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

Zoznam obrázkov

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 4asto 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.

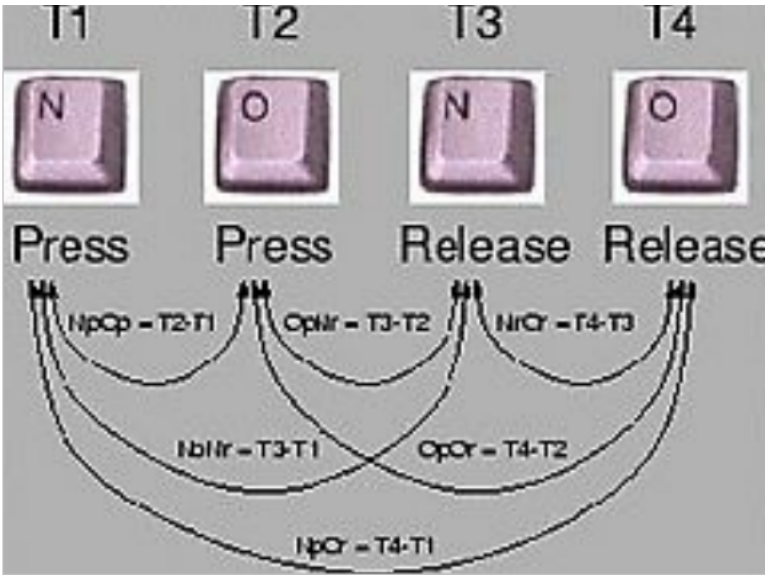
2.1.2 Metódy rozpoznávania používateľa

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 zásobnika *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 odošľú 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.

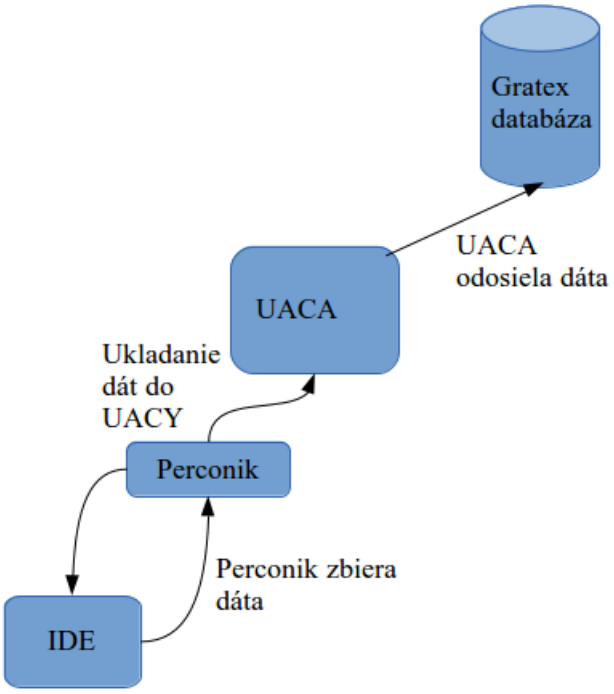
Perconik dokáže logovať používateľov v nasledujúcih IDE:

- Eclipse

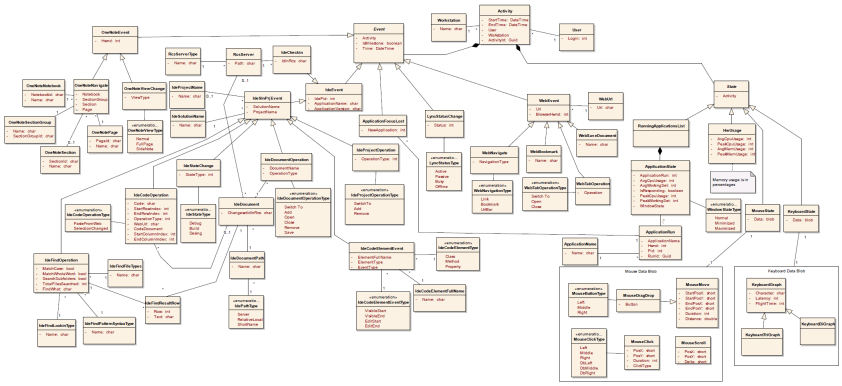
- MS Visual Studio od verzie 2010

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

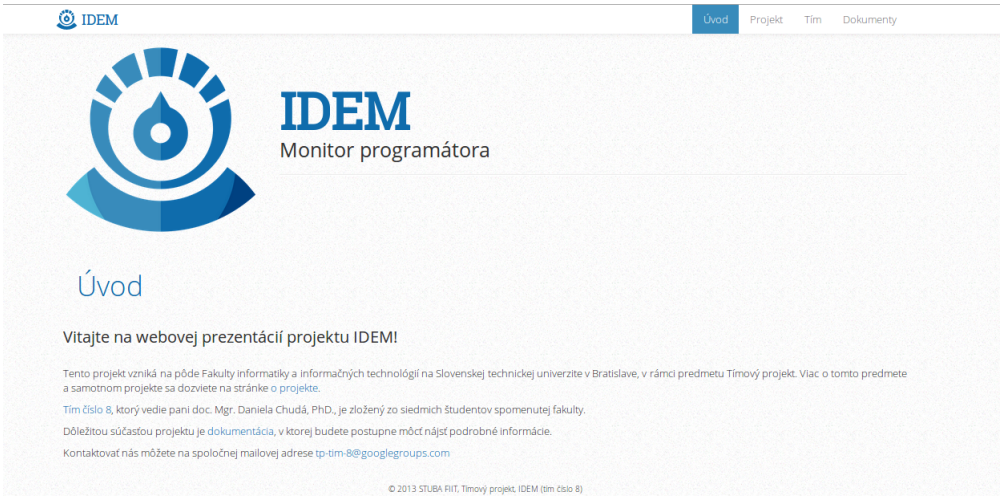
Monitorovanie práce a manažment úloh je pri tímovej práci veľmi dôležitý a preto sme museli začať používať softvér, ktorý by náam to uľahčil. Z veľkého množstva sovtvéru, ktorý bol pre tento účel vytvorený, sme sa rozhodli použiť Redmine. Redmine nám beží na školskom serveri a môžeme ho navštíviť na 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/team08issi/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 naištaloval 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 naištalovať 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 prihlásenïm 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 najnovešj verzie. Po jeho nainštalovaní aplikácia nasadená na tímový server. Tým sa aplikácia stala prístupná zvonka a môže byť využitá kým koľvek 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

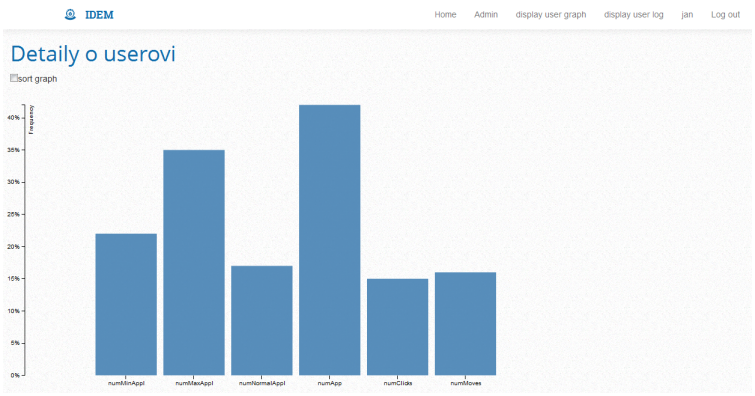
Ďalší dôležitý krok bol vytvorenie triedy agregovaného vektora, táto trieda sa skladá zo všetkých vektorov jednotlivého používateľa. Agregovaný vektor je štruktúra obsahujúca

- 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 madelu používateľa. Táto trieda zobrazí všetky hodnoty vektora na grafe. Graf zobrazuje hodnoty dynamicky, pri zobrazení sa hodnoty najprv zobrazí a v štandartnom 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 predchádzajucom 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

