čitateľnosť programového kódu
  - Priemerná dĺžka riadka
  - Dĺžka najdlhšieho riadka
  - Priemerný počet identifikátorov na riadok
  - Maximálny počet identifikátorov na riadok
  - Priemerná počet príkazov v riadku
  - Maximálny počet príkazov v riadku

- Selekcia textu
  - String, v ktorom je uložený momentálne vybraný (označený) kód

  - Tento údaj sa ďalej neposiela, slúži iba pre modul podobnosti programového kódu

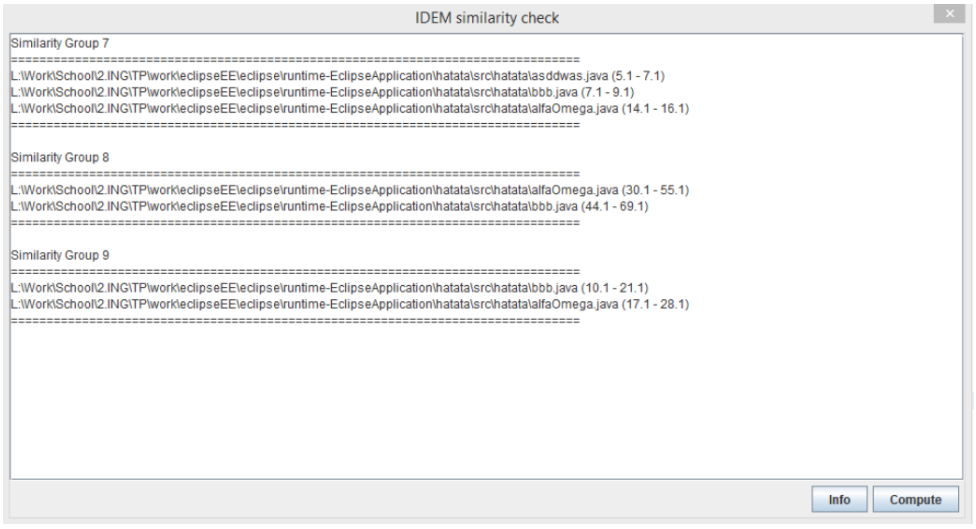
- Modul podobnosti
  - Vyhodnocovanie programového kódu a hľadanie podobných riadkov kódu kvôli znovupoužiteľnosti

Keďže Eclipse plugin nie je viditeľnou časťou práce z pohľadu používateľa, teda okrem modulu podobnosti programového kódu, preto uvádzame zopár veľmi krátkych ukážok zdrojového kódu a obrázok modulu podobnosti.

```
if (!dirty) {
            final long time = System.currentTimeMillis();
            final PluginProperties p = new PluginProperties();
            final String projectId = p.getProjectId(key);
            String path = buffer.getLocation().toString();
            String first = GlobalVariables.source.get(path);
            String all_text = TextFromFile.getTextFromFile(buffer);


if (!(first == null)) {
            LocMetricsObject lmo = new LocMetricsObject();
            lmo.getLocCount(first, all_text);
      SendData.process(time, part, "&IDEMplugin&{\"Version\":\""
                                                        +
Activator.getVersion() + "\",\"IdProject\":\"" + projectId +
"\",\"Path\":\"" + path
+ "\",\"DocLines\":" + lmo.getLine_document()
+ ",\"InsertLinesAll\":" + lmo.getLines_all_insert()
+ ",\"InsertCodeLines\":" + lmo.getLines_code_insert()
+ ",\"InsertCommentLines\":" + lmo.getLines_comment_all_insert()
+ ",\"DeleteLinesAll\":" + lmo.getLines_all_delete()
+ ",\"DeleteCodeLines\":" + lmo.getLines_code_delete()
+ ",\"DeleteCommentLines\":" + lmo.getLines_comment_all_delete()
+ "}")
}
GlobalVariables.source.put(path, all_text);
}
```

Obr. 33: *Ukážka kódu*



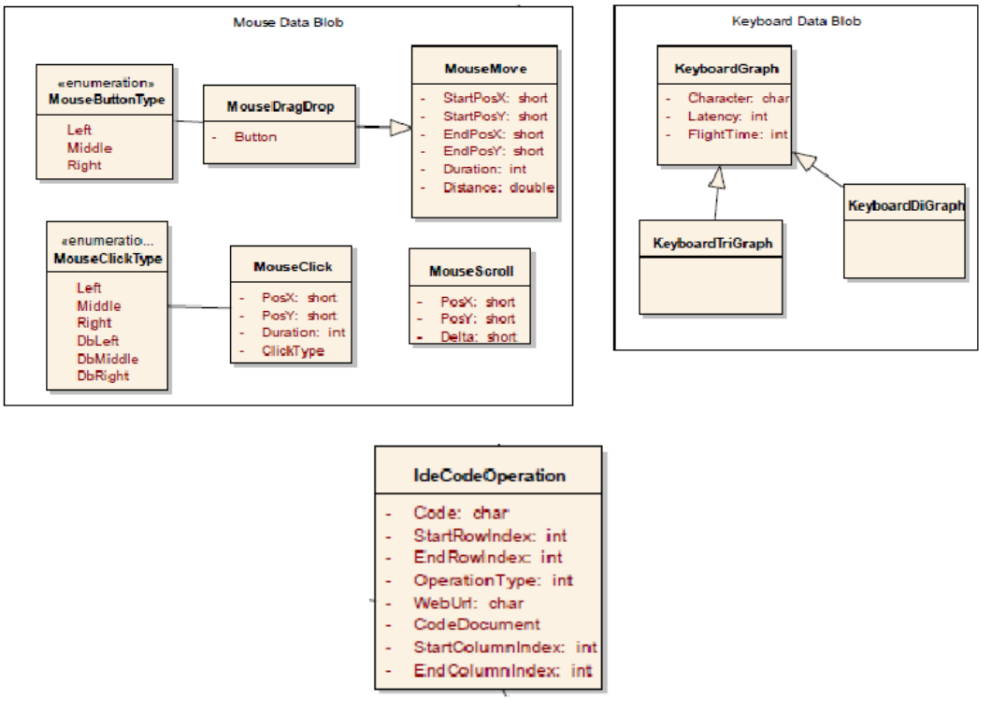
Obr. 34: *Ukážka pluginu*

12.5 Model databázy

V našej aplikácií používame databázu MongoDB, ktorá je key-value databázou a na organizáciu údajov používa kolekcie. Dáta MongoDb ukladá pomocou Json v ktorom sa aj píšu dopyty. Súhrn kolekcií v našej databáze predstavuje:

- userVectorBio – kde ukladáme biometrické dáta o používateľovi(používateľ, počet aktivitdigramy (pri digramoch aj latenciu, dobu letu a počet stlačení), klávesové skratky)
- userMetricsObject – ostatné metriky o používateľovi (používateľ, ID PerConIK, projekt, dátum (začiatok, koniec), názov metriky, hodnota metriky, relatívna cesta k súboru nad ktorým sa ráta metrika
- idemUser – kolekcia na ukladanie údajov o používateľovi (meno, heslo, perConIK ID, mail, lastfetch)
- project – kolekcia v ktorej ukladáme informácie o projekte (názov, vlastník, členovia)
- projectAccess – kolekcia ktorá obsahuje údaje o právach používateľov v projektoch

Údaje do týchto kolekcií získavame z PerConIK databázy. Konkrétne údaje o metrikách získavame tak, že pri posielaní údajov do UACA a ich následným preposlaním do PerConIK databázy maskujeme naše dáta za event IdeCodeOperationDto, aby sme nemuseli zasahovať do PerConIK databázy. Maskovanie prebieha pomocou tvorby Json objektu. Z celej PerConIK databázy teda používame najmä triedy: *IdeCodeOperation, MouseButtonType, MouseDragDrop, MouseMove, MouseClickType, MouseClick, MouseScroll, KeyboardGraph, KeyboardDigraph.*



Obr. 35: *Ukážka databázového modelu*

12.6 Testovanie

Náš produkt sme poskytli na testovanie každému kto mal záujem. Vytvorili sme dotazník, v ktorom sa jednotlivci, respektíve skupiny mali vyjadriť k nášmu produktu. V rámci testovania nám tím 04 v zložení Igor Homola, Michal Čerešňák, Michal Petráš, Martin Adámik, Samuel Benkovič, Matej Bádal a Vladimír Bošiak poskytol vo forme vyplneného dotazníka takúto spätnú väzbu:

pozitívne stránky:

- Použiteľnosť aplikácie tím 04 ohodnotil na 2 z 5, kde 1 reprezentuje výbornú použiteľnosť a 5 slabú použiteľnosť
- Tím na otázku, či sa metriky stotožňujú s ich pohľadom na vlastný kód odpovedal číslom 2 z <1-5>, kde číslo 1 reprezentuje úplne stotožnenie a 5 úplné nestotožnenie. Z toho vyplýva, že uvedené metriky celkom dobre korešpondujú s vlastným pohľadom na kvalitu kódu.

- Počas testovania tímom 04 nenastala chyba v našom produkte
- Tímu 04 sa na našej aplikácií páčil najmä dizajn a celková myšlienka projektu
- Odozva a celková rýchlosť aplikácie bola ohodnotená na 2 z 5, čo znamená relatívne dobrú odozvu

Hlavným prínosom nášho produktu podľa tímu 04 je:

- Lepší odhad jednotlivca, na akej úrovni kvality programovania v jazyku JAVA sa nachádza
- Zistenie, kto v rámci tímu dodáva najväčší prírastok čo sa kódu týka
- Možnosť zistenia rozdielov v kvalite programovania medzi jednotlivcami, čo môže pomôcť k motivácií a aj v prípade, ak jednotlivec chce druhému členovi tímu pomôcť svoje návyky vylepšiť
- Z manažérskeho hľadiska produkt prispieva k sprehľadneniu údajov o vykonanej práci.

