Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

**Pavol Michálek**

Interaktívna vizualizácia

vývoja zdrojového kódu v čase

Bakalárska práca

Vedúci práce: Mgr. Alena Kovárová, PhD.

december 2013

Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

**Pavol Michálek**

Interaktívna vizualizácia

vývoja zdrojového kódu v čase

Bakalárska práca

Študijný program:

Študijný odbor:

Miesto vypracovania:

Vedúci práce:

Informatika

9.2.1 Informatika

Ústav informatiky a softvérového inžinierstva, FIIT STU Bratislava

Mgr. Alena Kovárová, PhD.

december 2013

Anotácia

Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

Interaktívna vizualizácia

vývoja zdrojového kódu v čase

Študijný program:

Autor:

Vedúci práce:

Odovzdanie:

Informatika

Pavol Michálek

Mgr. Alena Kovárová, PhD.

December 2013

Cieľom práce je teoretická analýza využitia vizualizácií v oblasti vývoja softvéru a prieskum existujúcich riešení. Hľadajú sa väzby medzi fragmentmi kódu a metadátami zozbieranými počas vývoja softvéru za pomoci nástroja vytvorený v projekte PerConIK. Špeciálna pozornosť je venovaná vzťahu vývojára a zmeny fragmentu zdrojového kódu, ktorú vykonal. V implementácií vyplývajúcej z analýzy je možné vidieť vizualizované dáta z databázy PerConIKu AST RCS.

Annotation

Slovak University of Technology Bratislava  
Faculty of Informatics and Information Technologies

Interactive Visualization

of Developing source code over time

Degree Course:

Author:

Supervisor:

Submission:

Informatics

Pavol Michálek

Dr. Alena Kovárová

December 2013

The aim of this labor is theoretical usage analysis of visualization in software development and exploration of existing solutions. Searching for couplings between code fragments and metadata collected during software development thanks to tool developed in the project PerConIK. Special attention is paid to the relationship between developer and modification of the source code fragment, he has done. In implementation resulting from the analysis we can see visualized data from the database PerConIKu AST RCS.

Zadanie

Pri vývoji softvérového projektu vzniká mnoho nezodpovedaných otázok, na ktoré sa nie vždy ľahko hľadá odpoveď. Z tohto dôvodu existujú mnohoraké vizualizačné techniky (metódy), ktoré svojím vyobrazením pomáhajú človeku tieto odpovede nájsť. Niektoré z nich zachádzajú až do detailov na úrovni zdrojového kódu.

Analyzujte existujúce metódy vizualizácie vývoja zdrojového kódu v čase. Na základe analýzy a výberu zámeru (za akým cieľom sa vizualizácia robí, čo v nej má byť vidno) navrhnite vlastnú vizualizačnú metódu. Vypracujte špecifikáciu a softvérový návrh vlastnej aplikácie alebo zásuvného modulu, ktorý bude túto metódu realizovať. Riešenie implementujte a otestuje na vhodných vstupných dátach. Testovanie vyhodnoťte. Výsledky testovania zovšeobecnite pre vašu metódu.

Obsah

[Anotácia v slovenskom jazyku iii](#_Toc373845555)

[Anotácia v anglickom jazyku iv](#_Toc373845556)

[Zadanie v](#_Toc373845557)

[Poďakovanie vi](#_Toc373845558)

[Obsah vii](#_Toc373845559)

[Zoznam pojmov 2](#_Toc373845560)

[1 Úvod 3](#_Toc373845561)

[2 Analýza 4](#_Toc373845562)

[2.1 Vizualizácia 4](#_Toc373845563)

[2.1.1 Dynamická vizualizácia 4](#_Toc373845564)

[2.2 Väzby v zdrojovom kóde 5](#_Toc373845565)

[2.2.1 Analýza dopadu zmeny 6](#_Toc373845566)

[2.2.2 Modularita softvéru 6](#_Toc373845567)

[2.2.3 Kvalita kódu a predikcia chýb 7](#_Toc373845568)

[2.3 Spôsoby hľadania väzieb v zdrojovom kóde 7](#_Toc373845569)

[2.3.1 Štrukturálne väzby 8](#_Toc373845570)

[2.3.2 Väzby v histórii 8](#_Toc373845571)

[2.3.3 Sémantické väzby 9](#_Toc373845572)

[2.4 Verziovacie systémy a ich vizualizácie 10](#_Toc373845573)

[2.4.1 Gource 11](#_Toc373845574)

[2.4.2 Git Extensions 11](#_Toc373845575)

[2.4.3 Code\_swarm 12](#_Toc373845576)

[2.4.4 FishEye 13](#_Toc373845577)

[2.4.5 Zhrnutie analýzy 13](#_Toc373845578)

[2.5 Vizualizácia vzťahu vývojára a zmien v zdrojovom kóde 14](#_Toc373845579)

[2.5.1 Repozitáre vs. ChangeLog-y 14](#_Toc373845580)

[2.6 Analýza implementácie 15](#_Toc373845581)

[3 Špecifikácia vlastnej vizualizácie 16](#_Toc373845582)

[3.1 Účel vizualizácie 16](#_Toc373845583)

[3.2 Technická špecifikácia 17](#_Toc373845584)

[3.2.1 Databáza AST RCS 17](#_Toc373845585)

[3.2.2 Serverová Java aplikácia 18](#_Toc373845586)

[3.2.3 Vizualizačná knižnica D3.js 18](#_Toc373845587)

[4 Softvérový návrh 19](#_Toc373845588)

[Záver 20](#_Toc373845589)

[Resumé 21](#_Toc373845590)

[Zoznam použitej literatúry 22](#_Toc373845591)

[Prílohy 24](#_Toc373845592)

Zoznam pojmov

**ChangeLog** - kanonický názov pre záznam o zmenách na softvérovom projekte, ktorý zvyčajne obsahuje poznámky odrážajúce zmeny medzi commit-mi na konkrétnom mieste v projekte.

**commit** – zaužívaný anglický názov v kontexte verziovacích systémov v preklade znamená uloženie setu zmien v zdrojovom kóde pripraveného na odovzdanie do repozitáru

**widget** - všeobecné označenie pre ovládací prvok, čiže zariadenie prispôsobené k čo najjednoduchšiemu ovládaniu iného zariadenia, alebo programu

# Úvod

Vývojár počas práce na projekte s väčším počtom kooperujúcich ľudí (4+) často hľadá pôvod, vlastníctvo a editora kusov zdrojových kódov. Je to z dôvodu, že na krátkom úseku projektu/komponenty/modulu/balíku/triedy/metódy pracuje viac vývojárov a to v dlhšom časovom rozpätí. Dochádza tak k prelínaniu pôvodu fragmentov zdrojového kódu a teda k potenciálne väčšej chybovosti kódu. Aby vývojár predišiel takýmto chybám (napríklad v čase, keď sa zdrojový kód odladzuje alebo sa na ňom robí oprava chýb), jeho prvé kroky vedú k ľuďom, ktorý sú vlastníkmi týchto fragmentov. Od týchto ľudí získava cenné a rýchle informácie o vlastnostiach, závislostiach a účele daného zdrojového kódu. To vedie k efektívnej a spoľahlivej spolupráci na tvorbe projektu. Okrem toho vývojára zaujímajú aj ďalšie vzťahy okolo tohto fragmentu zdrojového kódu – kedy došlo k zmenám, a k akým ďalším zmenám došlo pri zmene tohto fragmentu. Ak zistí závislosť na nejakej ďalšej funkcionalite v inej časti zdrojového kódu, pomáha mu to k porozumeniu biznis hodnoty danej časti projektu.

