Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

# ANALÝZA VÝSLEDKOV VÝSKUMU

## Dokumentácia k riadeniu projektu

**Tím 10** ResearchRank
**Vedúci projektu:** Ing. Nadežda Andrejčíková, PhD.
**Členovia tímu:** Bc. Michael Gloger
Bc. Rastislav Kostrab
Bc. Šimon Kompas
Bc. Tomáš Jánošík
Bc. Daniel KĺČ
Bc. Stanislav Kubica
Bc. Michal Walder
**Školský rok 2013 / 2014**

# 1 Úvod

Táto dokumentácia bola vytvorená za účelom sumarizácie všetkých dokumentov súvisiacich s riadením projektu v našom tíme na predmete Tímový projekt. Všetky dokumenty boli tvorené priebežne a sumarizujú čas a obsah vynaloženej práce na projekte.

## 1.1 Prehľad dokumentu

1. Ponuka
2. Úlohy členov
3. Metodiky
4. Zápisnice zo stretnutí (príloha A)

V prvej časti dokumentu sú predstavení členovia nášho tímu a sú tu uvedené ponuky pre tri vybrané témy, usporiadané podľa priority. V druhej časti sú uvedené manažérske role členov tímu a zodpovednosti každého z nich. Tretia časť obsahuje metodiky - pravidlá vykonávania jednotlivých procesov v rámci kontextu tímového projektu. Medzi prílohy sú zaradené zápisnice z tímových stretnutí.

## 1.2 Autori jednotlivých častí dokumentácie

| **Kapitola** | **Autor** |
|---|---|
| 1 Úvod | Bc. Michael Gloger |
| 2 Ponuka | Celý tím |
| 3.1 Manažment komunikácie | Bc. Michael Gloger |
| 3.2 Manažment monitorovania | Bc. Tomáš Jánošík |
| 3.3 Manažment rizík | Bc. Stanislav Kubica |
| 3.4 Manažment plánovania | Bc. Rastislav Kostrab |
| 3.5 Manažment podpory vývoja | Bc. Šimon Kompas |
| 3.6 Manažment dokumentácie | Bc. Michal Walder |

| | |
|---|---|
| | Bc. Daniel Kĺč |
| 3.7 Manažment kvality | Bc. Daniel Kĺč |
| 4.1 Metodika tvorby user stories | Bc. Michael Gloger |
| 4.2 Metodika manažmentu zberu požiadaviek | Bc. Michael Gloger |
| 4.3 Metodika vybraných procesov čitateľnosti a konvencií kódu | Bc. Daniel Kĺč |
| 4.4 Metodika testovania | Bc. Daniel Kĺč |
| 4.5 Metodika manažmentu vývoja modulov | Bc. Daniel Kĺč |
| 4.6 Metodika manažmentu chýb | Bc. Stanislav Kubica |
| 4.7 Metodika evidencie úloh | Bc. Rastislav Kostrab |
| 4.8 Metodika dokumentácie zdrojových súborov | Bc. Michal Walder |
| 4.9 Metodika verziovania | Bc. Šimon Kompas |
| 4.10 Metodika prípravy podkladov na stretnutie so zákazníkom | Bc. Tomáš Jánošík |
| 4.11 Metodika iterácií projektu | Bc. Tomáš Jánošík |
| 4.12 Metodika manažmentu chýb | Bc. Stanislav Kubica |
| 4.13 Metodika plánovania úloh | Bc. Rastislav Kostrab |
| 4.14 Metodika manažmentu | Bc. Šimon Kompas |

| | |
|---|---|
| projektového repozitára a verzií | |

# 2 Ponuka

## 2.1 Predstavenie tímu

### 2.1.1 Bc. Michael Gloger

sa orientuje na tvorbu webových a mobilných aplikácií pre platformu Android. Pracuje s Oracle, PostgreSQL, webovými službami a s webovou grafikou. Jeho preferovaný jazyk je Java.

### 2.1.2 Bc. Rastislav Kostrab

sa orientuje na technológie rozsiahlych informačných systémov. Okrem toho sa zaujíma aj o vývoj natívnych aplikácií pre platformu Android. Preferuje open-source technológie a jeho silnou stránkou je programovací jazyk Java. Okrem toho ovláda prácu s databázovými technológiami PostgreSQL a MySQL.

### 2.1.3 Bc. Šimon Kompas

ovláda technológie ako Java, SQL databázy, trocha menej GWT, spring a ďalšie biznis technológie spojené s jazykom Java. Popri štúdiu sa venuje grafovým algoritmom nad sieťami malého sveta a po novom ho začala zaujímať bioinformatika, konkrétne oblasť DNA a skladania fragmentov do celku. Po technologickej stránke chce obohatiť svoje vedomosti o funkcionálny jazyk Scala, ktorý je možné kombinovať s klasickou Javou a je spustiteľný na JVM.

### 2.1.4 Bc. Tomáš Jánošík

sa venuje databázovým technológiám, vývoju aplikácii využívajúcich veľké dáta a spracovaniu týchto dát sofistikovanými algoritmami. Skúsenosti má aj s vývojom webových aplikácii a komunikáciou cez Internet, či cez iné siete.

### 2.1.5 Bc. Daniel Kĺč

sa orientuje popri škole na refaktorizáciu starších a vývoj nových algoritmov v oblasti experimentálnej a jadrovej fyziky na OJF FÚ SAV. V diplomovej práci sa chce venovať v rámci bioinformatiky sekvenovaniu DNA. Špecializuje sa na paralelizáciu a aspekty implementácie spojené so zložitosťou algoritmov a jej redukciou. Vyvíja predovšetkým v jazyku C++.

