Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Bc. Lukáš Sekerák

Interaktívna vizualizácia akcií vývojára softvérového produktu

Diplomová práca

Vedúca práce: Mgr. Alena Kovárová, PhD.

Máj, 2015

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Bc. Lukáš Sekerák

Interaktívna vizualizácia akcií vývojára softvérového produktu

Diplomová práca

Študijný program: Informačné systémy
Študijný odbor: 9.2.6 Informačné systémy
Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU Bratislava
Vedúca práce: Mgr. Alena Kovárová, PhD.
Máj, 2015

ANOTÁCIA

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Informačné systémy

Autor: Bc. Lukáš Sekerák
Diplomová práca: Interaktívna vizualizácia akcií vývojára softvérového produktu
Vedúci diplomového projektu: Mgr. Alena Kovárová, PhD.
Máj, 2015

Pri vývoji softvérového produktu vzniká mnoho nezodpovedaných otázok, na ktoré sa nie vždy ľahko hľadá odpoveď. Mnohé vizualizácie neponúkajú možnosť zobraziť vývoj softvérového produktu z pohľadu správania vývojára. Správanie vývojára identifikujeme zo zachytených akcií vývojára. Cez vizualizáciu týchto akcií alebo aktivít, používateľ môže identifikovať rôzne charakteristiky práce vývojára. Včítane tých, ktoré majú neblahý vplyv na vývoj.

Medzi akcie, ktoré sledujeme patria napríklad používanie IDE, prehliadača alebo rôznych komunikačných nástrojov na riešenie problémov. Tieto akcie spájame napríklad s ukazovateľom ako je napríklad prepisovanie kódu. Spomenuté akcie sú súčasťou vývoja softvérového produktu a ich vizualizácia tak môže odpovedať napríklad na otázky ako: "Ak vývojár často používa prehliadač (resp. vybraný portál), pri písaní svojho kódu, ako často je potom tento kód prepisovaný? Platí to u každého iného vývojára rovnako?" Riešenie je nasadené a overené. Obsahuje RESTful službu na spracovanie dát a interaktívny webový nástroj, ktorý umožňuje exploratívnu analýzu akcií.

ANNOTATION

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Study program: Information systems

Author: Bc. Lukáš Sekerák
Diploma thesis: Interactive visualization of developer's actions
Supervisor: Dr. Alena Kovárová
May, 2015

Software product development provokes numerous unanswered questions which may not be easily resolved. A lot of visualizations do not offer the option of observing development process from the perspective of developer's behaviour. His behaviour is tracked into sequence of actions, actions are composed into activities. These are visualized so a user can identify characteristics of developers' work. Including those, which have negative impact on development.

Various actions have been taken during developers work – such as actions from IDE, browser, communication channels. We link these actions with indicators, e.g. re writing a code. Our visualizations may answer questions, such as: "If a developer uses a browser (particular portal) frequently, during code - writing, how often is the code rewritten? If so, does this apply to every other developer in the same manner?" Solution was deployed and evaluated. It contains RESTful service to process data and interactive web tool, which allows exploratory analysis of actions.

Obsah

1. Úvod do témy

Softvérový vývoj je jedna z najdôležitejších tém v modernom softvérom inžinierstve.[20] To môže byť vysvetlené faktom, že najväčšie náklady v softvérovom inžinierstve súvisia s vývojom a udržateľnosťou. Potreba vizualizácie je preto opodstatnená. Môže prispieť k zníženiu nákladov, respektíve priniesť nové pohľady na vývoj.

Zároveň proces udržiavania softvéru ma negatívny vplyv na zvyšovanie veľkosti a komplexnosti systému. Časom sa tak mení pôvodný dizajn a kvalita softvéru. Pochopenie vývoja je teda vítané najmä u softvérových spoločností.

Pri vývoji softvérového produktu vzniká mnoho nezodpovedaných otázok, na ktoré sa nie vždy ľahko hľadá odpoveď. Mnohé vizualizačné techniky neponúkajú možnosť zobraziť vývoj softvérového produktu z pohľadu správania vývojára. Správanie používateľa by vychádzalo zo zachytených akcií vývojára. Cez vizualizáciu týchto akcií by sme mohli identifikovať rôzne vzory správania vývojára pri jeho práci. Mohli by sme identifikovať zaujímavé akcie či anomálie, ktoré majú neblahý vplyv na vývoj.

Hlavným cieľom tejto práce je teda vytvorenie alebo použitie existujúcej vizualizácie, ktoré sa bude zameriavať na správanie používateľa. Doterajšie vizualizačné techniky sa zameriavajú na iné oblasti, čo dokážeme v analýze.

Medzi akcie, ktoré by nás mohli zaujímať by patrili napríklad používanie prehliadača alebo rôznych komunikačných nástrojov na riešenie problémov. Ďalej vývojové prostredie a pod. Vývojár pri svojej práci používa mnoho nástrojov, preto sa zameriame len na určité akcie. Akcie si vyberieme na základe otázok, na ktoré budeme hľadať odpovede.

Pre splnenie cieľa je potrebné nájsť model bohatý na množstvo údajov (teda odkrokovaných akcií). Ďalej identifikovať typy vizualizácií a nájsť vizualizáciu, ktorá by poskytla najlepšiu možnosť zobrazenia týchto akcií vývojára, respektíve výsledky určitých analýz.

Vďaka takejto vizualizácii, by sme dokázali odpovedať na viacero zaujímavých otázok pri vývoji softvérového produktu. Napríklad: "Ak vývojár často používa prehliadač (resp. portál) pri písaní svojho kódu, ako často je potom tento kód prepisovaný? Platí to u každého iného vývojára rovnako?"

Odpovede na tieto otázky, nám napríklad identifikujú problémy pri vývoji softvérového produktu. Pričom pri vývoji ďalších produktov nám pomôžu predchádzať týmto problémom, pripadne sa z efektívni vývoj. Zároveň tieto odpovede poskytujú objektívnu spätnú väzbu na prácu vývojára.

V úvode diplomovej práce sme krátko spomenuli motiváciu na riešenie tejto témy. Na softvérový vývoj sa dá pozerať z viacerých uhlov. Rovnako existujú rôzne metódy na analýzu vývoja. Na tieto metódy a pohľady sa pozrieme v časti analýza. Časť analýza bude opísaná zo širšej perspektívy. Budeme sa snažiť každú metódu opísať krátko a zreteľne. Cieľom je identifikovať či existujúce metódy by bolo možné použiť a jednoducho ich len vylepšiť.

Problematika vizualizácie vývoja je pomerne široká. V rámci nej sú potrebné vedomosti o spôsobe riadenia a sledovania vývoja, rozdelení úloh, správe a ukladaní verzií softvéru. V poslednom rade vizualizácia informácii je sama o sebe široká téma.

Rozhodli sme sa rozdeliť analýzu na nástroje, ktoré už vizualizujú vývoj. Cieľom bude identifikovať či je možné spojiť staré metódy s novou. A v druhom rade na projekty, ktoré zachytávajú správanie vývojára. Teda zachytávajú jednotlivé akcie vývojára.

V závere práce sa nachádza slovník pojmov, zoznam skratiek, zoznam použitých obrázkov a prílohy. V prvej kapitole príloh, príloha a - technická dokumentácia opisujeme zoznam príloh a technickú časť projektu.

1.1. Softvér ako produkt tímu

Na vývoji softvéru sa môže podieľať tím vývojárov alebo jedinec. Základom tímu je nakoniec len vývojár, preto sa v našej práci zameriame práve naňho. Z pohľadu tímu sa budeme snažiť skôr o porovnanie vývojárov medzi sebou.

2. Analýza existujúcich metód vizualizácie vývoja

V tejto časti sa pozrieme na rôzne pohľady na vývoj softvéru. Tieto pohľady sledujú určité črty softvérového projektu. Tieto črty sú následne merané a vyhodnocované. Vďaka čomu je možné vedieť v akej fáze projekt je. Tieto pohľady teda môžeme rozdeliť:

A. Sledovanie štruktúry kódu
B. Sledovanie vývoja pomocou metrík pre zdrojový kód
C. Sledovanie plnenia úloh[15]

V ďalšej časti identifikujeme nástroje, ktoré sledujú tieto črty. Často tieto nástroje sledujú viaceré črty naraz. Výsledok analýzy zhrnieme v závere kapitoly.

2.1. Analýza existujúcich nástrojov na zobrazenie vývoja softvéru

Medzi nástroje, ktoré sa priamo zameriavajú na vizualizáciu vývoja softvéru patria:

- Evolution Radar od D'Ambrosa, kde vizualizácia je založená na zobrazení vzťahov medzi súbormi a modulmi (združuje logicky podobné informácie)[20].
- Vonea a Telea vytvorili framework, ktorý analyzuje zdrojový kód uložený v CVS repozitári. Vývoj zobrazuje pomocou n-snapshot matice. Každý stĺpec matice zobrazuje vývoj jednej metriky.[20]
- Evolution Matrix od Lanza, ktorý poskytuje metódy na analýzu aspektov vývoja tried.[20]
- Lagrein [13] nástroj zobrazuje vývoj softvéru v podobe grafov.
- Nástroj vytvorený Wu, Holtom a Hassan , ktorý používal spektrografy. Vývoj spektrografu kombinuje čas, spektrum a meranú vlastnosť. Zobrazuje to v rôznych farbách, tak aby charakterizoval softvérový vývoj. Farebnosť schémy pomáha ukázať vzory vo vývoji.[20]

Hore spomenuté nástroje sme vypísali s cieľom poskytnúť prehľad aktuálnych nástrojov. Všetky nástroje sa zameriavajú na zobrazenie vývoja softvéru, avšak nie sú také dôležité, preto neuvádzame pri nich hlbší popis ani náčrt. Každý nastroj ma svoje špecifické vlastnosti. Mi sme z článku identifikovali iba najdôležitejšiu vlastnosť, ktorá by nám mohla pomôcť vytvoriť lepšiu vizualizáciu.

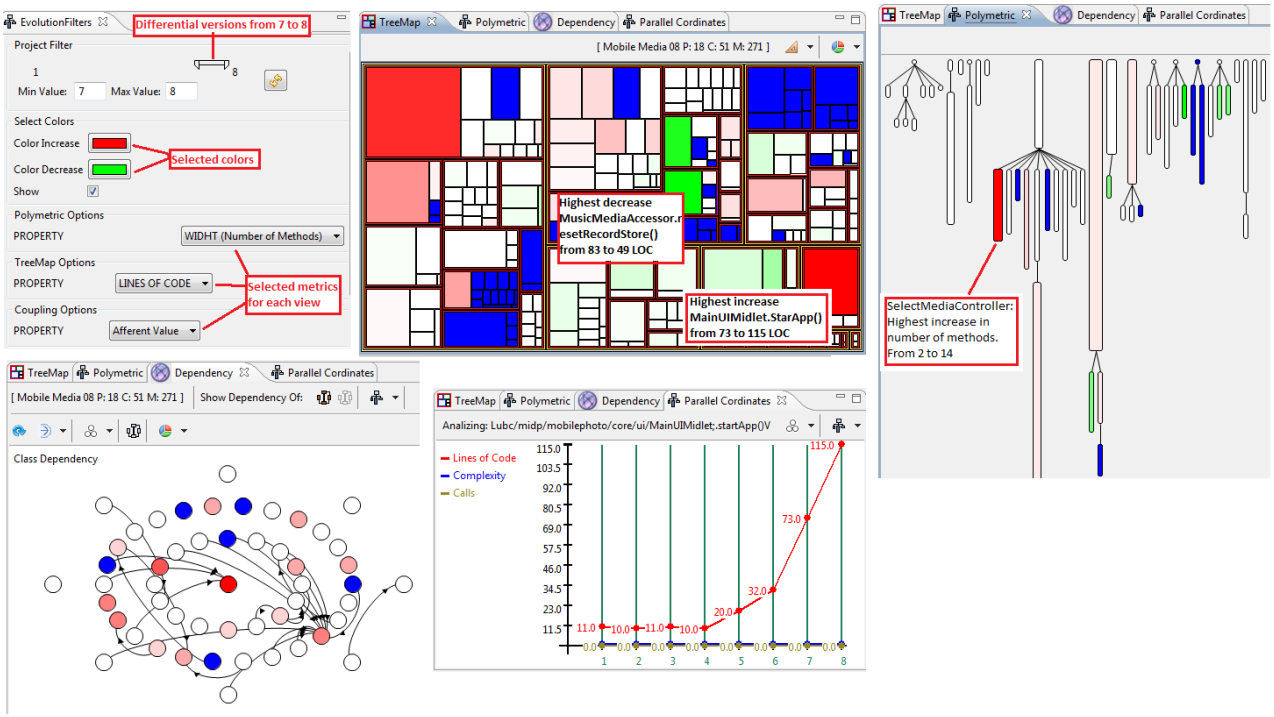
Opisujeme ďalej ďalšie nástroje, ktoré sa tiež zameriavajú na vizualizáciu vývoja softvéru, no sú k nášmu cieľu najbližšie a preto sú opísané samostatne. Nástroje nerozoberáme do hĺbky, predmetom tejto analýzy je stále poskytnúť prehľad medzi nástrojmi. Týchto nástrojov je pomerne málo a každý sa snaží zobraziť vývoj inak. Preto nie je objektívne tieto nástroje porovnávať medzi sebou. Vhodnejšie je ich opísať a z každého vybrať najlepšie črty. Nakoniec vybrať ten, ktorý je k nášmu cieľu najbližšie.

2.1.1. SourceMiner Evolution (SME)

V tejto kapitole opíšeme nástroj, ktorý používa rôzne pohľady pri zobrazovaní vývoja softvéru. Nástroj načíta dve rôzne verzie zdrojového kódu. Poskytuje tak vývojárovi možnosť dynamicky porovnávať všetky dvojice verzií softvéru. Porovnanie verzií zobrazuje priamo vo vývojovom prostredí (Eclipse).

Hlavnou súčasťou SME je analyzér. Tento analyzér porovnáva abstraktný syntaktický strom dvoch verzií zdrojového kódu. Výsledok analýzy tvorí hlavnú časť vizualizácie. Vizualizácia obsahuje celkom štyri pohľady. Tri pohľady sa zameriavajú len na vizualizáciu zdrojového kódu. Štvrtý pohľad sa zaoberá problémom, kde sa porovnávajú dve verzie zdrojového kódu. Samozrejme sa myslí aj na ich vývoj medzi dvoma časovými bodmi.

Súčasťou posledného pohľadu je časová priamka a množina metrík, zvoleného softvérového elementu. Za pomoci časovej priamky a metrík sa tak zobrazuje vývoj metrík v čase. Pre opis vývoja, SME používa osem metrík, definovaných podľa Lanza a Marinescu.[20]



Obr. 1: Jednotlivé okná nástroja SME. Každé okno reprezentuje pohľad.

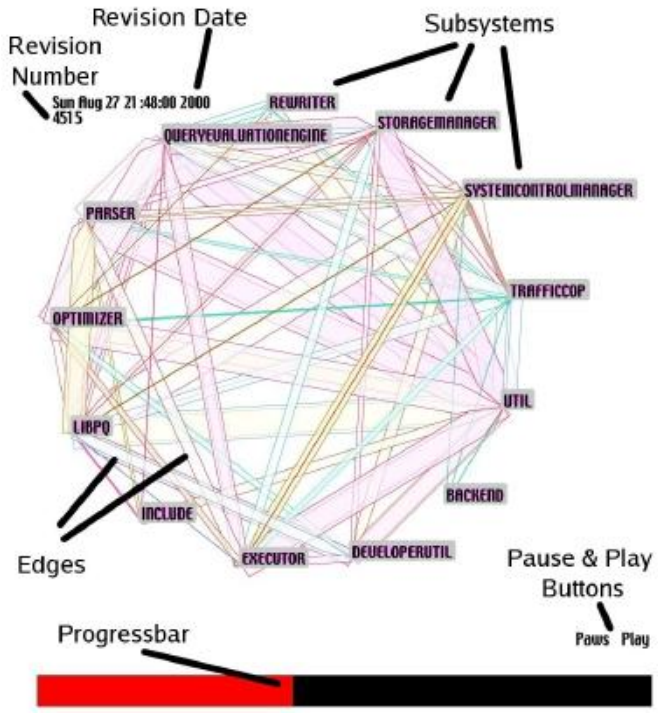
Súčasťou nástroja sú filtre podľa projektu, vývoja (času). Podrobné údaje o týchto filtroch sú v článku[20].

2.1.2. YARN

Je to ďalší nástroj, ktorý sa pozerá na zmeny v softvérovom produkte pomocou verziovacieho systému. Tieto zmeny analyzuje a následne ich animovane zobrazuje. Zameriava sa najmä na zmeny v architektúre. Animáciou zmien sa snaží intuitívne a prirodzene zobraziť zmeny v čase. Nástroj dokáže porovnať dve verzie softvéru medzi dvoma časovými bodmi. Toto porovnanie zobrazuje v tzv. „kumulatívnom pohľade“.[12]

V práci autori poukazujú, že sa spracováva pomerne veľké množstvo údajov. Pre svoj nástroj zvolili metódu zobrazovania informácii pomocou animovanej vizualizácie z dôvodu, že textové informácie často zaberajú viac priestoru.[12]

YARN (Yet Another Reverse-engineering Narrative) zobrazuje animované zmeny v závislosti (pomocou hrán) a zmeny v subsystémoch (vrcholy). Základom zobrazenia je teda graf závislosti. Zmeny v závislosti sú reprezentované zmenou farby a šírky hrany. Šírka hrany predstavuje silu závislosti. Algoritmus mení aj jas hrán a niektoré hrany môže extrémne zvýrazniť (oranžová farba) napríklad, ak ide o veľkú zmenu v architektúre. Vrcholy sú staticky umiestnené.[12] Snímka z animácie, kde sú použité vrcholy a hrany, je na obr. 2.



Obr. 2: Zachytená snímka z animácie vývoja softvéru (PostgreSQL) v nástroji YARN.

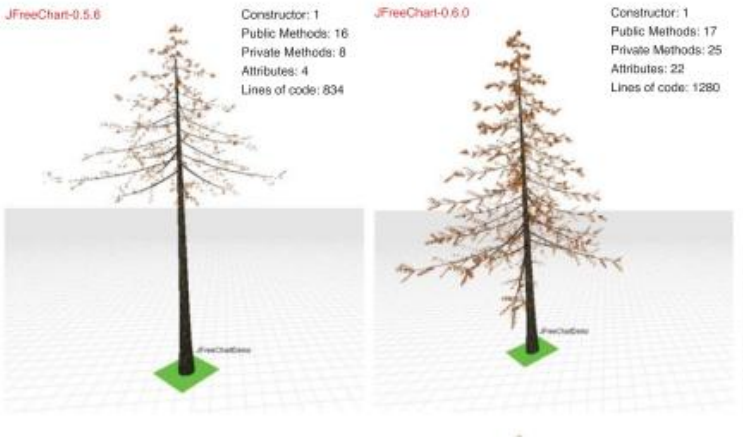
Tento graf je postavený nástrojom C-REX a HistODiff. C-REX extrahuje informácie z verziovacieho systému. HistODiff zase spracuje tento výsledok a počíta ďalej silu závislosti medzi sub systémami, kde sa za pomoci filtrov snaží ignorovať malé zmeny v zdrojovom kóde, ktoré nemali vplyv na architektúru.

2.1.3. Forest Metaphor

Erra, Scanniello vo svojej práci [6] poukazujú na veľkú výhodu vizualizácie pomocou stromov v lese. Každý strom v lese patri objektu. Zameriavajú sa teda na objektovo orientované systémy. Les je zobrazený v trojdimenzionálnom prostredí a vyvíja sa v čase (tak ako sa vyvíjal zdrojový kód). Vývojár má možnosť interaktívne pracovať so stromom a každou vetvou stromu. Vetvy priamo reprezentujú softvérové metriky: LOC, CLOC, NOA, NEOM, NEM, NOM.[6] Zobrazovanie informácií je riadené myšlienkou ,,Poskytni najprv prehľad, potom priblíženie s filtrami a nakoniec detaily na požiadanie“[23], ktorú prvý krát použil Schneiderman.

Prehľad v tomto význame predstavuje les. Les sa skladá zo stromov a stromy sú označené názvom triedy, ktorú reprezentujú. V tejto časti používateľ má možnosť interaktívne pracovať s prostredím a pohybovať sa po ňom. Filtre je možné aplikovať na výber tried. Pri priblížení na strom, môže byť tento strom vybraný a je ho možné následne pozorovať. Detaily sú poskytnuté pri opise stromu, prípadne ďalšie (napríklad hodnoty metrík) sa môžu zobraziť po interakcii so stromom.

Výhodou tejto vizualizácie je, že môže zobraziť veľké množstvo informácii (metódy, atribúty, dokonca komentáre). Autor opisuje aj nevýhodu, ktorá spočíva v tom, že pozorovateľ musí byť pripravený (naučený) pochopiť veľkú zložitosť takého množstva informácií.

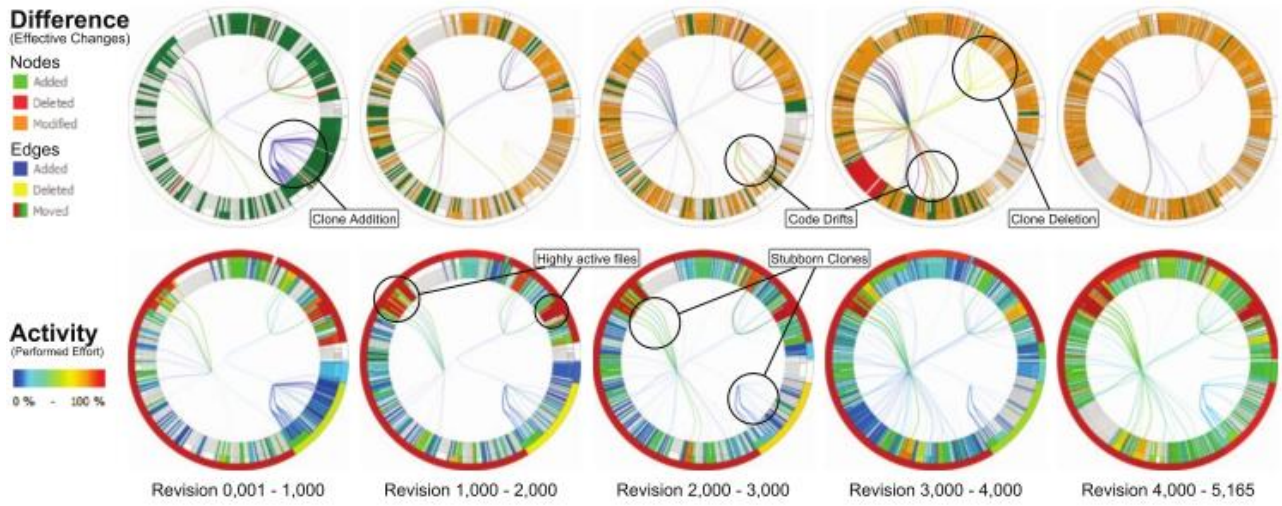


Obr. 3: Vývoj zdrojového kódu knižnice JFreeChart2 zobrazený pomocou Forest Metaphor.

Na obr. 3 je vidieť porovnanie 2 verzií triedy. Trieda (ľavý strom na obrázku) na začiatku má 24 metód a 834 riadkov kódu. Daná trieda sa vyvíja v čase, tak ako bola upravovaná vývojármi. Strom v pravej časti obrázka ma 42 metód a 1280 riadkov kódu. Listy stromu sa preto javia viac hustejšie a vetvy stromu sú až po koreň.[6]

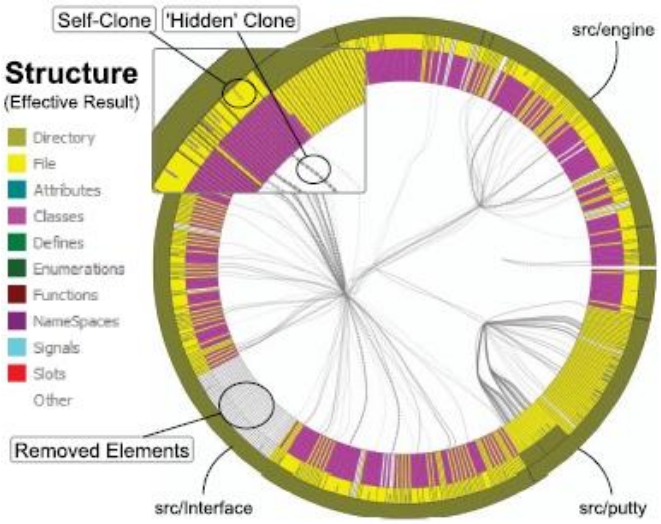
2.1.4. ClonEvol

Nástroj, ktorý spolupracuje s verziovacím systémom pre zber údajov. Taktiež s Doxygen-om a Simian-om pre statickú analýzu kódu. Hlavnou časťou vizualizácie je „mirrored radial tree". Tento strom sa mení podľa časovej osi, tak sa zobrazuje vývoj softvéru. Konečné výsledky vývoja získava aj pomocou objavovania znalostí. Veľkou výhodou tohto nástroja je škálovateľnosť.[11]



Obr. 4: Vývoj repozitára FileZilla od revízie 1 ku 5165 pomocou ClonEvol.[11]

Na obr. 4 sú zobrazené vrcholy usporiadané do kružníc (Mirrored radial tree). Tieto vrcholy predstavujú elementy zdrojového kódu a sú farebne odlíšené. Zelenou farbou sú nové elementy, červenou sú vymazané a oranžovou sú modifikované. Vrcholy na základe vzťahu elementov sú prepojené hranami. Rovnako ako sa elementy pridávajú, tak aj vznikajú nové vzťahy. Z toho dôvodu sú taktiež farebne odlíšené.



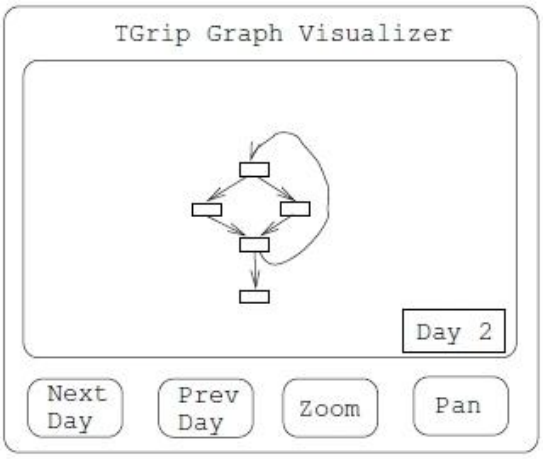
Obr. 5: Detail na kružnicu (Mirrored radial tree) a jej ďalšie časti.

Na obr. 5 je vidieť detail na konkrétnu revíziu. Detail zobrazuje štruktúru zdrojového kódu len na úrovni priečinkov, súborov a tried. Táto štruktúra predstavuje jeden stav vo vývoji. Ďalšie detaily sa zobrazia až po priblížení alebo kliknutí. Detaily nie sú poskytnuté na tejto úrovni, aby nezmiatli používateľa zobrazovaním príliš veľkého množstva informácií. V obrázku je vidieť vysokú mieru kohézie na časť „src/engine“.[11]

2.1.5. Gevol

Gevol nástroj používa systém na spravovanie verzií [4]. Využíva ukladanie postupných verzií zdrojového kódu na spätné analyzovanie vývoja. Tieto jednotlivé commit-y tak analyzujú a rekonštruujú vývoj.