Ako sa však vývojár dozvie o súvislosti fragmentov zdrojového kódu a ich autorov? Pomôcť by mu k tomu mala vizualizácia tejto väzby. Mal by sa niekoľkými rýchlymi kliknutiami dostať z prehľadu projektu až na konkrétne zmeny zdrojového kódu a priamo identifikovať autorov. Táto funkcionalita pripomína ChangeLog, ktorý však len stroho zobrazuje zmeny v textovej podobe. V správnej vizualizácií by mohol poskytnúť oveľa viac.

V nasledujúcej časti práce budem analyzovať aké aspekty vývoja zdrojového kódu je možné vizualizovať.

V časti práce „3. Špecifikácia prototypu“ uvediem opis hlavných komponentov, z ktorých bude vizualizácia pozostávať a čerpať dáta.

V časti práce „4. Softvérový návrh“ poskytnem opis fungovania a komunikácie komponentov prototypu.

# Analýza

Začiatok analýzy bude venovaný všeobecnému pohľadu na vizualizáciu. Následne sa zameriam na oblasť vývoja zdrojového kódu a budem hľadať väzby, ktoré je možné znázorniť vizualizáciou, aby poskytla pridanú hodnotu programátorovi participujúcemu na vývoji.

## Vizualizácia

Vizualizácia zahŕňa techniky vytvárania obrazov, animácií, diagramov pre potreby zviditeľnenia jednotlivých častí problému alebo vytvárania rôznych pohľadov na problém. Vizualizácia je definovaná (Brodlie, 1991) ako „vedný obor zaoberajúci sa využitím dát a informácií tak, aby prinieslo pochopenie a pohľad na dáta”. V súčasnej dobe sa najtypickejšie vytvárajú tieto zobrazovania a pohľady pomocou počítačovej grafiky. (1)

Vizualizácia má uplatnenie vo všetkých vedných oboroch a taktiež v samotnej informatike. V rámci informatiky sú to konkrétne počítačové vizualizácie, ktoré pomáhajú preskúmavať veľké množstvá dát. Vizualizácia dát zahŕňa výber, transformáciu a reprezentáciu abstraktných dát v zrozumiteľnej forme.

### Dynamická vizualizácia

Cieľom pre dizajnérov moderných vizualizačných metód je pomôcť používateľom zodpovedať ich otázky a dostať sa k odpovediam čo najrýchlejšie. Významným aspektom je teda interaktivita zobrazovania a jej dynamika. (2) Dynamické dotazy na databázu aj s miliónmi položiek umožňujú aktualizovať grafy v rádoch milisekúnd. Nie je nutné mať potvrdzujúce tlačidlo pre zmenu parametrov v danej vizualizácii – malo by stačiť, aby používateľ menil pomocou posuvníkov, tlačidiel, checkbox-ov a iných widget-ov výstup, ktorý sa nepretržite zobrazuje ako vhodná vizualizácia. Tou môžu byť stromové diagramy, mapy, časové osi, korelačné diagramy alebo akékoľvek iné inovatívne spôsoby vizualizácie.

Christopher Ahlberg vypracoval nasledujúce princípy manipulácie s grafickým rozhraním dynamických dotazov (3):

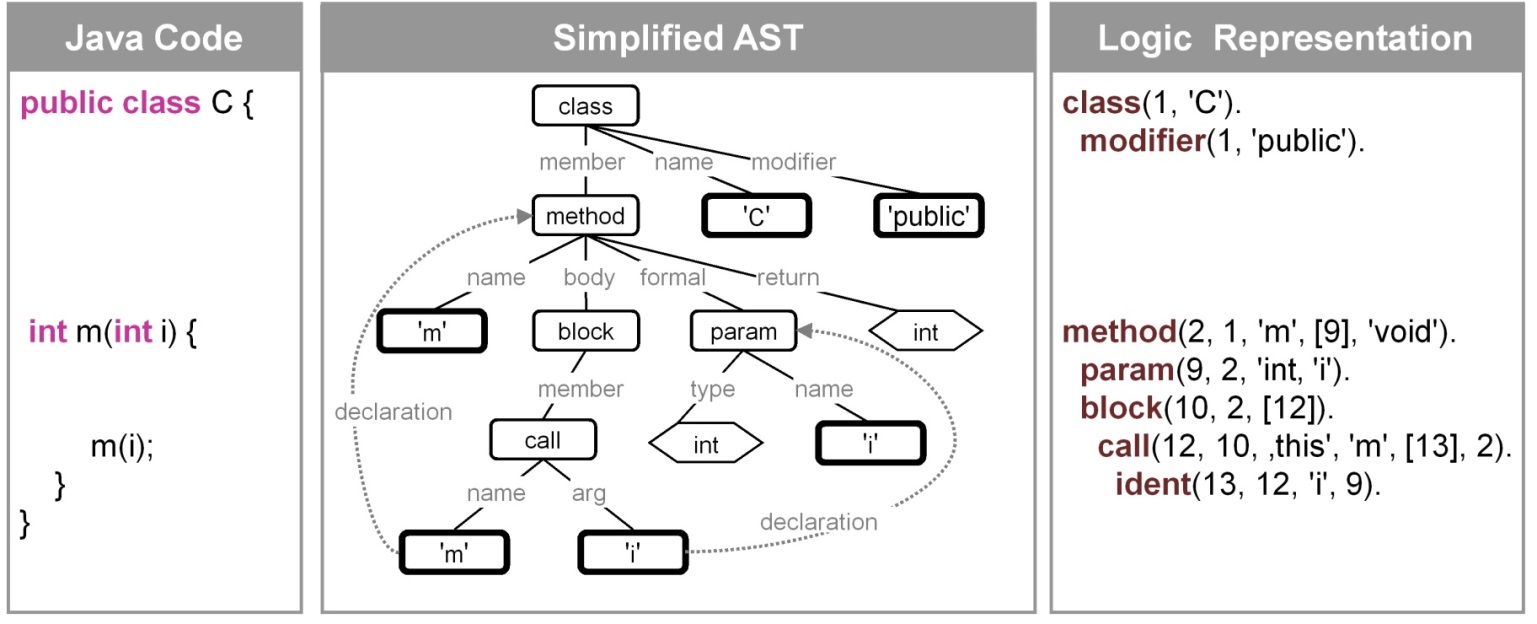
* nepretržitá reprezentácia objektov a akcií
* rýchle, prírastkové a reverzibilné akcie
* výber ukázaním (nie písaním)
* okamžité a kontinuálne zobrazenie výsledkov

Ak sa tieto princípy vhodne použijú, vizualizácia má ďalšie prospešné vlastnosti:

* nováčikovia sa rýchlo učia pri demonštrácii skúsenejším používateľom
* chybové hlášky sú málokedy nutné, pretože povolené sú len oprávnené hodnoty
* reverzibilné akcie znižujú pocit obmedzenia používateľa
* používateľ nadobúda istotu a odbornosť, pretože je iniciátorom akcií, takže môže predpovedať reakcie systému

## Väzby v zdrojovom kóde

Programovacie jazyky poskytujú syntax, ktorá hierarchicky organizuje zdrojový kód. Syntaktické elementy zdrojového kódu sú uzlami v AST - *abstract syntact tree* (abstraktný syntaktický strom), ktoré sú v tomto strome nazývané entity kódu. Sú atomickými elementmi dátového modelu. Môžu reprezentovať identifikátory, výrazy, bloky, metódy, ale aj vysokoúrovňové reprezentácie ako sú triedy alebo rozhrania. Táto široká definícia umožňuje použiť rôznorodé vizualizačné metódy.