### 2.1.6 Bc. Stanislav Kubica

sa orientuje na technológie ako Java, SQL databázy, PHP, JavaScript (jQuery), HTML5, CSS3 a iné webovo orientované technológie. Ovláda databázy MySQL a PostgreSQL. Jeho preferovaným jazykom je Java, prípadne PHP ak sa jedná o webové aplikácie.

### 2.1.7 Bc. Michal Walder

sa orientuje hlavne na jazyk Java, na vývoj J2EE aplikácií ale aj mobilných aplikácií pre platformu Android. Orientuje s v Enterprise aplikáciách pre platformu Java a ovláda technológie ako JSF, Struts, EJB, JPA a pod. Ovláda databázy MSSQL, MySQL i Oracle. Je orientovaný na open-source riešenia.

## 2.2 Ponuka 1: Analýza výsledkov výskumu

### 2.2.1 Návrhy na implementáciu:

Téma ohodnocovania výsledkov výskumu je zaujímavá téma podstatná pre správne rozdeľovanie prostriedkov na výskum, meranie určitej prestíže a v neposlednej rade i zaujímavosti témy, ktorej sa ten ktorý výskum venuje. Tento problém sa dotýka aj nás študentov, pretože každý čerpal z nejakého zdroja informácií, ideálne z nedávneho výskumu. Je teda zaujímavé pre študenta vedieť, či zdroj, z ktorého sa chystá čerpať je hodnoverný, uznávaný i aktuálny.

Podstatnou vlastnosťou nášho tímu je komplexnosť s akou by sme dokázali pristúpiť k problému. Máme v tíme ľudí, ktorí sú schopní a nadšení pre matematickú stránku algoritmov rozlišovania medzi entitami, máme ľudí, ktorí dokážu prísť s inovatívnou vizualizáciou dát a taktiež skúsenosti s veľkým objemom dát.

Zaujímavé by bolo odhaľovať a graficky reprezentovať rôzne závislosti medzi dokumentami, pričom faktorov je veľa a je potrebných veľa experimentov. Je zaujímavé zistiť, aké závislosti ovplyvňujú zvýšený výskyt daného dokumentu v citačných indexoch, čo by malo byť spravidla hlavne kredit jeho autora, avšak môžeme nájsť aj závislosti nečakané. V oblasti riešenia nejednoznačnosti existuje mnoho algoritmov, z ktorých by sme chceli vybrať vhodný pre naše účely. Možnosťou bolo riešiť problém nejednoznačnosti sémantickým prístupom, avšak naše úsilie by sme chceli venovať skôr iným aspektom.

Chceli by sme vytvoriť systém, ktorý by bol nielen schopný údaje získavať a vyhodnocovať, ale taktiež aj prezentovať výsledky ohodnocovania koncovým používateľom, aby sme tak pomohli používateľom pri hľadaní dokumentov.

Systém by sme mohli vylepšiť o grafickú vizualizáciu dokumentov v podobe grafov.

Graf by reprezentoval, ktoré dokumenty majú referenciu na iné dokumenty, a takisto by reprezentoval čas ich vytvorenia a relevantnosť k danej téme, čo by sprehľadnilo evidenciu a vyhľadávanie zdrojov pri výskume. Podobne by bolo možné spraviť aj vyhľadávanie.

## 2.3 Ponuka 2: Virtuálna FIIT na mobile

### 2.3.1 Cieľ práce:

Navrhnúť a implementovať mobilnú aplikáciu pre platformy Android, Windows phone, iOS ako aj web, ktorá by so svojím používateľským rozhraním a funkcionalitou zodpovedala požiadavkám každého študenta našej školy. Cieľom je zjednodušiť hľadanie informácií týkajúcich sa štúdia a skrátiť čas ich vyhľadávania.

### 2.3.2 Výstup z práce:

Multiplatformová webová aplikácia zabalená pomocou PhoneGap–u do natívnych aplikácií. Vzhľadom na to, že používanie notebookov na našej škole je veľmi bežné, tak sme premýšľali sme aj nad čisto webovou variantou informačného systému, ktorý by sa od svojej mobilnej verzie líšil používateľským rozhraním prispôsobeným pre väčšie obrazovky.

### 2.3.3 Cieľoví používatelia (Personas):

Študenti, učitelia, návštevníci školy, …

Počas vývoja aplikácie by sme chceli prototyp testovať za pomoci jej budúcich používateľov. Takýmto spôsobom by sme zhromažďovali pripomienky a skúmali spôsob interakcie ľudí s navrhnutým používateľským rozhraním.

### 2.3.4 Manažment tímu

Pre zvýšenie efektivity práce plánujeme použiť známy nástroj pre riadenie rozdeľovania úloh v tímových projektoch: JIRA. Svoje zdrojové súbory, dokumentácie a iné digitálne dáta súvisiace s našou prácou budeme medzi zdieľať pomocou verziovacieho systému GIT (resp. podľa potreby a možností SVN).