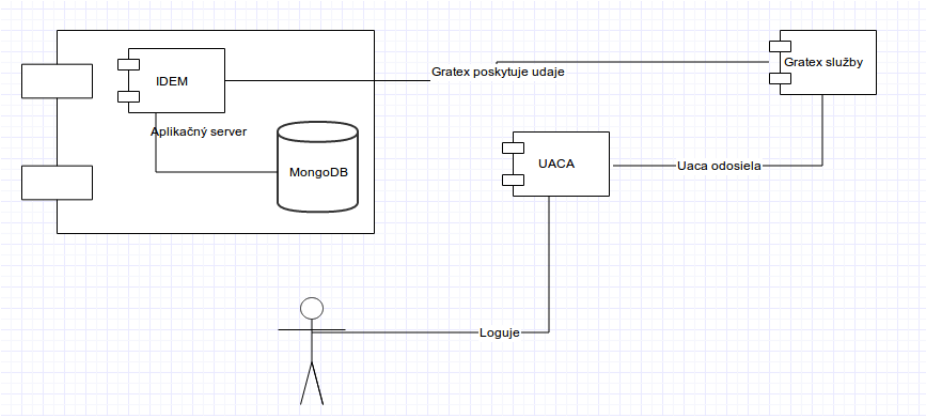
Aplikácia je vyvíjaná v Jave 1.7 a frameworku Spring MVC. Vzhľadom na technológie a webový charakter aplikácie sme sa ju rozhodli nasadiť na aplikačný server GlassFish. Tento server už bol nainštalovaný aj na náš tímový server. Aplikácia pracuje s NoSql databázou MongoDB.

- 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 Súčastná architektúra systému

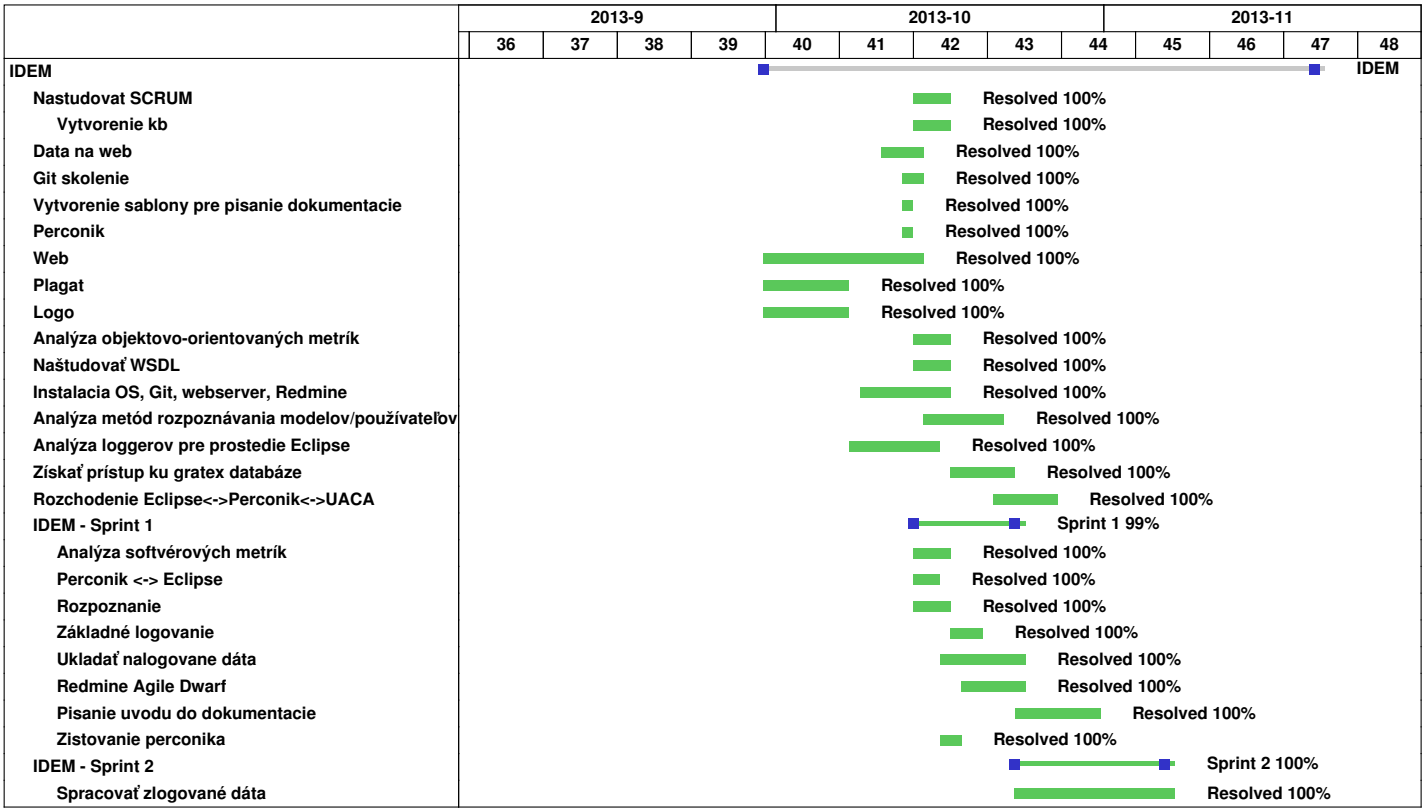
Súčastná architektúre systému je vyjadrená nasledujúcim obrázkom:

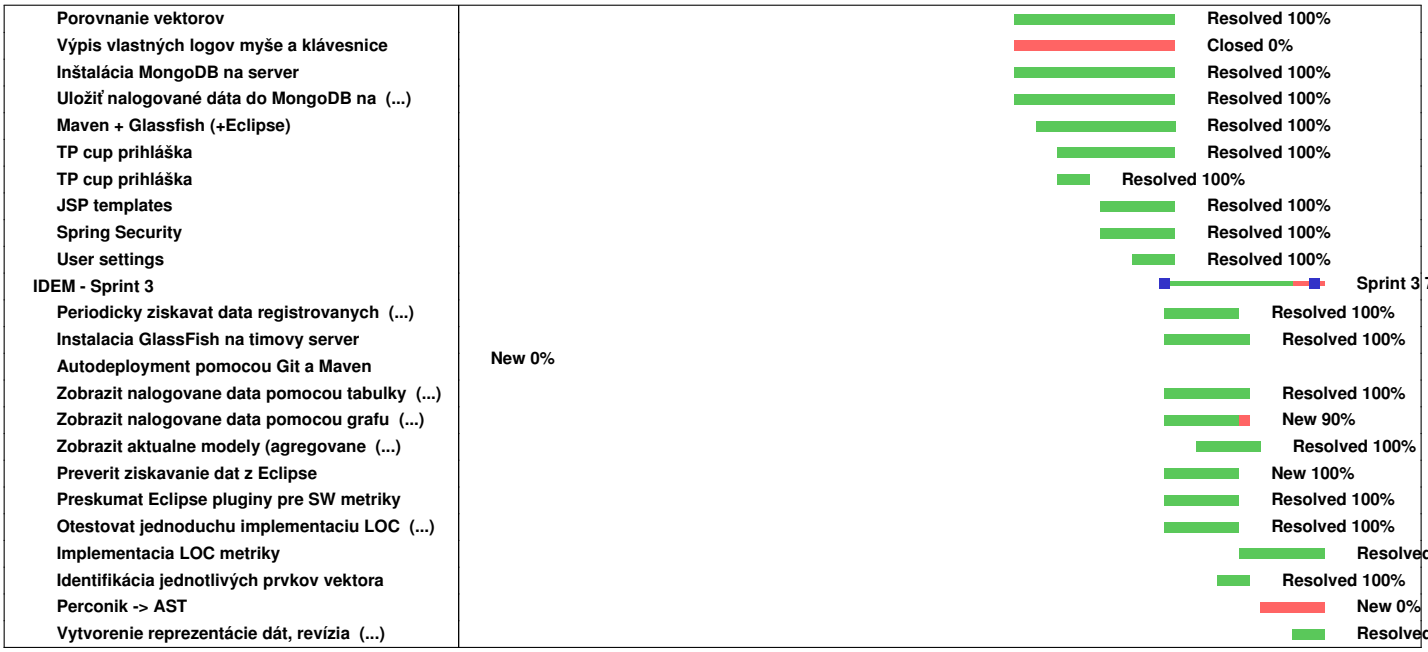


Obr. 6: *Ukážka architektúry systému*

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

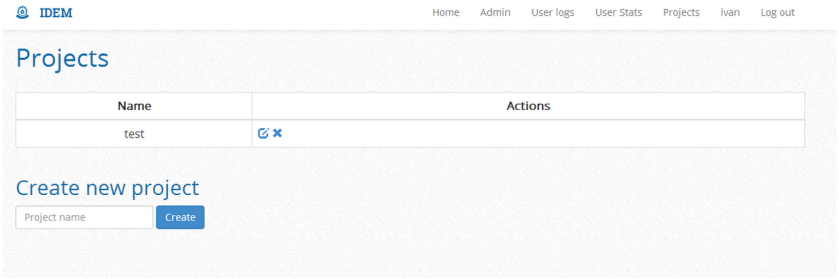
Systém monitorovania programátora pozostáva v podstate z dvoch dôležitých častí: klientskej aplikácie zaznamenávajúcej aktivitu programátora a webovej aplikácie zobrazujúcej získané a zanalyzované údaje. Pre rozšírenie možností klienskej aplikácie potrebujeme vytvoriť vlastné rozširenie pre prostredie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

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 anlýzu 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, priemerovanie 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 riadiacích 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 kľučové 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.