Obrázok 1 Reprezentácia zdrojového kódu Javy vytvorené pomocou nástroja JTransformer tvoriaceho Abstract Syntax Tree  
(http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/prologast)

Hlavná zložka vývoja softvéru je tvorba fragmentov zdrojového kódu a spojení medzi nimi. Testovacia a ladiaca fáza vývoja vyžaduje sledovanie a ošetrovanie týchto vzťahov. Taktiež počas údržby a rozširovania softvéru sa musia neustále brať do úvahy tieto spojenia (4). Spojenia prinášajú informácie o väzbách medzi dvoma entitami kódu. Aby sa mohli odlíšiť sily týchto spojení, väzba môže niesť *váhu*.

### Analýza dopadu zmeny

Pokiaľ sa bude v priebehu vývoja softvéru neustále meniť čo i len jeden fragment kódu je potrebné sledovať, aké ďalšie neželané zmeny to môže spraviť. Predikcia a analýza vplyvu zmeny zdrojového kódu je známa ako *analýza dopadu zmeny*. Používa sa na rozhodnutia ako uskutočniť zmenu v existujúcom systéme, prioritizáciu testovacích prípadov v súvislosti s regresným testovaním, alebo na identifikáciu chýb-tvoriacich zmien.

Zmeny sa realizujú v nejakých súvislostiach, napr. na štruktúre vytváraného softvéru, jeho histórie alebo sémantickej súvislosti. Analýza väzieb zdrojového kódu teda môže odhadnúť dopad zmeny. (1)

### Modularita softvéru

Rozdelenie softvérového systému na menšie časti poskytuje modularizáciu systému. „Iba modularita robí komplexné systémy, ako sú softvérové systémy, spravovateľnými (5).“

Modularita môže existovať na viacerých úrovniach abstrakcie: metódy a atribúty sú zostavené v triedach, rozhrania a triedy sú hierarchicky organizované v balíčkoch, tie tvoria komponenty alebo subsystémy. Nezávisle od tohto levelu abstrakcie existujú väzby a metadáta v zdrojovom kóde, ktoré pomáhajú modularizovať systém, napr. keď sa identifikujú autori tried.

Obrázok 2 Modularizácia SOA systému

### Kvalita kódu a predikcia chýb

Z vnútorného hľadiska má kód ďalšie vlastnosti, ktoré s kvalitou súvisia - ako znovupoužiteľnosť, bezpečnosť, čitateľnosť, štruktúra a ďalšie. Celkovú kvalitu kódu teda vyjadríme ako súhrn kvalít jeho jednotlivých vlastností. (6)

Ohodnotenie kvality kódu a identifikácia entít kódu náchylných na chybu sa používajú softvérové metriky. A tieto sú často založené na väzbách v zdrojovom kóde. Napríklad metrika súdržnosti je definovaná štrukturálnymi vzťahmi v kóde.

## Spôsoby hľadania väzieb v zdrojovom kóde

„Pokiaľ modifikácia entity kódu vyžaduje zmenu inej entity, tak existuje väzba“ (7). Aj keď budúce zmeny nie je možné precízne odhadnúť, rôzne indikátory v kóde a jej histórii môžu napovedať možné spoločné zmeny. Preto je nutné hľadať spoločné znaky.

Na demonštráciu väzbovosti v zdrojovom kóde je nevyhnutné použiť jednotný programovací jazyk. Najlepšou ukážkou bude objektovo-orientovaný jazyk Java, ktorý ponúka jasný náhľad na jeho štruktúru a sémantiku. Taktiež sú jednoducho získateľné historické údaje viacerých verzií z repozitáru projektu.

Obrázok 3 Prehľad zdrojov väzieb v softvérovom projekt

V ďalšej časti sa budem snažiť opísať koncepty hľadania väzieb.

### Štrukturálne väzby

Zdrojový kód nevyhnutne obsahuje cykly, podmienky, skoky a volania metód. Zdrojový kód sa teda vykonáva sekvenčne. Existujú však aj spojenia, ktoré sa správajú iným spôsobom – prístup k entitám (premenné, iné triedy, ...).

**Štrukturálne závislosti** sú priamym spojením dvoch entít kódu. Tieto väzby sú nevyhnutné, pretože vykonanie jedného fragmentu závisí na inom. V objektovo orientovanom jazyku je tiež možné vidieť väzby na vyššej štruktúrovej úrovni ako sú *dedenie* a *agregácia*. Poskytujú špeciálne možnosti interakcie s cieľom väzby. Všetky tieto závislosti jasne indikujú možnú budúcu spoločnú zmenu, pretože zmena jednej entity bude vyžadovať zmenu inej, napr. zmena parametrov metódy bude vyžadovať zmenu všetkým volaní tejto metódy.

**Štrukturálne podobnosti** sú nepriamou väzbou medzi entitami (8). Keďže dve rôzne entity môžu volať rovnaké metódy alebo pracovať s rovnakými entitami kódu, musí existovať vzťah – podobnosť vo funkcionalite. Samozrejme je celý koncept založený na priamej štrukturálnej závislosti. Ak sa teda zmení funkcionalita implementovaná v jednom fragmente, je pravdepodobné, že bude mať dopad aj na modifikáciu iného fragmentu.

### Väzby v histórii

Dolovaním údajov z repozitáru môžeme získať informácie o minulom ale aj budúcom evolučnom vývoji projektu a teda fragmentov zdrojového kódu. Základným rozdelením môže byť podľa toho, ktoré entity boli v minulosti menené spoločne a kto je zodpovedný za zmeny.

**Evolučné väzby** hovoria o vzťahoch fragmentov kódu z pohľadu spoločnej zmeny v čase ich vytvorenia alebo zmeny počas ladenia programu. Otázkou je, či skutočne došlo k zmene na týchto fragmentoch počas vykonávania jednej úlohy. Závisí to od presnosti, s akou programátori vykonávajú commit-y do repozitáru. Ak vykonal zmenu na jednej úlohe v dvoch rôznych časoch, je nutné nazerať na obe zmeny dohromady ako na jednu. A pokiaľ aj zanedbáme časovú presnosť, stále zostáva problémom nemennosť niektorých súvisiacich fragmentov.

**Vlastník fragmentu zdrojového kódu** je vývojár chápaný v zmysle “kto vykonal zmenu”. Existuje jasná väzba medzi entitami kódu a vývojárom. Takže sa dá hovoriť o nepriamej väzbe medzi entitami kódu pokiaľ zdieľajú rovnakú alebo podobnú skupinu vývojárov. Od každého vývojára, ktorý urobil zmenu na danom fragmente, sa očakáva, že má nejaký stupeň znalosti o tejto oblasti. Vzniká tak podobnosť entít zdrojového kódu na základe oblasti, v ktorej sa vývojár pohybuje. Významným ukazateľom je to hlavne v tímových projektoch, kde sa dá odhaliť, kto je zodpovedný za daný kód a jeho budúcnosť. Bowman a Holt (9) predstavili myšlienku používania vlastníctva kódu na spájanie entít zdrojového kódu. Porovnávajú vysokoúrovňovú architektúru založenú na vlastníctve fragmentov kódu a súčasnú architektúru a nachádzajú významnú zhodu.