### 2.3.5 Nápady pre zlepšenie funkcionality:

- offline synchronizácia (nie každý má mobilný internet, či eduroam konto)
- informácie o jedálňach, učebniach, študijnom oddelení, ...
- umožnenie vyhľadávania profesora v budove (podľa jeho rozvrhových akcií)
- umožniť používateľom pridávať skratky na jednotlivé funkcionality, ktoré využíva najčastejšie. Podobne ako hlavná stránka nášho AIS-u, alebo HOME screen smartfónov.
- prehľadné pozeranie osobného rozvrhu (prípadne rozvrhov kamarátov)
- rýchle dohľadanie informácií k jednotlivým predmetom (prípadne prelinkovanie s prácami ostatných tímov - napríklad CQA komunitný systém otázok a odpovedí).
- pripomienkovanie (notifikácie) k vybraným rozvrhovým akciám
- push notifikácie pre rôzne školské eventy (info o prihlasovaniach na rozvrhové akcie... a iné informácie bežne oznamované iba mailom v rámci AIS)
- mapa školy
- lokalizácia pomocou QR kódov rozmiestnených po škole, zobrazenie cesty do učební

### 2.3.6 Potenciál aplikácie

Myslíme, že naša aplikácia by bola úspešná medzi študentmi a zamestnancami fakulty z toho dôvodu, že sami sme študenti a z vlastnej skúsenosti poznáme ako je občas problematické pristupovať k niektorým informáciám ohľadom štúdia.

To neznamená, že AIS je zlý, ale že veľa informácií nie je na tom istom mieste. Čas vyhľadávania sa tak výrazne zvyšuje až kým sa študent radšej opýta svojich kamarátov. Informácie z druhej ruky však už nemusia byť presné a študent si ich nemusel hneď zapísať na prehľadné miesto, kde by ich opäť rýchlo vedel dohľadať.

Chceli by sme aj svojich spolužiakov osloviť (napríklad pomocou ankety) aby sme zistili, aký typ informácií je pre nich najdôležitejší a tieto informácie by sme zaradili medzi tie najľahšie dostupné (počet kliknutí musí byť vždy minimálny- tj žiadne zbytočné ovládanie kt človek nechce používať). Dôležitou funkcionalitou bude možnosť používateľov nastaviť si, ktoré informácie ho zaujímajú a chce ich mať vždy rýchlo prístupné, prípadne ktoré chce mať schované.

## 2.4 Ponuka 3: Distribuované počítanie na FIIT

Náš cieľ by bolo sprístupniť FIIT GRID a otestovať ho (resp. jeho výkon v rôznych kategóriách). Sprístupniť obojstranne, tj. kontribútorom , ale aj konzumentom výpočtových prostriedkov. Chceli by sme sprístupniť počítanie na desktopoch/serveroch ale aj na mobilných zariadeniach.

V tíme máme členov, ktorí už majú skúsenosti jak s Linuxom tak s Androidom. Bolo by vhodné vytvoriť klienta pre desktop počítače/mobilné zariadenia, ktorý fyzicky nastaví a skonzumuje lokálne výpočtové prostriedky, ale aj webové rozhranie, kde sa konzumenti môžu jednoducho uchádzať o výpočtové prostriedky a na druhej strane kontribútori môžu jednoducho stiahnuť klienta na výpočet.

Funkcionalita klienta môže napríklad zahrňovať čas kedy budú prostriedky zapožičané, objem prostriedkov, alebo napríklad v mobilných zariadeniach aj určenie jednotlivých prostriedkov, ktoré im budú k dispozícii.

Funkcionalita webu môže obsahovať napríklad grafické zobrazenie mapy kontribútorov alebo metriky systému, možnosť stiahnuť si klienta alebo formulár potrebný na uchádzanie sa o prostriedky.

Otestovanie by malo prebehnúť nad reálnym problémom a výpočet by pokiaľ možno mal byť zmysluplný. Je možnosť osloviť týmto smerom napríklad Fyzikálny ústav SAV kde jeden z členov nášho tímu pracuje a získať od nich nejaké reálne, užitočné a dostatočne náročné úlohy. Ďalšou možnosťou je zamyslieť sa nad využitím pri riešení simulačných problémov, kde by sa dal využiť fakt, že údaje sú známe všetkým agentom, potrebné je však pre každého agenta počítať stratégiu. Uplatnenie = simulácia davu, simulácia rannej špičky v MHD...

Pri realizácií tohto projektu je taktiež nutné uvažovať nad teoretickými aspektami distribúcie výpočtov, nech sa maximalizuje využitie zapožičaných prostriedkov.

Vzhľadom na to, že tento projekt je predovšetkým orientovaný na OS Linux, Unix, (prípadne Android a iOS), má zmysel uvažovať nad spôsobom, ako by sa s čo možno najmenším úsilím do GRIDu dali zapojiť počítače s OS windows (a tým pádom by boli napojiteľné do GRIDu aj počítačové učebne lokalizované na FIIT).

K týmto cieľom je možne využiť naše individuálne schopnosti a skúsenosti, ale napríklad je možnosť získať odpovede na konkrétne otázky ľudí ktorý prispievali, konzumovali a vyvíjali CERN GRID, s ktorými sa jeden člen nášho tímu osobne pozná.

## 2.5 Príloha A: Priorita výberu tém

1. Virtuálna FIIT na mobile
2. Analýza výsledkov výskumu
3. Distribuované počítanie na FIIT
4. Webový komunitný systém otázok a odpovedí
5. Digital SweatShop
6. Trojdimenzionálne UML
7. Vizualizácia informácií v obohatenej realite
8. Zábavný systém pre spolucestujúcich v automobile
9. Interaktívne hry na mobile s multimediálnym obsahom
10. Monitor programátora v IDE
11. Prehliadka kódov v tímových projektoch
12. Robotický futbal
13. Sledovanie pohľadu pri používaní aplikácií