Gevol zobrazuje vývoj pomocou grafu. Graf je menený v čase, postupne sa pridávajú vrcholy tak ako sa pridával zdrojový kód. Vrcholy sú farebne odlíšené, pri zmene v zdrojom kóde menia farbu. Keď si používateľ prezerá vývoj v čase, tak jeho pozornosť je upútaná zmenou farby. Prostredie je zároveň interaktívne a používateľ si môže pozrieť ďalšie podrobnosti zmien. Nástroj využíva aj graf dedenia a control-flow graf.[4]



Obr. 6: Schéma vizualizácie pomocou Gevol-u. V strede obrazovky je graf, ten sa mení v čase. Ďalšie schémy a snímky programu je možné nájsť v článku Collberga a spol..[4].

2.1.6. EPOSee

EPOsee rovnako ako predošlé nástroje ťaží informácie z verziovacieho systému. Tieto informácie zobrazuje pomocou pixel mapy, orientovaného grafu a trojdimenzionálnej matice. Samozrejme každá takáto schéma sa zobrazuje vo vlastnom okne. Rovnako poskytuje pre používateľa legendu vizualizácie, zoznam pravidiel a filtrov.

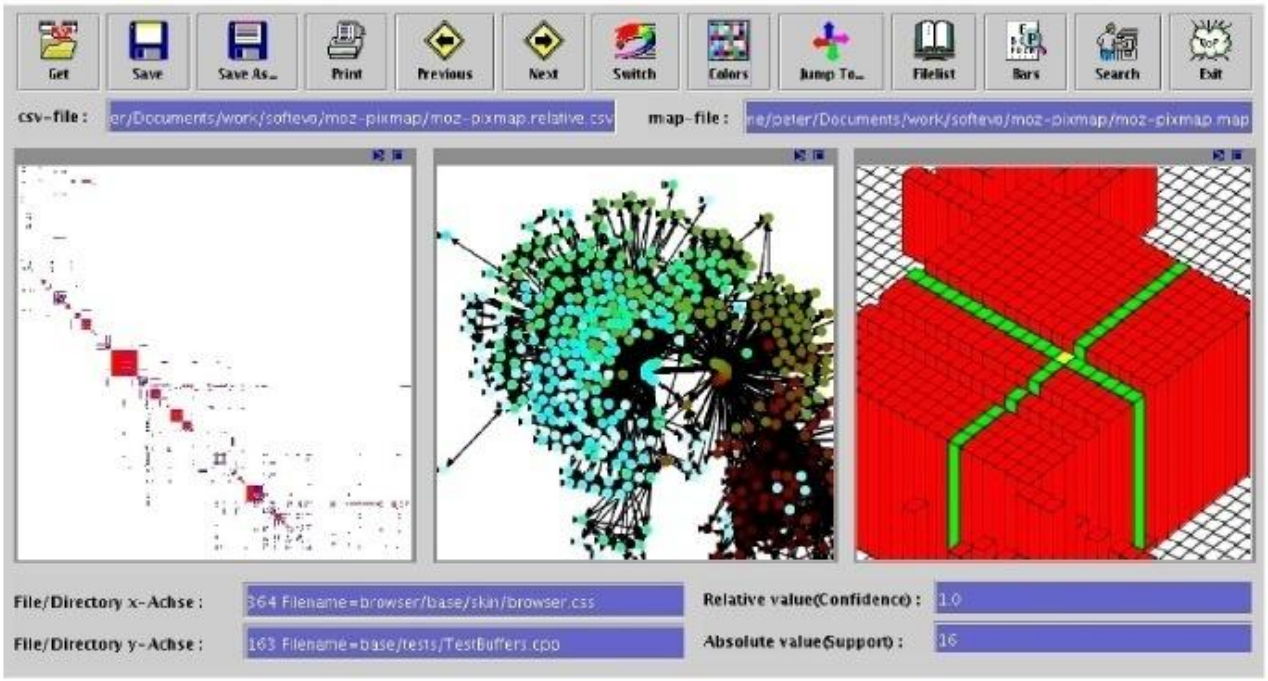
Vo farebnej pixel mape (obr. 7 ľavé okno), každý pixel reprezentuje softvérový prvok (súbor, triedu, metódu). Farba poukazuje na frekvenciu výskytu týchto prvkov v 2 verziách stiahnutých z verziovacieho systému. Čím je pixel červenší, tým je frekvencia vyššia. Mapa ukazuje napríklad, ktoré triedy sa najčastejšie upravovali (na základe commit-ov). [3]

Na ďalšej časti je grafový pohľad (obr. 7 stredné okno), kde každý vrchol opäť reprezentuje softvérový prvok. Farba vrcholu reprezentuje úroveň v hierarchii prvkov. Hrana reprezentuje spojenie medzi prvkami. Na tento pohľad sa dajú použiť rôzne filtre. [3]

Pri výbere určitej oblasti na pixel mape sa dodatočné informácie zobrazia v trojdimenzionálnej matici (obr. 7 pravé okno). Táto matica predstavuje histogram a ukazuje tak intervaly hodnôt. [3]

Všetky 3 obrazovky podporujú základný pohyb v priestore, približovanie a interakciu. Pri výbere prvku sa ihneď poskytnú detaily. Celý softvér bol navrhnutý myšlienkou „Poskytni najprv prehľad, potom priblíženie s filtrami a nakoniec detaily na požiadanie.“[23].

Vďaka princípom nástroja je možné nad údajmi objaviť vzory a získať nové znalosti. Prvé náznaky vzorov si používateľ môže všimnúť na pixel mape. Ak ho zaujímajú podrobnosti, tie si môže všimnúť na grafovom pohľade. Tam používateľ ma možnosť použiť filtre na obmedzenie zobrazovaných informácii.



Obr. 7: EPOSee pri práci.[3] **Základom aplikácie sú 3 okná.**

2.1.7. Gossip

Gossip (Graph Oriented System for Information Presentation) je systém na vizualizáciu informácii orientovaných na grafy. [9] Informácie zobrazuje v dvoj alebo v trojdimenzionálnom priestore. Zameriava sa najmä na 3D priestor. Podporuje rovnako pohyb a približovanie v tomto priestore. Umožňuje načítavať rôzne formáty súborov, podporuje vytváranie vlastných modelov údajov a využíva vlastné dátové štruktúry. Napríklad dokáže otvoriť a vizualizovať zdrojové súbory z lokálneho úložiska (projektu Visual studia), z verziovacieho systému (TFS) alebo informácie z RCS služieb (PerConIK služby).[1]

Gossip je vysoko dynamický a modulárny. Podporuje zásuvné moduly a tak jeho množina funkcii závisí od množiny pripojených modulov. Rovnako schémy, ktoré používa na vizualizáciu sú súčasťou doplnkov.

Opis prostredia

Prostredie v aplikácii je jednoduché a je rozdelené do týchto častí:

a) Project explorer – Zoznam aktuálne otvorených projektov. Je možné mať otvorené viacero projektov. Projekt môže obsahovať viacero modelov a pre každý model je možné vytvoriť viacero pohľadov.
b) Active view – Aktívny zvolený pohľad na model. Pohľad je možné si vybrať v „project explorer". S každým pohľadom je možné interaktívne pracovať.
c) Open views – V tejto časti sa nachádza zoznam otvorených pohľadov. Medzi nimi je možné prepínať.
d) Active view panels – V ľavej a pravej časti sa nachádzajú panely. V nich sú umiestnené rôzne nastavenia a filtre pre aktuálne zvolený pohľad. Nastavenia umožňujú manipulovať s pohľadom, prípadne cez filtre je možné nastaviť, ktoré údaje sa majú zobraziť.
e) logs – V dolnej časti sa nachádza panel pre záznamy, zoznamy chýb a ďalšie informatívne detaily.

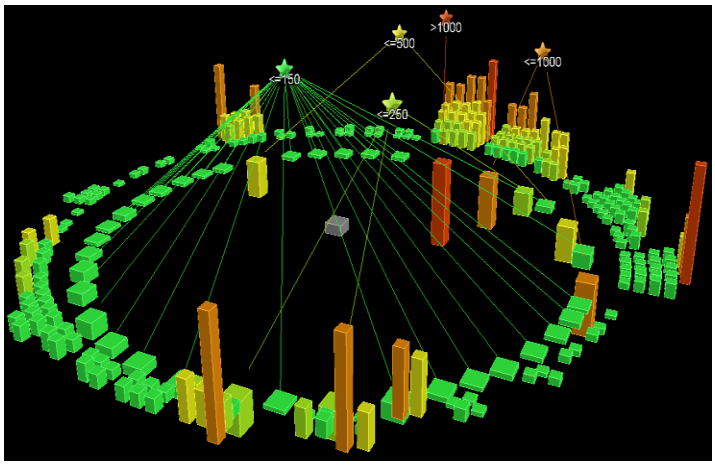
Obr. 8: Gossip prostredie vizualizácie.[2]

[1] Informácia prebratá z dokumentu Gossip používateľská príručka.

[2] Informácia prebratá z dokumentu Gossip používateľská príručka.

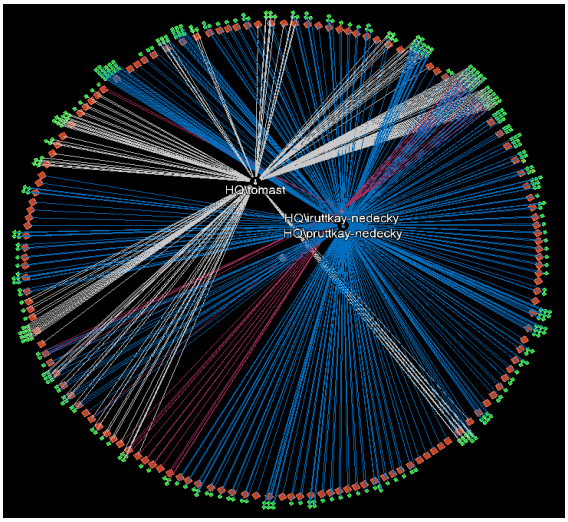
Gossip a vzťah ku PerConIK

Gossip sa nezameriava priamo na vývoj softvéru či jeho vizualizáciu. Je to skôr všeobecný nástroj na vizualizáciu. V práci ho spomíname, pretože v rámci projektu PerConIK bol tento nástroj použitý na vizualizáciu štruktúry zdrojového kódu a vizualizáciu metrík.[9] Príklady takého použitia sú na obr. 9 a obr. 10.



Obr. 9: Gossip pohľad na 3D Bar diagram v tvare kruhu. Jednotlivé hranoly predstavujú softvérové elementy. Výška hranolu reprezentuje hodnotu LOC.

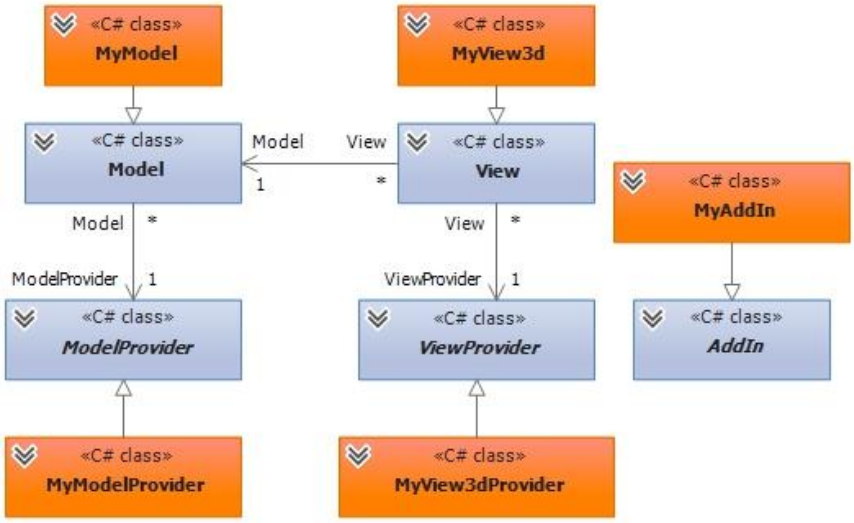
Jednoduchou animáciou týchto zobrazení podľa časovej priamky by sme mohli vizualizovať aj vývoj. Rovnako ako to robili ostatné spomenuté nástroje. Z tohto dôvodu toto riešenie nemá ďaleko k nášmu.



Obr. 10: Gossip pohľad na najčastejších prispievateľov zdrojového kódu. Mená sú zobrazené v strede a sú odlíšené farbou. Hodnota prispievania je ukázaná šírkou hranolu na konci kružnice.

Architektúry zásuvných modulov Gossipu

Nástroj Gossip má výbornú podporu na vizualizáciu informácii. Taktiež dokáže sťahovať informácie z PerConIK služieb. Obsahuje teda veľa výhod a je to vhodný kandidát na požitie. Preto sme tento nástroj analyzovali podrobnejšie a pozreli sa akým spôsobom môžu byť doplnky implementované.

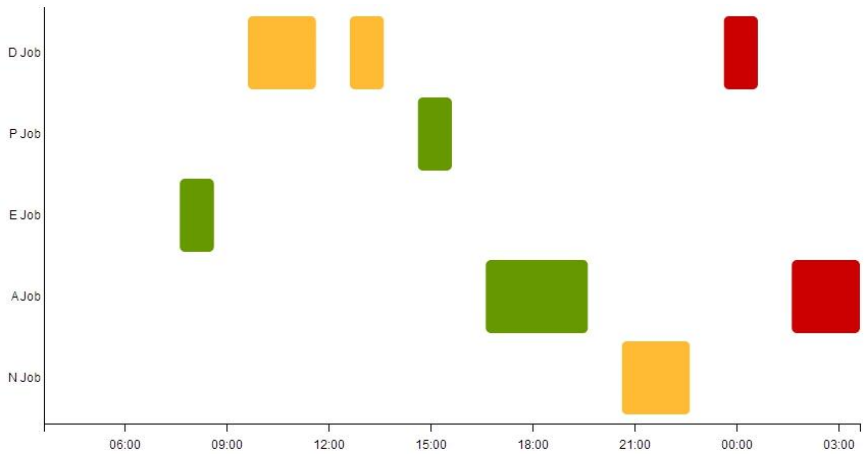


Obr. 11: Class diagram doplnku do nástroja Gossip. Triedy oranžovou farbou sú nové triedy, ktoré je potrebné implementovať v prípade vytvorenia doplnku. Triedy sivou farby sú základom pre rozšírenia.

2.2. Vizualizácia vývoja pohľadom na splnené úlohy

Vývoj softvérového produktu môžeme zobraziť pomocou Burndown, Gantt diagramu a mnohých ďalších PERT diagramov. Tieto diagramy sa používajú pri riadení a plánovaní projektu.[25]

Základom Gantt diagramu je časová priamka. V diagrame sú zaznačené plánované úlohy a aktuálny stav úloh. Rozdiel medzi úlohami poukazuje manažmentu na stav projektu. Pri plánovaní projektu sa odhaduje trvanie úloh. Odhadovanie trvania úloh však nie je objektívne.



Obr. 12: Gantt diagram vytvorený D3.js knižnicou.

Každá úloha má určitú časovú dĺžku, a je reprezentovaná obdĺžnikom. Tento diagram nám poukazuje, že na vizualizáciu aktivity alebo úlohy je vhodná časová priamka a na nej zobrazená aktivita formou obdĺžnika.

Tieto diagramy nám istou formou poukazujú na správanie vývojára, napríklad či si vývojár plní úlohy podľa plánu. Z týchto diagramov však nemôžeme získať žiadne ďalšie ukazovatele o správaní vývojára.

2.3. Záver z analýzy metód

Po základnej špecifikácii nástrojov je vidieť, že nástroje sa nezameriavali len na štruktúru kódu a metriky. Často krát tieto časti kombinovali. Avšak ani jeden nástroj sa nezameriaval na samotné kroky vývojára alebo na jeho správanie počas vývoja. Samotný vývoj softvéru následne simulovali tak, že tieto informácie animovali pomocou časovej priamky.

Väčšina nástrojov obsahovala rôzne filtre na údaje. Tieto nástroje sa pomerne dosť líšia spôsobom vizualizácie a rovnako aj používateľským prostredím.

3. Aktivita vývojára ako základ vývoja softvéru

V závere z analýzy metód sme určili, že mnohé vizualizačné techniky neponúkajú možnosť zobraziť vývoj softvérového produktu z pohľadu jednotlivých krokov vývojára. Ďalej je potrebné lepšie špecifikovať, ktoré kroky vývojára nás budú zaujímať a aký majú význam z globálneho hľadiska, teda vývoja.

Bude nás napríklad zaujímať, kedy vývojár otvoril prehliadač. Otvoril ho práve vtedy, keď implementoval funkcionalitu do systému, kopíroval niečo z internetovej stránky, alebo len čítal dokumentáciu. Podobných druhov informácií potrebujeme vo veľkom množstve. Pritom ich musíme správne skombinovať a interpretovať.

Tieto jednotlivé kroky alebo akcie vývojára, vzniknú pri vytvorení udalosti. Udalosť sama o sebe opisuje akciu, ktorá bola vykonaná. Sekvencia akcií tvorí aktivitu vývojára.

Výsledkom analýzy pracovných aktivít používateľa (vývojára) získame model používateľa ako súbor charakteristických čŕt používateľa a jeho správanie sa v čase a v pracovnom prostredí. Pracovná aktivita je množina činností používateľa za určitú dobu, v kontexte údajov s ktorými pracuje, pracovného prostredia resp. nástroja prostriedkov (HW aj SW) počítača. Aktivitu môžeme chápať ako ohraničenie dvoma významnými udalosťami, z ktorých pracovná aktivita nadobúda logický význam.[3]

Vykonávanie týchto aktivít v sekvencii, tak tvoria vývoj softvéru.

[3] Opis pracovnej aktivity je prebratý z dokumentu: PerConIK architektúra.

4. Nástroje na zachytenie správania používateľa

V článku H. Gall a M. Lanza o vizualizácii a analyzovaní vývoja softvéru [7] tvrdia, že je dobré mať prepojený systém na spravovanie verzií spolu so systémom na reportovanie chýb. Vďaka takému prepojeniu získame veľa užitočných informácii o vývoji. Spomína to aj Amor v článku o „Effort estimation by characterizing developer activity“, kde hovorí o potrebe sledovania aktivít vývojára z rôznych systémov.[2]

O podobné prepojenie sa snažíme aj my. Aby sme dokázali odpovedať na otázku v úvode, potrebujeme aby, model údajov obsahoval informácie z prostredia vývojára a verziovacieho systému. Z prostredia vývojára nás bude zaujímať, kedy vývojár napríklad otvoril prehliadač. Z verziovacieho systému nás bude zaujímať, koľko krát bol súbor prepísaný.

V tejto časti sa pozrieme na nástroje, ktoré sledujú kroky vývojára a majú takéto prepojenie.

4.1. PerConIK

Vývojom softvéru sa zaoberá aj výskumný projekt PerConIK (Personalized Conveying of Information and Knowledge, perconik.fiit.stuba.sk)[10]. Súčasťou projektu je aj riešenie, ktoré zaznamenáva a analyzuje pracovné aktivity používateľa za účelom analýzy správania sa používateľa v kontexte SW nástrojov, ktoré používa, ale aj v kontexte údajov (dokumentov, zdrojových kódov, Web, ...), s ktorými pracuje. Riešenie zbiera údaje o množstve stlačených kláves, kliknutí myšou, otvorení prehliadača a mnohé ďalšie. Pritom tieto údaje sú prepojené s verziovacím systémom.

Autori v článku o vizualizácií vývoja softvéru poukazujú na výhody v tvorbe poznámok ku zdrojovému kódu. [6] Tieto poznámky alebo tagy pomáhajú lepšie pochopiť softvérové zmeny. Podobné značkovacie tagy používa aj *PerConIK* . Projekt ponúka všetky údaje vo forme 3 webových služieb (RESTful služby).[4]

Riešenie na zaznamenávanie a analýzu aktivít vývojára je implementované v podobe viacerých aplikácií. Riešenie podľa funkcionality môžeme rozdeliť na tri časti:

- Sledovač aktivít
- Filtrovanie aktivít
- Uchovávanie aktivít

4.1.1. Sledovač aktivít

Je to aplikácia, ktorá sleduje činnosti vývojára, ako používateľa operačného systému. Ide o centrálnu aplikáciu v operačnom systéme, ktorá zbiera údaje aj z iných prostredí pomocou dodatočných komponentov. Tieto komponenty sledujú úpravu dokumentov, webový prehliadač atď. Všeobecne sa sledujú aj aktuálne bežiace procesy, otvorené okná a používateľove interakcie. Pre sledovanie aktivít sa využíva aj doplnok do vývojárskeho prostredia (sleduje operácie nad zdrojovým kódom).[5]

[4] Opis pracovnej aktivity je prebratý z dokumentu PerConIK architektúra.

[5] Informácia prebratá z dokumentu PerConIK používateľská príručka.

4.1.2. Filtrovanie aktivít

Mnohé zachytené činnosti majú vysokú granularitu. Za pomoci filtrov je ich možné konfigurovať a takto riadiť granularitu odosielania údajov. V tejto fáze sa činnosti filtrujú napríklad na základe významu činnosti alebo pre ochranu súkromia používateľa.

4.1.3. Uchovávanie aktivít

Všetky akcie vývojára sú uložené v databáze. V databáze sa rovnako ukladajú údaje o commit-och, ktoré sú prepojené na systém pre spravovanie verzií. Jednotlivé akcie sú prepojené na údaje o commit-och a nepriamo tak aj na zdrojový kód, ktorý bol upravovaný počas trvania aktivít. Tieto vzťahy sa určujú neskôr spracovaním.

4.1.4. Opis aktivity

Každá aktivita ma dodatočné atribúty. Tieto atribúty budú podrobne opísané neskôr. Každý commit môže obsahovať údaje o metrikách pre daný zdrojový kód. Zdrojové kódy zase môžu obsahovať značkovacie tagy, ktoré nám povedia viac o softvérovej časti alebo o aplikovaných zmenách. *PerConIK* podporuje 6 druhov používateľských značiek, zároveň aj vytváranie nových druhov či generovanie značiek.[6]

4.1.5. Gossip

Spolu s *PerConIK* – om súvisí aj nástroj *Gossip*, ktorý môže využívať údaje v databázach projektu. *Gossip* je všeobecný nástroj na vizualizáciu. Je rozšíriteľný pomocou zásuvných modulov a má práve jeden zásuvný modul, ktorý sa dokáže pripojiť na služby projektu. Nástroj *Gossip* a jeho vizualizačné možnosti sú opísané v sekcii 2.1.7.

4.2. Ďalšie projekty na sledovanie aktivít vývojára

Metódu sledovania aktivít vývojára použili pri práci „Using developer activity data to enhance awareness during collaborative software development“. V analýze tohto článku sa opisujú ďalšie nástroje. Článok tak poskytuje dobrý prehľad o nástrojoch, ktoré sledujú aktivity používateľa alebo vývojára. Tieto nástroje sledovali aktivity s cieľom vylepšiť používateľské prostredie alebo sledovali iba aktivity z IDE prostredia, a nešlo tak o generickú metódu.[21]

V článku „Automatically detecting developer activities and problems in software development work“ sledovali aktivity v IDE aj Web prostredí. Snažili sa odpovedať na viacero otázok o vývoji softvéru. Tieto otázky sú podobné našim. Zachytené aktivity spracovali pomocou skrytých Markových modelov. Nešlo tak o vizualizáciu údajov ale skôr o klasifikáciu aktivít a predikciu ďalších aktivít.[22]

4.3. Záver z výberu nástroja na zachytenie správania používateľa

Súčasti projektu *PerConIK* majú schopnosť zachytávať aktivity vývojára. Projekt je prepojený na verziovací systém. Informácie poskytuje jednoducho za pomoci služieb, pričom projekt beží už určitý čas a tak jeho databáza je už sčasti naplnená.

Pre náš pokus môžeme teda použiť jeho dáta. Tento projekt je teda ideálny pre naše riešenie.

---

[6] Informácia prebratá z priloženého dokumentu PerConIK používateľská príručka.

4.4. Analýza údajov v projekte PerConIK

Prácu vývojára môžeme rozdeliť do jednotlivých akcií. Tieto akcie môžu byť súčasťou nejakej aktivity. Tento pohľad na prácu vývojára má aj projekt *PerConIK*. Mechanizmus zachytávania aktivít a projekt *PerConIK* sme spomenuli v kapitole 4.1.

Presný spôsob zbierania dát, ich zachytávanie a posielanie dát v rámci *PerConIK*-u nie je predmetom tejto práce. Preto ďalšie časti budú opísané len pre vysvetlenie základného princípu a do tej miery, aby sme dokázali riešiť náš problém – hlavnú otázku: "*Ak vývojár často používa prehliadač (resp. portál) pri písaní svojho kódu, ako často je potom tento kód prepisovaný? Platí to u každého iného vývojára rovnako?*"

4.4.1. Schéma zachytených stavov a udalosti

Udalosti (činnosti používateľa) sú zachytené podľa možnosti chronologicky. Zachytávajú sa rôzne typy udalostí. Všetky udalosti sa nachádzajú v databázovej schéme, priloženej k práci .Pre našu potrebu nás budú zaujímať udalosti ako napríklad: spustenie prehliadača, otvorenie karty v prehliadači, otvorenie stránky, skopírovanie časti textu z webovej stránky do schránky a vloženie daného textu do zdrojového kódu.

Udalosť „skopírovanie *časti textu z webovej stránky do schránky a následne do zdrojového kódu.*“ sa nenachádza priamo v schéme. Táto udalosť sa ale dá poskladať z viacerých menších aktivít, ktoré sa vykonávajú v sekvencii. Napríklad za pomoci udalosti „prepnutie medzi aplikáciou“ a „stlačením presnej klávesy“ dokážeme zistiť či sa používateľ prepol do webového prehliadača a ak áno, či stlačil správnu klávesovú skratku pre skopírovanie textu. Podobným princípom dokážeme zistiť ďalšie zaujímavé udalosti.

Pre naše potreby nás budú ešte zaujímať udalosti nad daným vloženým textom. Tieto udalosti sa zachytávajú vo vývojom prostredí – pomocou doplnku do IDE. Sú označené typom IDE a nachádzajú sa v schéme. Takéto udalosti budeme musieť analyzovať a kontrolovať či prišlo k zmene vloženého textu – teda či táto časť zdrojového kódu bola prepísaná.

Táto analýza nebude ľahká. Dôvodom je veľké množstvo zachytených udalostí. Je potrebné si uvedomiť, že nad daným kódom mohol zmenu vykonať aj iný vývojár. Ďalej daná zmena bola aplikovaná až za určitú časovú dobu. Posledné 2 body je potrebné efektívne riešiť v rámci spracovania a filtrácie dát.

4.4.2. Dátový model údajov

V kapitole schéma zachytených stavov a udalosti sme spomenuli, že ďalšie udalosti sa dajú poskladať z iných udalostí. V projekte PerConIK sa pripravili aj na takúto možnosť. Po zachytení udalosti, sú niektoré udalosti automaticky spracované a analyzované.

Vďaka existujúcemu mechanizmu analyzovania udalostí sú výsledné údaje spracované v databáze. Nachádzajú sa tam aj tie, ktoré potrebujeme pre riešenie hlavnej otázky. Pre riešenie problému potrebujeme teda len prechádzať entitami v databáze.
Dátový model je znázornený na obr. 30. Každá entita vychádza z abstraktnej triedy *Event*. Každá entita je reprezentovaná unikátnym URI.