### Sémantické väzby

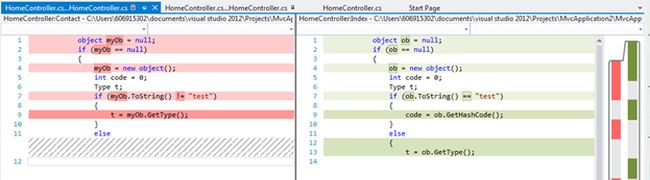
Sémantika programu opisuje abstraktný pohľad na zdrojový kód. Aj keď je ťažké hneď zadefinovať presné sémantické vzťahy, dá sa to pomocou postupného prechádzania zdrojového kódu a hľadania rôznych typov klonov entít zdrojového kódu.

**Klony entít zdrojového kódu** sú rovnaké alebo podobné fragmenty kódu, ktoré mohli vzniknúť *copy-paste* operáciou.

Rozoznávajú sa rôzne typy. *Typ 1* sú presné kópie fragmentov kódu. *Typ 2* môže mať variácie názvov a typov. *Typ 3* povoľuje rozdielne štruktúry. *Typ 4* zohľadňuje už len sémantické podobnosti. (10) Známou metrikou na rozoznávanie podobnosti zdrojového kódu je percentuálne ohodnotenie podobnosti dvoch daných fragmentov. Na základe škálovateľnosti rozsahu a tohto percenta sa dá zhodnotiť miera podobnosti.

Intuitívne sa dá poznať, že takéto podobné fragmenty zdrojového kódu nie sú správne. Pretože funkcionalita je implementovaná viac krát, môže vyžadovať viac snahy ju udržiavať, sledovať a opravovať na nej chyby. Môže to tiež znamenať zníženie výkonu počas behu systému.

Odhaľovanie takýchto fragmentov je samostatným predmetom aktívneho výskumu. Konkrétnymi výstupmi sú napr. zabudované funkcionality v integrovaných vývojových prostrediach (IDE), ktoré ponúkajú dohľadanie a vizualizáciu na úrovni zdrojového kódu.



Obrázok 4 Funkcionalita „Cloned Code feature“ vo Visual Studio 2012

**Sémantické podobnosti** sú spojením fragmentov na základe podobnosti použitých slov prirodzeného jazyka. Pri rozširovaní a zmene funkcionality, môže byť vyžadované zmeniť oba tieto fragmenty. Je opäť ťažšie rozpoznávať tieto súvislosti, ale je to možné napríklad vďaka heuristickým a štatistickým metódam uplatneným na použitú slovnú zásobu. (11)

## Verziovacie systémy a ich vizualizácie

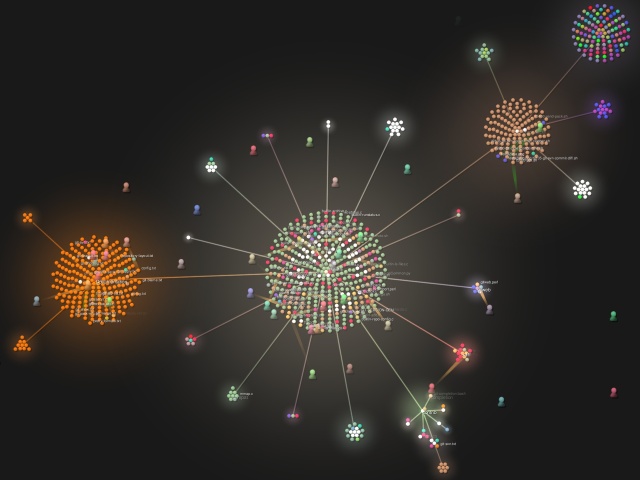
Vývoj softvérových produktov v dnešnej dobe je proces, ktorý trvá zvyčajne mesiace, ale niekedy aj niekoľko rokov. Verziovací systém je zodpovedný za spoluprácu najmä na zdrojových kódoch. Takýto podporný prostriedok má pre vývoj softvéru mnoho výhod:

* + viacerí môžu robiť súčasne na tom istom zdrojovom kóde
  + súčasná práca na viacerých verziách
  + zálohovanie bez zbytočne vynaloženého úsilia
  + jednoduchý návrat k predošlým verziám v prípade poškodenia súčasnej verzie
  + sledovanie zmien a aktuálneho stavu produktu
  + sledovanie práce jednotlivých vývojárov

Z vlastností verziovacích systémov sa dá vyvodiť priestor pre vizualizácie, a teda vhodné zobrazenie zaznamenaných informácií v čase týmto systémom.

### Gource

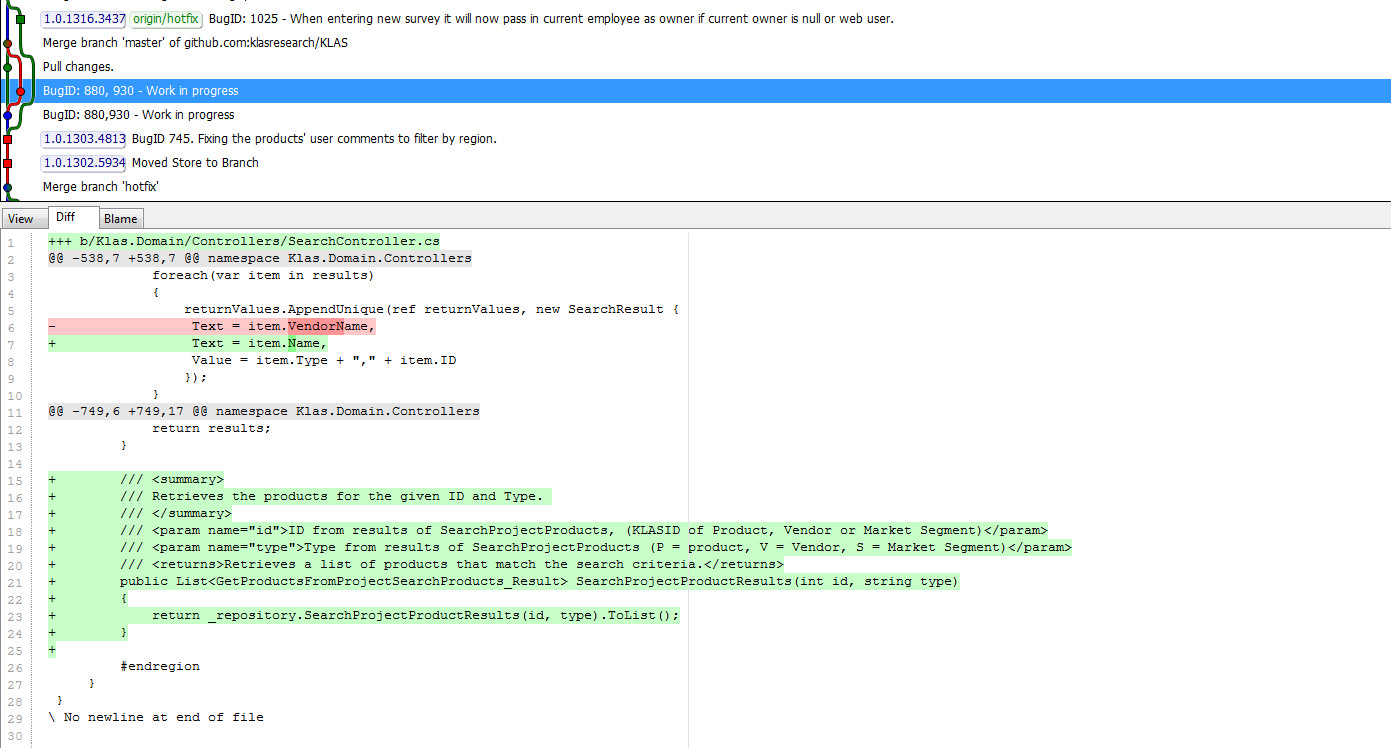
Gource je vizualizačným nástrojom, ktorý dokáže čerpať dáta z verziovacích systémov Git, Mercurial, Bazaar a SVN. Softvérový projekt je zobrazený ako animovaný rastúci strom s koreňovým adresárom v strede. Zložky sú jeho konármi a listami stromu sú jednotlivé súbory. Zobrazovanie je veľmi efektné a dynamické, avšak nemá veľkú výpovednú hodnotu – dá sa len ťažko vyčítať, ktorý vývojár vykonal zmeny na konkrétnych súboroch. Nie je tiež možné vidieť o aké zmeny sa jednalo pod úroveň zložiek.