## 2.6 Príloha B: Rozvrh členov tímu vyhradený na tímové stretnutia

| **Meno** | **PON** | **UTO** | **STR** | **ŠTV** | **PIA** |
|---|---|---|---|---|---|
| Rastislav Kostrab | 13 - 16 | 15 + | do 15 | 9 - 15 | - |
| Michael Gloger | - | 15 + | 11 - 13/14 | do 15 cca | dohodou |
| Stanislav Kubica | 13 - 16 | 15 + | do 15 | 9 - 15 | - |
| Michal Walder | 18 + | 15+ | do 13 | 11 - 15 | dohodou |
| Tomáš Jánošík | do 14 | do 11 U 15+ | do 13 | 11 - 15 | dohodou |
| Daniel KĺČ | do 14 | do 11, od 15 do 18 | do 13 | 11 - 15 | dohodou |

| | | | | | |
|---|---|---|---|---|---|
| Šimon Kompas | 12-16 U 18 + | 15 + | 11-15 | 18 + | - |
| **Prienik (cca)** | **-** | **15 +** | **11 - 14** | **11-14** | **-** |

# 3 Úlohy členov v tíme

Členovia tímu zastávajú nasledovné manažérske role:

| **Meno** | **Rola** |
|---|---|
| Bc. Michael Gloger | vedúci tímu |
| Bc. Rastislav Kostrab | manažér plánovania |
| Bc. Šimon Kompas | manažér podpory vývoja |
| Bc. Tomáš Jánošík | manažér monitorovania projektu |
| Bc. Daniel Kĺč | manažér kvality<br>manažér dokumentácie |
| Bc. Stanislav Kubica | manažér rizík |
| Bc. Michal Walder | manažér dokumentácie |

## 3.1 Manažment komunikácie

**Zodpovedná osoba:** Michael Gloger

Úloha manažéra komunikácie v našom tíme je vybrať vhodné komunikačné a kolaboračné nástroje a zabezpečiť aby ich členovia tímu vedeli efektívne využívať. Ďalšou hlavnou úlohou je organizovať tímové stretnutia a pripraviť si plán na každé stretnutie.

### 3.1.1 Stretnutia k tímovému projektu

V rámci tímového projektu sa konajú dva druhy stretnutí:

- Pravidelné stretnutia s vlastníkom produktu, pani Andrejčíkovou (pravidelne každý štvrtok od 11:00)
- Stretnutia tímu (v skole, niekoľko krát do týždňa)

V rámci pravidelných stretnutí s vedúcim sa preberá plnenie jednotlivých úloh a plánujú sa nové. Na stretnutiach sa zúčastňoval celý tím. Na stretnutiach s vlastníkom produktu sa zúčastňuje celý tím a trvajú približne 2 - 3 hodiny. Spoločné stretnutia mali nasledovný priebeh:

1. Vedúci tímu zhrnul plán na aktuálne stretnutie a uviedol aké výstupy sa z neho očakávajú. V tejto fáze sa jednotlivé body ešte nerozoberali
2. Vedúci tímu zhrnul vlastníkovi produktu čo sa stihlo od posledného stretnutia a dal priestor členom tímu vyjadriť sa jednotlivým bodom, podľa toho kto na čom pracoval.
3. Zhrnuli sa veci, na ktorých sa bude pracovať najbližšie
4. Vedúci tímu pokračoval s plánom stretnutia podľa potreby a diskutoval s členmi tímu a vlastníkom produktu

Stretnutia členov tímu sa konali 2-3 krát do týždňa vo forme stand-up meetingov. Na týchto stretnutiach členovia tímu konzultovali svoju prácu a svoje výstup s vedúcim tímu a s ostatnými členmi.

### 3.1.2 Komunikačné nástroje

Väčšina komunikácie prebiehala hlavne v rámci tímových stretnutí, avšak občas je nutné veci rieši'' aj pomocou online komunikačných a kolaboračných nástrojov. V rámci našej práce sme používali nasledovné nástroje:

#### *3.1.2.1 Email*

Použitý na komunikáciu v rámci celého tímu a komunikáciu s vlastníkom produktu. Pre komunikáciu bol použitý spoločný email tp-1314-10@googlegroups.com.

#### *3.1.2.2 Hangout (Google talk)*

Použitý na rýchle dohadovanie sa medzi členmi tímu. Využívaný najmä medzi členmi, ktorí na úlohách kolaborovali.

#### *3.1.2.3 Slack*

Slack je služba, ktorá ponúka tímom spoločnú komunikáciu vo forme webového rozhrania a mobilných aplikácií. Výhoda využívania tohto skupinového chatu je to, že konverzácie môžu byť rozdelené do rôznych kanálov a je možné jednoducho v týchto konverzáciách vyhľadávať. Rozhodli sme sa pre Slack namiesto Skype z toho dôvodu, že Skype v súčasnosti nepodporuje ukladanie správ na serveri a možnosti komunikácie sú tu obmedzené.

### 3.1.3 Kolaboračné nástroje

Pre zefektívnenie práce bolo nutné využívať kolaboratívne nástroje, vďaka ktorým vedeli členovia tímu spolupracovať na úlohách aj keď práve nesedeli pri jednom počítači. Medzi tieto úlohy patrí napríklad písanie dokumentácie a zdrojových kódov.

#### *3.1.3.1 Google Drive*

Manažér komunikácie vytvoril v Google Drive priečinok, do ktorého majú prístup všetci členovia tímu. V tomto priečinku sa nachádzajú všetky dokumentácie na ktorých sme potrebovali kolaboratívne pracovať.

#### *3.1.3.2 JIRA*

Pomocou JIRA evidujeme svoje šprinty a úlohy. Každá úloha má názov, popis prideleného člena tímu a iné atribúty, na základe ktorých vieme svoje úlohy triediť. Tento nástroj nám tiež umožňuje robiť reporty práce. Každý člen je povinný vykazovať si čas, ktorý danej úlohe venoval a písať sem reporty svojej práce. V každom reporte je uvedené čo stihol, čo nestihol a aký je záver z jeho práce.