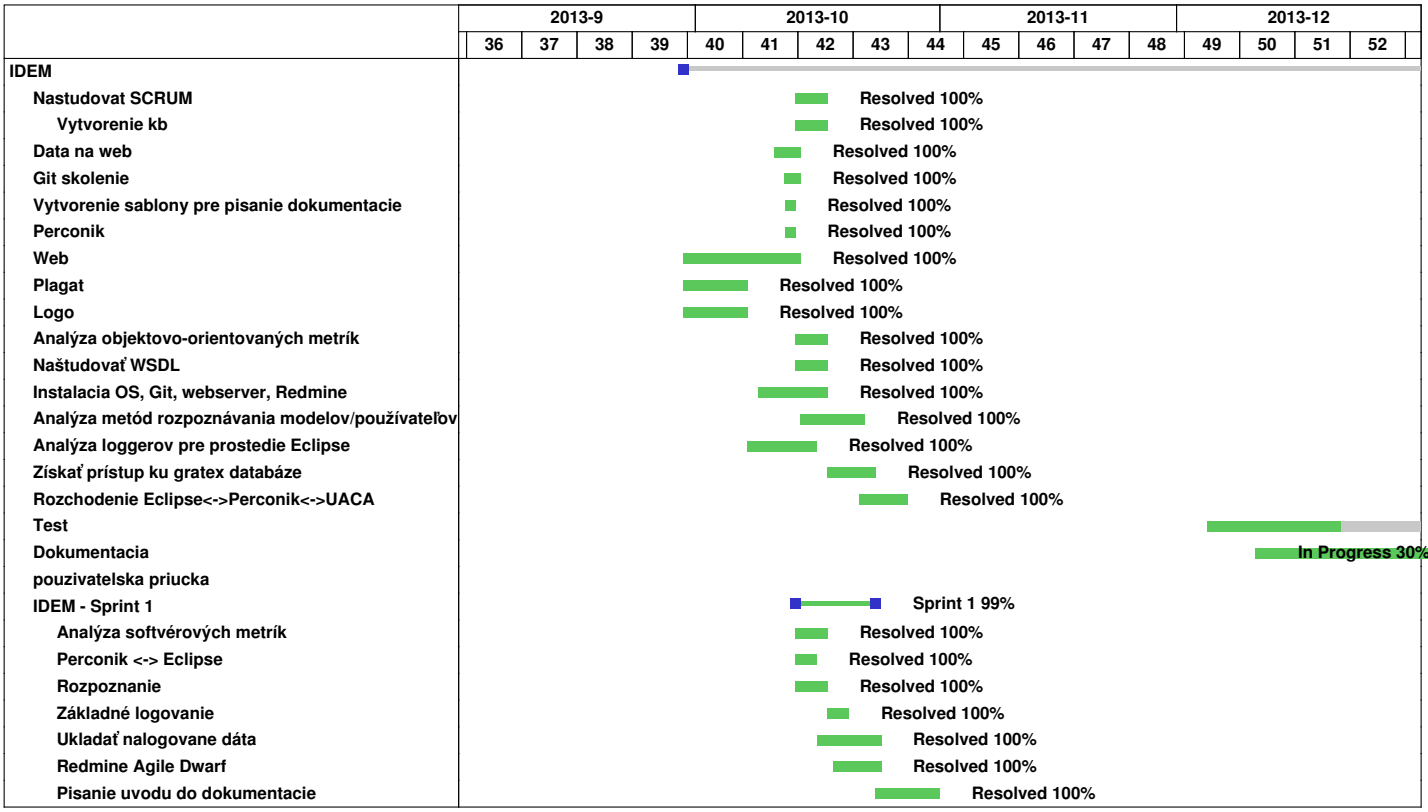
V ďalšej práci bude potrebné editovať tento plugin tak, aby dokázal vyrátať hodnoty McCabe Cyclomatic Complexity v rámci kódu, ktorý vytvoril jeden používateľ samostatne a tak určiť mieru kvality programovania daného používateľa.