Obrázok 5 Gource vizualizácia vývoja projektu verziovaného systémom Git

### Git Extensions

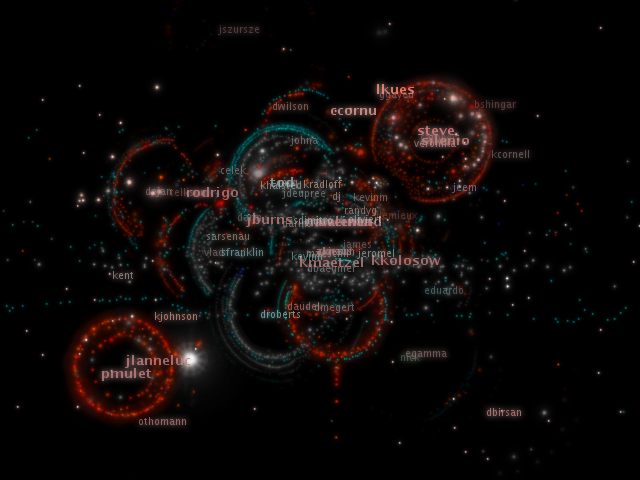
*Git Extensions* je jedným z grafických rozhraní pre verziovací systém Git, ktorý umožňuje jeho plnú kontrolu bez použitia príkazového riadku. Má vynikajúce vlastnosti zobrazovania vetiev a tiež zobrazenia detailných zmien na úrovni jednotlivých znakov v zdrojovom kóde. Pokiaľ teda vývojár vykonal zmenu vo svojej privátnej vetve, ostatný vývojári môžu vidieť jeho podrobné zmeny.



Obrázok 6 Zmeny zobrazené v Git Extensions

### Code\_swarm

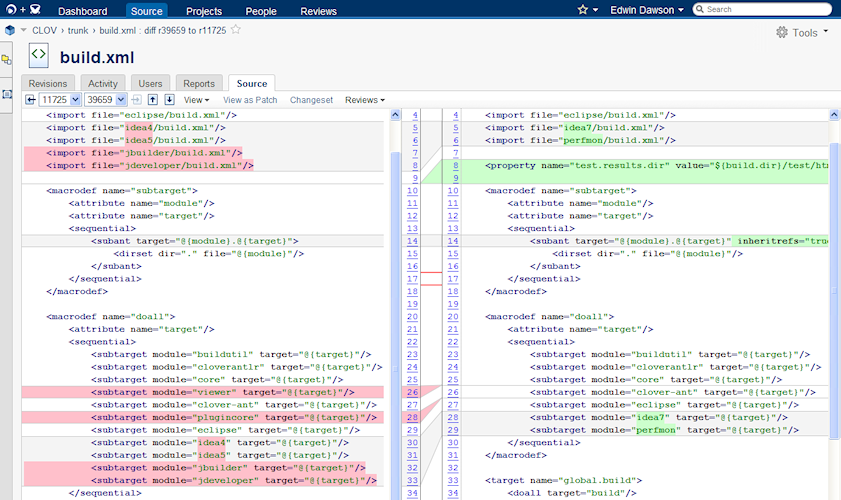
Code\_swarm je vizualizačným nástrojom podobným s Gource. Podporuje širšie spektrum zdrojových repozitárov (Subversion, CVS, Git, Mercurial, Perforce, ...). Zmeriava sa taktiež na interakciu vývojára a súborov zdrojových kódov, dokumentácií a ostatných typov súborov. Nesiaha však hlbšie pod úroveň súboru. Nie je teda možné vidieť detailné zmeny.



Obrázok 7 Code\_swarm vizualizácia vývoja projektu Eclipse

### FishEye

So zobrazovaním a vizualizáciou všetkých uložených informácií v rôznych repozitároch (Subversion, CVS, Perforce, Git, a Mercurial) pomáha nástroj FishEye od Attlasianu. Dokáže pomocou webového prehliadača zobrazovať zmeny, rozdiely, autorov, vetvy, značky na projekte podľa commit-ov. Vizualizuje, ako sa zdrojový kód vytváral, menil, kto zmeny robil a kedy boli uskutočnené zmeny. V realnom case tiež zobrazuje reporty zo všetkých zozbieraných informácií v nastavovateľných zobrazeniach.



Obrázok 8 Vizualizácia zmien v zdrojovom kóde v FishEye Atlassian

### Zhrnutie analýzy

Existujúce riešenia v oblasti vizualizácie väzieb a vlastností zdrojového kódu ponúkajú rôzne pohľady na zdrojový kód a jeho vývoj. Najčastejším zobrazením je zvýrazňovanie kódu, ktorý prešiel nejakou zmenou. Popri tom sa dá zistiť kto a kedy urobil zmenu. Chýba však komplexnejší pohľad na zmenu v rámci zložitejšieho zásahu do zdrojového kódu. Naopak, ak sa pozeráme na zmenu v rámci celého jedného commit-u, chýba nám informácia o viacerých zmenách na danom fragmente.

Z pohľadu vzťahu vývojára a zmien v zdrojovom kóde teda chýba zobrazenie, ktoré jednoducho zobrazí fragmenty zdrojových kódov a ich autorov. V praxi sa takéto informácie uchovávajú ChangeLog-och. Tie tiež priamo slúžia vývojárom pre identifikáciu zmien na zdrojovom kóde. Bolo by preto veľmi užitočné vizualizovať ChangeLog-y pre ich lepšiu čitateľnosť a výpovednú hodnotu.