#### *3.1.3.3 Stash*

Stash poskytuje git repozitáre, ktoré sú priamo integrované s JIRA. Pri commitovaní stačí do sprievodnej správy zadať číslo úlohy a JIRA automaticky tento commit spáruje s danou úlohou.

#### *3.1.3.4 Confluence*

V Confluence vytvárame wiki stránky k jednotlivým problematikám, ktoré sme riešili. Každý člen je počas svojej práce do týchto stránok pridávať výstup svojej práce aby ho ostatní členovia odtiaľ mohli čítať.

#### *3.1.3.5 Crucible + FishEye*

Ďalší nástroj integrovaný s JIRA. Slúži na vytváranie review kódu k obsahom repozitárov.

## 3.2 Manažment monitorovania

**Zodpovedná osoba:** Tomáš Jánošík

Manažér monitorovania projektu je zodpovedný za viditeľnosť stavu projektu, za to, aby poskytoval informácie o tomto stave, za vykazovanie plnenia úloh a za kompletnosť výstupov.

V rámci tejto funkcie teda dochádza ku kontrole ostatných členov tímu, aby zaznamenávali svoje výstupy do prostredia na manažment úloh (JIRY). Vďaka tomuto nástroju máme mnohé činnosti zautomatizované a je jednoduchšie vykazovať výsledky práce. Takúto kontrolu ostatných členov vykonáva manažér monitorovania projektu v súčinnosti s vedúcim tímu. Počas uplynulého času boli priebežne kontrolované úlohy, aby boli kompletné, boli monitorované z hľadiska stavu, množstva vykonanej práce a množstva práce potrebnej na dokončenie.

Tieto metriky boli brané do úvahy v zmysle počtu hodín odhadovanej práce a vykázanej práce. Na podporu týchto metrík slúžia tri atribúty každej úlohy obsiahnuté v JIRE v sekcii Time Tracking:

- Estimated - odhadovaný čas potrebný na splnenie úlohy
- Remaining - zostávajúci čas potrebný na splnenie úlohy
- Logged - čas spotrebovaný na plnenie úlohy

Na základe týchto metrík sa dá pomerne dobre usudzovať, v akej fáze sa daná úloha nachádza. Podpore predstave o stave úlohy napomáha taktiež percentuálne vyjadrenie týchto metrík taktiež obsiahnuté v JIRE.

Okrem týchto metrík bolo monitorovanie stavu úloh realizované na stretnutiach tímu, kde boli členom tímu kladené otázky ohľadne postupu na projekte. Túto časť sme preberali doteraz veľmi neformálne, pričom sme z týchto informácii nevyvodzovali dôsledky, preto by sme chceli túto oblasť viac prepracovať a z výsledkov monitorovania vyvodzovať dôsledky.

Monitorovanie stavu projektu prebiehalo pravidelne na stretnutiach so zákazníkom (pedagogickým vedúcim tímu), kde mu boli odprezentované výsledky práce, ktoré boli následne zákazníkom zhodnotené. V tomto smere môžeme zhodnotiť postup prác ako uspokojivý, pretože vyriešené úlohy, ako vidno zo záznamov zo stretnutí, dosahovali vždy aspoň polovicu všetkých úloh.

| **Šprint** | **Počet úloh splnených** | **Počet úloh čiastočne alebo vôbec nesplnených** |
|---|---|---|
| 1 “Amstel” | 3 | 2 |
| 2 “Budweiser” | 4 | 1 |
| 3 “Carlsberg” | 7 | 7 |
| 4 “Duff” | 11 | 2 |

Tretí šprint bol z hľadiska náročnosti ostatných predmetov najťažší, preto sme dosiahli len polovičnú úspešnosť vyriešených úloh. Taktiež sme sa v tomto šprinte rozhodli prehodnotiť dôležitosť niektorých úloh tak, aby sme mali hotový prototyp a aby sme ho podľa zásad agilného vývoja ponúkli zákazníkovi čo najskôr ponúkli na vyskúšanie.

V šprinte 4 sme teda splnili takmer všetky úlohy, ktoré boli zadané, vďaka prístupom, ktoré sme sa naučili, kvalitnému plánovaniu a dobrému pracovnému prístupu.

Na skvalitnenie a podporu priebehu stretnutia so zákazníkom bola vytvorená metodika prípravy na stretnutie, aby toto stretnutie prebiehalo hladko a riešili sa dôležité otázky týkajúce sa ďalšieho vývoja. Táto metodika

má za cieľ odstrániť nedostatky v komunikácii tímu ohľadne stavu ostatných úloh, pretože toto je podstatné pri udržiavaní si predstavy o veľkom obrázku projektu.

V rámci spomínanej metodiky boli vyvinuté postupy prípravy na stretnutie, kde využívame podporné nástroje, ako je JIRA, pričom tieto nástroje sme sa najprv museli naučiť používať, preto výsledky zautomatizovaných a metodicky pokrytých procesov sme očakávali v neskorších šprintoch. Niektoré nástroje sme sa používať naučili, naopak nástroj na prehliadky kódu Fisheye sme doteraz využili len málo.

Využívané mali byť grafy podporujúce transparentnosť stavu projektu. Tieto grafy zobrazujú vyriešené úlohy v zmysle prioritizácie úloh a v opozícii k nevyriešeným úlohám s vysokou prioritou. Bližšie sú tieto grafy rozobrané v spomínanej metodike. Používali sa ako príprava na stretnutie a ako základ prezentovaných splnených a nesplnených úloh.