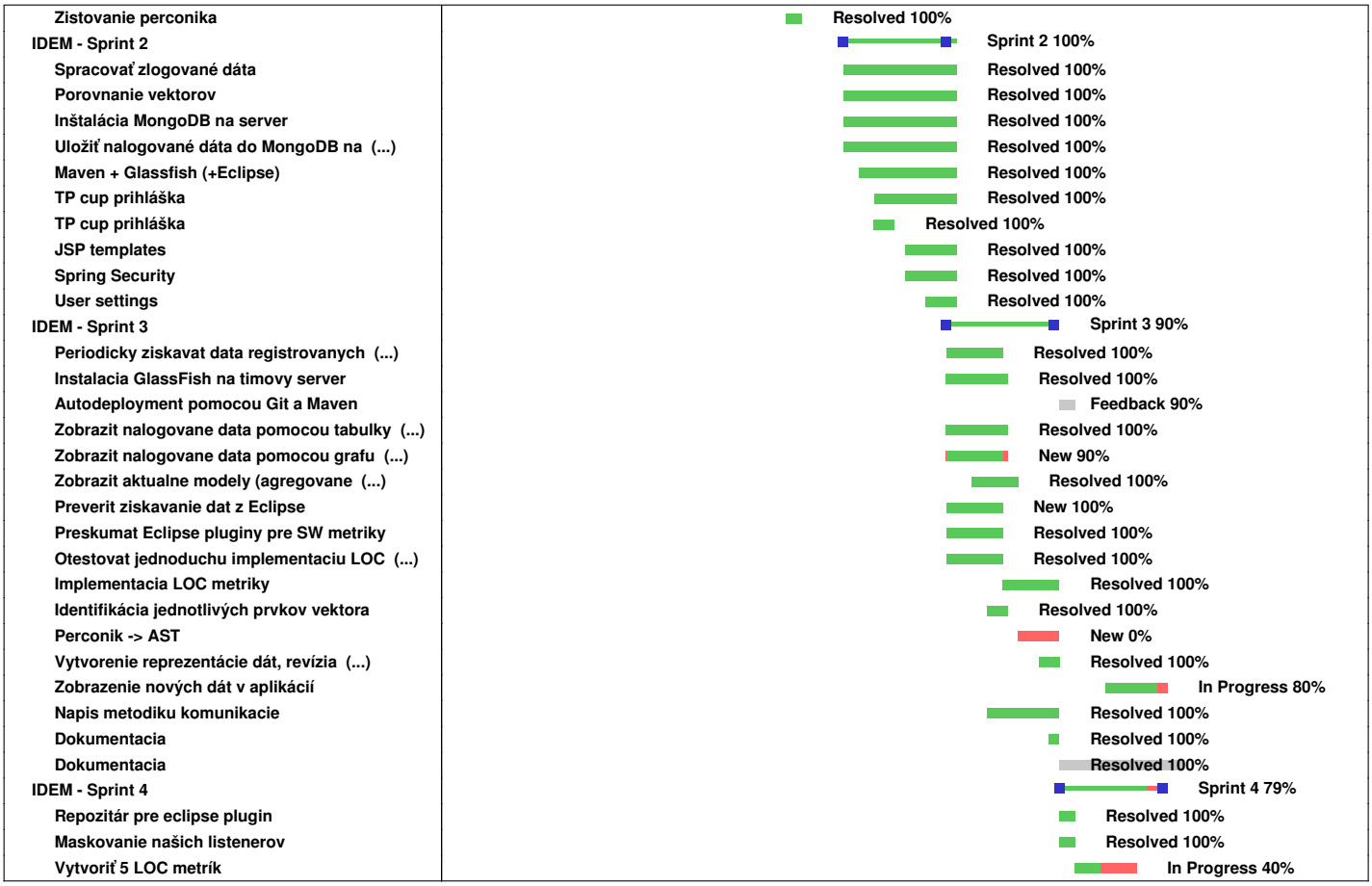
5.7 Zhodnotenie šprintu 4

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