Negatívne stránky:

- Tím používal náš produkt len v rámci testovania, nepoužívajú ho pravidelne.
- Skupina nám mierne vytkla orientáciu v aplikácií, kde by uvítali nejaký help, ktorý by ich naviedol.
- Ohodnotenie inštalácie pluginu dostalo známku 4 z <1-5>, kde 1 je jednoduchá a 5 je ťažkopádna, teda inštalácia im pripadala relatívne ťažkopádna.
- Podľa úsudku skupiny môže byť naša aplikácia nápomocná pri zlepšovaní kvality programátorských zručností zhruba na 50
- Tím 04 by vylepšil modul porovnania podobnosti, kde by namiesto momentálneho zobrazenia uvítali automatický značkovač duplicitného kódu, nie len statický výpis riadkov.
- Chýbal popis jednotlivých metrík, kde by uvítali možnosť výpisu informácie k metrike

- Vhodnosť metrík v rámci posúdenia kvality programátorských zručností bola ohodnotená na 3 z 5, čo je priemerné ohodnotenie – čiže tak 50 na 50. To znamená že celková vhodnosť nie je zlá, ale niečo tomu ešte chýba.
- Prínosnosť modulu podobnosti bola ohodnotená na 4 z 5, čo znamená, že modul podobnosti nebol pre tím 04 veľmi prínosný, čiže ho nepoužívali často
- Tím 04 na otázku či by si vedeli predstaviť náš produkt u svojho zamestnávateľa odpovedal ohodnotením 3 z 5, čo znamená, že aj áno, aj nie.

12.7 Zhodnotenie

Celkovo sa nám teda podarilo vytvoriť produkt, ktorý dokáže používateľovi zobraziť spätnú väzbu o celkovej kvalite jeho programátorských zručností.
Výsledný produkt teda obsahuje:

- Eclipse plugin
  - Vyhodnocuje kód počas jeho tvorby pomocou 6 špecializovaných metrík, z ktorých niektoré obsahujú viaceré podmetriky
  - Obsahuje modul porovnania podobnosti kódu
- Webová aplikácia
  - Umožňuje celkový manažment používateľov a projektov
  - Umožňuje zobrazovať štatistiky pre zvolené metriky

Čo sme nestihli:

- Počas tvorby finálneho dokumentu sme mali hotový Eclipse plugin do takej miery, aby sme s ním boli spokojní. Stihli sme dopracovať potrebné dolovanie dát vo všetkých smeroch, aké sme si na začiatku určili. Avšak najmä kvôli nutnosti viacerých opráv (menila sa stratégia orientovania projektu – autorstvo sme začali vyhodnocovať pomocou metódy DIFF, čomu sme museli prispôsobiť pluginy), sme nestihli dopracovať niektoré zobrazenia metrík. Konkrétne sme nestihli dopracovať:

  - Zobrazovanie čitateľnosti kódu
  - HĺbkA dedenia
  - Zobrazenie biometrických dát (používané klávesové skratky, efektívny čas v IDE atď...)

Čo sme sa naučili:
Práca na tímovom projekte nám dala veľmi veľa z pohľadu programátorského ale aj z pohľadu manažmentu projektu.

- Programátorské prínosy
  - Zdokonalenie schopností v jazyku JAVA
  - Naučenie sa nových technológii ako MongoDB, Jspring, D3.js, Json
  - GIT
- Prínosy v manažmente
  - Zvládanie vypätých situácií
  - Zdokonalenie sa v komunikácií
  - Zodpovednosť
  - Celkovo práca v tíme nám ukázala, aké sú správny manažment a motivácia dôležité

13 Používateľská príučka

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

TÍMOVÝ PROJEKT

Monitorovanie programátora v IDE

Autori:
Bc. Michal JURANYI
Bc. Ivan KOŠDY
Bc. Jozef MARCIN
Bc. Tomáš MARTINKOVIČ
Bc. Matej NOGA
Bc. Ján PODMAJERSKÝ
Bc. Juraj RABČAN

Školiteľ:
Doc. Mgr. Daniela CHUDÁ, PhD.

2013/2014

POĎAKOVANIE

Celý tím ďakuje Doc. Mgr. Daniele Chudej, PhD. za jej vedenie a cenné rady počas práce na projekte.

Obsah

1 Inštalácia UACA **2**

2 Eclipse plugin **2**

2.1 Inštalácia 2

2.2 Používanie 5

2.3 Prepojenie s projektami vo webovej aplikácii 7

2.4 Modul podobnosti 8

3 Webová aplikácia **10**

3.1 Registácia 10

3.2 Prihlásenie 11

3.3 Zmena nastavení 11

3.4 Vytvorenie nového projektu 12

3.5 Zrušenie projektu 13

3.6 Správa používateľov na projekte 14

3.7 Zaradenie do projektu 15

3.8 Zobrazenie štatistík pre používateľa 16

Zoznam obrázkov

1 Nastavenia odosielania 2
2 Inštalácia. 3
3 Inštalácia. 3
4 Adresa pluginu. 4
5 Inštalácia. 4
6 Inštalácia - upozornenie. 5
7 Preferences. 6
8 Listenery. 6
9 Nastavenie ID. 7
10 ID projektu. 8
11 Otvorené triedy. 9
12 Spustenie modulu podobnosti. 9
13 Na to sa nám objaví výsledok. 10
14 Registračný formulár 11
15 Formulár pre zmenu hesla 12
16 Formulár pre vytvorenie nového projektu 13
17 Formulár pre zrušenie projektov 13
18 Formulár pre správu používateľov 14
19 Formulár pre správu používateľov a zmenou rolí 15
20 Formulár pre zarsadenie do projektu 15
21 Zobrazenie štatistík 16

Používateľská príručka

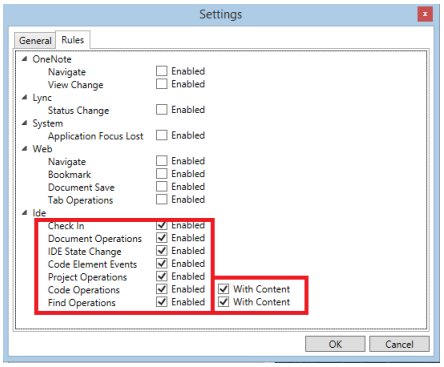
V tomto dokumente nájdete používateľskú príučku, ktorá slúži na ovládanie systému vyvíjanému na predmete Tímový projekt. Systém umožňuje monitorovanie a hodnotenie správania programátora v IDE. Skladá sa z webovej aplikácie, ktorá slúži na zobrazovanie nalogovaných dát; a z Eclipse pluginu. Na fungovanie systému je nevyhnutné mať nainštalovaný UACA Perconik, návod na túto inštaláciu je v samostatnom súbore, ktorý prichystali jej tvorcovia.

1 Inštalácia UACA

1. Zo stránky *http://perconik.fiit.stuba.sk/public/PerConIK_Tools_Install.exe* stiahneme inštalačný súbor a spustíme inštaláciu

2. Podľa príručky dostupnej na *tomto odkaze* nastavíme nástroj UACA, kde nás zaujíma len časť "Konfigurácia sledovača aktivít".

3. Po inštalácií a nastavení je potrebné nastaviť odosielanie podľa obrázka



Obr. 1: *Nastavenia odosielania*

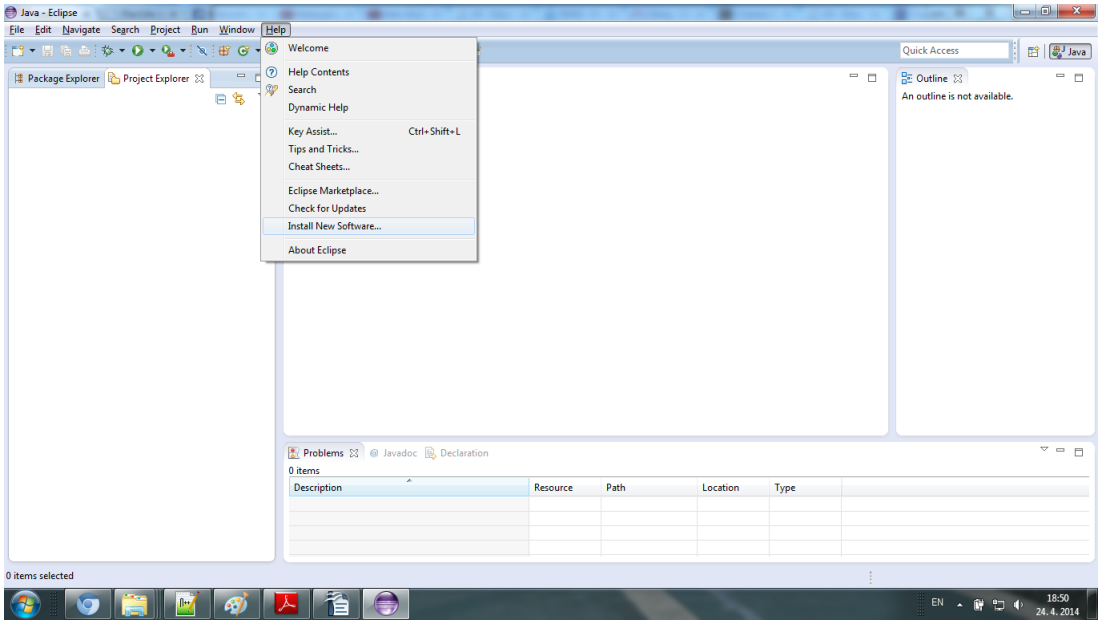
2 Eclipse plugin

Plugin je možné nainštalovať v Eclipse jednoducho pomocou update site.