## Vizualizácia vzťahu vývojára a zmien v zdrojovom kóde

Vývojár počas práce na projekte s väčším počtom kooperujúcich ľudí (4+) často hľadá pôvod, vlastníctvo a editora kusov zdrojových kódov. Je to z dôvodu, že na krátkom úseku projektu/komponenty/modulu/balíku/triedy/metódy pracuje viac vývojárov a to v dlhšom časovom rozpätí. Dochádza tak k prelínaniu pôvodu fragmentov zdrojového kódu a teda k potenciálne väčšej chybovosti kódu. Aby vývojár predišiel takýmto chybám (napríklad v čase, keď sa zdrojový kód odladzuje alebo sa na ňom robí oprava chýb), jeho prvé kroky vedú k ľuďom, ktorý sú vlastníkmi týchto fragmentov. Od týchto ľudí získava cenné a rýchle informácie o vlastnostiach, závislostiach a účele daného zdrojového kódu. To vedie k efektívnej a spoľahlivej spolupráci na tvorbe projektu. Okrem toho vývojára zaujímajú aj ďalšie vzťahy okolo tohto zdrojového kódu – kedy došlo k zmenám, a k akým ďalším zmenám došlo pri zmene tohto fragmentu. Ak zistí závislosť na nejakej ďalšej funkcionalite v inej časti zdrojového kódu, pomáha mu to k porozumeniu biznis hodnoty danej časti.

### Repozitáre a ChangeLog-y

Problémom je dostať sa k týmto informáciám jednoduchou a prehľadnou formou. V repozitároch je síce možné k týmto informáciám sa dostať, avšak je ťažké prehľadávať veľké množstvo commit-ov v mnohých vetvách projektu, kde k daným zmenám došlo. Ponúkajú zobrazenie zmien v závislosti od autora a času publikácie, náročnejšie je však zobrazenie zmien v závislosti na danej oblasti zmeny (fragmentu kódu) v chronologickom poradí. Takéto informácie je možné vidieť v  ChangeLog-och. Sú to väčšinou automatizovane zapisované súbory uložené pri súboroch, kde sa zmeny diali. (12) Zachytávajú zmeny na danom súbore, ktoré môžu pochádzať od viacerých autorov v dlhšom časovom rozmedzí (Obrázok 9). Prispievajú tak ku kvalitnejšej nadchádzajúcej práci na tomto fragmente. Niektoré vývojové prostredia priamo podporujú vytváranie a zápis automatických ChangeLog-ov. Stále sa ale jedná len o jednoduchý zápis do textovej podoby, ktorý je ťažko čitateľný pri viacerých záznamoch. Pokiaľ by boli tieto záznamy prevedené do graficky pochopiteľnejšej podoby, mohli by napomôcť vývojárovi dohľadať potrebnú informáciu rýchlejšie. Pridanou hodnotou má byť tiež zobrazenie zmien, ktoré nastali v rovnakom čase, čiže v rovnakom commit-e.

**1999-03-24 Stan Shebs address@removed**

**\* configure.host (mips-dec-mach3\*): Use mipsm3, not mach3.**

**Attempt to sort out SCO-related configs.**

**\* configure.host (i[3456]86-\*-sysv4.2\*): Use this instead of**

**i[3456]86-\*-sysv4.2MP and i[3456]86-\*-sysv4.2uw2\*.**

**(i[3456]86-\*-sysv5\*): Recognize this.**

**\* configure.tgt (i[3456]86-\*-sco3.2v5\*, i[3456]86-\*-sco3.2v4\*):**

**Recognize these.**

Obrázok 9: Príklad zápisu ChangeLog-u (http://www.gnu.org/software/guile/changelogs/guile-changelogs\_3.html)

## Analýza implementácie

Tímová spolupráca na projekte vývoja softvéru si vyžaduje vzájomné prepojenie a istú úroveň centralizácie. Takýto model poskytujú webové aplikácie. Vďaka nim sa na serveri môžu zhromažďovať dáta a v kombinácii s verziovacím systémom tvoria základ tímovej spolupráce. Rovnaký prístup je teda možné uplatniť aj pre zobrazovanie, resp. vizualizáciu týchto informácií a metadát. Nutnosť pripojenia na server s databázou verziovacieho systému tvorí predpoklad použitia modelu *client-server*. V tomto prípade sa osvedčila trojvrstvová architektúra, ktorú je možné implementovať na inom serveri, ako sú uložené dáta.

V databázovej vrstve sa k nim dá pristupovať rôznymi technológiami a protokolmi zabezpečujúcimi rýchlu komunikáciu. Databáza, z ktorej bude čerpať moja aplikácia, bola vytvorená a naplnená vzorovými dátami v rámci projektu **PerConIK** (Personalized Conveying of Information and Knowledge) prebiehajúceho na našej fakulte Informatiky a informačných technológií Slovenskej Technickej Univerzity v Bratislave v spolupráci s firmou Gratex. Jedná sa o výskum metód získavania a analýzy personalizovaných informácií a znalostí potrebných pri vývoji softvéru so zameraním na softvérové metriky.

Prezentačná vrstva bude riadiť dotazy od používateľa, vytvárať dynamický obsah a produkovať zobraziteľné dáta.

# Špecifikácia prototypu

Identifikoval som nasledujúce požiadavky na program zobrazujúci dvojicu vzťahu vývojára a zmenu v zdrojovom kóde:

* **Jednoduchý prístup** k zobrazeniu vizualizácie, aby bola okamžite použiteľná pre potreby vývojára
* **Škálovateľnosť** do rozsahu zadaného času, ktorý chce používateľ zobraziť
* Zobrazenie úplného **zoznamu** podieľajúcich sa **vývojárov**
* **Zobrazenie stromu projektu**, aby bolo možné vidieť jeho hierarchiu

## Účel vizualizácie

Moja vizualizácie má ambíciu zobrazovať z iného pohľadu, ako je možné vidieť na vizualizačných nástrojoch rôznych repozitárov. Má sa totižto viac približovať dátam, ktoré sú význačnejšie pre ChangeLog-y. Budem sa pohybovať medzi týmito dvomi pohľadmi, pretože dáta potrebné na zobrazenie budú pochádzať z databázy čerpajúcej z repozitáru TFS.

Obrázok 10 Transformácia dát zo záznamov repozitáru do sémantiky ChangeLog-u

Úlohou je vyextrahovať dáta z databázy, aby som získal dvojice informácií – zmena fragmentu zdrojového kódu a autora zmeny. Všetky tieto dáta by mali byť na jednej obrazovke. Vyžaduje si to teda zoznam mien (príp. aliasov) vývojárov a strom zobrazujúci celý zdrojový kód projektu. K tomuto výrazne pomôže databáza zachytávajúca AST (Abstract Syntax Tree). Koncové listy tohto stromu budú najatomickejšími zachytenými časťami z vývoja zdrojového kódu. Po označení tohto fragmentu, alebo ktoréhokoľvek nadradeného fragmentu by mal používateľ zistiť, ktorí vývojári zasiahli do danej oblasti, a aký typ zmeny vykonali (vytvorenie, zmena, zmazanie). Týmto je možné vidieť práve informáciu o pôvode fragmentu. Vďaka jednoznačným dvojiciam fragmentu zdrojového kódu a vývojára je možné vyhľadať zmeny, ktoré vykonal práve jeden vývojár.

## Technická špecifikácia

Model fungovania celého prototypu sa dá opísať v troch základných komponentoch. Každý z nich pritom bude bežať na samostnaných hardvéroch s odlišnými platformami. Komunikujú vzájomne pomocou štandardizovaných protokolov internetovej komunikácie SOAP a HTTP.