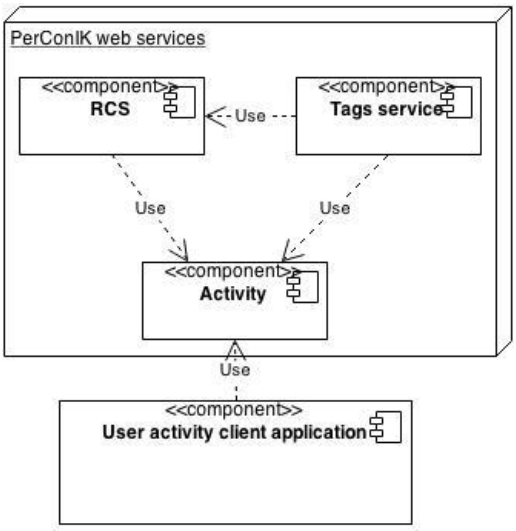
4.4.3. Obmedzenia projektu PerConIK

Projekt *PerConIK* je stále vo vývoji. V momentálne fáze má vytvorené webové doplnky do *Chrome* a *Firefox*. Ďalej doplnky do vývojového prostredia *Visual studio* a *Eclipse*.[9] Teda nepodporuje všetky nástroje, ktoré sa používajú pri vývoji. Ak vývojár použije iný nástroj, jeho akcie nemusia byť správne zachytené.

4.5. Stav architektúry PerConIK

PerConIK projekt je postavený na architektúre orientovanej na služby. Dana architektúra tak môže byť rozšírená, pridaním novej služby. Momentálne v projekte sú pripravene 3 základne služby. Zachytené údaje sa spracovávajú a sú uložené v týchto službách. Prepojenie týchto služieb je možné vidieť na diagrame komponentov na obr. 13. Tieto služby sú:

A. Aktivity služba
B. Služba na spravovanie verzií (RCS - Revision control system)
C. Služba na spravovanie značiek (Tags service)



Obr. 13: Diagram komponentov služieb v projekte PerConIK.

4.5.1. Aktivity služba

Aktivity služba je jedna zo služieb v rámci *PerConIK* projektu. Táto služba ma priamo na starosti zachytávanie a spracovávanie akcií vývojára. Tieto akcie sú po spracovaní uložené a neskôr sprístupnené. Spracúvajú sa napríklad akcie vo vývojom prostredí, kde sa údaje spájajú so službou RCS a hľadajú sa prepojenia na systém spravovania verzii. Je to *RESTful* služba. Údaje sú sprístupnené vo forme *JSON*. Vyžaduje sa autorizácia klienta. Dátový model tejto služby je na obr. 30.

4.5.2. Revision control system

Služba RCS - Revision control system - je súčasťou *PerConIK* služieb. Povinnosťou tejto služby je analýza verzií zdrojových súborov, s ktorými vývojár pracuje. Služba rozdeľuje jednotlivé verzie a softvérové elementy do entít. Entity sú spájané do grafu. Graf predstavuje zdrojový kód. Na základe histórie entít je možné zrekonštruovať celkovú históriu akéhokoľvek zdrojového súboru alebo jeho časti.

Základom služby je ďalej spolupráca s verziovacím systémom. Služba tak ukladá rôzne meta údaje o aktivitách používateľa v rámci verziovacieho systému. Napríklad poslanie commit-u. Je to *RESTful* služba. Údaje sú sprístupnené vo forme *JSON*. Vyžaduje sa autorizácia klienta. Dátový model tejto služby je na obr. 31 .

4.5.3. Tags service služba

Služba na spravovanie značiek je rovnako súčasťou *PerConIK* služieb. Služba sa zaoberá anotovaným zdrojovým kódom. Tieto anotácie sú tu zozbierane a analyzovane. Anotácie nám dodávajú dodatočné informácie k zdrojovému kódu. Význam anotácii bol spomenutý v článku o vizualizácii vývoja softvérového

systému. [6] Tieto poznámky alebo tagy pomáhajú lepšie pochopiť softvérové zmeny. Je to SOAP služba. Údaje sú sprístupnené vo forme *XML*. Vyžaduje sa autorizácia klienta.

4.6. Analýza údajov v Aktivity službe

V tejto kapitole sa pozrieme na údaje v „Aktivite" službe. V dátach pravdepodobne môžeme nájsť odpovede na zaujímavé otázky. Z dátového modelu aktivity služby na obr. 30 vidíme, že sa zachytávajú aktivity pre:

A. Webový prehliadač (Web event)
B. Integrovane vývojárske prostredie (IDE event)
C. Operačný systém, zachytáva sa zoznam procesov (Processes Changed Since Check Event)

4.6.1. Aktivity vo webovom prehliadači

Programy projektu PerConIK na zachytávanie správania vývojára nedokážu sledovať celkovú aktivitu ale len definované aktivity. V tejto časti sa pozrieme aké aktivity sa sledujú v rámci webového prehliadača.

A. Navštívenie webového cieľa kliknutím na adresu
B. Navštívenie cieľa napísaním adresy
C. Navštívením webovej stránky cez uložený odkaz (bookmark)

Ďalej sa sleduje práca vo webovom prehliadači. Konkrétne:

A. Otvorenie, zatvorenie, prepnutie tabu
B. Uloženie webovej stránky alebo stiahnutie súboru
C. Uloženie webovej adresy medzi obľúbene (bookmark)

4.6.2. Aktivity v integrovanom vývojom prostredí

V integrovanom vývojom prostredí sa sleduje najviac aktivít. Tieto aktivity sú pre nás najdôležitejšie, pretože vývojár pracuje so zdrojovým kódom. Teda nasledujúce aktivity sú priamo súčasťou vývoja.

A. Zmena stavu v IDE (napríklad zmena z design na build mód)
B. Práca s projektom (otvorenie, prepnutie, pridanie, zatvorenie, ...)
C. Udalosť nad elementom kódu (element je viditeľný, úprava elementu)
D. Vývojár odovzdal verziu zdrojového kódu do systému na spravovanie verzií
E. Práca so súborom (otvorenie, prepnutie, pridanie, zatvorenie, ...)
F. Udalosť nad zdrojovým kódom (kopírovanie, vloženie, zmena)
G. Udalosti pri vyhľadávaní

4.6.3. Aktivity v operačnom systéme

Pri práci vývojára sa sledujú zmeny aj v operačnom systéme. Sledujú sa len zmeny v procesoch. Tieto zmeny sa posielajú v intervale 5min. Sleduje sa, ktoré procesy boli vytvorené a ktoré zatvorené. Z týchto údajov nie je možné zistiť presný čas zapnutia procesu. Je ale možné aproximovať tento čas s toleranciou 5 min. Viď obr. 30.

V operačnom systéme sa nesledujú zmeny aktívnej aplikácie. Nie je teda možné zistiť, ktorú aplikáciu vývojár práve používal. Je možné zistiť iba, ktoré aplikácie mal zapnuté počas jeho práce. Tento záver neplatí pre aktivity vo webovom prehliadači a v integrovanom vývojom prostredí. Keďže tam zachytená aktivita priamo predstavuje vykonanie akcie v prostredí alebo v aplikácii.

5. Analýza knižníc pre fázu Implementácie

5.1. Analýza knižníc pre Timeline vizualizáciu

Pre naše riešenie by sme mohli vybudovať vlastnú vizualizáciu. Z hľadiska systémovej práce sme sa ale rozhodli identifikovať open source knižnice, ktoré vytvárajú *Timeline* vizualizáciu. V súčasnosti sme našli tieto knižnice:

- Google charts: Timeline[7]
- Silverlight & WPF Timeline Control[8]
- SIMILE Timeline[9]
- Vis.js[10]

Z týchto knižníc je pre nás vhodná javascript knižnica *vis.js*, ktorá bola vytvorená pre webovú platformu. Spĺňa najviac požiadaviek na vizualizáciu. Ostatné knižnice sú pre iné programovacie jazyky alebo sú ťažko modifikovateľné.

5.2. Analýza knižníc na metriky zdrojového kódu

Na riešenie hlavného cieľa, koľko zdrojového kód bolo prepísaného, je potrebné kvantitatívne vypočítať, veľkosť. To sa dá za pomoci metrík zdrojového kódu. Treba si uvedomiť, že vývojári mohli pracovať na rôznych projektov, v rôznych programovacích jazykoch. Preto je potrebné nájsť knižnicu, ktorá podporuje rôzne jazyky a ma čo najviac metrík. Identifikovali sme tieto Java knižnice:

- *CodeAnalyzer*[11]
- *Metrics*[12]
- *JavaNCSS*[13]
- *LOCC*[14]

Najviac softvérových metrík ma implementovaných knižnica *CodeAnalyzer*, ktorú sme sa rozhodli použiť. Ako základná metrika pre experiment bude použitá LOC. Použitie softvérovej metriky bude ale konfigurovateľné.

---

[7] https://developers.google.com/chart/interactive/docs/gallery/timeline

[8] http://www.nuget.org/packages/Timeline

[9] http://www.simile-widgets.org/timeline/

[10] http://visjs.org/

[11] http://www.codeanalyzer.teel.ws/

[12] http://metrics.sourceforge.net/

[13] http://www.kclee.de/clemens/java/javancss/

[14] http://csdl.ics.hawaii.edu/research/locc/

6. Analýza vhodného nástroja pre porovnanie

V analýze diplomovej práci sme uviedli 6 nástrojov. Nástroj ClonEvol sleduje zmeny nad zdrojovým kódom, vizuálne deteguje možné problémy, napríklad vysokú závislosť modulov nejakého systému. Nástroj tak rieši určitý typ problémov. Nesleduje však akcie vývojára. Nedokáže tak upozorniť na iný typ problému. Napríklad či nadmerná komunikácia alebo práca počas nočných hodín ovplyvňuje výkon vývojara.[14] Podobne je na tom SME, Lagrein, Yarn, Forest metaphor, Gevol, EPOSee. Hlavným rozdielom je, že nástroje sa pozerali na vývoj softvéru **z pohľadu vývoja štruktúry kódu**. Tieto pohľady a náš pohľad, pohľad na aktivity a kroky vývojára sú odlišné, preto to nie je korektné porovnať.

V článku „Combined visualization of structural and metric information for software evolution analysis“ sa autor zameral na aktivity vývojára z verziovacieho systému. Sledoval koľko vývojár upravil zdrojového kódu, ako často vykonal commit a pod.[8] Náš nástroj je v tomto smere podobný, pozerá sa na aktivity v rámci verziovacieho systému ale aj ďalších systémov. Spomenutý nástroj v článok dokonca rovnako používa vizualizáciu pomocou časovej priamky. Na určenie veľkosti zmeny používa podobné softvérové metriky. Nástroj je už ale **nedostupný** a bola ukončená jeho podpora.

V článku o detekcii aktivít vývojára od Roehm používajú skryté Markove modely na predikciu ďalších aktivít vývojára. Informácie zobrazujú len vo forme textu.[22] Náš nástroj zobrazuje doterajšie aktivity vývojára časovou osou a bol by tak vhodný doplnok k tomuto nástroju. Táto práca tiež nie je vhodná na porovnanie, pretože sa nejedná o žiadnu vizualizáciu

Analýza v práci „Using developer activity data to enhance awareness during collaborative software development“ od Ferguson-a a spol. poskytuje dobrý prehľaď nástrojov[22], určených na rôzne účely a venujúcich sa aktivitám vývojára. Medzi tieto nástroje patrí aj Team Tracks. Je to vizualizačný nástroj, ktorý zobrazuje aktuálnu aktivitu vývojára. Napríklad, ktoré súbory sú aktuálne menené a kto ich mení. Doplnok do IDE prostredia sleduje, ktoré súbory vývojár najčastejšie navštívil a predpokladá, že tie môžu byť problémové. Nástroj je určený pre tímy a riadenie kolaboratívnej práce. V popise nástrojov nie je uvedený nástroj, ktorý by riešil vizualizáciu vývoja softvéru pohľadom, podobným nášmu. Článok však vyznačuje **dôležitosť** zachytávania **údajov o aktivitách** vývojára a samozrejme aj vizualizáciu týchto údajov. Preto sme sa rozhodli vybrať nástroje spomenuté v tomto článku a porovnať ich cez **Gutwinové prvky**. To sme urobili v kapitole porovnanie cez gutwinové prvky.

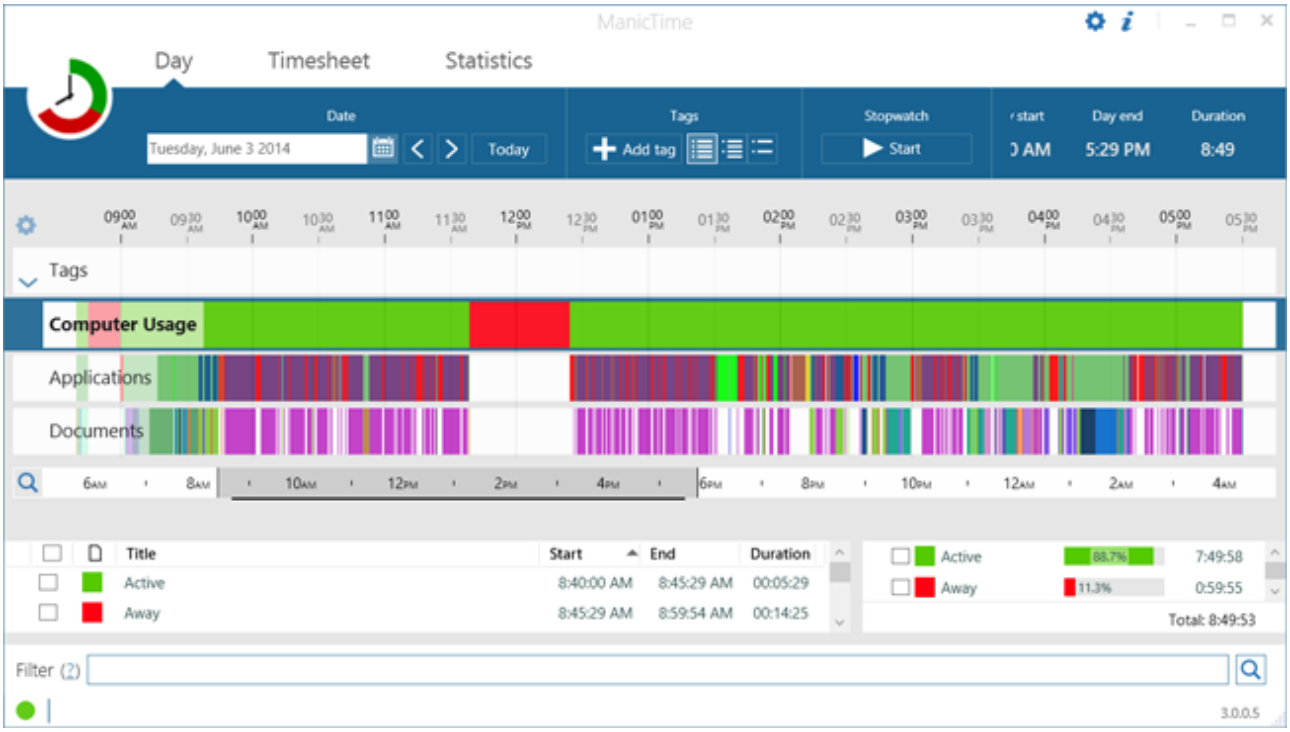
M. Lanza a R. Minelli sú známe osoby vo vizualizácií dát. Spolu napísali článok o vizualizácii toku práce vývojára[18] a druhý článok o vizualizácií interakcií vývojára [17]. Autori vizualizovali používanie IDE prostredia vývojárom. V porovnaní s naším nástrojom sa teda venovali len jednému prostrediu. Na druhej strane rovnako použili vizualizáciu dát cez časovú os. Pre aktivity nezobrazovali metriky.

Niektoré spomenuté nástroje sú účelovo odlišné a niektoré zase je nemožné použiť pre porovnanie použiteľnosti. Napríklad z dôvodu nedostupnosti nástroja. Z týchto dôvodov sme nevybrali ani jeden nástroj. Následne sme identifikovali komerčný nástroj **ManicTime**. Opis nástroja je v nasledujúcej kapitole.

6.1. Opis nástroja ManicTime

ManicTime automaticky nahráva používanie počítača pri akejkoľvek práci. Zameriava sa na použité aplikácie (prostredie) a dokumenty (kontext), ktoré sa používali cez aplikáciu. Sleduje sa ako dlho tieto aktivity trvali.

Zachytené dáta je možné vidieť na časovej osi a slúžia používateľovi ako prehľaď práce na pracovnej stanici. Nástroj dokáže vypracovať výkazy práce a štatistiky používania aplikácii.[26]



Obr. 14: Rozhranie aplikácie ManicTime.

Na rozdiel od IVDA nástroja program sleduje aktivitu vývojára aj ju vizualizuje. Pohyb a ovládanie na časovej osi ako aj princípy vizualizácie sú veľmi podobné.. Ďalšie rozdiely sú v tabuľke nižšie.

7. Sumarizácia kľúčových problémov

7.1. Veľa údajov

Pri vizualizácií má veľký zmysel zobrazovať najmä podstatné informácie. Projekt PerConIK zachytáva akcie vo vysokej granularite (napríklad každé otvorenie karty, jej zatvorenie, úpravu kódu), preto tento model obsahuje veľa údajov. Z toho dôvodu je potrebné tieto údaje prefiltrovať a zobraziť len to podstatné.

7.2. Neporiadok vo vizualizácií

Jeden problém súvisí s ľudskou pamäťou a Millerovým zákonom[16]. Druhý s vizuálnym neporiadkom, pričom Ellis, Dix a Aris uvádzajú ako hlavný dôvod problému skutočnosť, že sa zobrazuje príliš veľa dát na príliš malej časti displeja.[5][24] Tento problém bude potrebné riešiť kvalitným používateľským prostredím, kde používateľovi zobrazíme len to podstatné.

7.3. Nájsť odpovede na otázky

Tretím problémom, pre nás najdôležitejším, je potreba sformulovať ďalšie otázky, podobné tým, ktoré sme si uviedli v zadaní. Pred definovaním ďalších otázok bude vhodné sa pozrieť na možnosti použitého zdroja údajov. Vybrať zaujímavé údaje a sformovať ich do modelu údajov. Následne v modeli údajov opäť potrebujeme údaje prefiltrovať, podobné údaje zoskupiť a na základe otázky spraviť analýzu a výsledok.

7.4. Vybrať správne metriky

Mnohé spomenuté nástroje používali rôzne softvérové metriky, ktoré sa potom použili pri vizualizácii. Každý nástroj používal inú množinu metrík. Týmto metrikám je potrebné venovať samostatnú kapitolu a vybrať najvhodnejšie. Teda také, ktoré poukazujú na vývoj softvéru smerom k odpovediam na naše otázky.

8. Špecifikácia

Podľa článku o vizualizácii a analyzovaní vývoja softvéru od H. Gall a M. Lanza[7] má zmysel prepájať rôzne systémy s cieľom získať čo najviac údajov o vývoji softvéru. Dnes sa pri vývoji často používa systém na spravovanie verzií, systém na manažovanie úloh. V našom návrhu sa budeme snažiť ukázať vývoj novou metódou. Základom tejto metódy bude prepojenie systému na spravovanie verzii a biometrických informácii vývojára a zachyteného správania vývojára z vývojárskeho prostredia.

Základom metódy budú prevažné zachytené udalosti vývojára. Tieto udalosti vznikajú krok po kroku pri jeho práci. Mnohé akcie budeme ignorovať. Tie dôležité má zmysel zobraziť tak, aby bolo jasné o akú udalosť ide, a kedy nastala. K tomuto cieľu sa najviac blíži vizualizácia pomocou časovej priamky, podobná schéme *Gantt diagramu* (pozri obr. 12). Tento spôsob vizualizácie nebol opísaný v žiadnom existujúcom nástroji na vizualizáciu vývoja. Podobný princíp použili na vizualizáciu vývoja softvérových metrík v článku.[8]

Pôjde o vizualizáciu, kde udalostiam bude priradená časová stopa a tá bude zobrazená na časovej osi (pozri obr. 12). Používateľovi by malo byť jasné, kedy udalosť nastala a ako dlho trvala. Zároveň na základe susedných objektov, používateľ dokáže identifikovať akcie pred a po udalosti.

Vizualizácia pomocou **časovej priamky** priamo neodpovedá na definovaný cieľ – "Ak vývojár často používa prehliadač (resp. portál) pri písaní svojho kódu, ako často je potom tento kód prepisovaný?" Tá len poukazuje kedy dané kroky boli vytvorené, kedy vývojár použil prehliadač a kedy bol jeho kód prepísaný. Zobrazením týchto akcií zobrazujeme, ako často používal prehliadač a ako často jeho kódy boli prepísané.

Práca vývojára na projekte môže mať rôznu časovú lehotu. Môže trvať deň alebo rok, preto má zmysel ohraničiť časovú os na určité obdobie. Môžeme tiež ponúknuť používateľovi možnosť upraviť toto ohraničenie aby si mohol pozrieť konkrétne obdobie, ktoré ho zaujíma.

Používateľovi poskytneme koláčové grafy vedľa časovej priamky, ktoré spresnia koľko používal prehliadač z jeho celkovej aktivity v rámci definovaného obdobia. Pomocou ďalších grafov ukážeme pomer, koľko kódu bolo prepísaného, ako často dochádzalo k zmenám, o akú veľkú zmenu ide a pod. Veľkosť zmeny sa vypočíta pomocou softvérových metrík. Je potrebné poskytnúť aj ďalšie grafy tak, aby riešenie bolo generické a odpovedalo aj na otázky mimo hlavnej hypotézy. V závere má byť tento nástroj všeobecný a odpovedať na viaceré otázky ohľadom vývoja.

Stanovujeme si vytvorenie minimálne 3 grafov. Graf zobrazujúci prehlaď všetkých udalostí v čase, graf zobrazujúci používanie prehliadača a graf pre používanie IDE prostredia.

Porovnanie dvoch vývojárov bude možné jednoduchým porovnaním dvoch časových priamok alebo grafov. Takto zistíme, či takáto tendencia prepisovania kódov platí pre každého vývojára. Toto sme definovali na začiatku ako druhý cieľ.

8.1. Softvérové a hardvérové požiadavky

Našu aplikáciu plánujeme nasadiť na *Google Cloud*[15], ktorý ponúka servery a jeho zdroje zadarmo, má však rôzne obmedzenia. Naše obmedzenia sa budú odvíjať od týchto limitov. Naše požiadavky sú[16]:

- Zdrojový kód a statický obsah môže maximálne zaberať 1G HDD miesta.
- Aplikácia nemôže používať databázu, je povolené iba použitie „*Stored Data*“ (max. 1G priestoru).
- Denný limit 1G na prenosové pásmo pre vstup a výstup, max. 28 inštancií za hodinu.
- Aplikácia musí bežať na *Java 7* v „*App engine Sandboxe*“ (mnohé triedy Java 7, budú obmedzené).
- Zapisovanie a čítanie údajov z disku mimo aplikáciu je zakázané.
- Maximálny čas na spracovanie požiadavky je 60 sekúnd, maximálna veľkosť pamäte RAM je 128MB.

8.2. Ochrana súkromia vývojára

Zachytené akcie vývojára sa týkajú všetkých procesov na pracovnej stanici vývojára. A teda aj webového prehliadača. Sledujú sa aj webové adresy, ktoré vývojár navštívil. Tieto procesy môžu narúšať súkromie vývojára.

Projekt *PerConIK* sa sčasti zaoberá ochranou súkromia pri zachytávaní týchto aktivít. Napríklad ponúka vývojárovi možnosť obmedziť toto sledovanie. Náš nástroj tieto aktivity zobrazuje, preto má zmysel sa zaoberať aj ochranou súkromia a nezobraziť aktivity, ktoré by narušili toto súkromie.

Z tohto dôvodu náš nástroj bude obsahovať konfigurovateľný black-list zoznam procesov, ktoré budú ignorované pri spracovaní. Zároveň v rámci experimentu budú mená vývojárov **cenzurované**.

[15] Aplikácia bude nasadená do App Engine. App engine umožňuje prekročiť limity. Prekročenie limitov je spoplatnene. Cieľom je dodržať obmedzenia a vyhnúť sa tak plateniu.

[16] App engine limity: https://cloud.google.com/appengine/docs/quotas

9. Návrh riešenia

V nasledujúcej časti prezentujeme návrh, ktorý považujeme za najlepší pre vytvorenie vizualizácie jednotlivých akcií vývojára. Mnohé prezentované nástroje v analýze nie sú vhodné pre tento účel. To z dôvodu, že sa zameriavali na iný pohľad na vývoj softvéru.

Na základe spomenutých nástrojov je vhodné, aby sa zobrazovanie informácii riadilo rovnakou myšlienkou ako v *EPOSee*[3] a v *Forest Metaphor*[6]. Teda „Poskytni najprv prehľad, potom priblíženie s filtrami a nakoniec detaily na požiadanie“ [23]. Mnohé nástroje používateľovi ponúkali legendu pojmov a nástrojov. Použijeme tieto techniky.

Časová priamka bude v jednoduchom dvojdimenzionálnom priestore. Bude možné približovanie a pohyb v priestore, vďaka ktorému bude možné definovať hľadané obdobie, ktoré používateľa zaujíma. Toto sme spomenuli v kapitole špecifikácia.

Z úvodných požiadaviek je nutné použiť nástroj, ktorý bude prepojený na systém spravovania verzií. Niektoré spomenuté nástroje mali takéto prepojenie, ale ani jeden nástroj priamo nesledoval aktivity používateľa. V kapitole o nástrojoch na zachytenie správania používateľa sme podrobne opísali projekt PerConIK, jeho výhody (bohaté množstvo informácií, sledovanie aktivít vývojára, jednoduché použitie cez služby). Preto sa v ďalších etapách zameriame na tento projekt.

V rámci základnej analýzy sa nástroj *Gossip* javí ako výborné riešenie. Jeho celkové výhody boli už spomenuté. Ako sme spomenuli v analýze projektu PerConIK, tento nástroj s ním kolaboruje cez doplnkový modul. V rámci vizualizácie podporuje trojdimenzionálne prostredie, rôzne typy vizualizácii (grafu, schém, stromu). Mnohé ďalšie vizualizačné prvky sa dajú doňho ľahko implementovať. Nástroj *Gossip* je modulárny, ale práca s ním je náročná a tento nástroj bol vytvorený najmä pre vizualizáciu 3D štruktúr. Rozhodli sme sa pre viac interaktívnejšie riešenie.

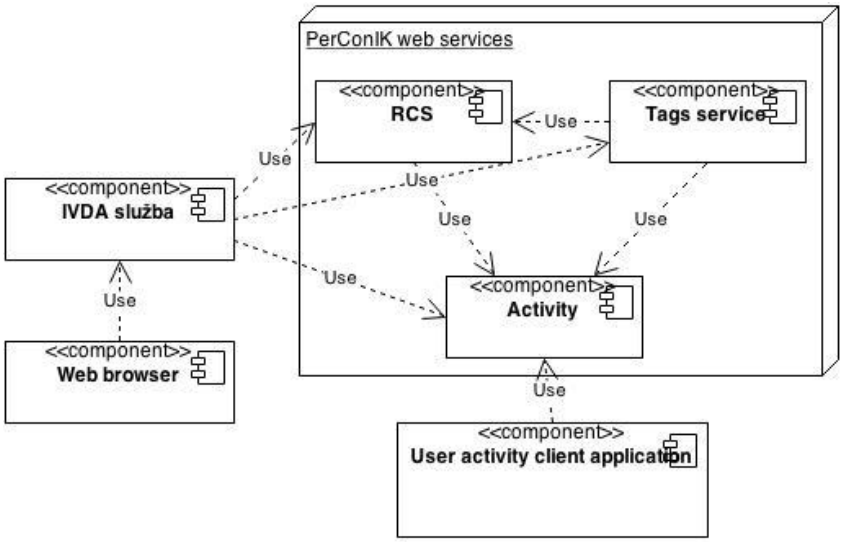
Našu vizualizáciu zabalíme do nástroja vo forme dynamickej a interaktívnej stránky (tzv. *web-based*). Vďaka tomu bude nástroj fungovať na desktope a aj na mobilných zariadeniach. Zaručí sa tak flexibilita a mobilita. Interakcia sa špeciálne využije na mobilných zariadeniach. Týmto spôsobom vizualizácia môže byť aj zabudovaná do nástrojov pre správu verzií alebo nástrojov na riadenie vývoja.