2.1 Inštalácia

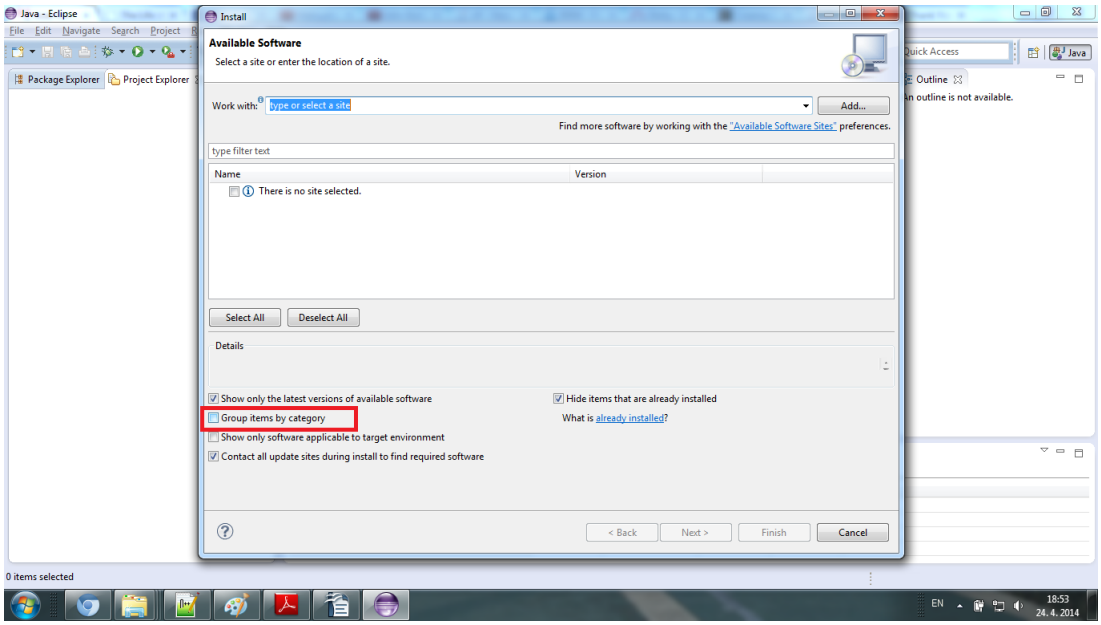
1. Spustite Eclipse

2. Zvoľte Help -> Install New Software



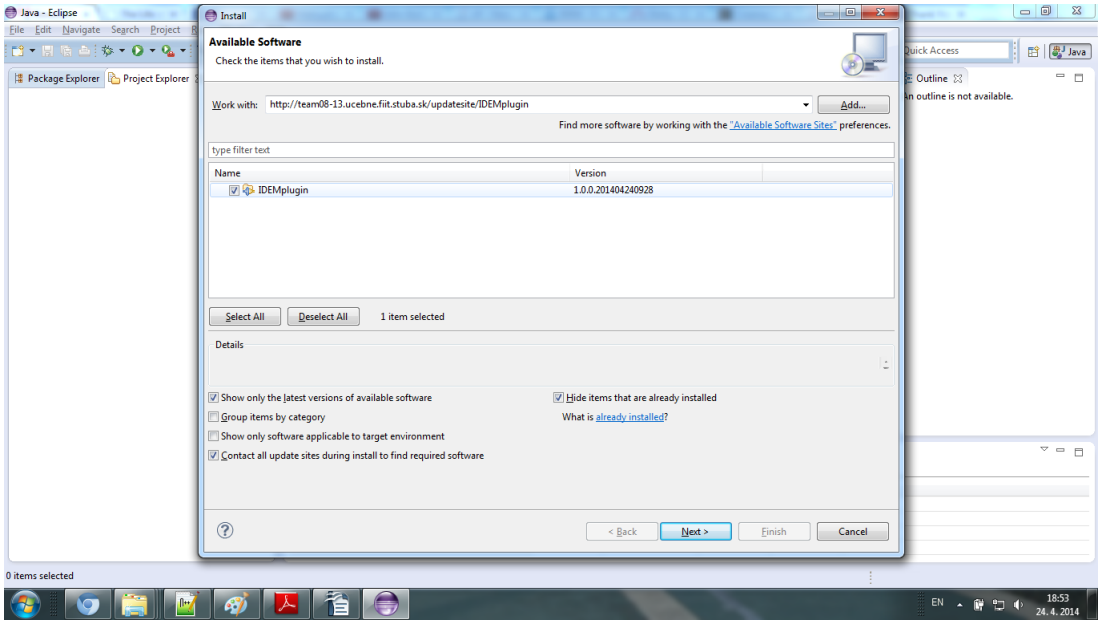
Obr. 2: *Inštalácia.*

3. Odškrknite Group items by category



Obr. 3: *Inštalácia.*

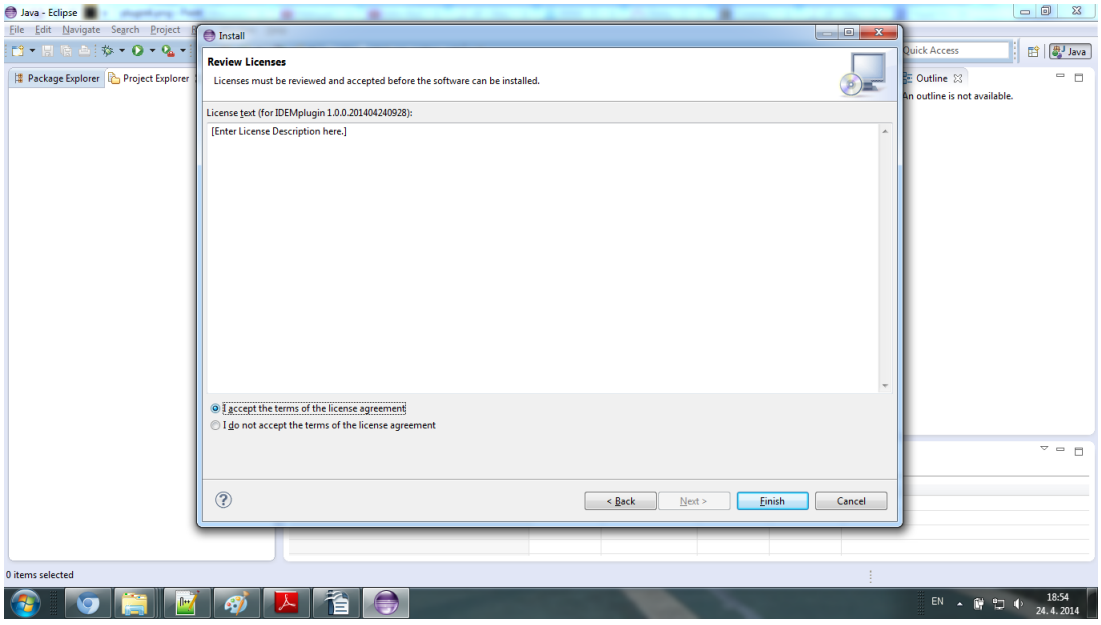
4. Do políčka Work with napíšte: *http://team08-13.ucebne.fiit.stuba.sk/updatesite/IDEMplugin* a kliknite Next.