### Databáza AST RCS

Back-end poskytujúci zozbierané informácie o činnosti vývojárov je relačnou databázou s názvom *AST RCS* bežiacou na *Microsoft Internet Information Server* (IIS) serveri.



Obrázok 11 Zbieranie informácií o činnosti programátora do databázy AST RCS [[1]](#footnote-1)

Rozhraním pre prístup k databáze AST RCS je webová služba *AstRcsWcfSvc* dostupná na koncovom bode *http://perconik.fiit.stuba.sk/AstRcs/AstRcsWcfSvc.svc*.

### Serverová aplikácia

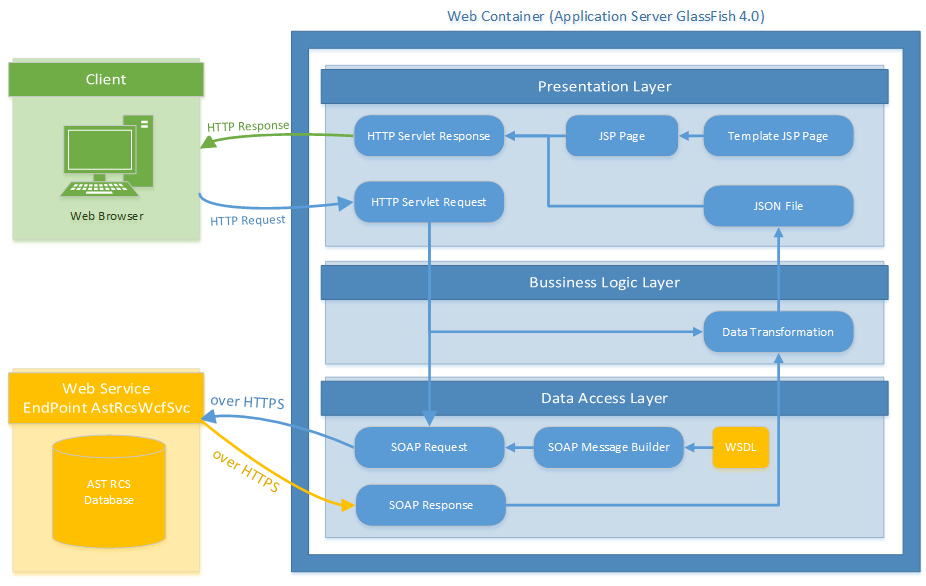
Aplikácia pobeží na Java aplikačnom serveri *GlassFish 4*, čo je aplikácia s otvorenou licenciou implementovaná na JavaEE platforme. Pre účely vývoja prototypu bude tento aplikačný server umiestený priamo na *desktopovej* stanici, kam sa na pripojím na *localhost* na port 8080. Z hľadiska funkčnosti architektúry však ide o úplnú zhodu s reálnym použitím hardvérového serveru s bežiacim aplikačným serverom GlassFish. Bude pozostávať z niekoľkých webových komponentov. Jednou je potrebné zabezpečiť komunikáciu s webovou službou. Ďalší komponent bude dáta z databázy transformovať na JSON *(JavaScript Object Notation)* súbory doťahované Ajax-ovým dopytom na (*Asynchronous JavaScript & XML* - súhrnné označenie pre technológie vývoja interaktívnych webových aplikácií, ktoré umožňujú meniť obsah stránok bez potreby ich kompletného znovunačítania zo servera) na základe požiadavky používateľa s internetového prehliadača. Prezentačný komponent vyskladá dynamicky generovanú webovú stránku založenú na HTML *(HyperText Markup Language)*, JavaScript, pomocou technológie JSP *(Java Server Pages)*.

### Vizualizačná knižnica D3.js

*JavaScript*-ová knižnica D3 slúži na manipuláciu dát na webovej stránke založenej na vstupných dátach. Používa technológie HTML, SVG *(Scalable Vector Graphics)*, CSS *(Cascading Style Sheets)* a dátovo riadený DOM *(Document Object Model)* prístup k  manipulácií s objektmi HTML. Selektory definované konzorciom W3C v *Selectors API*[[2]](#footnote-2)sú natívne podporované všetkými modernými prehliadačmi, preto nehrozí nekompatibilita. Štýly, atribúty a iné nastavenia objektov môžu byť špecifikované funkciou, nie len jednoduchými konštantami . V tomto nachádzam podobnosť s knižnicou *jQuery,* vďaka ktorej je možné urobiť vizualizáciu dynamickú, atraktívnu, a rozšíriteľnú o ďalšie prvky metadát. Knižnica vychádza priamo z webových štandardov, je preto dobre spravovateľná a pochopiteľná. Je tu možnosť štýlovať elementy externými štýlovacími súbormi CSS, čo umožňuje mojej vizualizačnej aplikácii neskôr dorobiť rozhranie pre správu vlastného zobrazovania. V reálnej komerčnej aplikácií by to dovolilo upravovať vizualizáciu bez zásahu do kódu podľa požiadaviek klienta.