Vo forme webovej stránky bude samozrejme iba používateľské prostredie. Spracovanie udalostí a ich analýza sa bude vykonávať na pozadí (vo forme *RESTful* služby). Takto chceme podporiť ďalšie možné projekty, ktoré bežia na službách v *PerConIK* projekte. Ostatné projekty budú môcť využiť aj našu novú službu. Službu implementujeme v jazyku *Java*.

Aby mohol nástroj pracovať správne bude vyžadované internetové pripojenie na služby PerConIK.

9.1. Nová architektúra

Architektúra PerConIK služieb je orientovaná na služby. Všetky spomenuté služby v PerConIK projekte sú vzájomne prepojené. Povoľujú prístup iných služieb, za pomoci autorizácie. Preto sme navrhli vytvoriť naše riešenie v podobe novej služby a webovej stránky, ktorá používateľovi zobrazí finálnu vizualizáciu.



Obr. 15: Návrh architektúry tvorí webový prehliadač, IVDA služba a PerConIK služby. V PerConIK balíku sú 3 služby, tie už existujú. User activity client application je nástroj na zachytenie krokov vývojára.

Na používanie nástroja používateľ bude potrebovať iba webový prehliadač. Služba „*IVDA"* bude spracovávať dáta pre vizualizáciu. Jej význam je opísaný v kapitole .

9.2. IVDA služba

V PerConIK projekte sú údaje uložené na rôznych službách. Tieto údaje musia prejsť procesom spracovania, aby sme ich mohli vizualizovať v podobe informácií, ktoré už majú význam pre používateľa. Túto formu spracovania zabezpečí *RESTful* služba.

Služba bude ďalej zabezpečovať:

- prepojenie 3 nezávislých služieb a zjednodušenie prístupu k údajom
- filtráciu a cachovanie údajov (cieľom je klásť menší nápor na služby)
- párovanie udalosti procesov (cieľom je zistiť, kedy sa proces začal a kedy sa ukončil)
- klasifikácia udalostí a výpočet metrík
- budovanie aktivít z akcií vývojára
- vypracovanie údajov pre grafy, histogramy
- skrytie autentifikácie na služby a cenzúra údajov

IVDA služba má jasne definované úlohy. Jednotlivé body sú rozanalyzované v samostatných kapitolách.

9.2.1. Spracovanie udalostí v službe

Služba bude reagovať na požiadavky tým, že na základe parametrov vráti množinu aktivít. Tieto aktivity budú už filtrované a budú obsahovať údaje z ostatných služieb na základe jej prepojenia. Zároveň tieto aktivity prejdú procesom cenzúrovania a cachovania. Proces cachovania je opísaný v kapitole 10.1 a význam cenzúrovania dát v kapitole 8.2.

Najdôležitejším krokom pri spracovaní udalosti je klasifikácia aktivít a výpočet metrík pre ňu.

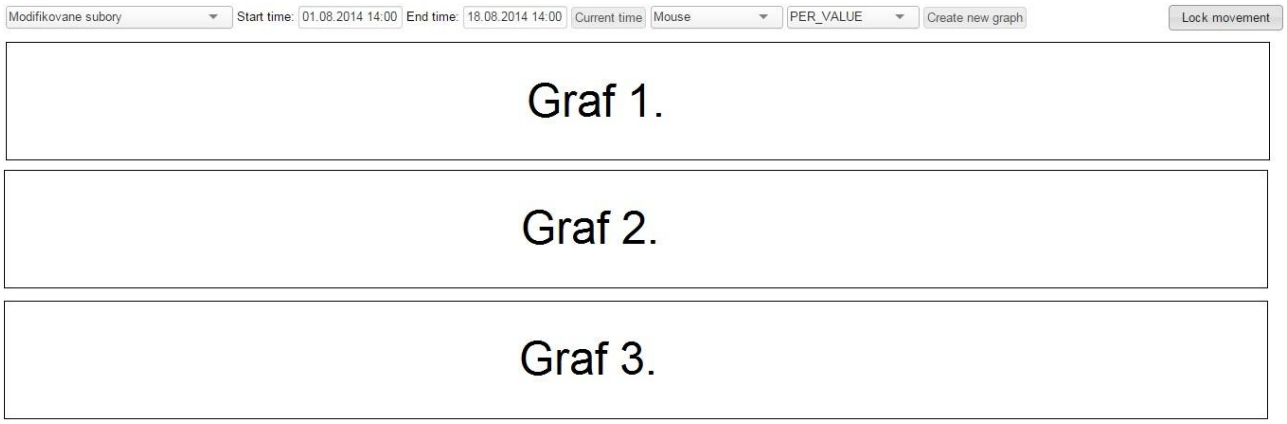
9.2.2. Klasifikácia procesov a webového cieľa vývojára

Klasifikácia udalosti nám pomáha určiť, o aký typ udalosti ide. PerConIK služba udalosti v súčasnosti určuje typ udalosti, teda zdroj jej vzniku. Mnohé udalosti ako spustenie určitého procesu nemajú ďalšie vlastnosti. Spustenie procesu však z hľadiska vývoja softvéru môžeme považovať za dôležitú akciu. Pri procese je ale potrebne klasifikovať o aký proces sa jedná, či program, ktorý vývojár použil súvisí s vývojom.

Klasifikácia udalostí ma význam aj z hľadiska samotnej vizualizácie. Pre tento pohľad je klasifikácia opísaná v kapitole " Využitie metrík". V kapitole sme použili triviálne riešenie. Cieľom tejto časti je aj poukázať, motivovať na ďalšiu prácu, na použitie lepších možností pre klasifikáciu procesov, webových cieľov.

9.3. Návrh používateľského prostredia

Vizualizáciu pomocou časovej osi zabudujeme do nástroja. V tejto kapitole prinášame náčrt používateľského prostredia, na základe požiadaviek na vizualizáciu. Používateľské prostredie bolo rozdelené do častí a tieto časti sú opísané v rôznych kapitolách.



Obr. 16: Návrh používateľského prostredia.

Základom používateľského prostredia je panel nástrojov (obr. 16: návrh používateľského prostredia. horná časť) a množina grafov. Za pomoci panelu nástrojov si vývojár môže vybrať aký graf sa vytvorí. Môže špecifikovať ďalej časove ohraničenie, vývojára a granularitu údajov.

- Množinu grafov opisujeme v kapitole 9.5
- Časové ohraničenie, môže používateľ zadať manuálne napísaním, alebo pri kliknutí na „dátum" sa otvorí interaktívny kalendár, kde používateľ môže vybrať požadovaný dátum troma klikmi.
- Vývojári sú vypísaní krycími menami
- Sledovaná granularita údajov závisí od grafu. Všeobecne podporujeme granularitu na úrovni mesiaca, dňa, hodiny, minúty a „PER_VALUE" tzv. údaje sa nebudú vôbec zoskupovať.

Množina grafov je usporiadaná pod sebou. Jednotlivé grafy je možné presúvať nad a pod seba, čím sa zmení poradie grafov. Grafy je možne presunúť do koša, čím sa grafy vymažú z vizualizácie. Kôš je na obr. 17, zobrazí sa až po interakcií s grafom. V pravej časti panela nástrojov sa nachádza tlačidlo „LOCK MOVEMENT", ktoré spôsobí to, že pri pohybe jedného grafu sa všetky grafy prispôsobia. Takto je možné jednoducho sledovať nejaké obdobie na viacerých grafoch naraz.

Obr. 17: Návrh interakcie s grafmi. Grafy je možné presúvať. Po presunutí do koša, graf sa odstráni.

9.4. Hlavná vizualizácia krokov pomocou časovej osi

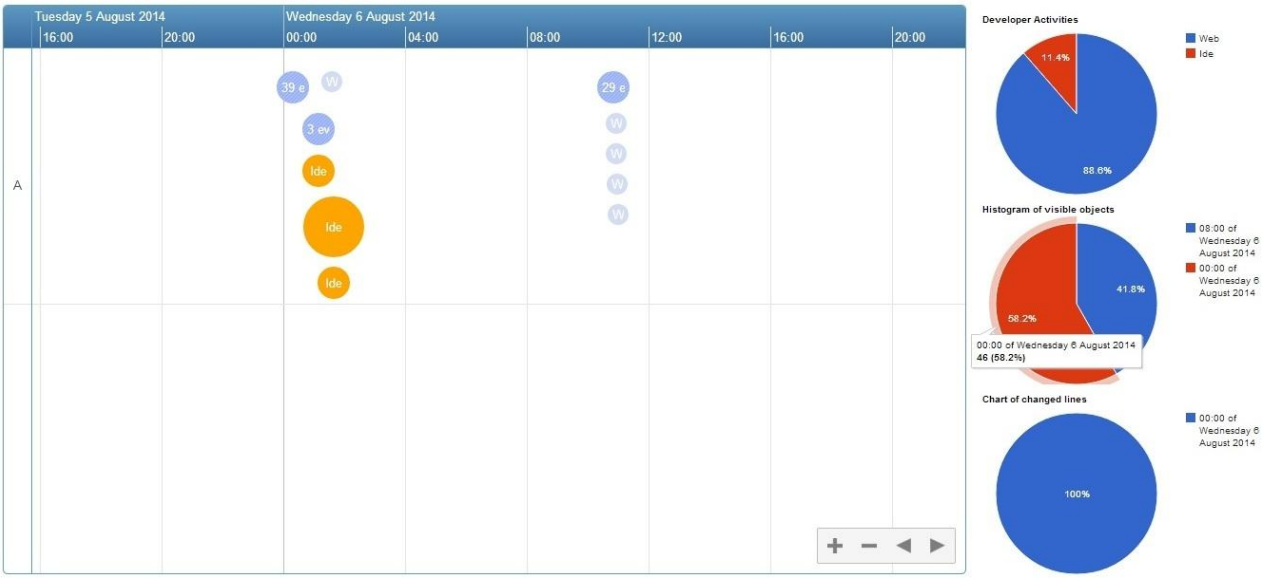
9.4.1. Požiadavky na vizualizáciu

Výsledkom záveru analýzy je potreba vytvorenia nového typu vizualizácie. Zo špecifikácie je potrebné, aby základom vizualizácie bola časová os. V tejto časti prinášame ďalšie požiadavky na vizualizáciu, opis jednotlivých časti a opis ich významu.

Základom vizualizácie musí byť anotovaná časová os, ktorá jasne definuje kedy nastala aktivita. Horizontálna dimenzia tak bude reprezentovať časový udaj. Vertikálna dimenzia je rozdelená na úseky (swim line), každý jeden pre vývojára. Týmto spôsobom chceme umožniť jednoduché porovnanie aktivít viacerých vývojárov.

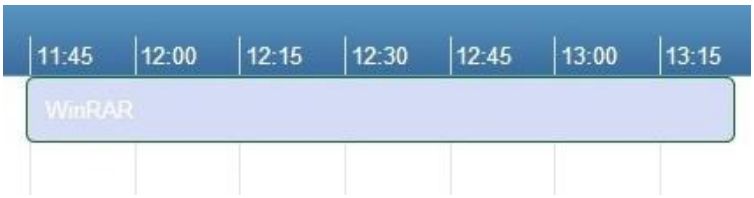
Jednotlivé aktivity sú zobrazené na časovej osi. Farba aktivity môže reprezentovať typ udalosti. Tvar aktivity môže reprezentovať vlastnosť aktivity alebo metriky, napríklad počet upravených riadkov. Aktivity sú rozdelené na vertikálnej súradnici tak, aby sa neprekrývali.

9.4.2. Samotný návrh grafov



Obr. 18: Návrh vizualizácie krokov softvérového vývojára.

V náčrte sa dominantne nachádza **Timeline** (ohraničená modrou čiarou), na pravej strane sú potom ďalšie pomocné grafy. Grafy sú opísané neskôr. Podľa časovej osi, je plocha rozdelená do stĺpcov (sú ohraničené vertikálnymi čiarami) a úseky pre vývojárov (swim line, horizontálne čiary). Na obr. 18 je zobrazený úsek pre vývojára „A“. Farba, tvar a pozície udalosti na osi sú vysvetlené v kapitole 9.4.3. Na obr. 19 zase vidno trvanie aktivity.



Obr. 19: Ukážka vizualizácie aktivity – proces Winrar.

9.4.3. Pozícia udalosti na časovej osi

Zvolili sme si kruh, ako základný tvar pre zobrazenie udalosti. Veľkosť kruhu bude reprezentovať vypočítanú metriku pre udalosť, ako je uvedené v kapitole 9.6. Dve časové udalosti sa tak môžu prekrývať v prípade, že udalosti vznikli hneď po sebe, sú dosť veľké a na časovej osi je vybraná malá granularita. V tomto prípade sa dokonca môže udalosť schovať za inú udalosť.

Všetky udalosti vnímame ako informáciu pre používateľa. Vizualizácia má intuitívne poukázať na sekvencie týchto udalosti, čím v prípade schovania alebo prekrytia jednotlivých udalosti strácame dôležitú informáciu. Preto sme sa ďalej rozhodli, že tieto udalosti nebudú zobrazené vedľa seba. V prípade možného prekrytia, sa udalosť zobrazí nižšie na časovej osi. Informácia o časovej stope udalosti, ktorá je reprezentovaná kruhom tak zostane zachovaná. Keďže X súradnica na časovej osi ostane neporušená. Stred kruhu bude reprezentovať bod na časovej osi, teda bod vzniku udalosti.

Udalosť sa posunie len o takú hodnotu, aby sa neprekrývala s inými udalosťami. Význam vzdialenosti medzi dvoma udalosťami má význam len na x-ovej súradnici a na y-ovej súradnici stráca na význame.

Obr. 20: Ukážka pozície udalostí (a posunu) na Y súradnici.

V prípade veľkého počtu udalostí na malom úseku, aplikujeme metódu zoskupovania udalosti. Táto metóda je opísaná v nasledujúcej kapitole.

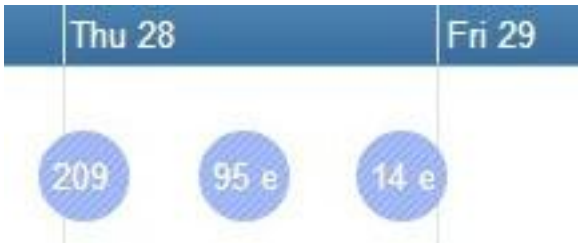
9.4.4. Zoskupovanie udalostí

Udalosti, ktoré sa zobrazujú na časovej osi, môžu pri veľkom počte znehodnotiť kvalitu vizualizácie. Tento problém bol opísaný v kapitole 7.2 a 9.4.2. V týchto kapitolách bol tento problém z časti vyriešený, pri nadmernom počte udalostí však stále môže pretrvávať.

Preto ďalej navrhujeme udalosti zoskupovať. Zoskupovať sa majú tie udalosti, ktoré sú blízko seba a je problematické ich zobraziť pod sebou. Udalosti je problematické zobraziť pod sebou v prípade, že zobrazujeme príliš veľa udalostí na súradnici y, udalosti sa nemôžu prekrývať, skladajú sa pod seba a tak sa zväčšuje potrebná plocha. Zoskupovanie sa teda môže použiť vtedy, keď sa veľké množstvo udalostí zobrazuje pod sebou. O aké veľké množstvo pôjde sa určí threshold-om, ktorý sa presne definuje po testovaní.

Výsledok zoskupenia

Skupina udalostí je zobrazená rovnakým tvarom ako iné udalosti, bude ale farebne odlíšená. V strede skupiny je vypísaný počet prvkov v nej. Skupina musí agregovať vlastnosti aktivít v nej. Napríklad veľkosť tvaru skupiny bude súčtom veľkosti tvarov v nej. Výnimkou je časový bod. Časový bod bude vypočítaný priemerom časových stôp udalostí. Preto tento bod na časovej osi, nie je presný, ale je stále informatívny.



Obr. 21: Tri skupiny udalostí na časovej osi. Prvá obsahuje 209 udalostí, druhá 95, tretia 14. Udalosti sú zoskupené pre zvolenú nízku granularitu časovej osi. Po priblížení sa udalosti rozdelia od skupiny.

9.4.5. Pomocné grafy umiestnené pri časovej osi

Pri časovej osi zobrazíme 3 koláčové grafy. Pozri obr. 18: návrh vizualizácie krokov softvérového vývojára. Tieto grafy sú na pravej strane. V tejto kapitole ich opíšeme.

A. Koláčový graf, ukazujúci pomer udalosti podľa typu, ktorý nám lepšie ukáže, ktorý typ udalosti prevažuje.
B. Koláčový graf, ukazujúci pomerom rozmiestnenie udalostí. Podľa neho je jednoduchšie určiť, kedy vzniklo veľa udalostí.
C. Koláčový graf, zobrazujúci pomer zmeny v zdrojovom kóde podľa rozmiestnenia. Podľa neho je jednoduchšie určiť, kedy došlo k najväčšej zmene.

Tieto grafy sú len nápomocné a prispôsobujú sa časovej osi, teda údaje v nich sa menia podľa zvoleného úseku na časovej osi.

9.4.6. Interakcia vo vizualizácii

Z cieľov práce je nutné, aby vizualizácia a celkovo používateľské prostredie bolo interaktívne. Preto všetky prvky na webovej stránke musia interaktívne reagovať na používateľove akcie.

Vizualizácia za pomoci časovej osi musí umožňovať pohyb a priblíženie. Takto umožníme používateľovi pozrieť si susedné udalosti. Prvky na časovej osi sa musia presúvať počas pohybu. Presun prvkov je najlepšie zobraziť animovane.

Všetky prvky na časovej osi musia jasne označovať, ktoré aktivity reprezentujú. Pri interakcii s nimi musia používateľovi poskytnúť ďalšie detaily o tejto aktivite. Prípadne ďalšie informácie o skupine udalostí. Viď obr. 22.

Používateľ by mal mať možnosť si pozrieť udalosti v určitom časovom období. Pohybom po časovej osi na jeho zvolený dátum by to trvalo príliš dlho. Preto používateľ môže zadať začiatočný a konečný dátum, ktorý ho zaujíma a systém mu ukáže aktivity pre tento interval. Po zadaní časových súradníc vizualizácia interaktívne reaguje na toto zadanie a používateľovi poskytne reakciu, že súradnice sú správne a presun na dané obdobie je v procese. Túto interakciu a presun na súradnice je opäť vhodné zobraziť **animáciou,** aby používateľ mal pocit že s niečím pracuje.[19]

Obr. 22: Interakcia pri posune kurzora nad skupinou udalostí.
Detail skupiny sa poskytne po kliknutí na skupinu.

9.5. Ďalšie podporované grafy

Časová os poskytuje používateľovi dostatok informácií o udalostiach, zachytených pri práci vývojára. Tieto informácie však nemusia byť prehľadné a niekedy nás viac zaujímajú vlastnosti udalosti ako to, kedy nastala. Pre lepšie zobrazenie **konkrétnych vlastností** poskytneme ďalšie grafy. Grafy sú v prílohe D.

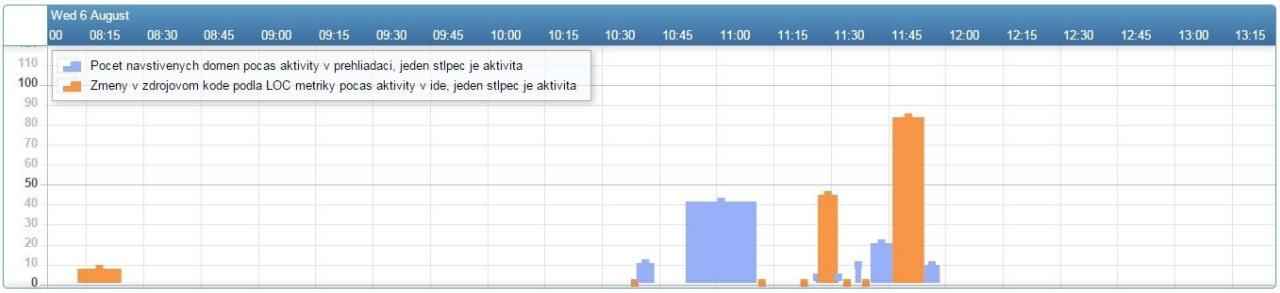
A. Dĺžka používania webových portálov – poukazuje na strávenú na portáloch. Pozri obr. 34.
B. Graf zmien pre zdrojový kód spresni kedy vývojár písal zdrojový kód a koľko napísal. Pozri obr. 35.
C. Graf špecifických vlastností pre aktivity. Pozri obr. 36.
D. Graf počtu udalosti rozdelených do dní. Pozri obr. 37.
E. Aktivita v prehliadači voči zmenám v zdrojovom kóde. Pozri obr. 38.
F. Graf zobrazujúci detail aktivít, zobrazuje kedy a koľko času strávil na aktivite. Pozri obr. 39.
G. Počet modifikácii súborov vykonanými vývojárom v rámci vybraného obdobia. Pozri obr. 40.
H. Navštívené domény pre vývojára Mouse pre zvolené obdobie. Pozri Obr. 41.
I. Vizualizácia pre detail procesov spustených na pracovnej stanici vývojára. Pozri obr. 42.
J. Aktivity vývojára za zvolený týždeň, ktoré sú zoskupené do hodiny. Pozri obr. 43.

Ako sú tieto grafy nápomocné?
Pri pozorovaní dát je dôležité zameriavať sa na ich vlastnosti. Tieto vlastnosti môžu byť rôzneho typu, napríklad môžu niesť časový údaj alebo údaj o počte. Rôzne typy údajov má zmysel zobrazovať rôznou metódou. Napríklad údaje s časovou stopou zobrazujeme na časovej osi. Údaje o počtoch zobrazujeme pomocou histogramu alebo grafu. Preto zobrazujeme viacero grafov a histogramov, zameraných na rôzne vlastnosti. V konečnom dôsledku tieto grafy nám pomáhajú odpovedať na viacero ďalších otázok súvisiacich s vývojom.

Pri grafe B, C, D, F, J, K sa používa graf, upravený histogram, je to nový vizuálny prvok, preto bližšie opíšeme v nasledujúcej kapitole.

9.5.1. Graf aktivít, podobný histogramu

Tento nový vizuálny prvok sme špeciálne vytvorili pre zobrazenie aktivít. Jedná sa o upravený histogram, kde na X-ovej osi je časová os, šírka jednotlivých stĺpcov ukazuje trvanie aktivity. Os Y poukazuje na kvantitu vybranej vlastnosti, napríklad veľkosť zmeny alebo počet navštívených portálov. Histogram je dynamický, pretože **šírka nemusí byť pre každý stĺpec rovnaká** a granularita časovej osi sa mení.



Obr. 23: Graf s rôznou šírkou stĺpcov, poukazujúc na metriku pre aktivity (výškou) ale aj na jej trvanie (šírkou).

Na obr. 23 zobrazujeme aktivity v prehliadači a aktivity v IDE prostredí, pre vývojára „Mouse“ od 8:00 do 13:20 dňa 6. Augusta. Z tohto „upraveného histogramu“ môžeme vidieť, kedy aktivity nastali. Tým, že sa pozrieme na šírku stĺpcov vidíme ako dlho trvali. Ďalej môžeme vidieť vlastnosť tejto aktivity, podľa výšky stĺpca. Napríklad pre aktivitu v prehliadači počítame počet domén, ktoré navštívil. Pre aktivitu v prostredí IDE počítame, koľko zmenil riadkov v rámci danej aktivity.

Z tohto grafu môžeme usúdiť, že o 8:15 sa približne 10 minút venoval IDE. Pri tom vykonal zmenu v zdrojovom kóde, pričom zmenil 8 riadkov. Medzi 8:25 a 10:30 vývojár pracoval s niečím iným ako je web alebo IDE[17], alebo bol off-line[18].

Vývojár navštívil najviac domén od 10:45 do 11:10. Navštívil približne 40 domén. Vývojár môže navštíviť taký počet domén, za tak krátky čas napríklad pri **riešení nejakého problému**. V čase 11:45 vývojár spravil zmenu alebo viacero zmien, kde zmenil približne 80 riadkov kódu za 10 minút. Tieto zmeny mohol vykonať napríklad komentovaním 80 riadkov alebo refaktorovaním.[19] Po 12:00 vývojár nevykonal žiadnu aktivitu. Histogram hlavne poukazuje na aktivity a ich striedanie.

---

[17] Ak vývojár pracoval na inom programe, vykonával teda inú aktivitu. Tak túto aktivitu zobrazuje graf procesov. Vďaka, ktorému používateľ môže vidieť akej inej aktivite sa vývojár venoval.

[18] Predpokladáme, že vývojár mal stále zapnutý program na sledovanie aktivít.

[19] V ďalšom grafe ukazujeme, aké súbory boli zmenené a ukazujeme aj detail zmeny. Manažér tak môže vidieť, o akú zmenu ide.

9.6. Využitie metrík zdrojového kódu

Na časovej priamke vo vizualizácii sa zobrazujú rôzne udalosti. Tieto udalosti sú vždy v správnom poradí tak, ako boli zachytené. Postupnosť týchto udalosti stráca na význame pri veľkom počte udalostí, kedy kvantita prevyšuje nad kvalitou. Preto sme sa rozhodli, že dôležité aktivity **zvýrazníme**. Na to využijeme dimenziu veľkosti udalosti (tzv. polomer kruhu zobrazujúcu danú udalosť).

Dôležitosť udalostí sa dá chápať rôznymi smermi. My na určenie dôležitosti udalosti sme vypracovali systém klasifikácie, ktorý klasifikuje udalosť podľa jej vlastnosti a následné priradí ohodnotenie. Pre udalosti z webového prehliadača sledujeme napríklad zadaný webový cieľ. Toto ohodnotenie sa deje na strane služby.

Klasifikácia udalosti nie je predmetom tejto práce. Preto v našom systéme bol použitý len triviálny mechanizmus. V implementácii používame viaceré **definované zoznamy**. Následne udalosti klasifikujeme, podľa toho do ktorého zoznamu patria. Sledovaná vlastnosť vždy patrí iba do jedného zoznamu. Napríklad pre webový cieľ zisťujeme do akého zoznamu patrí doména. Takto zistíme, či používateľ navštívil odbornú stránku. Zoznamy fungujú aj ako white listy, keď daná doména nepatrí do žiadneho zoznamu, nezobrazí sa.

Udalosť môže pridať na význame na základe klasifikácie a kontextu udalosti. Pretože udalosť úprava zdrojového kódu, kde bolo upravených viac riadkov kódu ako v inej udalosti úprava zdrojového kódu, má alebo môže mať väčší význam.