Obr. 4: *Adresa pluginu.*

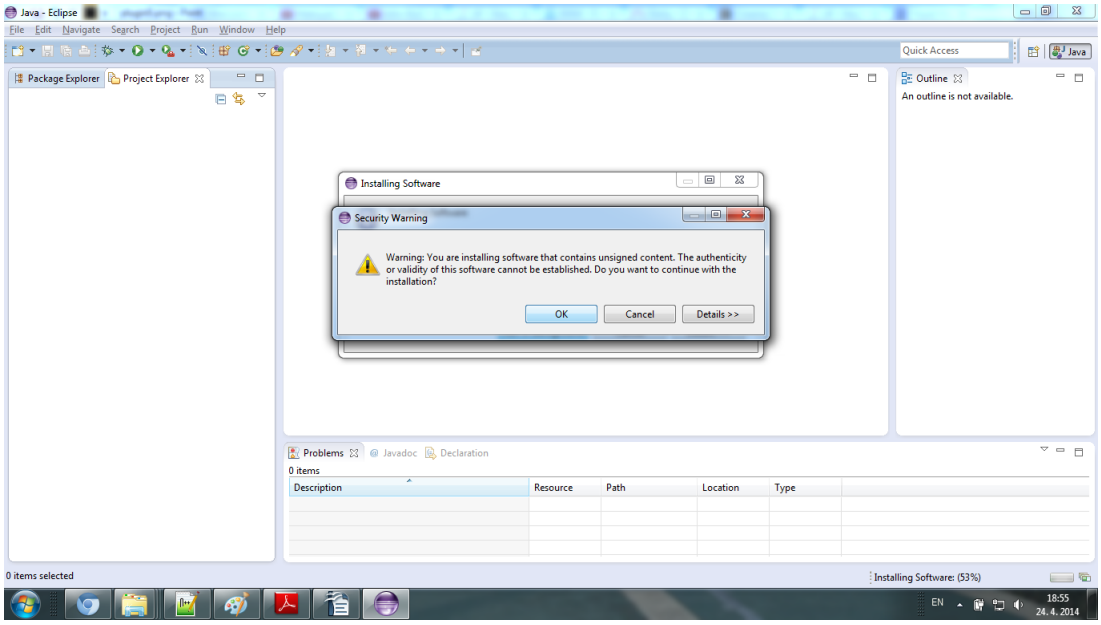
5. Opäť kliknite Next

6. Kliknite I accept the terms of the license agreement a následne Finish



Obr. 5: *Inštalácia.*

7. Podtvrd'te Warning



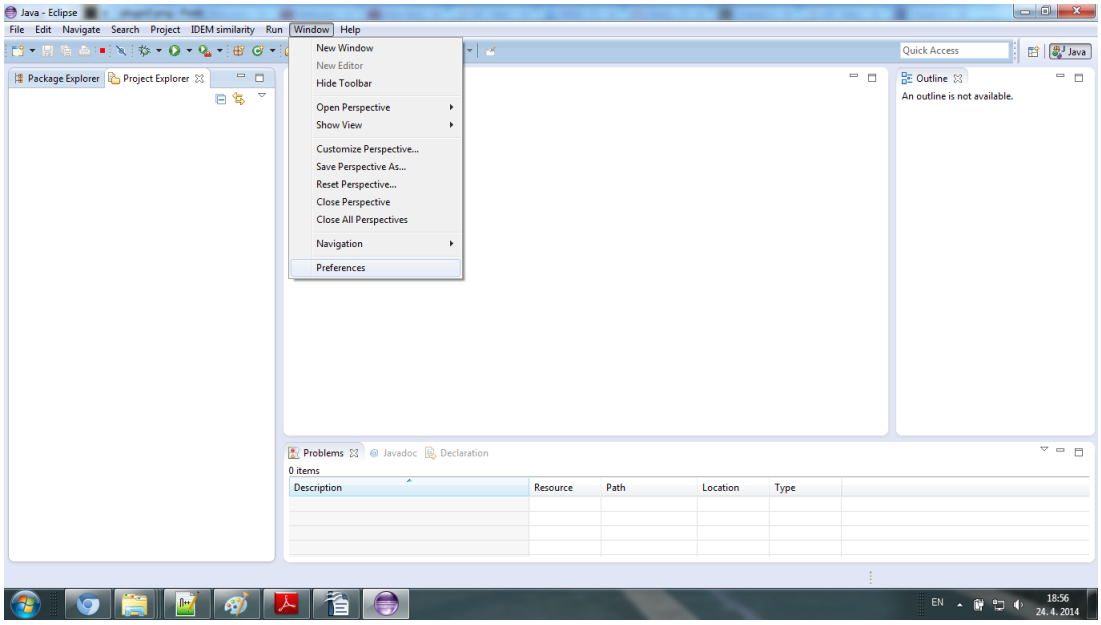
Obr. 6: *Inštalácia - upozornenie.*

8. Reštartujte eclipse

2.2 Používanie

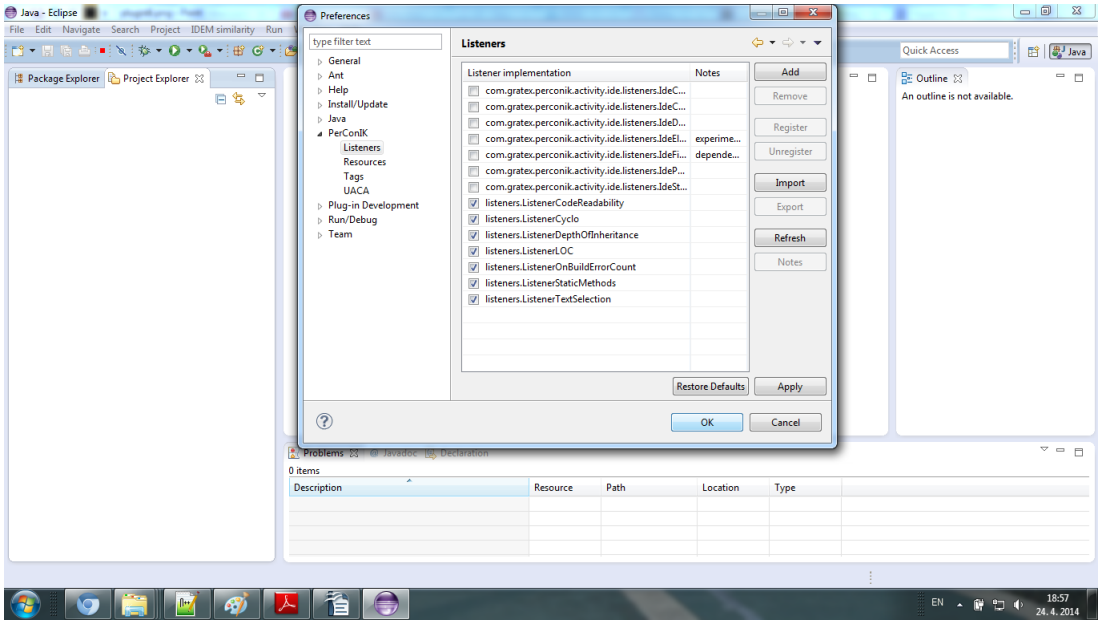
Na korektné používanie pluginu je potrebné vykonať niektoré nastavenia.

1. Zvoľte Window -> Preferences



Obr. 7: *Preferences.*

2. V pravo kliknite Perconik -> listeners a zaškrknite listenery, ako vidíte na obrázku.



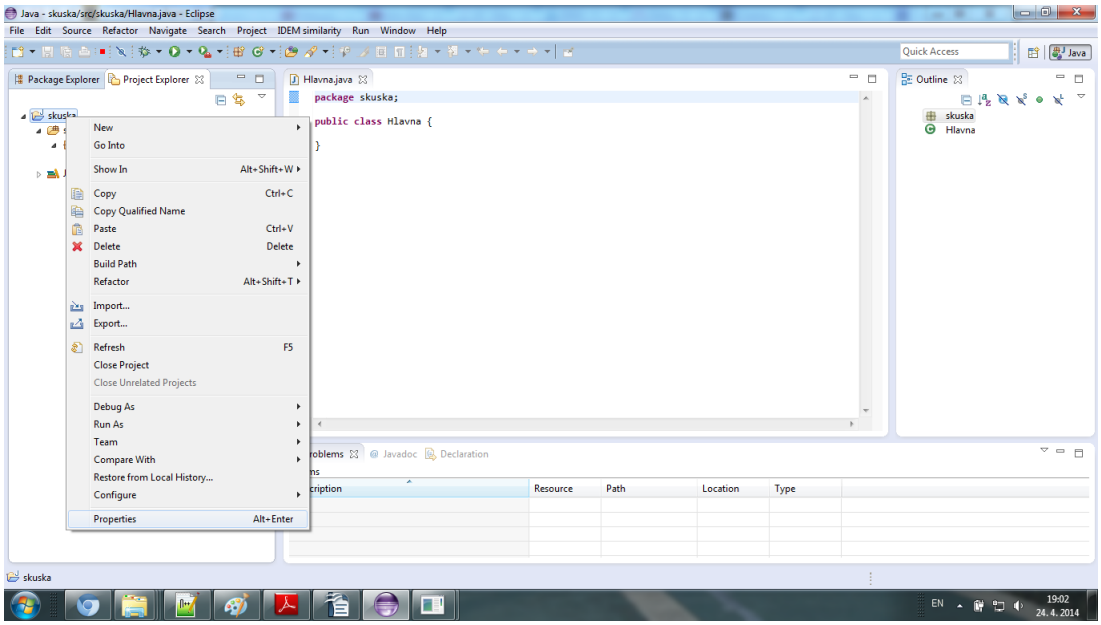
Obr. 8: *Listenery.*

3. Kliknite Apply a následne OK

2.3 Prepojenie s projektami vo webovej aplikácii

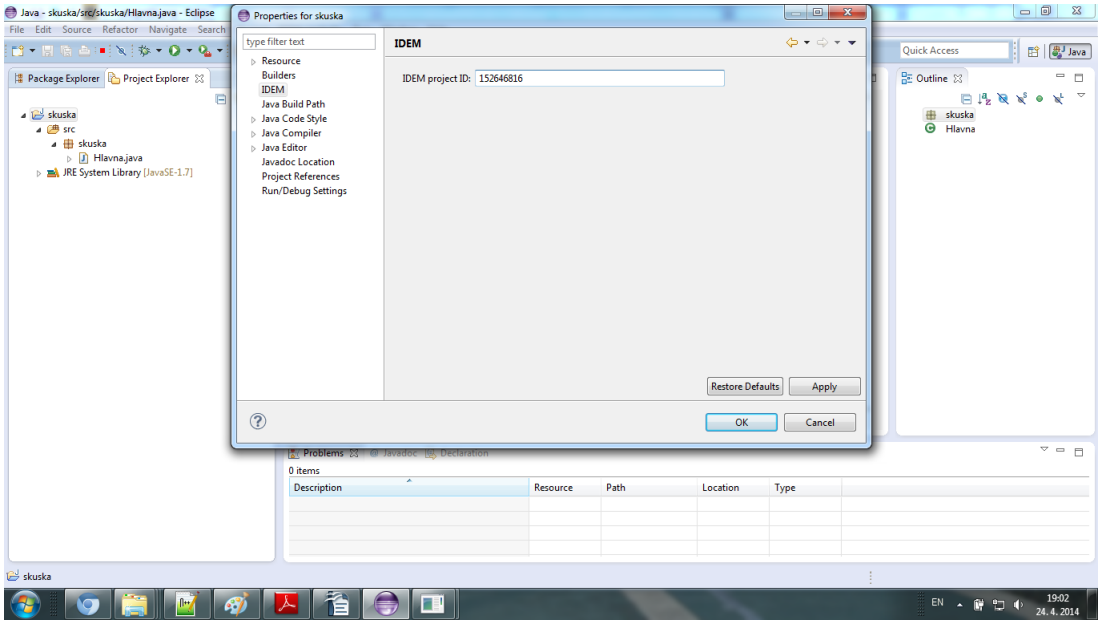
Postup v tejto kapitole je možné vykonať až po zaregistrovaní sa vo webovej aplikácii. Systém potrebuje vedieť prepojiť nalogované dáta z pluginu a webovej aplikácie. Používatelia potrebujú získať prístup k projektom vo webovej aplikácii, k tomu slúži návod, ktorý sa nachádza v ďalšej kapitole. Ak už máte vygenerované ID projektu z našej webovej aplikácie môžete pokračovať v tejto kapitole.