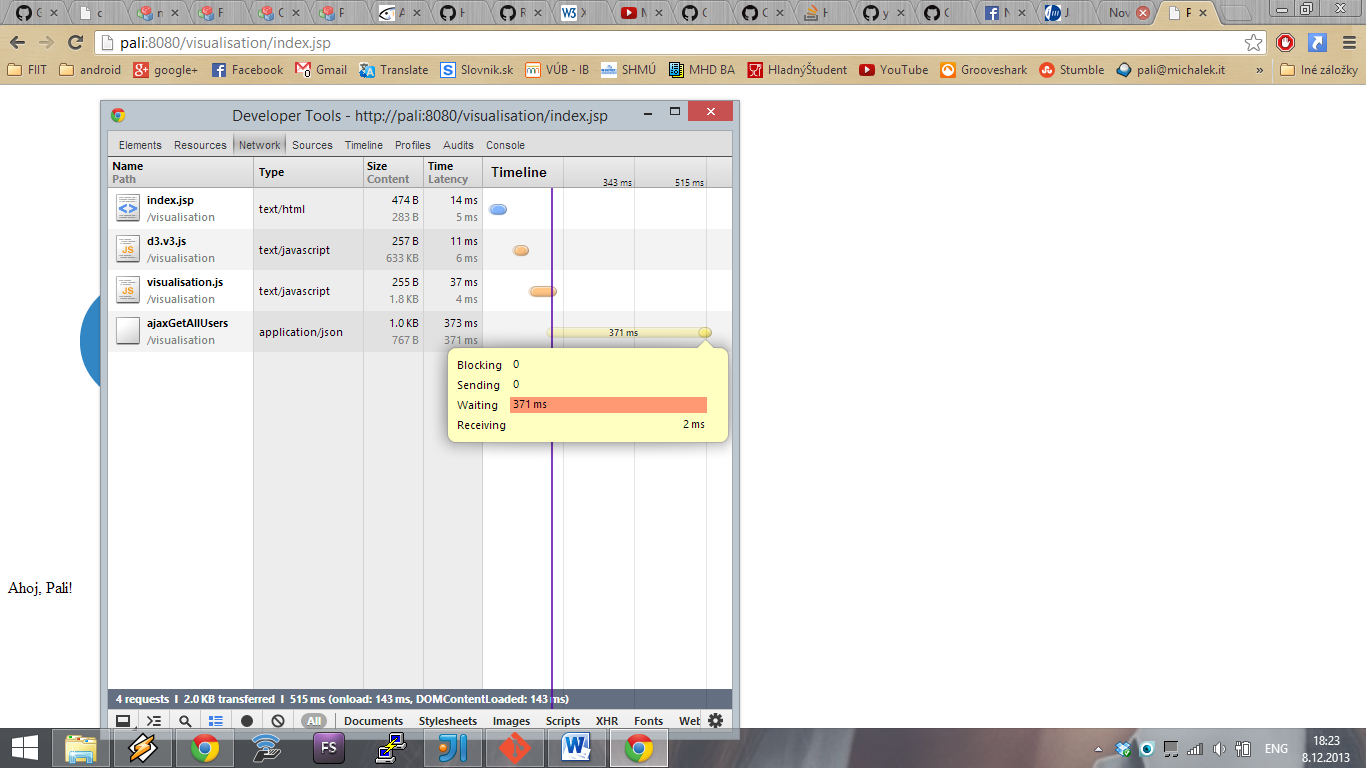
# Softvérový návrh

V tejto časti opíšem fungovanie mnou vytvorenej časti softvéru, ktorá zahŕňa serverovú aplikáciu napojenú na webovú službu *AstRcsWcfSvc.* Aplikácia je spustená HTTP dopytom na URL, kde sa nachádza jej domovská stránka *index.jsp.*



Obrázok 12 Model fungovania aplikácie

Na obrázku 13 je možné vidieť priebeh HTTP dopytov na aplikačný server z klientskeho prehliadača pri volaní stránky s prototypom mojej aplikácie. Odpoveď servera s domovskou stránkou vo formáte JSP, vizualizačná JavaScript-ová knižnica D3 a samotná vizualizácia tiež v JavaScript-e sa prijímala v primeraných časoch v rádoch desiatok milisekúnd. Zvislá fialová čiara na časovej osi znázorňuje spustenie Ajax-ového dopytu, teda požiadavky na server na dotiahnutie samotných dát, ktoré zavolal môj vizualizačný JavaScript. V tomto uvedenom čase 371ms, ktorý sa dá považovať za priemerný čas vykonávania tejto akcie, prijal servlet požiadavku na dotiahnutie všetkých dát potrebných na zobrazenie vizualizácie v prototype. Ten vytvoril spojenie s webovou službou *AstRcsWcfSvc* a vykonal SOAP (Simple Object Access Protocol) dopyt. Bol to malý vzorový objem dát z databázy AST RCS, konkrétne zaslanie všetkých informácií o používateľoch participujúcich na projektoch zaznamenaných v databáze. Samotné prijímanie dát sa uskutočnilo za zanedbateľné 2ms. Vtedy bol vizualizačný JavaScript schopný pokračovať vo vykonávaní zobrazenie



Obrázok 13 Časová os priebehu HTTP dopytov z klientskeho prehliadača

Podstatnou časťou Java serverovej aplikácie je transformácia prijatých dát s webovej služby na výsledné JSON objekty, ktoré je schopné vizualizačná knižnica čítať. Vďaka opisnému rozhraniu WSDL (Web Services Description Language), ktoré vystavila webová služba *AstRcsWcfSvc*, som pomocou nástroja *wsimport* (štandardne dodávany v rámci *Java Development Kit 1.6*) vygeneroval Java triedy poskytujúce webové metódy potrebné na vytváranie SOAP dopytov na webovú službu a triedy typu *DTO* (Data transfer object) uchovávajúce odpovede do Java objektov. Knižnica *JSON.simple* poskytla metódy na vytváranie JSON objektov, ktoré som namapoval dátovými objektmi. Ako HTTP odpoveď som tento JSON objekt poslal zo servletu spravujúceho aj AJAX dotazy.

Záver

Analyzoval som rôzne prístupy k zobrazeniu veľkého množstva dát zhromaždených počas vývoja softvéru a písania zdrojového kódu pomocou nástrojov revíznej kontroly a sledovača aktivít vývojára. Aj keď som sa zameral na oblasť závislosti zmeny zdrojového kódu a programátora, v ktorej už existujú mnohé vizualizácie aj od veľkých komerčných firiem, myslím si, že je tu priestor zobrazovať tieto dáta inak. Vhodnou transformáciou dát z repozitárov sa totižto dá pozrieť na zmeny fragmentov zdrojového kódu z pohľadu lokalizácie danej zmeny v chronologickom poradí, tak ako to zachytávajú ChangeLog-y, namiesto existujúcich zobrazení podľa commit-ov zmien programátorom do repozitáru. Tento pohľad sa pokúsim premietnuť aj do finálneho produktu vizualizácie, ktorá by mala priniesť prehľadný spôsob zobrazenia vývoja zdrojového kódu viacerými programátormi. V prototype som si otestoval funkčnosť použitej trojvrstvovej architektúry a vhodnosť vybranej vizualizačnej knižnice *D3.js* .

Zoznam použitej literatúry

1. *Understanding Multi-Dimensional Code Couplings.* **Beck, Fabian.** University of Trier : s.n., 2013. PhD thesis.

2. **Bederson, Benjamin B and Shneiderman, Ben.** *The craft of information visualization: readings and reflections.* San Francisco, CA, USA : Morgan Kaufmann, 2003. ISBN: 978-1-55860-915-0.

3. *Visual information seeking using the FilmFinder.* **Ahlberg, Christopher and Shneiderman, Ben.** New York, NY, USA : ACM, 1994. s. 2.

4. *Using origin analysis to detect merging and splitting of source code entities.* **Godfrey, Michael W and Zou, Lijie.** s.l. : IEEE, 2005, Software Engineering, IEEE Transactions on, Zv. 31, s. 166-181.

5. **Clark, C. Y. Baldwin and K. B.** Design Rules. *The MIT Press.* 1st, 2000, Zv. Vol. 1: The Power of Modularity.

6. **Greguš, Peter.** Improving code quality by using refactoring techniques. [aut. knihy] PhD. prof. Ing. Mária Bieliková. *Essays collection.* 2013.

7. *Reducing coupling.* **Fowler, Martin.** 2001, IEEE Software, Zv. 18.

8. **Beck, Fabian and Diehl, Stephan.** On the congruence of modularity and code coupling. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.* s.l. : ACM, 2011.

9. *Software Architecture Recovery Using Conway's.* **Holt, I. T. Bowman and R. C.** s.l. : IBM Press, 1998. Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research. s. 6.

10. **Roy, C. K. a Cordy, J. R.** *A Survey on Software Clone Detection Research.* Kingston : Queen's University, 2007.

11. *The Conceptual Coupling Metrics for Object-Oriented Systems.* **Poshyvanyk, D. a Marcus, A.** s.l. : IEEE Computer Society, 2006. Proceedings of the 22nd International Conference on Software Maintenance.

12. **Blandy, Jim.** http://www.gnu.org/software/guile/changelogs/guile-changelogs\_toc.html. *www.gnu.org.* [Online] http://www.gnu.org/software/guile/changelogs/guile-changelogs\_toc.html.

Prílohy

1. zdroj:  Výskum PeWe (Personalized Web), Slovenská technická univerzita v Bratislave, http://wiki.fiit.stuba.sk/research/seminars/pewe/program-2013-2014/PerConIK\_data.pdf [↑](#footnote-ref-1)
2. http://www.w3.org/TR/selectors-api/ [↑](#footnote-ref-2)