V konečnom dôsledku je udalosť ohodnotená funkciou:

$$y(x) = g(x) + v(x)$$
$$x - Udalosť$$
$$y(x) - Kvantitatívne\ ohodnotenia\ udalosti$$
$$g(x) - Ohodnotenie\ na\ základe\ klasifikácie\ typu\ udalosti$$
$$v(x) - Ohodnotenie\ na\ základe\ vnutorných\ vlastností\ udalosti\ (kontextu)$$

Implementácia funkcie $v(x)$ je len v základnej forme:

$$v(x) = text\_lines(x)$$

Funkcia $text\_lines(x)$ vracia počet neprázdnych riadkov. Ignorujú sa prázdne riadky alebo riadky, kde sú iba medzery. Funkcia na ohodnotenie vnútorných vlastností môže byť implementovaná v rôznej podobe. V našom riešení sme sa zamerali na jednoduchosť. Takáto funkcia napríklad vracia ohodnotenie s číslom 0, pre udalosti z webového prehliadača, pretože pri tomto type udalosti nedošlo k zmene žiadneho kódu.

Podľa kapitoly „Zoskupovanie udalostí“ sú vlastnosti skupiny agregované z udalosti v nej. Platí to aj pre softvérové metriky skupiny.

9.6.1. Použité white listy

Pre jednoduchú klasifikáciu procesov a webového cieľa sme použili zoznamy:

- *Work* - Zoznam procesov a zoznam webových domén, ktoré priamo súvisí s vývojom.
- *Communication*- Zoznam procesov a zoznam webových domén, ktoré slúžia na komunikácia. Napríklad proces „skype“ alebo proces pre emailového klienta.
- *Personal* - Zoznam procesov a zoznam webových domén, o ktorých si myslíme, že nesúvisia s vývojom.

Pri týchto zoznamoch je potrebné si uvedomiť, že záleží na kontexte práce vývojára. Tzv. vývojár môže pracovať, spustiť proces, ktorý nesúvisí s vývojom. Pri inom projekte však tento proces alebo webový cieľ, môže súvisieť s vývojom softvérového produktu.

Toto obmedzenie sme vyriešili tak, že white listy sú konfigurovateľné. Pre náš experiment budú tieto zoznamy nastavené na základe analýzy projektov v PerConIK úložiskách.

Vypísaná časť zo zoznamu webových domén typu „Personal“ (vľavo) a „Work“ (vpravo):

facebook.com	*github.com*
tv-program.sk	*localhost*
youtube.com	*docs.google.com*
thepianoguys.com	*stackoverflow.com*
autokelly.sk	*docs.mongodb.org*
ib.slsp.sk	*slideshare.net*

10. Riešenie kľúčového problému: veľa údajov a neporiadok vo vizualizácii

Timeline vizualizácia vyžaduje zoznam aktivít pre daného vývojára. Tieto aktivity sú sťahované z RESTful služby, nazvanej *IVDA*. Denne sa môže zachytiť niekoľko tisíc aktivít. Konkrétne množstvo dát závisí od nastavenia threshold-u v programe na zachytávanie aktivít a závisí ešte od aktivity vývojára, teda či pracoval.

Podľa kľúčového problému „ Veľa údajov“, ktorý bol opísaný v kapitole 7.1 nemá zmysel zobrazovať všetky údaje. Zároveň je problémové pre webový prehliadač zobraziť niekoľko tisíc aktivít, keďže každá aktivita je reprezentovaná niekoľkými HTML elementami.

Avšak aktivity, ktoré patria do časového obdobia, na ktoré sa používateľ pozerá ,musia byť vždy pripravené aby ich používateľ videl a mohol s nimi pracovať.

Tieto kľúčové problémy sme sa rozhodli vyriešiť niekoľkými praktikami naraz.

A. Posun a zoskupovanie blízkych aktivít (problém vyriešený v kapitole 9.4.2 a 9.4.4)
B. Cachovaním požiadaviek na našu službu
C. Cachovaním požiadaviek na PerConIK služby
D. Priradenie aktivít do chunkov

Cieľom je dosiahnuť rýchlu reakciu služby na danú požiadavku. Ďalej spracovanie a uloženie len tých aktivít, ktoré majú byť viditeľné. A tretím cieľom je cachovanie častých požiadaviek. Splnením týchto cieľov bude práca pre používateľa, ktorá bude mimoriadne interaktívna, pretože odozva používateľského prostredia bude rýchla (do 0.1ms).

10.1. Princípy cachovania

Navrhovaná vizualizácia sa dopytuje na množinu aktivít zo služby pre zvolené časové obdobie. Časové obdobie zadáva používateľ. Z predpokladu, že používateľa môže viac krát zaujímať to isté obdobie je efektívne zabudovať cachovanie požiadaviek smerom na službu. Používateľa môže zaujímať rovnaké časové obdobie, napríklad keď sa na vizualizáciu pozerá opakovane.

Predpoklad sa môže potvrdiť pri testovaní. O cachovanie požiadaviek smerom na službu sa postará samotný webový prehliadač používateľa. V prípade, že viacerých používateľov zaujíma rovnaká časová oblasť je efektívne vytvoriť cachovanie na strane našej služby. Toto riešenie už vyžaduje implementáciu.

Softvérový vývojár pri svojej prací používa rôzne nástroje. Týmto nástrojom sa venuje určité časové obdobie a tak aktivity, ktoré sa zachytili, môžu medzi sebou súvisieť. Respektíve tieto aktivity medzi sebou súvisia, kým používateľ nezačne používať iný nastroj. Tieto aktivity majú spoločný zdroj vzniku, nástroj.

Rovnakým princípom, môže vývojár pracovať nad určitým dokumentom. Zachytené aktivity pri tejto práci majú **spoločnú vlastnosť**, dokument, ktorý vývojár upravuje. Tieto aktivity vznikajú po sebe. Keďže architektúra PerConIK-u je orientovaná na služby a rôzne dáta sú uložené na rôznych službách je efektívne aplikovať cachovanie údajov smerom na tieto služby. Z hore spomenutých vlastností je vidieť, že aktivity, ktoré sú vytvorené po sebe, majú často spoločnú vlastnosť. Napríklad dokument, nástroj atď. Teda

cachovanie týchto údajov bude mimoriadne efektívne. Výsledkom bude menšia záťaž na *PerConIK* služby a rýchlejšia reakcia vizualizácie pre používateľa.

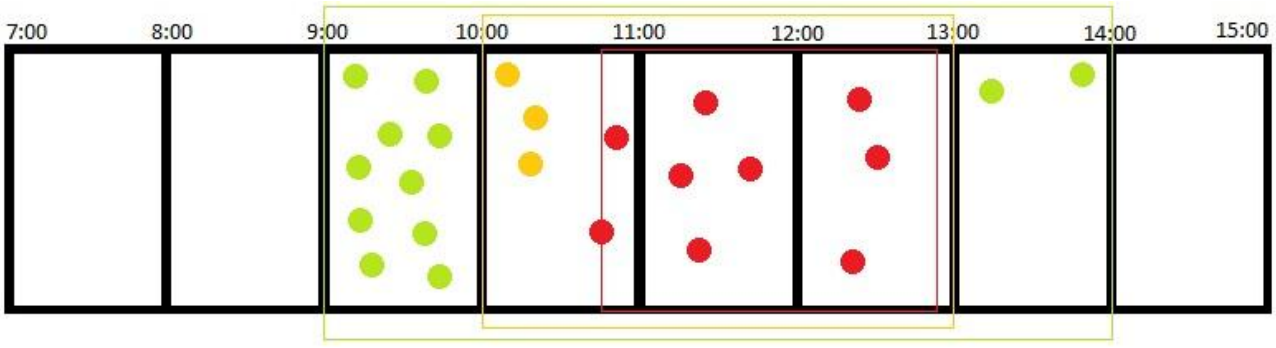
10.2. Priradenie aktivít do chunkov - dátového balíčku

Používateľ pri interakcii s vizualizáciou môže meniť časové obdobie. Môže ho meniť tým, že manuálne určí časovú hranicu alebo pohybom v časovej osi. Pohyb po časovej osi je jednoduchší a intuitívny. Týmto pohybom sa zmení časové obdobie a úlohou vizualizácie je ukázať nové aktivity.

Pred samotným zobrazením týchto aktivít, je potrebné ich načítať. Ako bolo spomenuté v úvode tejto kapitoly o kľúčových problémoch, nie je vhodné načítať všetky aktivity pre ich nadmerné množstvo. Lepšie je načítať len tie, ktoré sa majú zobraziť. Preto sa pri pohybe po časovej osi budú nové aktivity dynamicky **predčítavať. V smere pohybu** sa nové aktivity načítajú a v opačnom smere, sa staré aktivity vymažú. Dynamické načítavanie pomáha udržať vo vizualizácii len aktivity, ktoré sú potrebné. Pohybom používateľa sa dá predpokladať, ktoré aktivity bude chcieť a tak sa dajú aktivity predspracovať.

Pri tomto spôsobe načítavania aktivít vzniká problém s pôvodným cachovaním dopytov na našu službu. Pretože prehliadač cachuje dopyty na službu na základe adresy. Pri zmene parametrov v adrese dopytu (GET request) cachovanie v prehliadači neplatí.

Rozhodli sme sa preto rozdeliť časové obdobie na pevné intervaly a všetky aktivity, ktoré patria do definovaného intervalu sú súčasťou **skupiny** - chunku. Dĺžka intervalu je definovaná konštantou. Konštanta môže byť zvolená napríklad na hodinový interval. Časové obdobie tak bude rozdelené podľa hodín, napríklad 11:00 – 12:00 – 13:00 – 14:00. Princíp rozdelenia aktivít pomáha vysvetliť nasledovný obrázok.



Obr. 24: Náčrt princípu rozdelenie udalosti do chunkov. Udalosti sú rozdelené do blokov podľa hodiny. Červenou farbou sú udalosti, ktoré sú zobrazené používateľovi. Oranžovou sú pripravené udalosti, ktoré sú vo vizualizácii ale nie sú zobrazené. Zelenou sú udalosti, ktoré sa predpripravia. Červený obdĺžnik predstavuje viditeľnú časť pre používateľa na časovej osi.

Aktivity sú rozdelené na pozadí do chunk-ov. Pri pohybe používateľa sa nenačítajú len prvky, ktoré by mali byť viditeľné, ale načíta sa cely chunk, ak je tento chunk aspoň z časti viditeľný. Následne sa zobrazia prvky z tohto chunk-u, ktoré sú viditeľné. Priradením aktivít do chunk-ov, získame **pevné dopyty** na našu službu. Týmto spôsobom cachovanie v prehliadači bude opäť pracovať efektívne a aktivity na službe môžu byť uložené v celom chunku – zložke. Tým sa získa priamy prístup k celej požadovanej množine.

10.3. Záver z riešenia kľúčových problémov

Mnohé navrhované riešenia majú za účinok **rýchlu reakciu** PerConIK **služieb**, našej služby a celkovo používateľského prostredia pre používateľa. Zároveň sa vyrieši problém s veľa údajmi a neporiadkom vo vizualizácii. Efektivita navrhovaného riešenia sa overí testovaním.

11. Aproximácia riešenia alebo hypotézy

V nasledujúcich riadkoch robíme analýzu hlavnej hypotézy, opisujeme rôzne obmedzenie, pre ktoré nedokážeme na hypotézu odpovedať priamo. Na dané obmedzenie sa snažíme nájsť riešenia. Pomocou vizualizácie sa snažíme potvrdiť alebo vyvrátiť hypotézu: "*Ak vývojár často používa prehliadač (resp. portál) pri písaní svojho kódu, ako často je potom tento kód prepisovaný?"*

Pre potvrdenie tejto hypotézy sú potrebné informácie o používaní prehliadača, písaní alebo prepisovaní zdrojového kódu. Údaje týkajúce sa aktivít vývojára. Údaje naberajú na význame (teda stavajú sa pre nás informáciou) po tom čo ich spracujeme (tzv. prejdú procesom ohodnotenia). Pre lepšie pochopenie rozoberieme bližšie kľúčové pojmy našej hypotézy.

11.1. Používanie prehliadača

Pojem *„používanie prehliadača"* je pomerne široký. PerConIK zachytáva rôzne druhy udalostí (kapitola 4.6.1) vo webovom prehliadači. Tieto udalosti vznikajú pri používaní prehliadača. Každá udalosť vznikla použitím špecifickej časti prehliadača. Teda za pomoci filtrovania udalostí, dokážeme povoliť, ktoré udalosti budú zobrazené a takto špecifikujeme formu používania prehliadača. Ako typickú formu používania považujeme **navštívenie webového cieľa**.

Pravdepodobne existujú portály, ktoré keď vývojár navštívi, jeho kód je menej prepisovaný. Alebo opačne, navštívi špecifický portál a jeho kód je viac prepisovaný. Na túto hypotézu sa snažíme odpovedať čiastkovo, tým, že zobrazujeme dĺžku trvania návštevy používateľa na portáli, portál identifikujeme podľa webové cieľa.

11.2. Prepisovanie zdrojového kódu

Prepisovanie zdrojového kódu vývojára, iným vývojárom je vo vývoji softvéru častý jav. My nebude určovať autora zdrojového kódu a neskôr prepisovanie iným autorom. Budeme teda ignorovať autora a ignorovať typ zmeny (pridanie, vymazanie, úprava) nad zdrojovým kódom.

Granularita zmeny

Úpravu zdrojového kódu dobre zaznamenávajú systémy na správu verzii. Tie zaznamenajú konkrétnu verziu po commit-nutí vývojárom. Z týchto systémov je tak možné zistiť, koľko krát bol daný zdrojový kód prepísaný a o akú veľkú zmenu išlo. Verzie v týchto systémoch však závisia od commit-ovania vývojárom. Commit-ovanie vývojárom však nemusí byť časté a tak tieto informácie nemusia dávať spoľahlivú informáciu o prepisovaní.

PerConIK projekt zachytáva aj zmeny nad zdrojovým kódom, zachytáva ich na menšej granularite, respektíve akúkoľvek zmenu nad blokom kódu. Poskytuje tak **vyššiu úroveň** granularity.

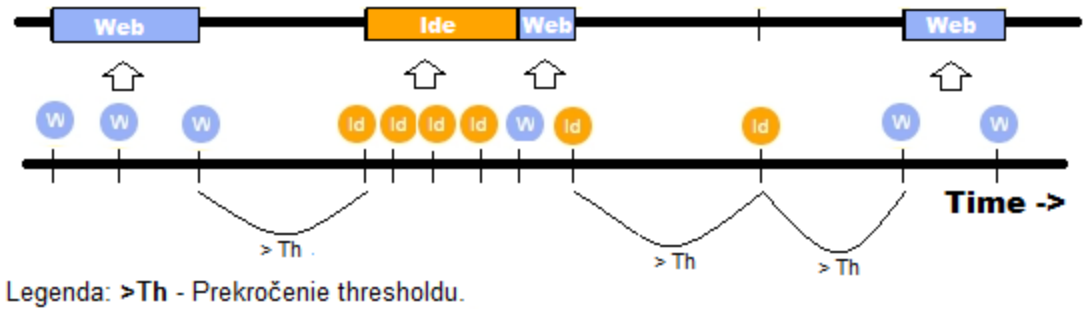
11.3. Aproximácia aktivít

PerConIK projekt poskytuje len udalosti, ktoré boli zachytené pri akciách vývojára. Postupnosť udalostí (akcií) tvorí aktivitu. Z PerConIK udalostí sa snažíme vybudovať aktivity vývojára, ktoré by nám priniesli dôležitú informáciu o jeho práci.

Udalosti sa nezachytávajú genericky, ale len pri práci s konkrétnymi nástrojmi. Takže my dokážeme zmerať aktivitu len pre tieto nástroje. Zároveň, *UACA*, program na sledovanie vývojára, nezachytáva, s akým oknom vývojár aktuálne pracuje (focus). To sa snažíme vyriešiť aproximáciou, aby sme v konečnom dôsledku získali informácie o jeho aktivitách a ktoré budú pomerne presné.

Pri aktivite vývojára totiž vzniknú dve udalosti, napríklad otvorenie a zatvorenie karty v prehliadači. Počas tohto časového úseku, sa mohol vývojár pozerať na webovú stránku ale aj nemusel. V prípade, že vývojár otvoril kartu a zatvoril ju v krátkom časovom úseku, je vysoko pravdepodobné, že vývojár čítal túto webovú stránku. Tento časový úsek bude definovaný **thresholdom** a ten bude **určovať tvorbu**[20] aktivít z udalostí. Ak udalosť rovnakého typu vznikne do daného časového intervalu, povieme, že vývojár pracoval na aktivite, ktorá je udaná typom týchto udalosti.

V prípade, že nastala udalosť nejakého typu a druhá udalosť nenastala, napríklad vývojár otvoril kartu a nezavrel ju, tak túto akciu nebudeme chápať ako aktivitu a používateľovi zobrazíme iba udalosť, nie aktivitu. Je ďalej na používateľovi, aby identifikoval, či udalosť je súčasťou nejakej aktivity.

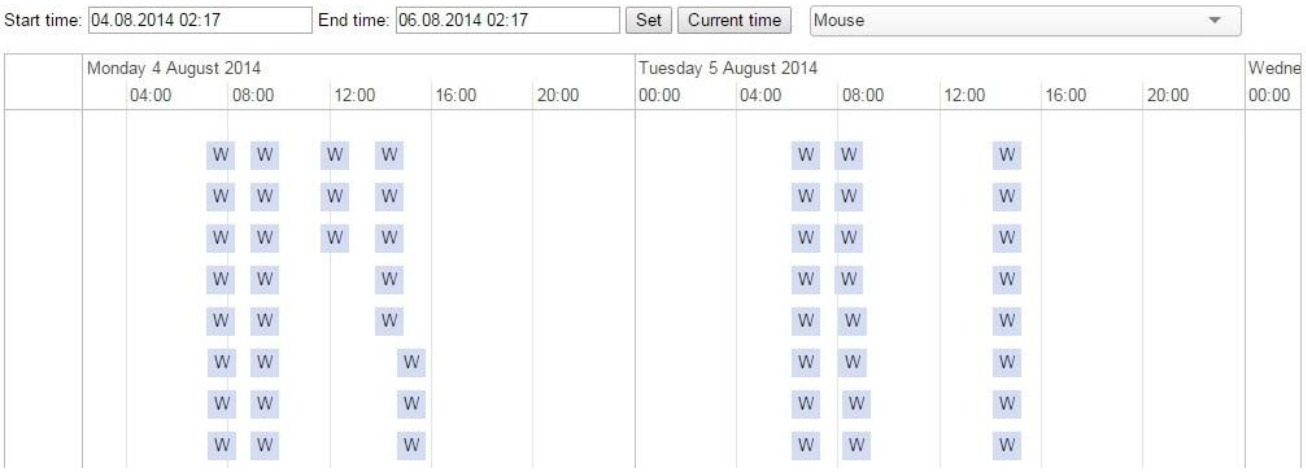


Obr. 25: Znázornená tvorba aktivít (prvá časová os) z akcií vývojára (druhá os).

[20] Tvorba aktivít nemusela byť aproximovaná v prípade, že programy na sledovanie aktivít vývojára by sledovali prepínanie aplikácií (tzv. focus pre okná na používateľskej ploche).

12. Implementácia riešenia

Vývoj softvéru prebiehal implementovaním služby podľa návrhu a prototypovaním[21] vizualizácie. Každý prototyp sa testoval[22] a vyhodnocovala sa jeho použiteľnosť. Počas vývoja nástroja sa potvrdila správnosť prístupu ku kľúčovým problémom. Konkrétne časť cachovanie požiadaviek, pri ktorej sa doladili prahové hodnoty.



Obr. 26: Ďalší prototyp zobrazuje akcie vývojára. Akcie sú zobrazené ako štvorec, nie sú zoskupené a sú tak neprehľadné. Pri takomto zobrazení nie je jasná postupnosť udalostí. Vizualizácia ešte nemá pripravený dizajn. Po tejto verzii sme vytvorili zoskupovanie, ide o dôležitú časť vizualizácie.

V návrhu používateľského prostredia došlo k zmenám farieb. Ďalej anonymné mená[23] boli nahradené **používateľsky príjemnejšími** názvami. Pre vhodnejšie používateľské prostredie na mobilných zariadeniach má webová aplikácia responzívny dizajn. V návrhu vizualizácie nastali tiež zmeny. Úseky (swim line), boli pôvodne určené pre rôznych vývojárov. Neskôr sa ukázalo, že používatelia majú stále problém so spracovaním nadmerného množstva informácií. Preto tieto úseky budú rozdeľovať akcie podľa typu. Celá vizualizácia sa tak bude venovať iba jednému vývojárovi. Používateľ si môže stále porovnať vývojárov, tým, že si vytvorí druhý graf s vizualizáciou.

Vývoj celého nášho softvéru bol zachytený v repozitári na *github-e* ako *open source*. Teda je možné pozrieť si históriu zmien. Dôvodom na toto rozhodnutie, je fakt, že naša vizualizácia môže byť súčasťou iného nástroja. Napríklad nástrojov na správu verzií, kde by popri konkrétnej verzie softvérového kódu bola zobrazená súvisiaca aktivita vývojára. Otvorením implementácie svetu sa snažíme používateľovi poskytnúť **garanciu** bezpečnosti, že údaje sú naozaj chránené a cenzurované.

---

[21] Prvý prototyp vizualizácie bol vytvorený v Gossipe. Pri tomto prototype sa ukázali nevýhody tohto nástroja. Neskôr sa vytvorilo tzv. web-based riešenie za pomoci knižnice VIS. Knižnica získala niektoré nové črty aj vďaka tejto práci, keď sme sa podelili o nápady s autormi knižnice.

[22] Prototypy boli testované interne a následne externe malou skupinou študentov a doktorandmi. Napríklad pri skupine PeWe (6 ľudí) a výskumného seminára p. Návrata (12 ľudí) a zamestnancami Gratex-u (3 ľudia).

[23] Pôvodné mená „Developer A“, „Developer B“, „Developer C“ boli nahradené „Cat“, „Dog“, „Mouse“ atď.

13. Experiment a porovnanie

Navrhnutú metódu vizualizácie sme implementovali a v tejto fáze sa ju snažíme uvádzame, ako sme ju vyhodnotili. Najprv porovnávame nástroj ako celok, teda porovnávame ho na základe závislosti od operačného systému. Je to dôležité zistiť pred samotným testovaním, aby sme mohli určiť potrebné prostredie. V našom teste Testovaním porovnáme predovšetkým použiteľnosť. Potom nástroje porovnáme podľa čŕt. V závere podľa prvkov vedomia pracovnej plochy. Tieto prvky definoval Gutwin (1996). Potrebu porovnania pomocou týchto otázok zdôvodňujeme v nasledujúcej kapitole.

Vizualizáciu je vhodné vyhodnotiť na úrovni použiteľnosti. Keďže IVDA nástroj je pomerne špecifický, preto sme hľadali nástroj alebo nástroje, ktoré sa funkčne a účelovo podobajú nášmu nástroju. Tieto nástroje opisujeme v kapitole analýza vhodného nástroja pre porovnanie.

13.1. Porovnanie vlastností IVDA a ManicTime

V nasledujúcej tabuľke sme sa pozreli na vlastnosti 2 nástrojov a porovnali ich.

Porovnávaná vlastnosť	**IVDA**	**ManicTime**
Typ nástroja	Open source	Komerčný nástroj
Cieľ nástroja	Určený na riešenie hypotéz o aktivitách vývojára, zobrazenie aktivít.	Výpis a zobrazenie aktivít. Výpočet výkazov práce.
Vyobrazenie aktivity	Zobrazuje dĺžku aktivity, kedy nastala a kto ju vykonal a metriku pre ňu.	Zobrazuje dĺžku aktivity, kedy nastala a kto ju vykonal
Maximálna granularitu času sledovania aktivít	Zobrazuje až prehľaď roka.	Maximálne jeden deň.
Minimálna granularitu času sledovania aktivít	Milisekundy	Sekundy
Zobrazuje dáta vývojárov	Všetkých	Jedného
Podpora zobrazenia prostredia	Web, IDE prostredie a spustené procesy	Sleduje väčšinu spustených prostredí .
Podpora zobrazenia kontextu aplikácii	Pre Web prostredie sa zobrazujú navštívené domény a pre IDE prostredie cesta k súborom.	Sleduje kontext pre väčšinu prostredí.
Cenzurovanie údajov	Áno	Nie
Spôsob ukladania citlivých údajov	Centrálny server (služba}	Lokálne úložisko
Podporuje grafy	10 rôznych typov grafov	Časová os aktivít a histogramy

		používania aplikácii
Hlavný spôsob vyobrazenia	Časová os aktivít	Časová os aktivít

13.2. Nástroje a ich príprava na experiment

Počas experimentu budeme overovať použiteľnosť 2 nástrojov. Tieto nástroje budú vizualizovať dáta, aktivity vývojára. Tieto dáta je potrebné nadobudnúť zachytením práce vývojárov, čo je nutné vykonať pred experimentom. Pričom v tejto kapitole opisujeme aj špecifické kroky prípravy pre konkrétne nástroje.

13.2.1. Príprava nástroja IVDA

Použité dáta

Experiment bude vykonaný na uložených dátach v PerConIK projekte. Tieto dáta sú záznamom práce vývojárov, ktorí boli súčasťou projektu. Osem vývojárov, najmä študentov a doktorantov pracovalo na rôznych projektoch jeden rok a tak dáta sú postačujúce pre experiment a porovnanie vývojárov. Tieto dáta budú chránené ochranou súkromia, spomenuté v kapitole 8.2.

Nezaznamenané aktivity

Zachytávače aktivity vývojára sledujú len určité prostredie, len určité aplikácie a len definované aktivity. Napríklad v kapitole o 4.1.1 sme opísali, že sa sleduje len jeden typ prehliadača, z dôvodu obmedzení PerConIK projektu. Vývojár tak môže vykonať aktivitu, ktorá súvisí s vývojom ale jeho aktivita nemusí byť zaznamenaná. Preto sa zameriame len na aktivity, ktoré sa sledujú.

Príprava nástroja na experiment

Aby experiment mohol byť vykonaný, našu službu a vizualizáciu sme nasadili na *Google Cloud*. Používatelia budú mať prístup k nástroju navštívením webovej adresy[24].

13.2.2. Príprava nástroja ManicTime

Nástroj bude zachytávať aktivity jedného vývojára počas 3 dní intenzívnej práce. Tieto dáta nástroj ukladá do lokálnej databázy, preto tento nástroj a jeho databáza údajov bude skopírovaná na pracovnú stanicu, ktorá bude použitá pri testovaní. Testovací používatelia budú musieť iba zapnúť nástroj, údaje budú už pripravené.

13.3. Vykonanie experimentu

Experiment pre otestovanie použiteľnosti nástroja sme navrhli a formálne zapísali do dokumentu *Formal experiment report*, ktorý je v prílohe **F**. Dokument je štruktúrovaný podľa *Common Industry Format* a obsahuje aj krátku diskusiu k doterajším výsledkom experimentu.

13.4. Porovnanie cez Gutwinové prvky

[24] Adresa nasadeného nástroja: http://ivda.eu

Gutwin (1996) opísal množinu elementov, ktoré kolaboratívni pracovníci majú sledovať počas kolaboratívnej práce v zdieľanom priestore. Pre tieto elementy existujú relevantné otázky na ktoré sa majú pýtať.[21]

Element	Relevantné otázky
Identita	Kto sa zúčastňuje na aktivite?
Poloha	Kde sú oni?
Aktivity level	Sú aktívni v pracovnom priestore?
Akcie	Čo sa chystajú robiť? Kde sa chystajú byť?
Zámery	Kde sa chystajú byť?
Zmeny	Aké zmeny vykonávajú? Aké zmeny vykonali?
Objekty	Aké objekty používajú?
Rozsahy	Čo môžu vidieť?
Schopnosti	Čo môžu robiť?
Sféra vplyvu	Kde môžu mať vplyv?
Očakávania	Čo potrebujú, aby som urobil?