1. Otvorte projekt v Project Exploreri
2. Kliknite pravou myšou na projekt Properties



Obr. 9: *Nastavenie ID.*

3. V pravom menu klinite na IDEM
4. Do poľa IDEM projekt ID zadajte



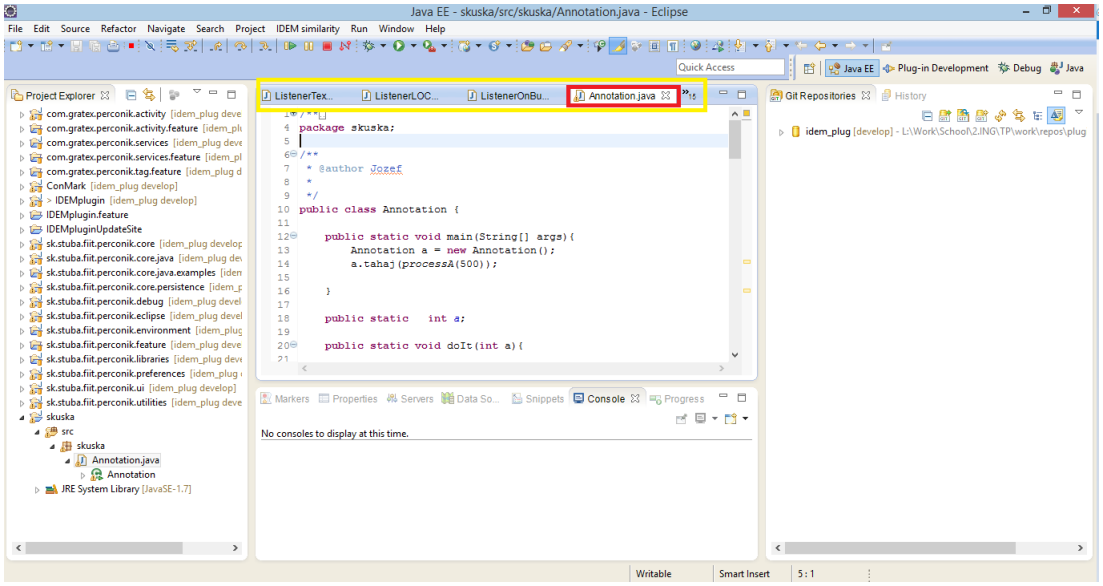
Obr. 10: *ID projektu.*

2.4 Modul podobnosti

Postup v tejto kapitole umožní vypočítať podobnosti medzi jednotlivými časťami kódu.

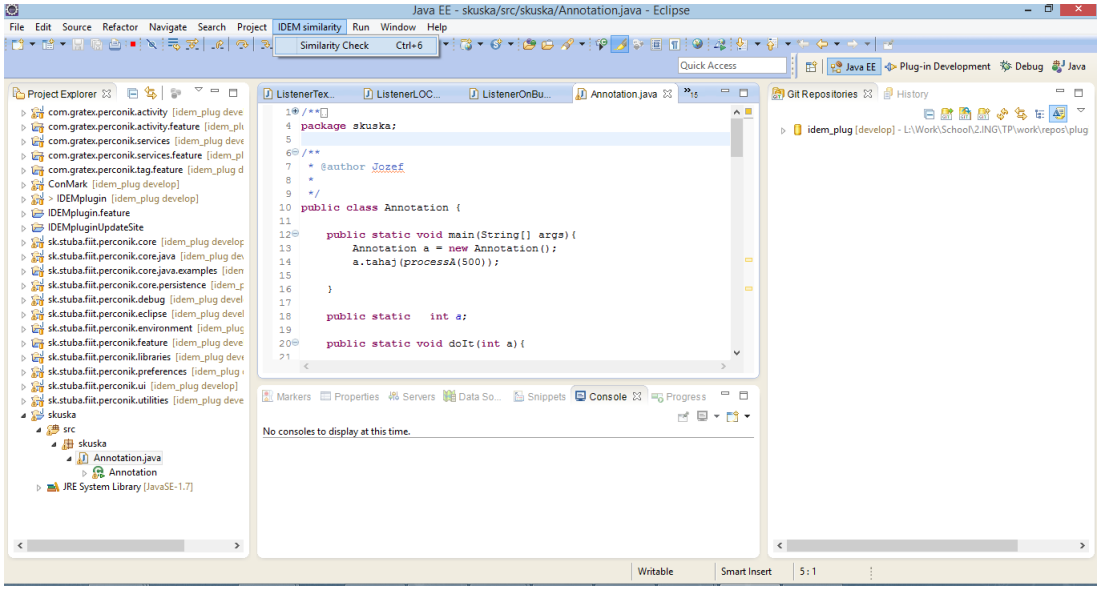
1. V project exploreri vyberieme triedu, nad ktorou chceme zisťovať podobnosť.

2. Je potrebné sa ubezpečiť, že danú triedu máme otvorenú v hlavnom okne.



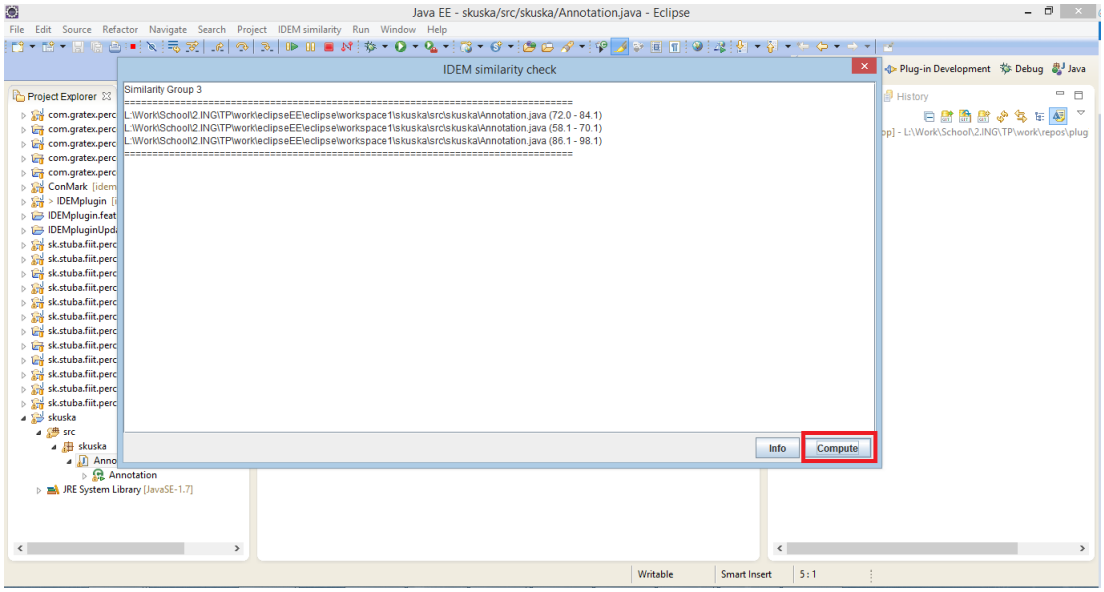
Obr. 11: *Otvorené triedy.*

3. klikneme na menu IDEM similarity a vyberieme Compute Similarity.



Obr. 12: *Spustenie modulu podobnosti.*

4. Vyskočí nám okno, kde v pravom dolnom rohu stlačíme.



Obr. 13: *Na to sa nám objaví výsledok.*

5. Tlačidlo Info vysvetľuje jednotlivé výpisy.

6. Ak nebolo schopné vykonať kontrolu podobnosti (neskompilovteľný kód), vypíše sa chybové hlásenie.

3 Webová aplikácia

K aplikácii sa dá prisupovať na *http://team08-13.ucebne.fiit.stuba.sk:8080/idem/*. Ovláda sa prostredníctvom hlavného menu, ktoré je umiestnené v pravej hornej časti obrazovky.

3.1 Registácia

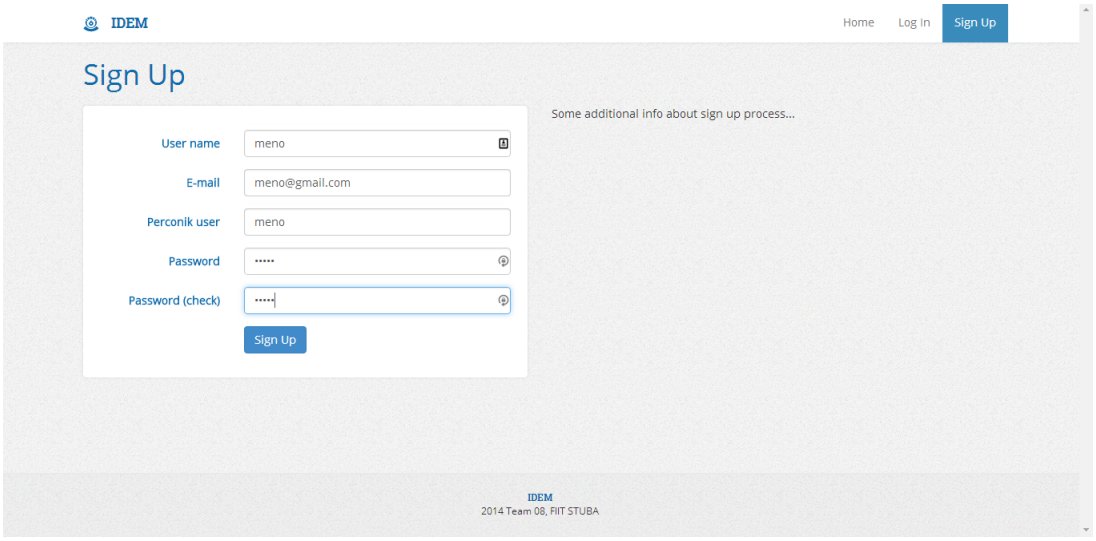
Pre registráciu je nutné vykonať nasledujúce kroky:

1. V hlavnom menu klikneme na položku Sign Up

2. Vyplníme všetky polia, email slúži len na obnovu hesla

3. Tlačidlom Sign Up potvrdíme registráciu

4. Po úspešnej registrácií sa užívateľ môže prihlásiť do aplikácie



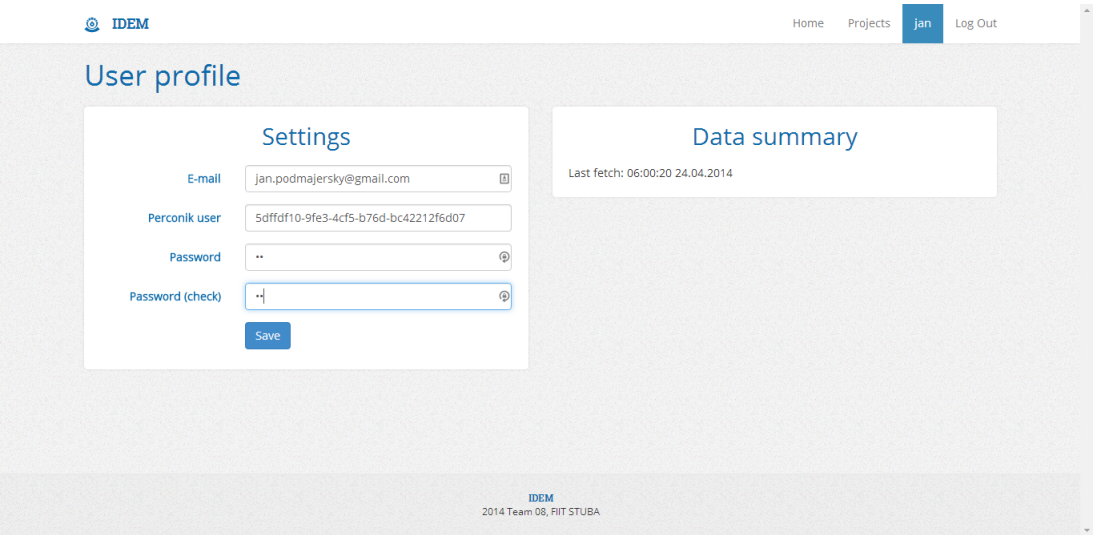
Obr. 14: *Registračný formulár*

3.2 Prihlásenie

1. V hlavnom menu vyberieme položku Log In
2. Do zobrazeného formuláru vyplníme meno a heslo
3. Po zadaní správnych údajov a potvrdení tlačidlom Log In je používateľ prihlásený

3.3 Zmena nastavení

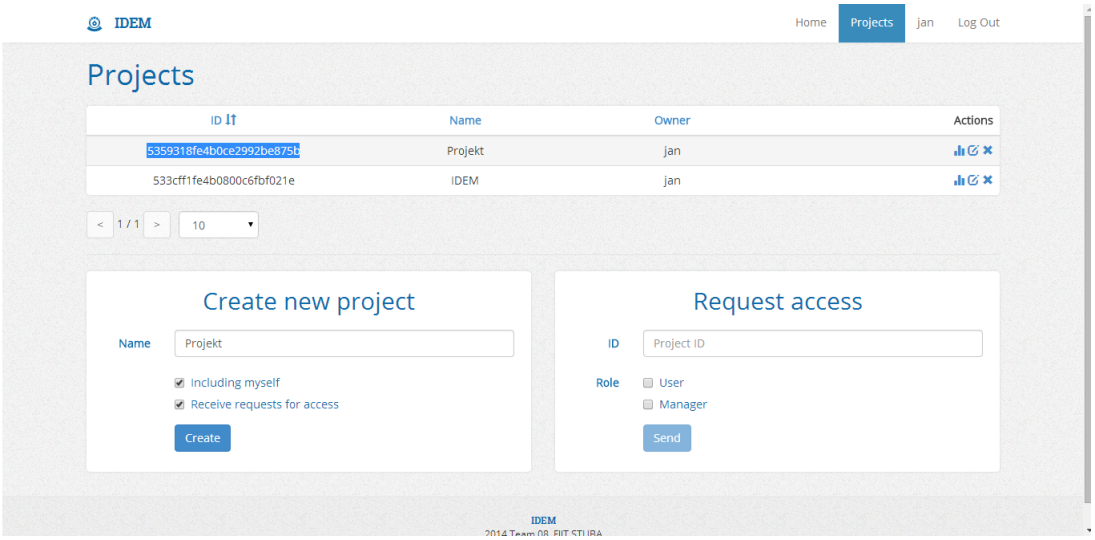
1. Prihlásený používateľ klikne na svoje prihlasovacie meno v hlavnom menu hore
2. Vo formulári môžeme zmeniť všetky údaje
3. Zmena sa potvrdí tlačidlom save



Obr. 15: *Formulár pre zmenu hesla*

3.4 Vytvorenie nového projektu

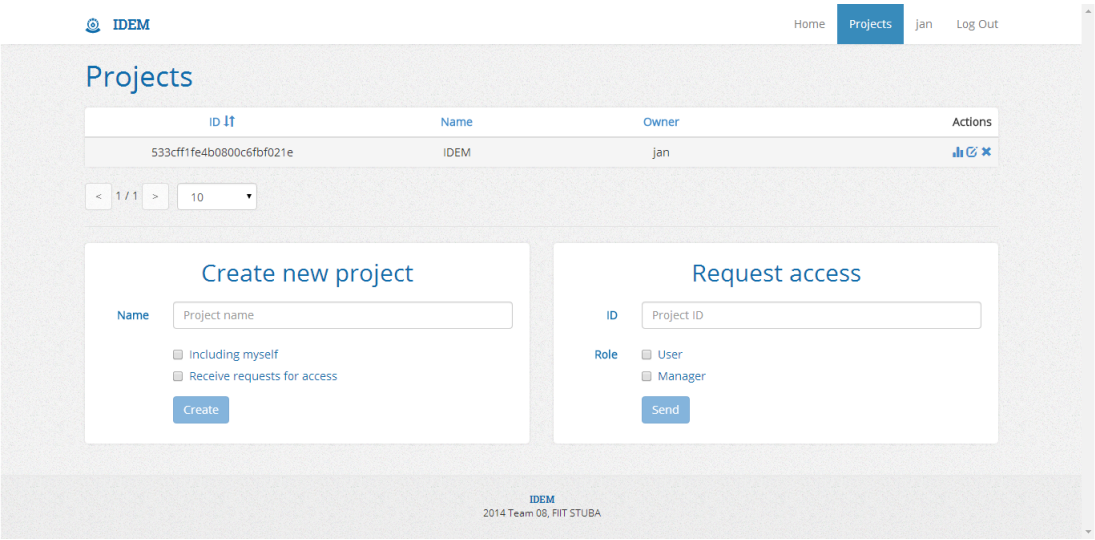
1. Prihlásený používateľ klikne v hlavnom menu na položku Projects
2. Vyplníme názov (Name) vo formulári Create new project v ľavom dolnom rohu
3. Ak chceme byť súčasťou projektu musíme zaškrknúť checkbox Including myself
4. Ak chceme aby sa do projektu mohli nahlasovať iný užívatelia musí zaškrknúť checkbox Including myself
5. Vytvorenie projektu potvrdíme tlačidlom Create a automaticky sa stane owner projektu s plnými právami
6. Projektu sa automaticky vygeneruje id, ktoré slúži na prepojenie eclipse pluginu a webovej aplikácie



Obr. 16: *Formulár pre vytvorenie nového projektu*

3.5 Zrušenie projektu

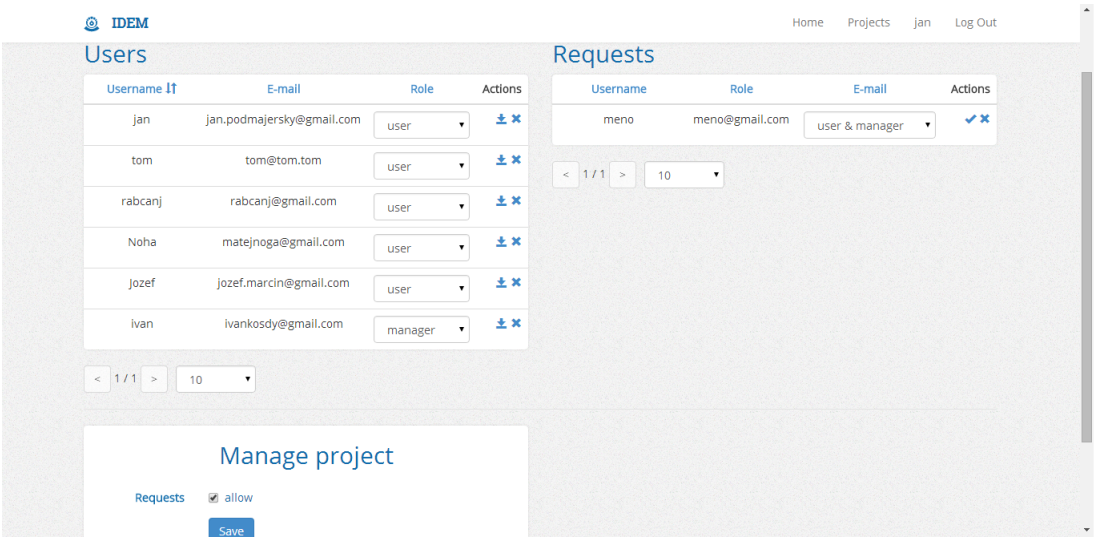
1. Prihlásený používateľ klikne v hlavnom menu na položku Projects
2. Používateľ vyhľadá v zozname zvolený projekt
3. Kliknutím na ikonu ✖ v stĺpci Actions sa projekt zruší