Dôležitým bodom podpory monitorovania projektu bolo výraznejšie využívanie tabuľky pri tímových stretnutiach, ako aj model architektúry systému znázorňujúci hotové a nehotové komponenty. Na poslednom stretnutí so zákazníkom bol tento model prezentovaný vo forme A2, čo pomohlo pri komunikácii so zákazníkom a pri prezentovaní výsledkov našej práce.

Tieto dva aspekty pomohli v rámci tímu udržiavať "veľký obraz" o projekte, pomohli identifikovať kritické komponenty, ktoré musia byť implementované čo najskôr, ako aj zlepšiť komunikáciu medzi tímovými kolegami, ktorí mali na starosti podobné úlohy.

Na stretnutiach so zákazníkom boli uskutočnené zistenia, že výstupy prác jednotlivých členov nie sú dostatočne popísané v systéme na evidenciu takýchto výstupov (Confluence). často tu chýbali podstatné údaje, ktoré by mali byť verejné pre všetkých členov tímu, avšak nachádzali sa len v privátnej podobe u človeka, ktorý tieto výstupy vyprodukoval.

Naviac bola podoba výstupov neformálna, preto sme v metodike prípravy pokryli aj kontrolu zapísania výstupov do Confluence, aby tak mali ostatní členovia tímu prístup k podstatným informáciám pri čo najmenšom úsilí.

## 3.3 Manažment rizík

**Zodpovedná osoba:** Stanislav Kubica

Úlohou manažéra rizík je identifikovať a predvídať problémy, ktoré môžu negatívne ovplyvniť vývoj projektu, prípadne ohroziť jeho celkovú realizáciu. Vo všeobecnosti sa rizikám vyhnúť nedá, je však možné ich do veľkej miery eliminovať vhodným spôsobom riadenia projektu.

### 3.3.1 Analýza rizík

Hlavné riziká identifikované v prvej fáze projektu:

1. Nesprávne pochopenie problematiky
2. Nedostatok času členov tímu
3. Nedostatočná dokumentácia minuloročného systému

4. Nerealistické časové odhady
5. Nesprávny manažment tímu
6. Nesprávny výber technológií

### 3.3.2 Popis rizík

**Nesprávne pochopenie problematiky:** Napriek snahe o čo najlepšie pochopenie problematiky daného projektu, takéto riziko existovalo. Nikto z nášho tímu sa totiž podobnej problematike ešte nikdy nevenoval, čo mohlo v začiatkoch spôsobiť veľké nedostatky pri analýze minuloročného projektu alebo novej funkcionality systému.

**Nedostatok času členov tímu:** Vzhľadom na náročnosť zimného semestra bolo pravdepodobné, že jednotliví členovia tímu nebudú mať dostatok času na plnenie svojich povinností v pozíciách manažérov tímu. Ďalším dôvodom zváženia tohto rizika bol fakt, že všetci členovia tímu popri štúdiu pracujú v rôznych firmách.

**Nedostatočná dokumentácia minuloročného systému:** Systém, ktorý bol výsledkom práce predchádzajúceho timu, bol naprogramovaný pomocou technológie C# a PHP, čo mohlo spôsobiť problémy pri spätnej analýze ich kódu. S programovacím jazykom C# totiž z nášho tímu vedeli pracovať len dvaja ľudia. Tiež dokumentácia, ku ktorej sme mali spočiatku prístup, neobsahovala všetky potrebné informácie z dôvodu licenčných podmienok SCOPUS-u a WoS. Vo všeobecnosti dokumentácia predchádzajúceho projektu neobsahovala informácie, potrebné na spätnú analýzu uvedeného projeku.

**Nerealistické časové odhady:** V jednotlivých šprintoch mohli byť naplánované úlohy s nerealistickými časovými odhadmi. Dôvodom predĺženia času vykonávania môžu byť technické problémy (napr. pri vytváraní databázy, parsovaní Google Scholar, parsovaní a importovaní CREPČ xml) alebo to môže byť dôsledkom druhého rizika a to nedostatkom času jednotlivých členov.

Tomuto riziku sa nebolo možné spočiatku vyhnúť, keďže sme nemali dostatok informácií o problémovej oblasti ktorej sme sa venovali. Neskôr sa manažment časových odhadov zlepšil aj vďaka použitia metodiky “planning poker cards”.

**Nesprávny manažment tímu:** Vzhľadom na našu neskúsenosť v úlohách manažérov tímu a malé možnosti v zmene manažmentu (a jeho spôsobu) je možným rizikom demotivácia tímu a následné nesplnenie vytýčených cieľov v určenom termíne. Riziko zvyšoval tiež fakt, že v praxi sme sa s agilnou metódou vývoja SCRUM stretli na tomto projekte po prvý krát.

**Nesprávny výber technológií:** Pri procese analýzy sme sa rozhodli použiť niektoré funkčné časti minuloročného projektu (napr. importéry WOS a SCOPUS) a tým pádom aj technológie, ktoré používajú (C# a MongoDB). Keďže náš systém bude bežať na technológiách Java, PostgreSQL a C, môže použitie ďalších dvoch technológií pôsobiť neprehľadnosť systému a zbytočné komplikácie pri ladení a testovaní kódu.

### 3.3.3 Aktuálny stav

V súčasnosti máme väčší prehľad o problémovej doméne, ktorej sa náš projekt týka. Tiež sme si osvojili prácu v softvérovom tíme a manažment tímu. Ten nie je stále na dostatočnej úrovni, keďže zastávame manažérske

role po prvý krát. Na predchádzajúcich chybách sme sa však naučili na čo si máme v budúcnosti dať pozor pri riadení tímu.