V článku o využití dát (zachytených z práce vývojára) pre kolaboratívne procesy[21] použili tieto elementy na porovnanie nástrojov. Nástroje sa venovali aktivitám vývojára a autori článku tvrdia, že tieto dáta o týchto aktivitách sa dajú použiť na tieto relevantne otázky. Vizualizácia aktivít vývojára tak nadobúda nový význam pri riadení a pozorovaní kolaboratívnej práce v zdieľanom priestore.

Nástroj IVDA a ManicTime vizualizujú údaje o aktivitách vývojára, preto dokážu odpovedať na tieto prvky. Porovnali sme nástroje spomenuté v článku a tieto dva nástroje.

Názov nástroja	Identita	Poloha	Aktivity level	Akcie	Zámery	Zmeny	Objekty	Rozsahy	Schopnosti	Sféra vplyvu	Očakávania
TagSEA	x				x	x	x				
Jazz	x				x	x	x	x		x	
Expertise browser	x	x				x			x	x	
Sysiphus	x	x			x	x					

Hipikat	x										
Palantír	x		x			x	x	x			
FASTDash	x	x		x		x	x				
Team tracks			x							x	x
CASS	x				x	x	x				
Augur	x				x	x	x				
Ariadne	x							x			
Mylyn		x	x	x		x	x				x
ManicTime	**x**	**x**	**x**								
IVDA	**x**	**x**	**x**			**x**	**x**				

13.5. Výsledky porovnania nástrojov

V úvode hlavnej kapitoly sme navrhli porovnanie nástroja z viacerých hľadísk. Porovnanie nástroja *ManicTime* a *IVDA* z hľadiska závislostí od operačného systému sme zapísali v časti testovacie prostredie. Toto porovnanie bolo potrebné najmä pre určenie prostredia pre experiment.

V kapitole analýza vhodného nástroja pre porovnanie sme hľadali nástroj, ktorý by sa podobal na IVDA, ktorý by mal podobnú vizualizáciu. Krátkym porovnaním a diskusiou ku každému nástroju sme sa snažili dokázať, že náš nástroj je **špecifický** a nájsť správne porovnanie bude ťažké. Nakoniec sme našli jeden komerčný nástroj.

Pre oba nástroje sme testovali použiteľnosť, najmä úspešnosť v plnení úloh. Výsledky testu vyšli veľmi podobné pre oba nástroje. Diskusiu k tomuto testu sme uviedli na konci reportu z testu, v kapitole výsledky experimentu. V teste sme sa pozerali aj na subjektívne ohodnotenie úloh pre prácu s nástrojom, podľa Likert stupnice[1]. To nám naznačilo relatívnu **spokojnosť** s používaním nástroja. Spätná väzba a sledovanie používateľov nám prinieslo ďalšie poznatky, ktoré sa dajú použiť na vylepšenie nástroja.

Doterajšie porovnanie nástrojov nám neukázalo jasné rozdiely.

Existujú nástroje, ktoré sa zaoberajú aktivitami vývojára ale nepoužívajú rovnaký spôsob zobrazenia dát. Preto náš nástroj je špecifický. To sme dokázali v analýze. Následne sme identifikovali komerčný nástroj, ktorý bol určený za iným účelom ale podobá sa nášmu najviac. Nástroj zobrazuje najmä aktivity, ich dĺžku a kedy nastali. Náš nástroj zobrazuje aj metriku pre každú aktivitu. Preto je **vhodnejší na riešenie hypotéz** o aktivitách vývojárov ale aj na kvantitatívne vyhodnotenie práce vývojára, čo je doležité pre manažéra tímu. Posledným porovnaním cez Gutwin prvky sme dokázali, že, náš nástroj je vhodnejší pre informovanosť v kolaboratívnom softvérovom vývoji ako ManicTime.

14. Zhodnotenie a záver

14.1. Záver z vykonanej práce

Na začiatku našej práce sa nám podarilo identifikovať metódu, ktorá sa zameriava na vývoj softvéru z **nového pohľadu,** a to z pohľadu správania vývojára. Analýzou nástrojov sme dokázali, že pôvodné metódy sa zameriavali na štruktúru zdrojového kódu, vývoj softvérových metrík alebo sledovali plnenie úloh. Viac v kapitole analýza existujúcich metód vizualizácie vývoja. Vývojár pri svojej práci používa viacero nástrojov, ako prehliadač, či nástroje na komunikáciu. Analyzovali sme, či sa spomenuté metódy venujú aktivitám vývojára, prípadne aj vplyvu týchto nástrojov na jeho prácu.

Stanovili sme si, že aktivita vývojára môže byť chápaná ako základ vývoja. Preto sme v analýze snažili identifikovať nástroje, ktoré aspoň zachytávali kroky vývojára. Spomenuli sme viaceré projekty. Projekt **PerConIK** sa zdal byť najdostupnejší a zachytával kroky na nízkej granularite. Preto sme tomuto projektu venovali viacero kapitol, kde sme analyzovali spôsob akým dáta zachytávajú, dátový model údajov s cieľom čo najlepšie pochopiť štruktúru dát, a aj celkovú architektúru služieb pre integráciu nášho riešenia. Projekt mal potrebné dáta ale neriešil vizualizáciu údajov.

V špecifikácii sme na základe doterajších nástrojov a tvrdení vytvorili množinu požiadaviek na aplikáciu. Hlavným cieľom bolo vytvoriť vizualizáciu s **časovou osou**, kde sa zobrazia aktivity vývojára, podobne ako u Gantt diagramu. Nástroj sme špecifikovali vo forme webovej aplikácie. Pre aplikáciu sme ďalej špecifikovali používateľské prostredie, softvérové požiadavky v rámci Google cloudu a ochranu súkromia vývojára.

Navrhnutá vizualizácia „Timeline“, za pomoci časovej osi, ukázala svoj význam, respektíve ukázala, že časový údaj o akciách alebo aktivitách vývojára je dôležitá informácia. Popri časovému údaju sme zobrazovali aj **metriku**, pre aktivity alebo akcie. Pre výpočet metriky sme museli klasifikovať akcie a procesy, ktoré sa zachytia na pracovnej stanici používateľa. Spôsob klasifikácie je v kapitole klasifikácia procesov a webového cieľa vývojára.

Pre efektívne zobrazenie informácie a dodržanie Millerovho pravidla[16] stanovili sme pozícia udalosti na časovej osi a zoskupovanie udalostí. K tejto základnej vizualizácii sme pridali špeciálny graf aktivít, podobný histogramu. Neskôr ďalších **desať** zaujímavých grafov a histogramov. Každý z týchto grafov je opísaný v tejto práci, v kapitole 9.5. Grafy majú názov, opis a obsahujú opisy osí. Nástroj umožňuje zvoliť si vývojára, časové obdobie a aj porovnanie vývojárov. Všetky dôležité časti sú preto pripravené. Cieľom bolo vytvoriť **generickú** vizualizáciu, čo sa nám sčasti podarilo.[25]

Vývoj softvéru prebiehal implementovaním služby podľa návrhu a **prototypovaním**[26] vizualizácie. Každý prototyp sa testoval[27] a vyhodnocovala sa jeho použiteľnosť. Počas vývoja nástroja sa objavili problémy

[25] Grafy sú vytvárané ako komponenty. Vytvorenie ďalších grafov je teda jednoduché.

[26] Prvý prototyp vizualizácie bol vytvorený v Gossipe. Pri tomto prototype sa ukázali nevýhody tohto nástroja. Neskôr sa vytvorilo tzv. web-based riešenie za pomoci knižnice VIS. Knižnica získala niektoré nové črty aj vďaka tejto práci, keď sme sa podelili o nápady s autormi knižnice.

s veľa údajmi a neporiadok vo vizualizácii. Problémy s veľa údajmi sme vyriešili použitím 3 typov cachovania, rýchlou serializáciou, chunk-ovanie udalostí. Neporiadok vo vizualizácii sme tiež riešili. Tieto kľúčové problémy sú opísané v kapitole 10.

Naša vizualizácia podporuje len čiastočne hypotézy, ktoré sme si definovali ako ciele na začiatku. Napriek tomu dosiahnuté výsledky sú postačujúce a vďaka generickej vizualizácii sme dokázali podporiť iné zaujímavé hypotézy. Ukázali sme, že tento nový pohľad na vývoj softvéru je dôležitý a má potenciál sa vyvíjať ďalej. Cieľom bolo vytvoriť nástroj, vizualizáciu, ktorá by odpovedala na hypotézy, na ktoré iné nástroje nedokázali odpovedať. Graf na obr. 38 neukazuje žiadnu priamu odpoveď. To môže byť zapríčinené nedostatkom dát (malý počet bodov, reprezentujúcich pomer medzi používaním prehliadača a následnému prepísaniu kódu). Grafy na obr. 34, obr. 35, obr. 36 **riešia inú hypotézu**, tie sú už prehľadné a poskytujú kvalitný pohľad na aktivity vývojára.

V poslednej časti sme porovnávali nástroj . Pre toto porovnanie sme museli vykonať ďalšiu analýzu nástrojov, tak aby sme našli čo najpodobnejší nástroj a porovnanie tak bolo korektné. Nástroj sme porovnali 3 spôsobmi. Výsledky porovnania pre každý spôsob je v príslušnej kapitole daného porovnania. Celkový záver je v kapitole výsledky porovnania nástrojov. Z tohto záveru môžeme vybrať, že náš nástroj je naozaj špecifický, používatelia sú spokojní s používaním a pre určovanie hypotéz je efektívnejší ako ostatné nástroje.

V ďalšej práci navrhujeme zamerať sa na zachytávanie údajov ešte na nižšej úrovni. Napríklad, aby bolo možné zachytiť každú zmenu kódu. Analýzou toku týchto zmien, by sme získali ešte zaujímavejšie informácie o trendoch vývojára pri písaní zdrojového kódu. Rovnako sledovanie používania prehliadača, má pri vývoji softvéru význam, a tak odporúčame sledovať ďalšie údaje, napríklad, ktoré časti stránky vývojár číta. Treba si ďalej uvedomiť, že pri sledovaní práce vývojára je dôležitá otázka ochrany súkromia. V našej práci sme sa venovali aj tejto časti a ďalším prácam alebo nástrojom odporúčame vhodnosť **cenzúrovania** údajov.

Náš nástroj prezentujeme ako *RESTful* službu a webovú stránku. Služba je **znovu použiteľná** pre ďalšie *PerConIK* projekty. Vizualizácia zase môže byť súčasťou väčšieho nástroja. Napríklad systému na spravovanie verzií alebo nástroja na riadenie projektu. Tieto ďalšie nástroje sú častokrát tiež založené vo forme webovej stránky. Takto jeho používanie môže byť ďalej zjednodušené.

Nástroj môže byť použitý pre manažment alebo odborných pracovníkov. Používateľ vďaka vizualizácií dokáže identifikovať niektoré **trendy a artefakty** sledovaných vývojárov, čím môže odstrániť ďalšie náklady. Napríklad či zablokovanie určitého portálu nezvýši produktivitu alebo či vývojára neruší nadmerná komunikácia.

---

[27] Prototypy boli testované interne a následne externe malou skupinou študentov a doktorandmi. Napríklad pri skupine PeWe (6 ľudí), výskumného seminára p. Návrata (12 ľudí) a zamestnancami Gratex-u (3 ľudia).

14.2. Limity a obmedzenia

Súčasťou komplexného riešenia je používateľské prostredie v podobe interaktívnej webovej stránky, používateľský manuál a služba na spracovanie udalostí. Súčasťou nie sú pôvodné *PerConIK* služby a ani nástroj na zachytávanie udalostí. Na nasadenie riešenia do pracovného prostredia sú vyžadované všetky časti.[28]

[28] Nástroje je na stiahnutie na adrese: http://perconik.fiit.stuba.sk/UserActivity

Bibliografické odkazy

[1] ALGOZZINE, B. Likert Rating Scales. In *Development* . 1932. s. 2. .

[2] AMOR, J. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 ...* [online]. 2006. s. 5–8. Dostupné na internete: <http://dl.acm.org/citation.cfm?id=1139116>.

[3] BURCH, M. et al. EPOSee: A Tool For Visualizing Software Evolution. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis* [online]. 2005. s. 1–2. Dostupné na internete: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1684322>.

[4] COLLBERG, C. et al. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization - SoftVis '03* [online]. New York, New York, USA: ACM Press, 2003. s. 77. [cit. 2013-12-07]. Dostupné na internete: <http://dl.acm.org/citation.cfm?id=774833.774844>.

[5] ELLIS, G. - DIX, A. A Taxonomy of Clutter Reduction for Information Visualisation. In *IEEE Transactions on Visualization and Computer Graphics* [online]. 2007. Vol. 13, no. 6, s. 1216–1223. Dostupné na internete: <http://eprints.lancs.ac.uk/12933/>.

[6] ERRA, U. et al. Visualizing the Evolution of Software Systems Using the Forest Metaphor. In *2012 16th International Conference on Information Visualisation* [online]. 2012. s. 87–92. [cit. 2014-03-24]. . Dostupné na internete: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6295797>.

[7] GALL, H. - LANZA, M. Software evolution: analysis and visualization. In *Proceedings of the 28th international conference on Software engineering* [online]. 2006. s. 1055–1056. [cit. 2013-12-07]. . Dostupné na internete: <http://dl.acm.org/citation.cfm?id=1134502>.

[8] GONZALEZ, A. et al. Combined visualization of structural and metric information for software evolution analysis. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops* [online]. 2009. s. 25–29. [cit. 2013-12-07]. . Dostupné na internete: <http://dl.acm.org/citation.cfm?id=1595815>.

[9] GRATEX Gossip. In [online]. [cit. 2014-11-19]. Dostupné na internete: <http://perconik.fiit.stuba.sk/Methods/UserModelingandPersonalization/Gossip.aspx>.

[10] GRATEX PerConIK. In [online]. [cit. 2013-12-07]. Dostupné na internete: <http://perconik.fiit.stuba.sk/>.

[11] HANJALIC, A. ClonEvol: Visualizing software evolution with code clones. In *Software Visualization (VISSOFT), 2013 First IEEE ...* [online]. 2013. s. 1–4. [cit. 2014-03-24]. . Dostupné na internete: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6650525>.

[12] HINDLE, A. et al. YARN: Animating Software Evolution. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis* [online]. 2007. s. 129–136. Dostupné na internete: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4290711>.

[13] JERMAKOVICS, A. et al. Visualizing software evolution with lagrein. In *Companion to the 23rd ACM SIGPLAN conference on Object oriented programming systems languages and applications - OOPSLA Companion '08* [online]. 2008. s. 749. Dostupné na internete: <http://portal.acm.org/citation.cfm?doid=1449814.1449843>.

[14] KONÔPKA, M. Software Metrics Based on Developer ' s Activity and Context of Software Development. In . 2013. s. 101–102. .

[15] MALETIC, J.I. et al. A task oriented view of software visualization. In *Proceedings First International Workshop on Visualizing Software for Understanding and Analysis* . 2002. .

[16] MILLER, G.A. The Magical Number Seven. In SALA, S. DELLAEd. *The Psychological Review* [online]. 1956. Dostupné na internete: <http://www.musanim.com/miller1956/>.

[17] MINELLI, R. et al. Visualizing Developer Interactions. In *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on* . 2014. s. 147–156. .

[18] MINELLI, R. - LANZA, M. Visualizing the workflow of developers. In *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013* . 2013. .

[19] NIELSEN, J. *Usability Engineering* [online]. . Ed. J Nielsen. [s.l.]: Morgan Kaufmann, 1993. ISBN 0125184069.

[20] NOVAIS, R. - LIMA, C. An interactive differential and temporal approach to visually analyze software evolution. In *Visualizing Software for Understanding and Analysis* [online]. 2011. s. 1–4. [cit. 2014-03-24]. . Dostupné na internete: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6069460>.

[21] OMORONYIA, I. et al. Using developer activity data to enhance awareness during collaborative software development. In *Computer Supported Cooperative Work* [online]. 2009. Vol. 18, no. 5-6, s. 509–558. Dostupné na internete: <http://link.springer.com/article/10.1007%2Fs10606-009-9104-0>.

[22] ROEHM, T. - MAALEJ, W. Automatically detecting developer activities and problems in software development work. In *Proceedings - International Conference on Software Engineering* [online]. 2012. s. 1261–1264. Dostupné na internete: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6227104>.

[23] SHNEIDERMAN, B. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages* . 1996. .

[24] SHNEIDERMAN, B. - ARIS, A. Network visualization by semantic substrates. In *IEEE Transactions on Visualization and Computer Graphics* [online]. 2006. Vol. 12, no. 5, s. 733–740. Dostupné na internete: <http://www.ncbi.nlm.nih.gov/pubmed/17080794>.

[25] WILSON, J.M. Gantt charts: A centenary appreciation. In *European Journal of Operational Research* . 2003. s. 430–437. .

[26] Time management software - ManicTime. In [online]. [cit. 2015-03-14]. Dostupné na internete: <http://www.manictime.com/>.

Slovník pojmov

Aktivita	V kontexte tejto témy aktivita často predstavuje aktivitu vývojára, čo chápeme ako sekvenciu udalosti.
Cachovanie	Rýchla vyrovnávacia pamäť zvyšujúca výkonnosť počítačového systému.
Commit	Termín sa používa pri systéme na spravovanie verzií, keď vývojár sa ukladá ďalšiu verziu do systému.
Chunk	Skupina objektov. Všetky skupiny majú rovnakú vlastnosť, napríklad rovnaký časový interval, pre prvky v skupine.
Repozitár	Schránka, úschovňa, skladisko.
Revision control system	Softvér alebo systém na správu verzií.
Softvérová metrika	Softvérová metrika je spôsob merania určitej softvérovej veličiny alebo špecifikácie. Metriky poznáme tradičné, objektovo-orientované a procesné.
Softvérový element	Softvérový element predstavuje projekt, balík, súbor, trieda, metóda, fragment kódu a pod.
Swim-line	Je vizuálny element, ktorý vizuálne zoskupuje a oddeľuje prvky do kategórií. Je používaný najmä vo vývojových diagramoch.
Tagy	Elektronické štítky.
Threshold	Prahová hodnota.
White-list	Termín používaný pre zoznam entít, ktorým sú poskytované zvláštne privilégia.

Zoznam skratiek

IDE	Integrované vývojárske prostredie.
LOC	Lines of code je druh softvérovej metriky.
PerConIK	PerConIK je skratka „Personalized Conveying of Information and Knowledge“. Je to výskumný projekt, ktorý sa zaoberá širokou oblasťou. Pre nás je zaujímavý tým, že zbiera údaje o vývojárovi. Podrobne je opísaný je v kapitole 4.1.
RCS	Revision control system – Systém na správu verzií. Ide o jednu zo služieb PerConIK projektu.
REST	Representational State Transfer – je architektúra rozhraní, navrhnutá pre distribuované prostredie
TFS	Skratka predstavuje „Team Foundation Server“, čo je systém na spravovanie verzií.
UACA	PerConIK nástroj na zachytávanie krokov vývojára.
URI	Uniform resource identifier - je kompaktný reťazec znakov, používaný na identifikáciu alebo pomenovanie zdroja

Zoznam použitých obrázkov

Príloha A - Technická dokumentácia

V prílohe sú uvedené všetky základné dokumentácie od nainštalovania až k používaniu nástroja. V príloha:

A. Štruktúra dátového disku
    - Návod na nainštalovanie
    - Návod na jednoduché spustenie už nasadeného nastroja
B. Používateľská príručka
C. Diagramy pre fázu Návrh
    - Prípady použitia
    - Diagram balíkov
    - Diagram tried
    - Diagram toku údajov
D. Obrázková príloha
    - Obsahuje dátové modely služieb, prototypy a logo nástroja

Každá kapitola opisuje systém z rôzneho aspektu.

Algoritmus jednoduchým pohľadom

Princípom služby je spracovávanie požiadaviek, sťahovanie údajov zo služieb *PerConIK-u*. Tieto údaje sú sťahované na začiatku požiadavky[29] a sú spracovávané v prúde údajov. Takže je to rýchly proces. Stiahnuté údaje sú ďalej poskytované metóde spracovania. Metóda spracovania určuje akým spôsobom sa údaje spracujú. Napríklad sa môžu spracovať do aktivity a ohodnotiť, alebo sa môžu poslať do výpočtu histogramu. Metódu spracovania určuje klient vo svojej požiadavke.

Po spracovaní údajov sa výsledok posiela klientovi. Výsledok sa spracuje na strane klienta (napríklad sa nastaví časová zóna) a neskôr sa tieto údaje zobrazia. Používateľ riadi vizualizáciu cez časovú os, kde môže nastaviť aké informácie uvidí. Systém na tieto požiadavky reaguje a automaticky ich spracuje. Ak je to nutné a údaje nie sú cachované tak sa údaje opäť spracujú na strane služby.

Na službu sú posielané viaceré požiadavky naraz, žiadajú sa asynchrónne.

Čo je v nástroji konfigurovateľné?

Aplikácia je konfigurovateľná za pomoci XML súboru, ktorý je načítaný pri spustení. Súbor obsahuje krycie mena vývojárov, threshold hodnoty, nastavenie metrík, prihlasovacie údaje a adresy na služby PerConIK, všetky nastavenia časov pre cache systém.

White listy je možné jednoducho upraviť, sú uložene v textovom súbore. Kde každý záznam je na jednom riadku. Sú uložené v priečinku „*catalog*“ a ďalej sa delia na:

- Web – Obsahuje personal, work, unknown zoznam.
- Processes – Obsahuje banned, communication, music, nodeveloper zoznam procesov.

[29] Niektoré operácie spúšťajú agregácie nad údajmi. Operácie agregácie dát boli náročné na zdroje. Pre zvýšenie odozvy systému tak tieto údaje boli predspracované. Rovnako nie sú zobrazené aktivity za posledných 12 hodín, keďže údaje vývojára na strane PerConIK služby sú spracované až po určitom čaše.

Príloha B – Používateľská príručka

Naše riešenie je sprístupnené vo forme webovej stránky. Jeho používanie je preto zjednodušené a mobilne. Pre používanie nástroja je potrebné pristúpiť na adresu nástroja. Nástroj neobsahuje pod stránky a tak všetky informácie sú priamo na hlavnej stránke.

Obr. 27: Panel nástrojov.

Ovládacie prvky panelu nástrojov

A. Výber črty – *Graf, ktorý sa vytvorí*
B. Start time – *Začiatok intervalu, pre ktorý sa zobrazia udalosti na časovej osi*
C. End time – *Koniec intervalu, pre ktorý sa zobrazia udalosti na časovej osi*
D. Current time – *Tlačidlo na nastavenie intervalu*
E. Pick a developer - *Vyber vývojára, pre ktorého sa ukážu výsledky*
F. Granularita – *Výber úrovne granularitu*
G. Create new graph – *Tlačidlo na vytvorenie nového grafu*
H. Lock movement – *Pohyb na jednom grafe spraví pohyb na všetkých grafoch*

Po nastavení požadovaných vlastností je potrebné kliknúť na tlačidlo a vytvoriť nový graf. Nový graf sa pridá do zoznamu grafov na koniec. Presne ako je uvedené v návrh používateľského prostredia. Graf je možné následne presúvať, napríklad presunutím do koša.

Ovládacie prvky na časovej osi - Timeline

1. Stlačením ľavého tlačidla na myši a následné pohybom myšou, alebo pomocou šípka na klávesnici, potiahnutím prsta (len mobilné zaradenia) .
   Je možné ovládať a meniť interval časovej osi.
2. Pohybom kolieska na myši, alebo stlačením pomocou 2 prstov a následne ťahaním (len mobilné zaradenia).
   Je možné ovládať granularitu času a meniť interval časovej osi.
3. Kliknutím na udalosť .
   Je možné poskytnúť ďalší detail o udalosti. Napríklad pre webovú udalosť sa zobrazí adresa webového cieľa, alebo pre udalosť vo vývojovom prostredí sa zobrazia informácie o zmene v kóde, veľkosti zmeny atď.

Ostatné ovládacie prvky

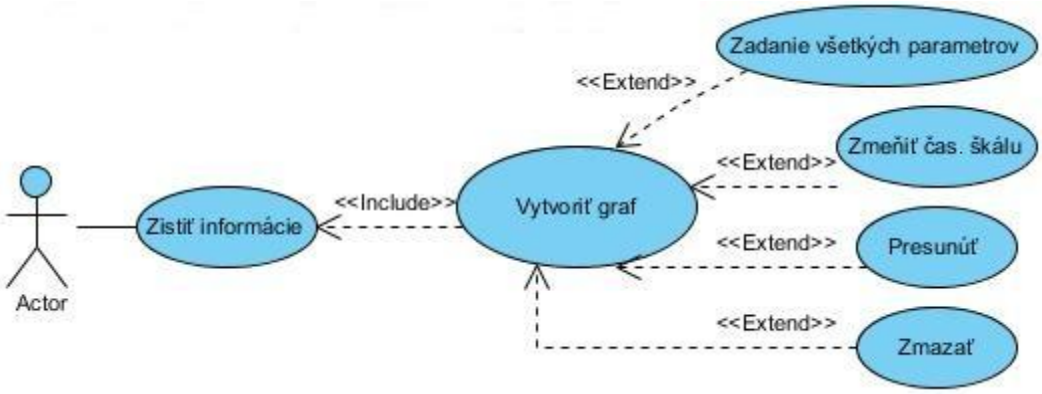
Všetky ostatné prvky na stránke automaticky reagujú na zmenu intervalu časovej osi. Svoje údaje tak automaticky prispôsobia. Ich ďalšie ovládanie je rovnaké ako pri časovej osi (platí pre histogramy, scatter graf sa môže iba približovať).

Porovnanie dvoch a viacerých vývojárov

Udalosti, aktivity a ďalšie informácie o vývojároch je možné porovnať vytvorením grafu pre každého vývojára. Grafy sa zobrazia pod seba. Je ich možné presunúť potiahnutím.

Príloha C – Diagramy pre fázu Návrh

Prípady použitia



Obr. 28: Diagram prípadu použitia.

Typický používateľ je napríklad manažér tímu alebo vedecký pracovník.

UC-01 Vytvorenie grafu

Používateľ chce zistiť informácie.

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa vygeneroval graf s ktorým môže ďalej pracovať.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si otvorí vizualizáciu.
2. Používateľ si vyberie sledovanú vlastnosť, časové obdobie, vývojára, granularitu.
3. Používateľ klikne na vytvorenie grafu.
4. Používateľ sa pozrie pod panel nástrojov, kde sa mu zjaví požadovaný graf.

Poznámka: Používateľ môže s grafom hýbať alebo ho môže zmazať.

UC-02 Rozloženie aktivít vývojárov

Používateľ tímu je zvedavý na časové rozloženie práce vývojárov, za posledný týždeň . Píšu vývojári zdrojový kód štandardne od 8:00 až do 16:00? Alebo to má posunuté?

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa zobrazil graf.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si vytvorí graf zmien pre zdrojový kód, podľa UC-1.
2. Používateľ si vytvorí graf, počet navštívení portálu daným vývojárov, podľa UC-1.

3. Používateľ si pozrie grafy.

Poznámka: Vytvorené graf mu spresní, kedy vývojár písal zdrojový kód a v akom množstve (LOC metrika). Graf sa vytvorí pre každého vývojára.