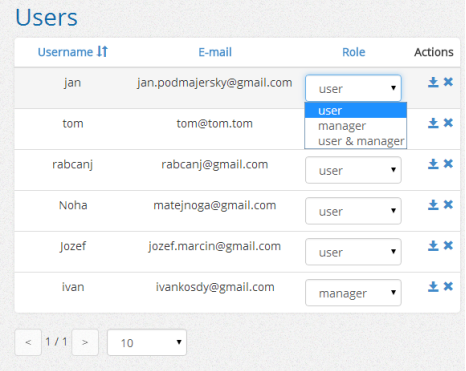
Obr. 17: *Formulár pre zrušenie projektov*

3.6 Správa používateľov na projekte

1. Prihlásený používateľ klikne v hlavnom menu na položku Projects
2. Používateľ vyhľadá zvolený projekt v zozname
3. Klikne na ikonu v stĺpci Actions sa otvorí ponuka projektu
4. Requests
   - Kliknutím na v Actions za menom používateľa sa potvrdí jeho požiadavka o prijatie do projektu
   - Kliknutím na ikonu v Actions za menom používateľa sa zamietne jeho požiadavka o prijatie do projektu
5. Users
   - Kliknutím na ikonu v Actions za menom používateľa sa vyhodí užívateľ s projektu
   - Kliknutím na selectbox Role môžeme zmeniť používateľovi rolu
6. Manage project
   - Checkbox request allow, povoluje žiadať o zaradenie do projektu



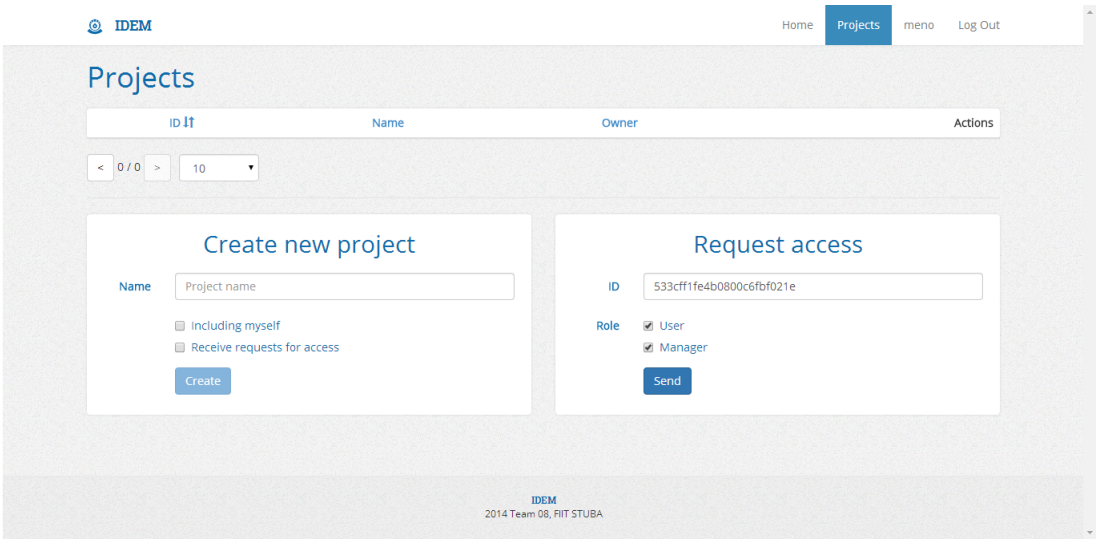
Obr. 18: *Formulár pre správu používateľov*



Obr. 19: *Formulár pre správu používateľov a zmenou rolí*

3.7 Zaradenie do projektu

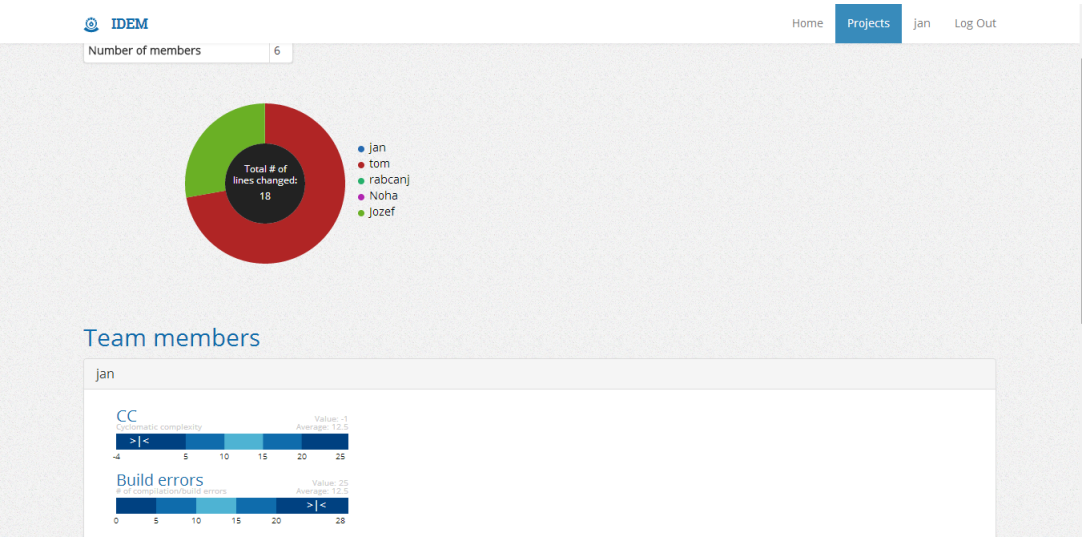
1. V hlavnom menu zvolíme položku *Projects*
2. Do pola ID v Requests napíšeme jedinečné ID projektu do ktorého máme záujem vstúpiť
3. Zvolíme požadovanú rolu, jednoduchý používateľ - user, alebo manager
4. Klikneme na tlačítko *Send*



Obr. 20: *Formulár pre zarsadenie do projektu*

3.8 Zobrazenie štatistík pre používateľa

1. V hlavnom menu zvolíme položku *Projects*
2. Kliknutím na ikonu na požadovanom projekte v Actions sa zobrazia štatistiky
3. Hore koláčový graf podielu práce používateľov na projekte
4. Nižšie, po kliknutí na meno sa zobrazia aj ďalšie štatistiky jednotlivých metrík



Obr. 21: *Zobrazenie štatistík*

14 Prílohy

14.1 Riadenie letný semester

V letnom semestri sme zápisnice zo stretnutí nahradili exportami z RedMine a retrospektívami.

IDEM - Úlohy

#	Stav	Priorita	Predmet	Priradené	Strávený čas	Odhadovaná doba
Sprint 6 (33)						
107	Resolved	Normal	Requestovanie pristupu k projektu	Ivan Košdy	9.0	8.0
104	Resolved	Normal	Css & html structure	Ivan Košdy	2.5	3.0
103	Resolved	Normal	Menu item highlight	Ivan Košdy	1.5	1.0
102	Resolved	Normal	Titulka stranky	Ivan Košdy	0.5	1.0
101	Resolved	Normal	Drobne upravy gui	Ivan Košdy	1.0	1.0
94	Resolved	Normal	Sort admin tabuliek	Ivan Košdy	2.0	2.0
93	Resolved	High	Web security anotacie	Ivan Košdy	1.0	
92	Resolved	Normal	Filtrovanie v admine	Ivan Košdy	1.0	1.0
91	Resolved	Normal	Oprava relativnych ciest na webe	Ivan Košdy	1.5	2.0
89	Resolved	Normal	Tabulky pri nizkom rozliseni	Ivan Košdy	0.5	1.0
88	Resolved	Normal	Autocomplete/suggestion username	Ivan Košdy	1.0	1.0
87	Resolved	Normal	Odobranie uzivatela	Ivan Košdy	1.0	1.0
86	Resolved	Normal	Strankovanie	Ivan Košdy	1.0	1.0
78	Resolved	Normal	Prezentacia - 24.2.2012	Ivan Košdy	3.0	
82	Resolved	Low	Modul monitorovania biometriky - edit povodneho, priprava dat pre zobrazenie	Jan Podmajersky	4.5	7.0
79	Resolved	Normal	onBuild Listener	Jan Podmajersky	18.0	
77	Resolved	Normal	Spojit pluginy + prerobit plugin na riadiace premenne	Jozef Marcin	14.0	7.0
175	Resolved	Normal	priebezna sprava po prvom semestri	Juraj Rabčan	4.0	4.0
173	Resolved	Normal	registrovanie listeneru do perconika	Juraj Rabčan	6.0	6.0
137	Resolved	High	Oprava posielania dat pri zistovani hlbky dedenia	Juraj Rabčan	2.0	
85	Resolved	Normal	Dokumentacia sprintov a stretnuti	Juraj Rabčan	0.5	
75	Resolved	Normal	hlbka dedenia	Juraj Rabčan	24.0	1.0
76	Resolved	Normal	Listener pre GIT (Eclipse/UACA)	Matej Noga	36.0	
109	Resolved	Normal	Vytvoriť grafy pre zobrazovanie metrik	Michal Juranyi	17.0	12.0
108	Resolved	Normal	Vytvorenie Properties strany v plugine	Michal Juranyi	4.5	4.0
80	Resolved	Normal	Preskumat GUI moznosti pluginov	Michal Juranyi	2.5	2.0
100	Resolved	Normal	Priebezna sprava iitsrc	Tomáš Martinkovič	2.0	4.0
99	Resolved	Normal	new LOC	Tomáš Martinkovič	1.5	1.0
98	Resolved	Normal	Static cez AST	Tomáš Martinkovič	3.0	8.0
97	Resolved	Normal	onSave	Tomáš Martinkovič	13.5	8.0
96	Resolved	Normal	LOC	Tomáš Martinkovič	2.0	2.0
57	Resolved	High	Vytvoriť 5 LOC metrík	Tomáš Martinkovič	8.0	5.0
90	New	Normal	Konfirmacia zmazania		0	