Ako najväčšie riziká v súčasnosti považujeme nedostatok času členov tímu a nerealistické časové odhady, ktoré spolu priamo súvisia. Hlavne prvé riziko je náročné eliminovať, keďže okrem tohoto projektu pracujeme na iných školských projektoch a zadaniach. Riziko časových odhadov sa nám darí zmenšovať, keďže naše vedomosti o problémovej doméne a teda aj náročnosti vykonávania jednotlivých úloh sú oveľa väčšie ako na začiatku projektu.

### 3.3.4 Tabuľka rizík

| **Riziková oblasť** | **Riziko** | **Pravdepodobnosť** | **Vplyv** |
|---|---|---|---|
| Znalosti | Neznalosť problémovej domény | vysoká | Potreba štúdia danej problematiky |
| Znalosti | Nízka kvalita informácií z knižníc | vysoká | Potreba kontroly dát importovaných z knižníc |
| Tím | Nesprávny manažment | stredná | Potreba študovania manažmentu softvérového tímu a zlepšovania riadenia |
| Tím | Nedostatočná komunikácia | stredná | Potreba správneho používania dohodnutých informačných kanálov |
| Znalosti | Neznalosť predchádzajúceho projektu | vysoká | Potreba štúdia predchádzajúceho projektu z projektovej dokumentácie a zdrojových kódov |
| Znalosti | Chýbajúca dokumentácia predchádzajúceho | stredná | Potreba štúdia zdrojových kódov, prípadne analýza danej funkcionality zo zdrojov |

| | | | |
|---|---|---|---|
| | riešenia | | dát (napr. CREPČ) |
| Tím | Prideľovanie na sebe závislých úloh rôznym ľudom | stredná | Potreba prerozdelenia úloh jednotlivým členom tímu |
| Tím | Nerovnomerné rozdeľovanie úloh | stredná | Potreba prerozdelenia úloh, aby všetci členovia tímu mali priradené rovnaké percento úloh v danom šprinte |
| Technológie | Nedostatočné testovanie | stredná | Potreba vytvorenia a priradenia úloh o testovaní vytvorenej funkcionality |
| Tím | Odchod člena tímu | nízka | Potreba prerozdelenia úloh a povinností |
| Znalosti | Podcenenie analýzy | vysoká | Potreba ďalšej analýzy danej problematiky. |

## 3.4 Manažment plánovania

Osoba zodpovedajúca za manažment plánovania: Rastislav Kostrab, Bc.

### 3.4.1 Míľniky

| **Dátum** | **Cieľ** |
|---|---|
| 5.12.2013 | Vytvorenie prototypu front-endu |
| 15.2.2014 | Dokončenie refaktorizácie minuloročného riešenia |
| 1.4.2014 | Počítanie publikačných indexov |

| | |
|---|---|
| 10.5.2014 | Vytvorenie webovej služby pre porovnavanie záznamov |

### 3.4.2 Harmonogram práce

Nasledujúca tabuľka znázorňuje plánovanie šprintov. Šprinty prebiehajú v rámci vývojovej metodiky SCRUM. Šprinty trvajú prevažne 2 týždne. Každý šprint má svoje poradové číslo, názov, vymedzené obdobie a stručný popis úloh (resp. činností), ktoré sa počas daného obdobia riešia.

### 3.4.3 Plán šprintov

| P.č. | Názov | Obdobie | Činnosti |
|---|---|---|---|
| 1 | Amstel | 10.10.2013 – 24.10.2013 | ● Vytvorenie webovej prezentácie tímu<br>● Inštalácia VM<br>● Zefektívnenie algoritmov párovania (Refactoring)<br>● Štúdie XML CREPČ<br>● Návrh možností získavania dát z Google Scholar |
| 2 | Budweiser | 24.10.2013 – 7.11.2013 | ● Analýza front-endu systému<br>● Vytvorenie PostgreSQL databázy<br>● Analýza citačných štýlov<br>● Porovnávacie metódy<br>● Školenie JIRA |
| 3 | Carlsberg | 7.11.2013 – 21.11.2013 | ● Naplnenie DB s dátami z XML CrepČ<br>● Realizácia základnej funkcionality frontendu<br>● Návrh systému pre správu používateľov<br>● Návrh normalizácie dát<br>● Vyriešenie blokovania Google Scholar |

| | | | |
|---|---|---|---|
| | | | • Naplnenie databázy z Google Scholar parsera<br>• Vytvorenie návrhu používateľského rozhrania<br>• Experiment a výber algoritmu na porovnávanie |
| 4 | Duff | 21.11.2013 – 5.12.2013 | Uskutočnenie zmien v databáze<br>Inštalácia Bamboo<br>Inicializácia GWT projektu<br>Inštalácia aplikačného servera<br>Vytvorenie prihlasovania používateľov<br>Analýza a návrh komunikácie C programov a Javy<br>Inicializácia komponentu Scheduler<br>Návrh normalizácie dát<br>Realizácia základnej funkcionality front-endu |

## 3.4.4 Krátkodobé plány

Nasledujúce tabuľky znázorňujú prehľad krátkodobých plánov, t.j. širšie vyjadrenie plánov pre každý šprint. Každá jedna tabuľka teda predstavuje jeden šprint a stručný popis činností, ktoré sa počas daného časového ohraničenia riešia. Za každú činnosť je vždy zvolený člen tímu, ktorý zodpovedá za úspešné zvládnutie danej činnosti vo vymedzenom čase.