UC-03 Prezeranie trendov vývojára / Sledovanie jeho výkonu.

Používateľ tímu je zvedavý, v ktorej časti dňa je jeho vývojár najefektívnejší. Písal vývojár zdrojový kód viac ráno alebo večer.

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa zobrazil graf.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si vytvorí graf zobrazujúci zoskupené vlastnosti aktivity, podľa UC-1 (Vyberie si granularitu zoskupenia na hodinu).
2. Používateľ si pozrie graf.
3. Používateľ uvidí rozloženie aktivít za celý týždeň.
4. Používateľ si porovná dni v týždni a zistí zaujímavé trendy správania jeho vývojára.

UC-04 Vplyv používania prehliadača na prácu vývojára

Používateľ je zvedavý, či používanie prehliadača ovplyvňuje prácu vývojára. Ak áno, ktorý portál má na tom najväčší podiel. Používateľa zaujíma konkrétna hodina. (Túto hodinu identifikoval v UC-02, kde našiel nízku aktivitu vývojára. Alebo používateľ tímu je zvedavý, či vývojár navštevuje súkromné stránky. Či ich navštevuje často a ako dlho sa im venuje.

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa zobrazili grafy.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si vytvorí 3 grafy (DĺžkaPoužívania webových portálov, Aktivita v prehliadači voči zmenám v zdrojovom kóde) , podľa UC-1.
2. Používateľ si pozrie graf. (Aktivitu v prehliadači voči zmenám v zdrojovom kóde mu ukáže pomer a teda aj vplyv.
3. Používateľ si pozrie druhý graf. (DĺžkaPoužívania webových portálov, mu ukáže portál, ktorý ma na to najväčší vplyv)

UC-05 Problémové projekty alebo zdrojové súbory

Používateľ kontroluje prácu nového vývojára a je zvedavý, či vývojár nemá problémy s novým projektom na ktorý bol pridelený. Používateľ chce ďalej zistiť, ktorý konkrétny súbor bol ním najčastejšie upravovaný.

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa zobrazili grafy.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si vytvorí 2 grafy (Graf zobrazujúci detail aktivít, Počet modifikácii súborov daným vývojárom v rámci vybraného obdobia), podľa UC-1.
2. Používateľ si pozrie graf. (Zobrazujúci detail aktivít, ktorý zobrazuje kedy a ako dlho strávil na aktivite, takto si môže pozrieť akým súborom sa ako dlho venoval, ďalej akým portálom sa ako dlho venoval)
3. Používateľ si pozrie druhý graf. (najčastejšie upravované súbory)

UC-06 Poukázanie na aktivity, ktoré nesúvisia s vývojom

Používateľ tímu je zvedavý, či existujú aktivity vývojára, ktoré nesúvisia s vývojom. Ďalej chce zistiť, kedy takéto aktivity vznikli, ako dlho trvali a čo sú to za aktivity.

Vstupné podmienky:

- Používateľ navštívil webový nástroj

Výstupné podmienky:

- Používateľovi sa zobrazili grafy.

Účastníci: Používateľ

Hlavný tok:

1. Používateľ si vytvorí 2 grafy (Graf zobrazujúci detail aktivít, zobrazuje kedy a aký čas strávil na aktivite, Vizualizácia pre detail procesov spustených na pracovnej stanici vývojára), podľa UC-1.
2. Používateľ si pozrie graf.

Poznámka: Graf mu ukáže všetky aktivity vývojára a procesy, ktoré spustil na svojej stanici. Aktivity, ktoré nesúvisia s vývojom sú vizuálne odlíšené na časovej osi. Prípady použitia sme overili pri testovaní.

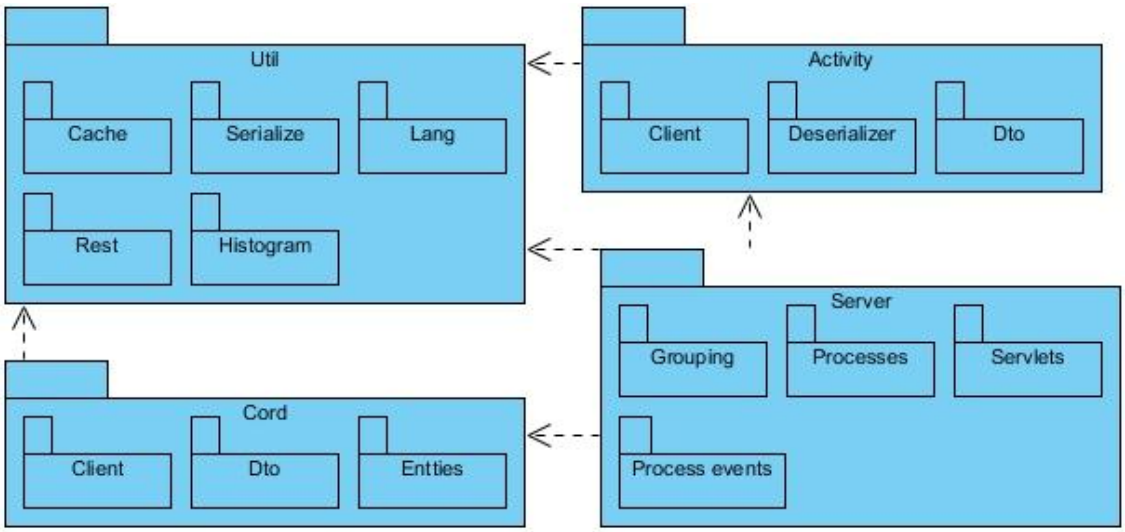
Diagramy štruktúry

Štruktúru systému rozdelíme na štruktúru služby a štruktúru webovej časti. Štruktúra služby bude pomerne komplikovanejšia, preto bude riešená objektovo orientovaným pohľadom. Architektúra programu je teda organizovaná do balíkov a tried. V ďalších podkapitolách prinášame diagram balíkov a diagramy tried pre jednotlivé kapitoly s cieľom opísať navrhovanú štruktúru.

Diagram balíkov

Základnými balíkmi navrhovaného systému, na strane serveru sú:

- *Activity* – Bude združovať zdrojový kód pre Activity službu. Obsahuje ďalšie balíky:
  - *client, deserializer, dto*
- *Cord* – Bude združovať zdrojový kód pre klienta na RCS službu.
  - *client, dto, entities*
- *Server* – V tomto balíku sa bude nachádzať zdrojový kód pre spracovanie požiadaviek na server. Spracovanie aktivít získaných zo služieb. Spracovanie procesov, zoskupovanie aktivít, výpočet histogramov. Ďalej výpočet metrík nad požadovanými aktivitami. Obsahuje pod balíky:
  - *fileStats, grouping, process, processesevents, servlets*
- *Util* – Bude združovať zdrojový kód pre prácu s dátumami, reťazcami a ďalšie pomocné triedy.



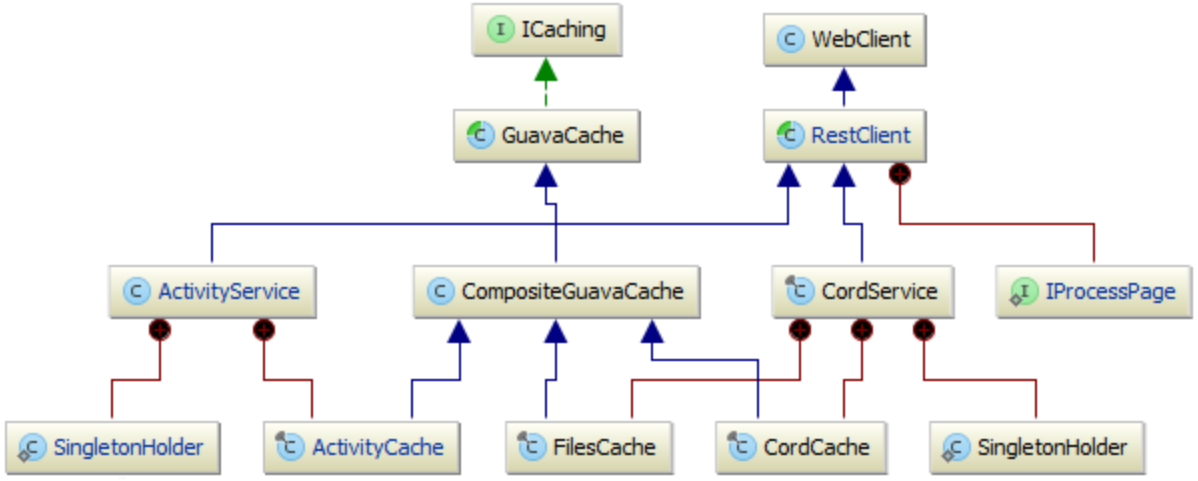
Obr. 29: Diagram balíkov serverovej časti.

Na strane klienta bude iba **jeden** základný balík a ten bude obsahovať triedy:

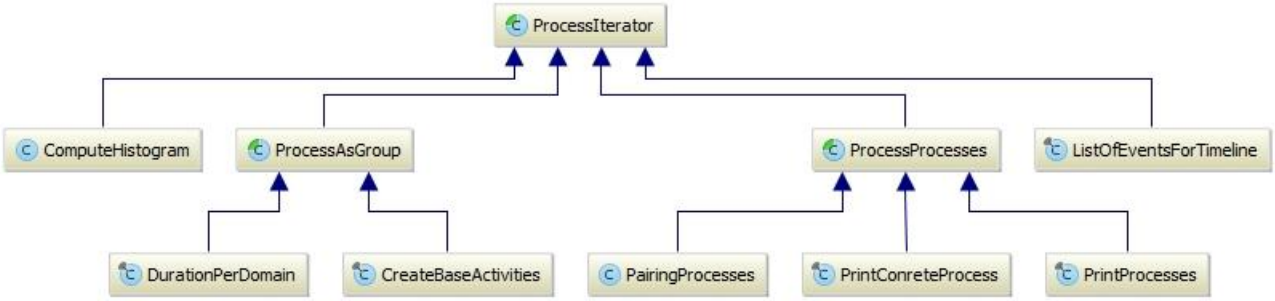
- *Timeline / Vis / GoogleChart component* – komponenty vizualizácie
- *ComponentManager* , *ChunksLoader, IvdaService, Preloader* – Majú na starosť načítavanie údajov.
- *Globals, GraphData, Toolbar, Util, ComponentManager* – Pomocné triedy.

Diagramy tried

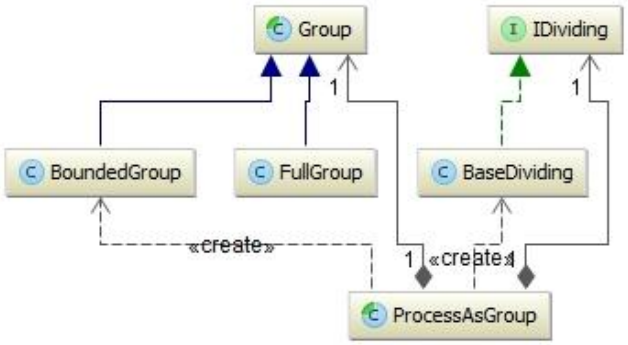
V tejto kapitole ukážeme iba tri základné diagramy tried. Pôjde o tieto diagramy: diagram klientov na PerConIK služby, diagram ukazujúci možné spracovanie udalostí a aktivít.



Obr. 30: Diagram tried reprezentujúci klientov na PerConIK služby.
Klient každej služby je rozšírením od „RestClient“, ktorý používa webového klienta na sťahovanie údajov. Údaje sú cachovane priamo v klientovi za pomoci pomocných tried.

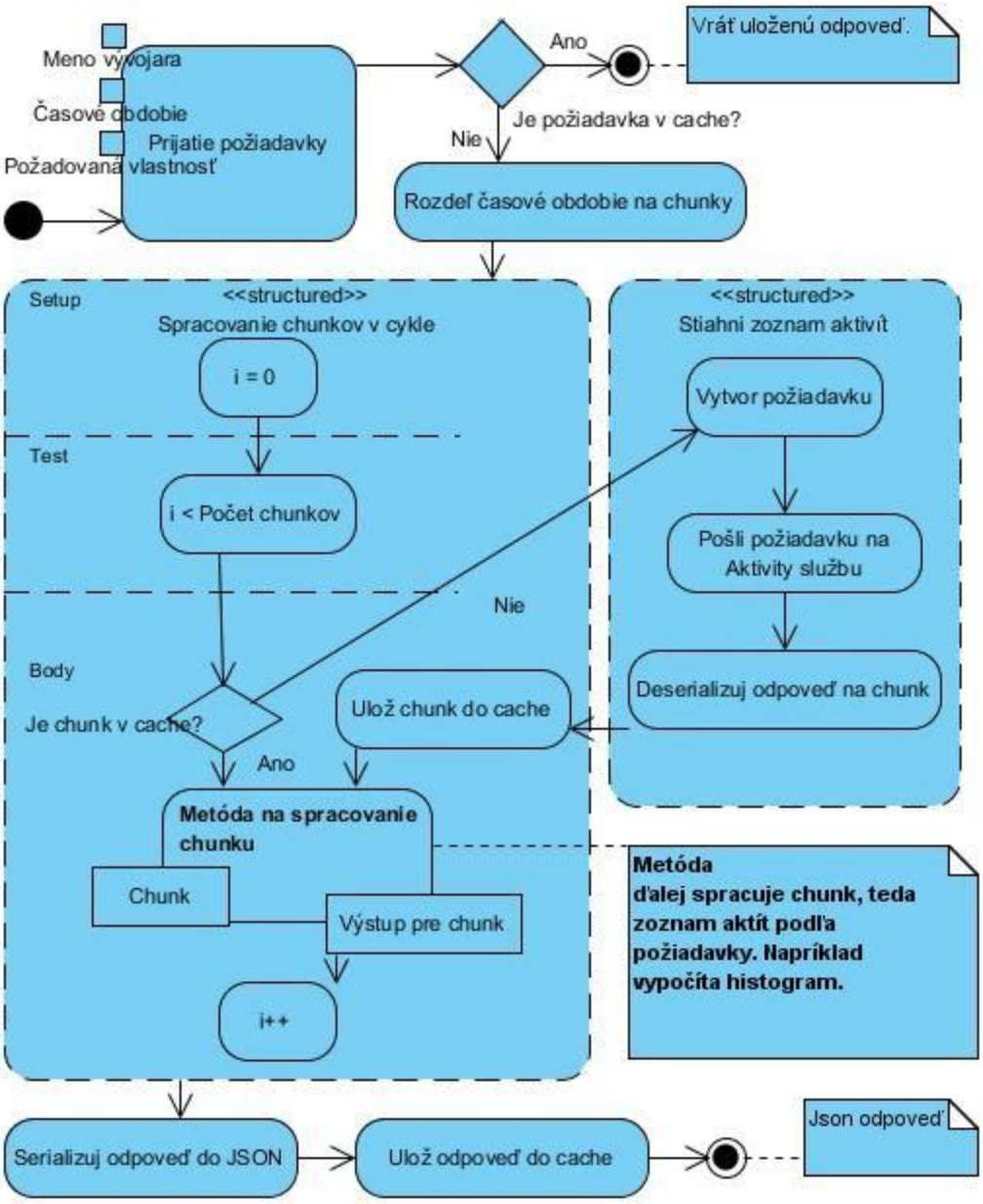


Obr. 31: Diagram tried pre spracovanie akcií vývojára.
Akcie vývojára môžu byť spracované pre histogram, tvorbu aktivít, výpis procesov alebo sa pošlú na časovú os. (To je opis tried v druhom riadku, z ľavá do pravá.)



Obr. 32: Triedy v balíku na zoskupovanie akcií do aktivít (skupín).

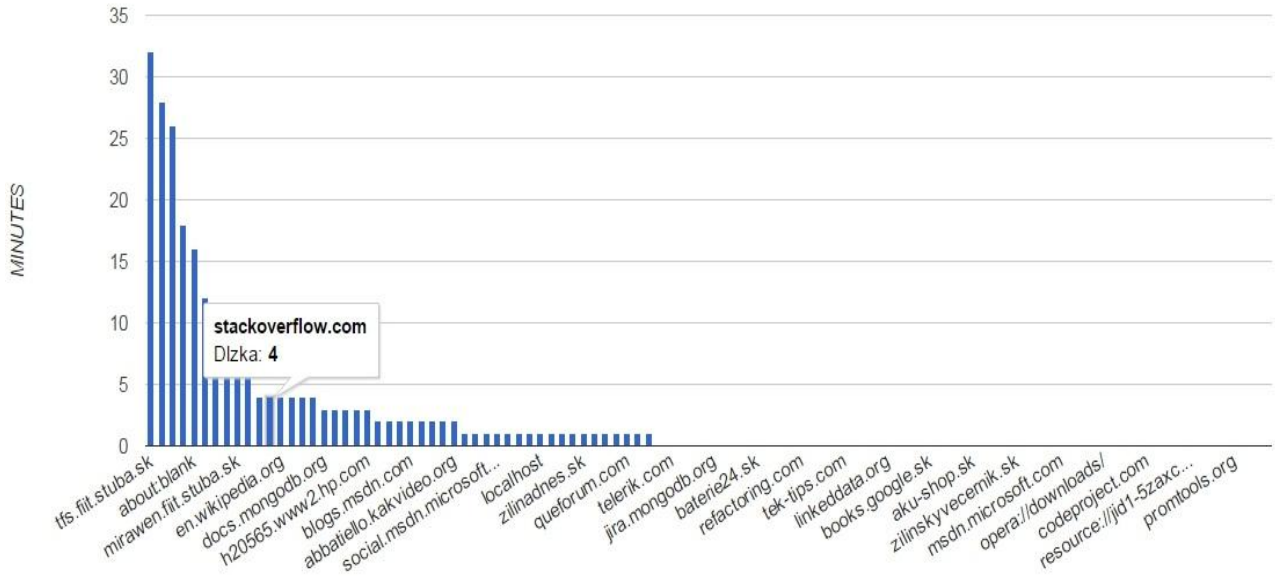
Diagram toku údajov



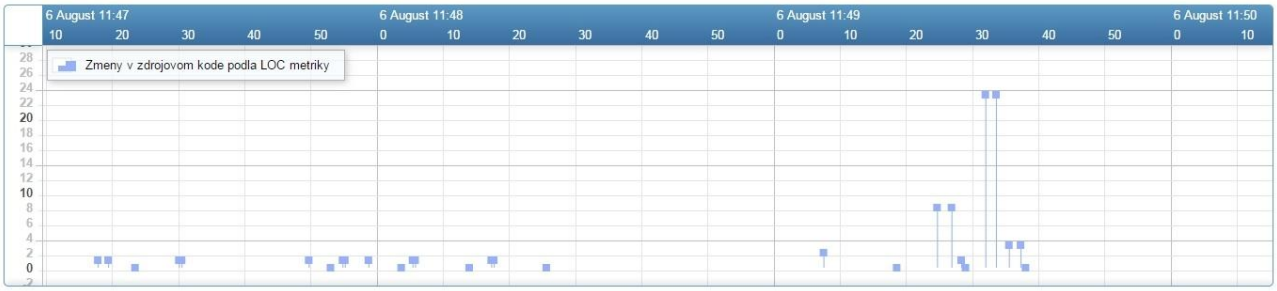
Obr. 33: Diagram toku údajov v službe IVDA.

Príloha D – Obrázková príloha vytvorených grafov

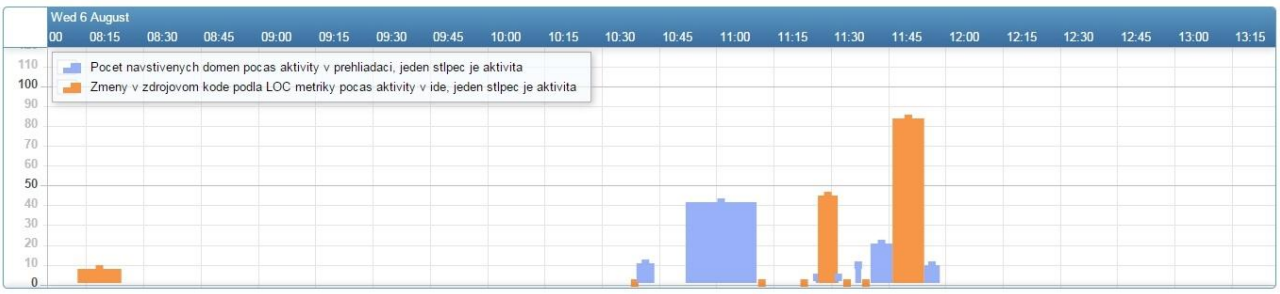
V kapitole 9.5 opisuje aké grafy môžeme vygenerovať v nástroji. Tu prezentujeme náčrt každého grafu.



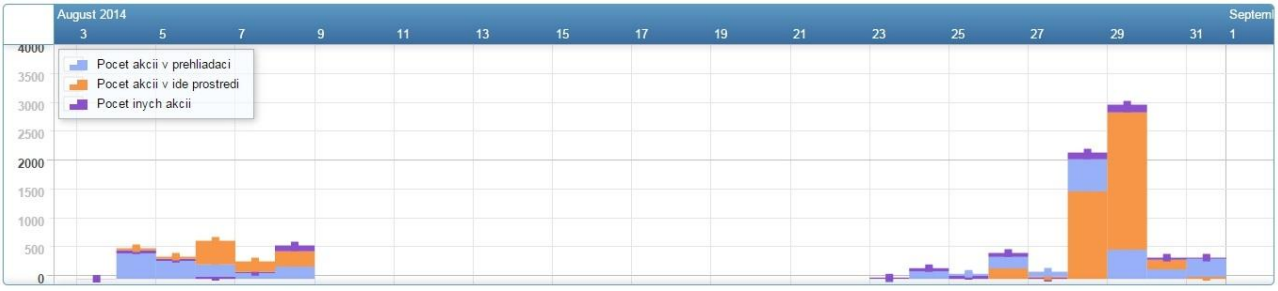
Obr. 34: Dĺžka používania webových portálov – poukazuje na strávenú dĺžku (v minútach pre 1 vývojára) na portáloch. Graf vygenerovaný pre vývojára Mouse v období 1.8-8.8.2014.



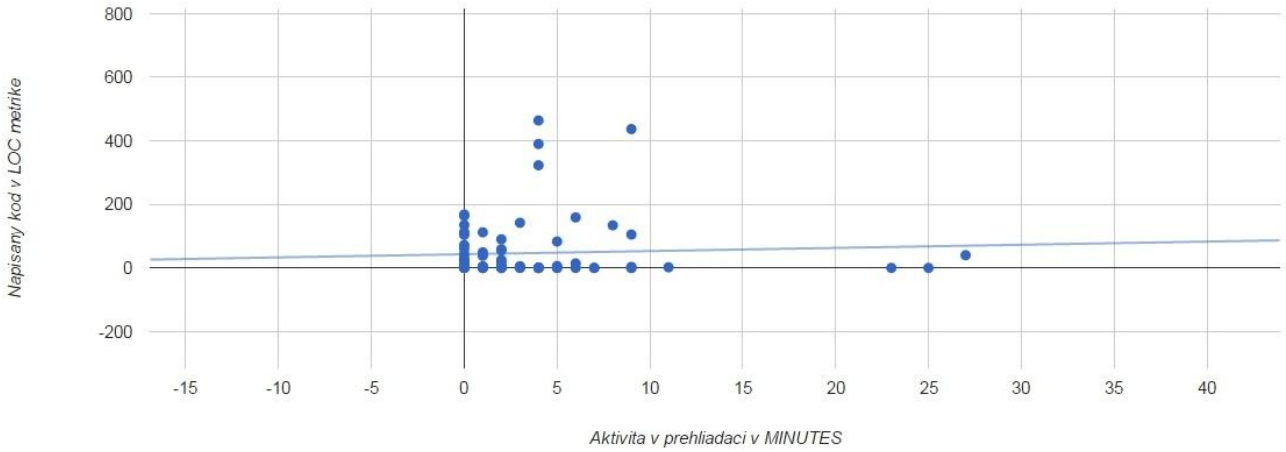
Obr. 35: Graf zmien pre zdrojový kód, spresni kedy vývojár písal zdrojový kód, v akom množstve (LOC).



Obr. 36: Graf vybraných dvoch vlastností pre aktivity

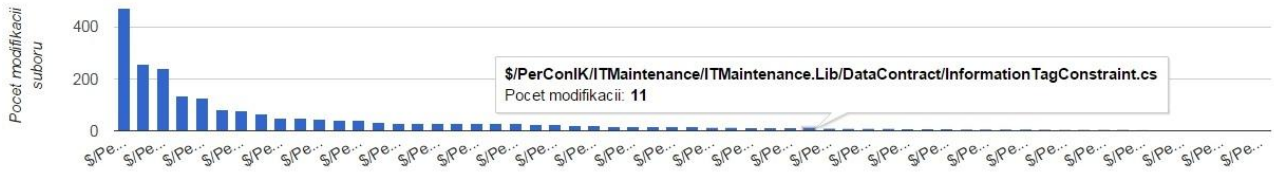


Obr. 37: Graf počtu udalostí rozdelených do dní.



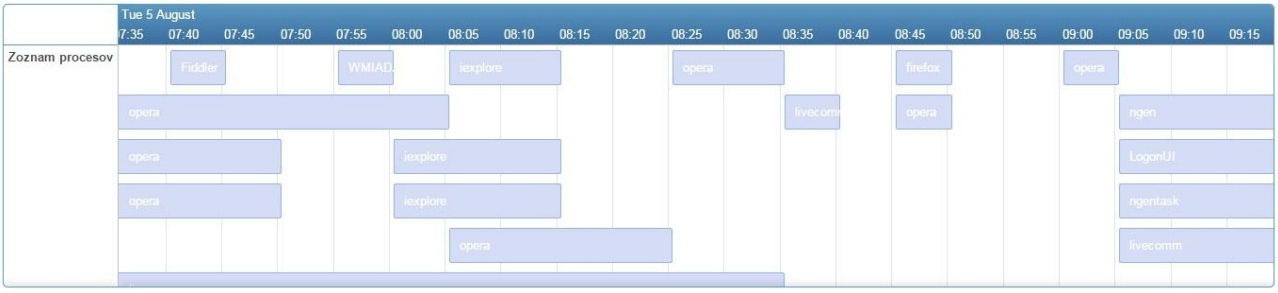
Obr. 38: Aktivita v prehliadači v minútach voči zmenám v zdrojovom kóde podlá LOC metriky.

Obr. 39: Graf zobrazujúci detail aktivít, zobrazujeme kedy a ako dlho strávil na aktivite (portály).



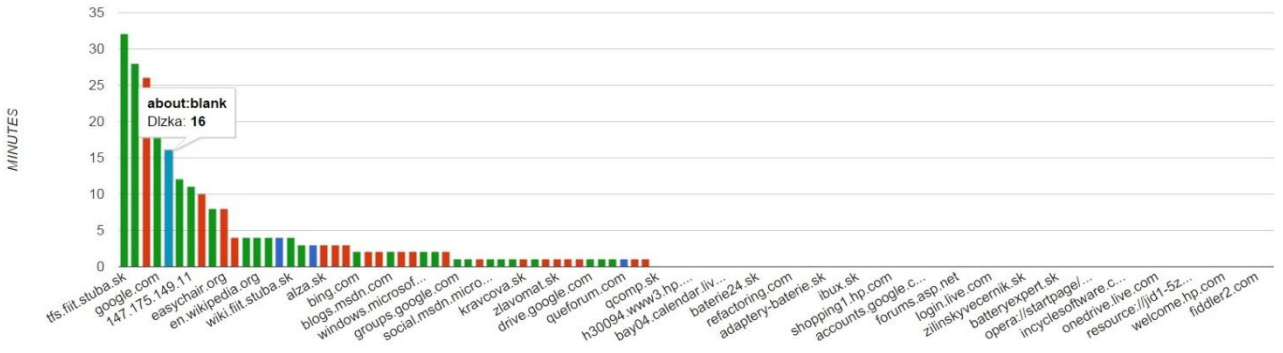
Obr. 40: Počet modifikácii súborov daným vývojárom v rámci vybraného obdobia. Vybraný vývojár bol Mouse a obdobie bolo zvolene 01.08-31.08.2014.