	Doing	Stop Doing	Start Doing
Riadenie	2 stretnutia za týždeň komunikácia pri problémoch	Nenazývať premenné podľa konvecie	Dodržiavanie harmonogramu Viac prototypovanie Dlhodobé plány
Produkt	Pamätať na koncových používateľov Dokumentovať Git commit – zlepšiť	Plytvanie časom	Začať s prípravou na odovzdanie Práca na všetkých častiach produktu

IDEM - Úlohy

#	Stav	Priorita	Predmet	Priradené	Strávený čas	Odhadovaná doba
Sprint 7 (23)						
130	Resolved	Normal	Perconik user unique & adding idemuser reference	Ivan Košdy	2.0	2.0
114	Resolved	Normal	Parser na data	Ivan Košdy	23.5	8.0
113	Resolved	Normal	Uprava css a pridanie footera	Ivan Košdy	1.0	1.0
110	Resolved	Normal	Alerty na uzivatelske akcie	Ivan Košdy	2.5	5.0
118	Resolved	Normal	Rozbehanie posielania	Jan Podmajersky	3.5	
117	Resolved	Normal	Registracia domeny	Jan Podmajersky	0	
116	Resolved	Normal	Pouzivatelska prirucka	Jan Podmajersky	6.0	3.0
128	Resolved	Normal	odstranovanie chyb v plugine	Jozef Marcin	3.0	3.0
127	Resolved	Normal	oprava nemoznosti mazania tried v eclipse	Jozef Marcin	1.0	1.0
122	Resolved	Normal	spojenie sucasti pluginu a update site	Jozef Marcin	7.0	6.0
121	Resolved	Normal	uprava listenerov pre pracu s DIFF	Jozef Marcin	8.0	8.0
111	Resolved	Normal	new Cyclo	Jozef Marcin	13.0	13.0
138	Resolved	Urgent	Oprava logovania dedenia	Juraj Rabčan	7.0	
135	Resolved	Normal	Upravit prava pre zobrazovanie statistik projektu	Michal Juranyi	2.0	2.0
133	Resolved	Normal	Vytvorit kolacovy graf	Michal Juranyi	6.0	3.0
115	Resolved	Normal	Finalizacia abstraktu na IIT.SRC	Michal Juranyi	1.0	1.0
83	Resolved	Normal	Zobrazenie udajov - vizualizacia	Michal Juranyi	8.0	
129	Resolved	Normal	oprava nemoznosti mazania tried v eclipse	Tomáš Martinkovič	1.0	1.0
126	Resolved	Normal	nevalidny json v plugine	Tomáš Martinkovič	2.0	1.0
125	Resolved	Normal	send builderrors	Tomáš Martinkovič	0.5	1.0
124	Resolved	Normal	diff LOC	Tomáš Martinkovič	8.0	8.0
123	Resolved	Normal	on save na viacej suborov	Tomáš Martinkovič	7.5	8.0
119	Resolved	High	Posielanie dat	Tomáš Martinkovič	5.0	4.0

	Doing	Stop Doing	Start Doing
Riadenie	zjemňovanie úloh vytvorenie skupiny na Skype	Nedôvera voči splneniu taskov navzájom Hádať sa	Uzatvárať tasky v RedMine Ak máme chybu tak sa pýtať
Produkt	Častejší komit kvoli rollbackom	Komitovať nefunkčný kód	Častejší release produktu

IDEM - Úlohy

#	Stav	Priorita	Predmet	Priradené	Strávený čas	Odhadovaná doba
Sprint 8 (18)						
147	Resolved	Normal	Reset hesla z adminu	Ivan Košdy	1.0	1.0
136	Resolved	Normal	Plagat	Ivan Košdy	10.0	6.0
134	Resolved	Normal	onBuild errors na projekt	Jan Podmajersky	13.0	4.0
120	Resolved	Normal	listener na kvalitu zdrojoveho kodu	Jan Podmajersky	17.0	
146	Resolved	Normal	opraveny listener hlbky dedenia	Jozef Marcin	3.0	3.0
143	Resolved	Normal	Verziovanie pluginu - odosielanie verzii pluginu spolu s metrikami	Jozef Marcin	1.0	1.0
81	Resolved	High	Modul porovnania podobnosti	Jozef Marcin	25.0	15.0
145	Resolved	Normal	Analyza zozbierania a zbieranie dat z mysi (biometrickych)	Juraj Rabčan	18.0	
144	Resolved	High	Retrospektivy + exporty z redmine	Juraj Rabčan	1.0	2.0
139	Resolved	Normal	Neuroph - skusanie a zistovanie ako sa snim pracuje	Juraj Rabčan	3.5	2.0
148	Resolved	Normal	Analyza zozbieravanych dat z metrik	Matej Noga	22.0	
170	Resolved	Normal	Nakreslenie obrazku architektury systemu	Michal Juranyi	2.0	2.0
167	Resolved	Normal	Bezpecnostna kontrola a zabezpecenie timoveho servera	Michal Juranyi	2.0	2.0
140	Resolved	Normal	Zobrazenie metrík pomocou grafov	Michal Juranyi	9.0	8.0
151	Resolved	Normal	duplicitne posielanie dat	Tomáš Martinkovič	1.0	1.0
150	Resolved	Normal	getProjektId	Tomáš Martinkovič	2.0	2.0
149	Resolved	Normal	Build errors, warning	Tomáš Martinkovič	9.0	4.0
141	Resolved	Normal	Posielanie verzie pluginu	Tomáš Martinkovič	2.0	2.0

	Doing	Stop Doing	Start Doing
Riadenie	Dodržiavanie metodík pri tvorbe kódu	Maily do celej skupiny, ak to nie je nevyhnutné	Používať fórum
Produkt	Kontinuálna práca Refaktoring	Prílišné analyzovanie problémov – menej výstupu	Ak sa zmení výstup úlohy, na ktorú nadväzuje ďalšia úloha je potrebné oboznámiť S tým všetkých členov tímu

IDEM - Úlohy

#	Stav	Priorita	Predmet	Priradené	Strávený čas	Odhadovaná doba
Sprint 9 (11)						
155	Resolved	Normal	tp zaverecna sprava	Jan Podmajersky	4.0	3.0
162	Resolved	Normal	GUI pre modul podobnosti	Jozef Marcin	5.0	5.0
142	Resolved	Normal	dotaznik	Jozef Marcin	1.0	1.0
172	Resolved	Normal	oprava funkcionality v v datach z myse	Juraj Rabčan	5.0	5.0
171	Resolved	Normal	Oprava funkcionality Biometrickych dat	Matej Noga	3.0	3.0
169	Resolved	Normal	Pridanie interaktivnosti ciarovemu grafu	Michal Juranyi	2.0	2.0
168	Resolved	Normal	Pridanie verzie logov pri ukladani do DB	Michal Juranyi	1.0	1.0
164	Resolved	Normal	Uprava zobrazenia chyb - kvoli zmene pluginu	Michal Juranyi	3.0	3.0
163	Resolved	Normal	Vytvorenie obsahu uvodnej stranky webappky	Michal Juranyi	1.0	1.0
154	Resolved	Normal	testovanie instalacie pluginu	Tomáš Martinkovič	2.0	2.0
153	New	Normal	Spravy / Stranky		2.0	

	Doing	Stop Doing	Start Doing
Riadenie	Dochvíľnosť Zlepšiť ohodnocovania	Písanie dokumentácie na poslednú chvíľu Zlý časový manažment	Písanie dátumov do RedMine Správna aktualizácia úloh V RedMine
Produkt	Častejší komit kvoli rollbackom Kontinuálna práca Refaktoring	Nerobiť komint and push, ale rozvrhnúť jeden push na viacero komitov	lepšie tesovať

IDEM - Úlohy

#	Stav	Priorita	Predmet	Priradené	Strávený čas	Odhadovaná doba
Sprint 10 (8)						
160	New	Normal	Zobrazenie biometrickych dat	Jan Podmajersky	12.0	
161	Resolved	Immediate	Finalna verzia pluginu za letny semester	Jozef Marcin	1.0	1.0
159	Resolved	High	Citatelnost kodu - vytvorenie vzorca na urcenie citatelnosti z metrik	Jozef Marcin	3.0	3.0
158	In Progress	Urgent	Záverečná dokumentácia	Juraj Rabčan	8.0	20.0
166	Resolved	Normal	Uprava rozlozenia stranky so statistikami timu	Michal Juranyi	2.0	2.0
165	Resolved	Normal	Vytvorenie CD pre zaverecne odovzdanie	Michal Juranyi	2.0	2.0
174	In Progress	High	zaverecna dokumentacia	Tomáš Martinkovič	5.0	8.0
156	Resolved	Normal	TPcup zaverecna sprava	Tomáš Martinkovič	2.5	2.0

	Doing	Stop Doing	Start Doing
Riadenie	2 stretnutia za týždeň komunikácia pri problémoch dlhodobé plány krátkodobé plány	Prestať sa nesústrediť na stretnutiach Označovať nezdokumentované a neotestované tasky ako hotové Písať lepšie návody Prestať porušovať konvencie a začať dodržiavať metodiky	Dochvíľnosť Dodržiavanie harmonogramu Viac prototypovanie Zlepšiť realizáciu Scrumu Zlepšiť testovanie
Produkt	Práca na všetkých častiach produktu	Nedotiahnutá implementácia	Vizualizácia dát Začať s prípravou na odovzdanie Dokumentácia