Obr. 41: Počet navštívení portálu daným vývojárom pre zvolené obdobie. Vybraný vývojár bol Mouse a zvolené obdobie 01.08-31.08.2014.



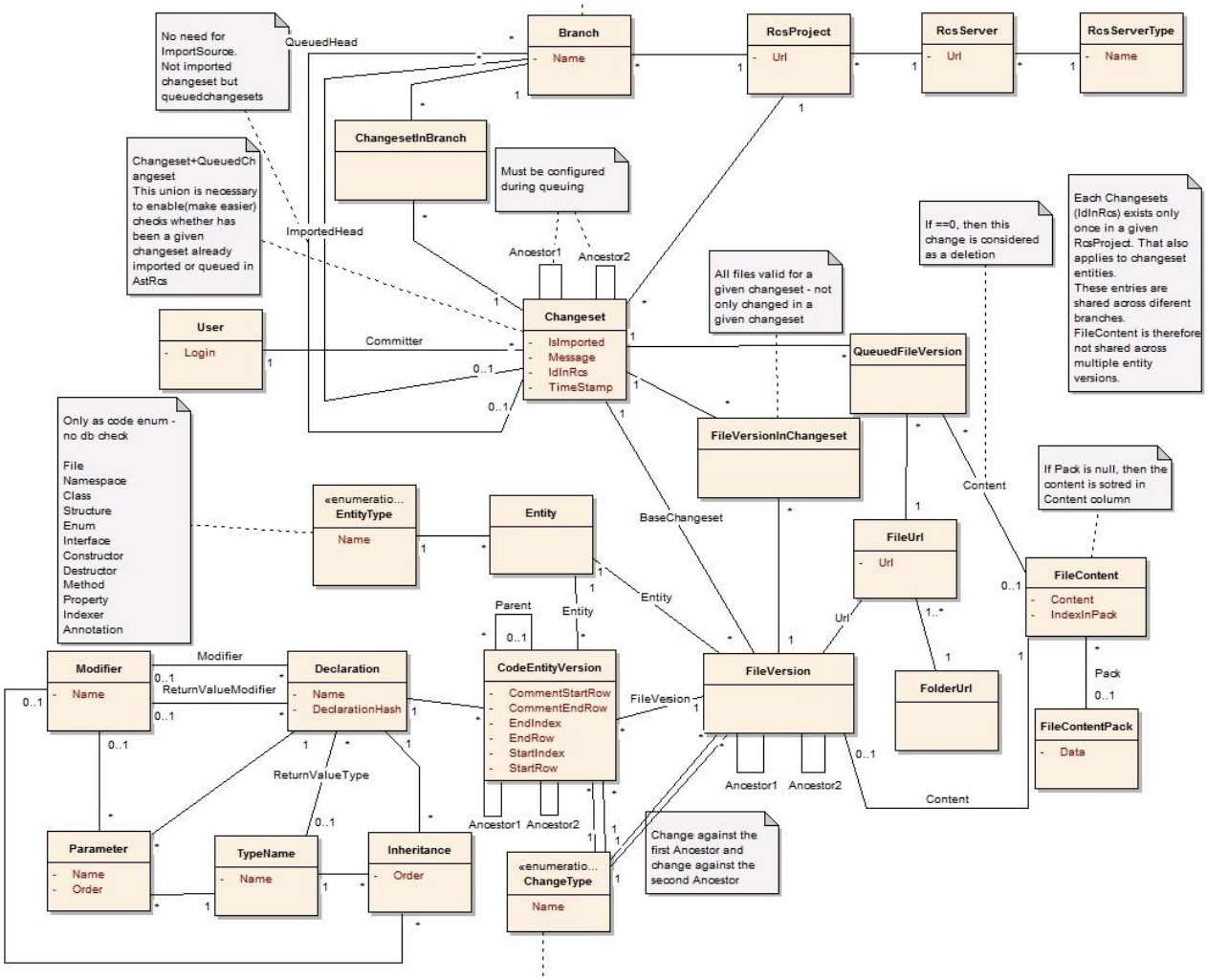
Obr. 42: Vizualizácia pre detail procesov spustených na pracovnej stanici vývojára. Intervaly na časovej osi ukazujú dĺžku trvania procesov.

Obr. 43: Aktivity vývojára Mouse za zvolený týždeň, ktoré sú zoskupene do hodiny. Výška stĺpca predstavuje dĺžku jeho aktivít v rámci hodiny.



Obr. 44: Graf zobrazujúci dĺžku strávenú na portály. Pričom tieto portály sú klasifikované do typov. Zelena odborné, červená neodborné a modrou neznáme portály.

Príloha E – Obrázková príloha

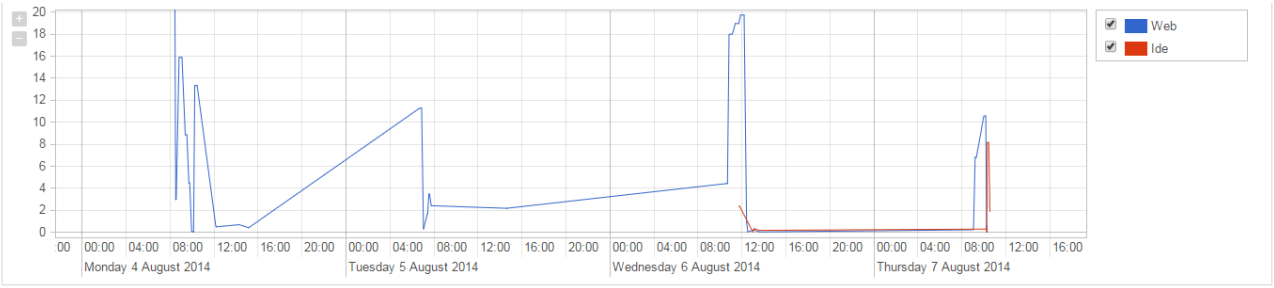


Obr. 45: Dátový model na službe Revision control system.
V databáze je uložená história entít, súborov a zoznamy commit-ov.

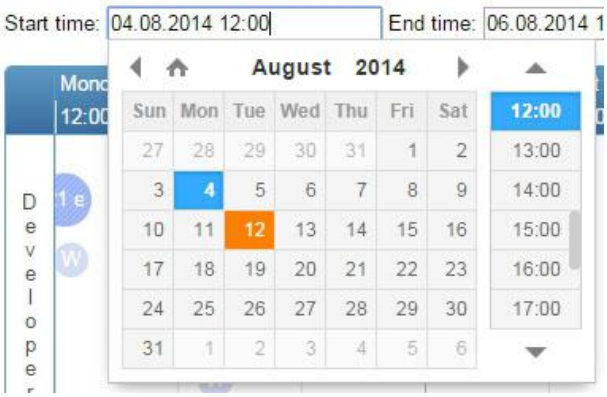
Obr. 46: Logo aplikácie.

	Start date	End date	Content	Group	ClassName
1	10.6.2014 9:18:00		checkin	A	available
2	10.6.2014 10:04:37		checkin	B	available
3	10.6.2014 10:14:48		checkin	B	available
4	10.6.2014 10:40:29		checkin	B	available

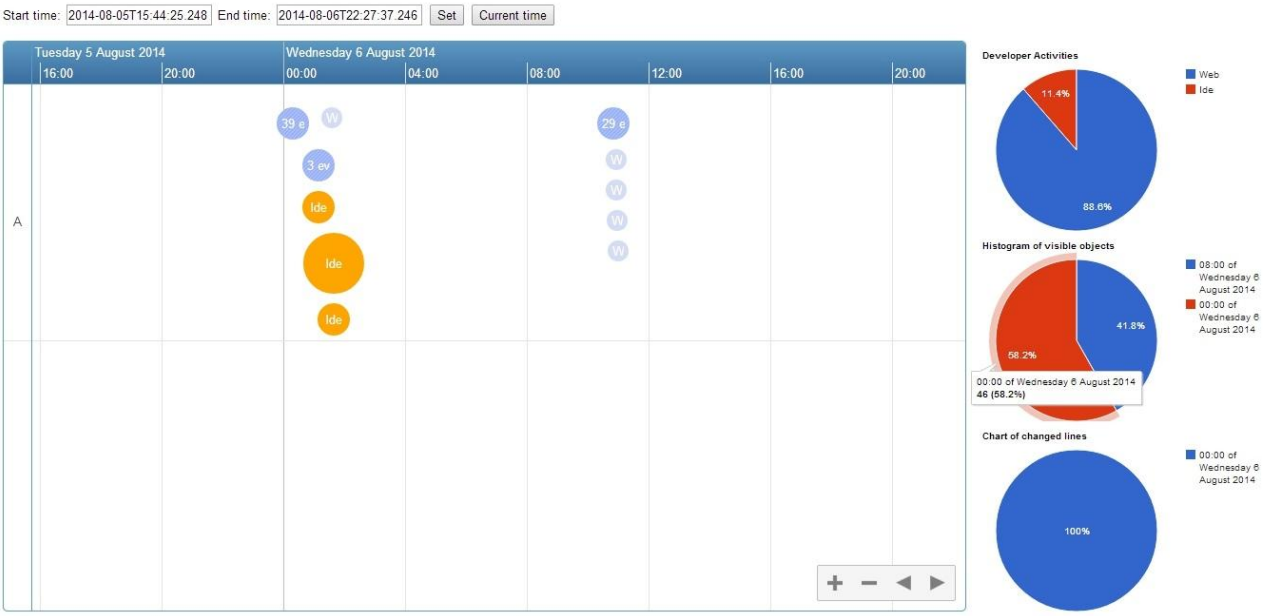
Obr. 47: Zahodený prototyp vizualizácie, kde sme zobrazovali informácie aj v tabuľke. Na časovej osi sme ukázali iba check-in udalosti. Prototyp sme zahodili, keďže systémy na správu verzie poskytujú detailnejší pohľad na checkin udalosti



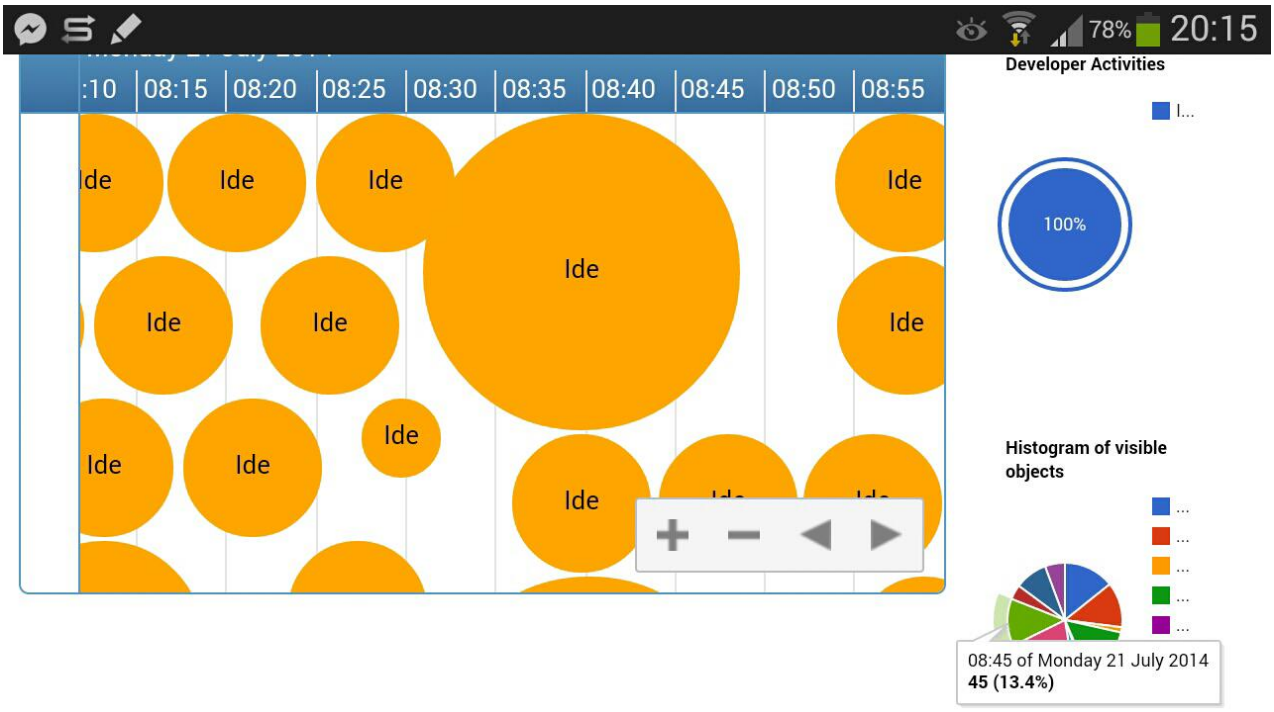
Obr. 48: Zahodený prototyp grafu na zobrazovanie aktivít. Po tomto prototype sme zistili, že aktivity je lepšie zobraziť pomocou intervalu.



Obr. 49: Interakcia používateľa s panelom nástrojov. Po kliknutí na časový údaj, zobrazí sa nápomocný kalendár, ktorý je taktiež interaktívny.



Obr. 50: Prototyp vizualizácie, panel nástrojov nie je dokončený.



Obr. 51: Prototyp aplikácie zobrazený na mobilnom zariadení. Vizualizácia je na malej obrazovke pomerne neprehľadná. V pravom rohu sa zobrazuje popis po interakcii.

Príloha F – Report z experimentu

1. Predstavenie testovacieho subjektu

Nástroj IVDA poskytuje prostredie pre generovanie grafov, ktoré zobrazujú údaje o aktivitách softvérového vývojára. Tieto údaje pomáhajú vizuálne odpovedať na otázky ohľadom vývoja softvéru. Nástroj je nasadený vo forme webovej stránky, pre jeho používanie je potrebný iba webový prehliadač.

ManicTime automaticky nahráva používanie počítača pri akejkoľvek práci. Zameriava sa na použité aplikácie (prostredie) a dokumenty (kontext), ktoré sa používali cez aplikáciu. Sleduje sa ako dlho tieto aktivity trvali. Zachytené dáta je možné vidieť na časovej osi a slúžia používateľovi ako prehľaď práce na pracovnej stanici.

Cieľom tohto testu je otestovať použiteľnosť a ukázať výsledky testovania.

2. Metodológia testu

2.1. Testovacia procedúra

Testovací subjekt vo forme webovej stránky bude poskytnutý skupine používateľov. Títo používatelia budú musieť mať pripravené definované prostredie. Každý používateľ vykoná sériu úloh, pri ktorých bude sledovaný. Po vykonaní týchto úloh každému používateľovi položíme niekoľko otázok a získame spätnú väzbu na používateľské prostredie. Štruktúra dokumentu je zapísaná podľa **Common Industry Format** (CIF) [ISO, 2006][30].

2.2. Testovací používatelia

Testu sa zúčastnili priatelia a kolegovia s nasledujúcou charakteristikou.

- skúsení s prácou na počítači, s používaním webového prehliadača a webu
- ľudia bez skúseností s riadením či manažovaním tímu vývojárov
- ľudia so základnou vedomosťou, ako vývojár pracuje a čo pri práci používa
- vo veku 18-25 rokov

	Pohlavie	**Vek**	**Vzdelanie, titul, odbor**	**Denne používa prehliadač**
TU-1	Muž	22	Vysokoškolské, Bc., Informatika	8 hodín
TU-2	Žena	21	Vysokoškolské, Geodézia	5 hodín
TU-3	Muž	22	Vysokoškolské, Bc., Informatika	8 hodín
TU-4	Muž	23	Vysokoškolské, Bc., Informatika	5 hodín
TU-5	Muž	24	Vysokoškolské, Informatika	7 hodín
TU-6	Žena	19	Vysokoškolské, Chémia	2 hodín

[30] Zdroj: http://courses.iicm.tugraz.at/iaweb/materials/fe/fe.html

TU-7	Žena	35	Vysokoškolské Dr., Informatika	5 hodín

2.3. Testovacie prostredie

Lokalita
Všetky testy budú vykonané na Fakulte informatiky a informačných technológií, v učebni 1.38.

Hardware a Software
Podpora sledovania aktivít vývojára podľa operačného systému je zobrazená v tabuľke vľavo. V pravej tabuľke je zobrazená podpora vizualizácie od operačného systému. IVDA je založená vo forme webu, je tak nezávislá od operačného systému.

	Win	Unix	Mac
ManicTime	x		
IVDA[31]	x	x	

	Win	Unix	Mac
ManicTime	x		
IVDA	x	x	x

Testovanie bude teda prebiehať na jednej pracovnej stanici. Na pracovnej stanici bude nasadený OS Windows aby bolo možné použiť *ManicTime*. Zároveň webový prehliadač s prístupom na internet, pre použitie zvyšných dvoch nástrojov.

2.4. Príprava a výcvik používateľov
Experiment je zameraný na užívateľov skúsených s prácou s webovým prehliadačom. Týmto smerom užívatelia neboli nijako pripravení. Testovacím subjektom boli poskytnuté základné informácie o nástroji, teda na čo je určený , akej oblasti sa venuje a ako pomáha manažérovi tímu či výskumníkovi.

2.5. Testovacie prípady
Každá úloha vychádza z predpokladu, že testovací subjekt navštívi domovskú stránku nástroja. Dátumy boli referencované od dňa *6.08.2014* a používateľom bolo odporúčané sledovať vývojára *Mouse*, čo sme si istý, že používateľ nájde nejakú informáciu.

Číslo úlohy	Popis úlohy	Odhad. čas[32]	Max čas
TC-1	Pozri aktivity ľubovoľného vývojára za posledný týždeň.	1 min	3 min
TC-2	Pozri aké aktivity prevažovali pre ľubovoľného vývojára za posledný deň.	30 sek	2 min
TC-3	Zisti aké procesy boli spustene na pracovnej stanici používateľa dňa	2 min	4 min

[31] Ivda nástroj nesleduje priamo aktivity vývojára, robia to iné PerConIK nástroje.

[32] Časové intervaly platia pre všetky nástroje. Po prekročení maximálneho času, bude úloha označená ako nesplnená.

	6.12.2014.		
TC-4	V ktorej časti dňa, za posledné 3 dni, bol vývojár najproduktívnejší?	4 min	8 min
TC-5	S akými súbormi a prostredím pracoval vývojár pri svojej práci včera?	3 min	6 min
TC-6	Ktorý konkrétny súbor bol vývojárom najčastejšie upravovaný za cely rok?	3 min	8 min
TC-7	Písal vývojár zdrojový kód viac ráno alebo večer za posledný týždeň?	2 min	4 min
TC-8	Ako dlho používal vývojár prehliadač po 11 hodine.	30 sek	2 min
TC-9	Skús identifikovať nejaký zvyk v správaní vývojára za posledný mesiac.	8 min	15 min
TC-10	Existujú aktivity vývojára, ktoré nesúvisia s vývojom. Ak áno, zisti, kedy takéto aktivity vznikli, ako dlho trvali a čo sú to za aktivity. Vyberte si jeden augustový deň, 2014.	6 min	12 min

2.6. Interview otázky

Po vykonaní testu položíme nasledujúce otázky každému používateľovi na získanie spätnej väzby.

1. Bolo tam niečo, čo by ste hodnotili ako veľmi dobré, alebo pozitívne?
2. Bolo tam niečo, čo by ste hodnotili ako mimoriadne zlé?

3. Výsledky experimentu

V tejto časti prezentujeme výsledky po vykonaní experimentu.

3.1. Úspešnosť plnenia úloh

Výsledok úspešnosti je zapísaný v tabuľke. Riadok v tabuľke predstavuje testovacieho používateľa a stĺpec testovaciu úlohu. Splnená úloha pre nástroj *IVDA* je označená 1, *ManicTIme* je označený 2. Úloha bola splnená, ak používateľ vykonal postup úlohy, dosiahol požadovaný cieľ do maximálneho času. V poslednom riadku je celkový výsledok pre úlohu.

	TC-1	**TC-2**	**TC-3**	**TC-4**	**TC-5**	**TC-6**	**TC-7**	**TC-8**	**TC-9**	**TC-10**
TU-1	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2
TU-2	1,2	1,2	1,2	1	2	1,2	1,2	1,2	1	1,2
TU-3	1,2	1,2	1,2	1,2	1,2	1	1		1	1,2
TU-4	1,2	1,2	1,2	1	2	1,2	1,2	1,2	1,2	1
TU-5	2	1,2	2	1	1,2	1,2	1,2	1,2	1	1
TU-6	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2

TU-7	1,2	1,2	1,2	1,2	2	1,2	1,2	1,2	1,2	1,2
Úspešnosť nástroja 1	**85%**	**100%**	**85%**	**100%**	**57%**	**100%**	**100%**	**85%**	**100%**	**100%**
Úspešnosť nástroja 2	**100%**	**100%**	**100%**	**57%**	**100%**	**85%**	**85%**	**85%**	**57%**	**71%**

3.2. Subjektívne ohodnotenie použiteľnosti

Pre každého používateľa sme sledovali jeho spokojnosť s plnením úloh, tento krát len pre nástroj IVDA. Spokojnosť sme vyjadrili mierkou od 1 (spokojný) do 7 (nespokojný), nazývanou tiež **Likert stupnicou.**[1]

	Celkové ohodnotenie	**TC1**	**TC2**	**TC3**	**TC4**	**TC5**	**TC6**	**TC7**	**TC8**	**TC9**	**TC10**	**Priemer**
TU-1	4	5	5	3	5	5	4	5	5	2	4	**4.3**
TU-2	4	5	4	5	2	4	3	4	4	2	4	**3.7**
TU-3	4	4	4	5	4	4	3	3	2	3	4	**3.6**
TU-4	3	3	4	4	4	3	4	3	4	3	3	**3.5**
TU-5	3	2	3	3	2	3	4	4	4	2	4	**3.1**
TU-6	4	4	4	4	2	5	3	3	4	4	2	**3.5**
TU-7	5	3	4	5	5	4	3	5	3	5	4	**4.1**

Ďalej každého používateľa sme sa opýtali na finálnu spätnú väzbu.

	Spätná väzba používateľa na prácu s nástrojom IVDA po vykonaní celého testu.
TU-1	Grafy sa generovali jednoducho, ale často som nevedel, ktorý mam vybrať. Tak som skúsil viaceré a po viacerých pokusoch som už vedel, ktorý graf je na čo. Posledné úlohy potom neboli ťažké.
TU-2	ManicTime bol neprehľadný, keď som musel hľadať aktivity za posledný mesiac. Naopak zase sa mi na ňom páčilo, že mal tri riadky pre aktivitu, programy, súbory.
TU-3	Posledné úlohy sa mi zdali príliš ťažké. V ManicTime som musel prepínať medzi dňami aby som si spravil prehľad čo robil vývojár za týždeň a viac. Bolo to otravné. Rovnako pri pozeraní akcií v ManicTime som musel počítať ako dlho trvali akcie, nechcelo sa mi to.
TU-4	V IVDA som sa pozeral koľko zdrojového kódu vývojár napísal a to som bral ako efektivitu práce. Pri tom druhom programe som nenašiel žiadne metriky.
TU-5	Pri IVDA som nevedel zistiť ako dlho trvali aktivity, musel som si to vypočítať. ManicTime mi krásne ukázal ako dlho trvali aktivity. Hodil by sa lepší popis pre oba nástroje.

TU-6	Je to zaujímavé vidieť, kto čo a kedy robil. Tie informácie som si spájala ťažko, dokedy mi to nevysvetlili. Potom som napríklad zistili, že oni trávili veľa času na Facebooku a nepísali veľa kódu.
TU-7	Mne sa nástroj páčil. Po testovaní som bola ďalej zvedavá. Napríklad pre 9 úlohu: Čo ak takéto správanie nie je typické len preňho? Ale pre všetkých? Stačilo zapnúť rovnaký graf pre ďalších 3 vývojárov a našla som odpoveď. Skvelé.

3.3. Diskusia a analýza

Výsledky v časti 3.1 poukazujú, že používatelia vykonali väčšinu požadovaných úloh v stanovený čas. Platí to pre oba nástroje. To môže byť spôsobené tým, že boli zadané pomerne jednoduché úlohy. Ďalej sme pozorovali, že používatelia častokrát mali problémy s prvými úlohami, robili ich dlhšie ako bol odhadovaný čas ale splnili maximálny dovolený čas. To môže znamenať, že nástroje nie sú veľmi intuitívne a vyžadujú určitý tréning používateľa. Posledné úlohy používatelia splnili do odhadovaného času.

Časť 3.2, zase poukazuje na relatívnu spokojnosť s používaním nástroja. Preto môžeme prehlásiť, že používanie tohto nástroja je pomerne jednoduché a pre používateľov príjemné ale vyžaduje tréning používateľa na prácu s IVDA nástrojom.

Zachytené poznatky pri pozorovaní používateľov boli zapísané a ich spätná väzba môže napomôcť vylepšiť používateľské prostredie. V ďalšej verzii sa odporúča vylepšiť implicitne ukazovať dĺžku aktivity vývojára a popisy k tlačidlám zobraziť výraznejšie.

Príloha G – Štruktúra dátového disku

Na priloženom dátovom disku sa nachádzajú nasledovné priečinky:

\ documents - priečinok obsahuje elektronickú verziu diplomovej práce v *DOCX* a *PDF* formáte

\ sk.stuba.fiit.perconik.ivda - koreňový adresár projektu
\ sk.stuba.fiit.perconik.ivda \doc - vygenerovaná java-doc dokumentácia k zdrojovému kódu
\ sk.stuba.fiit.perconik.ivda \conf - konfiguračné súbory pre aplikáciu
\ sk.stuba.fiit.perconik.ivda \test - jUnit testy k zdrojovému kódu
\ sk.stuba.fiit.perconik.ivda \web - webová časť projektu
\ sk.stuba.fiit.perconik.ivda \ src - zdrojové kódy pre serverovú časť

Ďalšie zaujímavé súbory:
\ sk.stuba.fiit.perconik.ivda \ README.md - obsahuje opis projektu, návod na spustenie
\ sk.stuba.fiit.perconik.ivda \ LICENCE.md - licenčný súbor pre celý projekt

Návod na nainštalovanie:

Pre spustenie programu je potrebné mať nainštalovanú Java verziu 7 a server, napríklad „Apache Tomcat 7.0.54“. Následné je potrebné nasadiť výstup (*deploy artifact*) na server. Serverová časť obsahuje konfiguračný súbor, tento súbor má už základné hodnoty nastavené. Konfiguračný súbor musí byť nastavený pred spustením servera. Vizualizácia sa zobrazí po navštívení adresy servera, kde bol výstup nasadený.

Súbor „README.md“ obsahuje podrobnejší návod.

Návod na jednoduché spustenie už nasadeného nástroja:

Hore definované súbory sú zároveň uložené v *GitHub* repozitári. Tam je možné vidieť históriu zmien a históriu vývoja tohto nástroja. Adresa repozitára je:

https://github.com/sekys/ivda

Adresa nasadeného nástroja:

http://ivda.eu