### *3.4.4.1 Šprint č. 1 (Amstel)*

| **Úloha** | **Zodp. osoby** |
|---|---|
| Vytvorenie webovej prezentácie tímu<br>• Vytvorenie webovej stránky pre tím č. 10 | Michal Glogner, Bc. |
| Inštalácia VM<br>• Inštalácia Ubuntu 12.04 Server na školský server<br>• Inštalácia a konfigurácia systému Jira | Michal Glogner, Bc.<br>Šimon Kompas, Bc. |
| Zefektívnenie algoritmov párovania (Refactoring) | Daniel KÍč, Bc. |

| | |
|---|---|
| | Tomáš Jánošík, Bc. |
| Štúdie XML CREPČ | Rastislav Kostrab, Bc<br>Stanislav Kubica, Bc. |
| Návrh možností získavania dát z Google Scholar | Šimon Kompas, Bc.<br>Michal Walder, Bc. |

#### *3.4.4.2 Šprint č.2 (Budweiser)*

| **Úloha** | **Zodp. osoby** |
|---|---|
| Analýza front-endu systému<br>● Analýza front-endu systému<br>● Skúmanie jeho prepojenia na DB<br>● Návrh riešenia (použijeme vytvorený vs. vytvoríme nový) | Bc. Michael Gloger |
| Porovnávacie metódy<br>● Implementácia porovnávacích metód<br>● Porovnanie na syntetických dátach | Bc. Daniel KĺČ,<br>Bc. Tomáš Jánošík |
| Vytvorenie PostgreSQL databázy<br>● Vytvorenie dátového modelu v PostgreSQL<br>● Naplnenie PostgreDB reálnymi dátami<br>● Pripravenie Tomášovi a Danovi dáta pre porovnávanie | Rastislav Kostrab, Bc.<br>Stanislav Kubica, Bc. |
| Školenie JIRA | Bc. Šimon Kompas, |

| | |
|---|---|
| • Zorganizovať školenie o používaní Jiry | Bc. Michael Gloger |

### *3.4.4.3 Šprint č.3 (Carlsberg)*

| **Úloha** | **Zodp. osoby** |
|---|---|
| Naplnenie DB s dátami z XML CREPČ<br>• Vytvorenie parsera XML CREPČ<br>• Implementácia systému pre import XML CREPČ do DB | Rastislav Kostrab, Bc. |
| Návrh systému pre správu používateľov<br>• Analýza riešenia správy používateľov z minulého roku<br>• Refaktoring správy používateľov z minuloročného projektu<br>• Návrh dátového modelu správy používateľov | Stanislav Kubica, Bc. |
| Realizácia základnej funkcionality frontendu<br>• Prihlasovacie okno do systému<br>• Prepojiť prihlasovacie okno s tabuľkou používateľov<br>• Testovanie prihlasovacieho okna<br>• Príprava prostredia na tvorbu frontendu | Šimon Kompas, Bc. |
| Návrh normalizácie dát<br>• Učiaci algoritmus na naučenie prahovej hodnoty pre porovnávanie mien<br>• Algoritmus na výber kandidátov pre porovnanie<br>• Comparacny engine | Daniel KĺČ, Bc.<br>Tomáš Jánošík, Bc. |
| Vyriešenie blokovania Google Scholar | Michal Walder, Bc. |
| Naplnenie databázy z Google Scholar parsera | Michal Walder, Bc. |

| | |
|---|---|
| Vytvorenie návrhu používateľského rozhrania | Michal Glogner, Bc. |
| Experiment a výber algoritmu na porovnávanie | Daniel KĺČ, Bc. |

#### *3.4.4.4 Šprint č.4 (Duff)*

| Úloha | Zodp. osoby |
|---|---|
| Uskutočnenie zmien v databáze<br><br>Pridanie stĺpca pre indikáciu nespracovaných záznamov<br>pridanie tabuľky histórií zmien v DB<br>Pridanie stĺpca zdroja a tabuľky obsahujúcej zdroje | Stanislav Kubica, Bc. |
| Inštalácia Bamboo<br><br>Získanie licencie k Bamboo | Michal Glogner, Bc. |
| Inicializácia GWT projektu<br><br>Vytvorenie Java balíčka pre GWT v Java projekte | Rastislav Kostrab, Bc. |
| Inštalácia aplikačného servera<br><br>Inštalácia aplikačného servera Tomcat 7.0 na server projektu<br>Nakonfigurovanie aplikačného servera | Rastislav Kostrab, Bc. |
| Vytvorenie prihlasovania používateľov<br><br>vytvorenie serverovej časti pre GWT pre správu prihlasovania používateľov | Michal Glogner, Bc. |
| Analýza a návrh komunikácie C programov a Javy<br><br>Zistenie možnosti volania C programov z Javy<br>Definicia komunikačného rozhrania medzi komparátormi a Javou | Daniel KĺČ, Bc. |
| Inicializácia komponentu Scheduler<br><br>Vytvorenie Java balíčka pre komponent Scheduler v Java projekte | Tomáš Jánošík, Bc. |
| Návrh normalizácie dát | Tomáš Jánošík, Bc. |

| | |
|---|---|
| Realizácia základnej funkcionality front-endu | Šimon Kompas, Bc.<br>Michal Walder, Bc. |

## 3.5 Manažment podpory vývoja

**Zodpovedná osoba:** Bc. Šimon Kompas

Úloha manažéra podpory vývoja v tíme je spravovať, konfigurovať, integrovať a poskytovať vybrané nástroje, ktoré pri vývoji a manažmente projektu uľahčujú kolaboratívny vývoj a vytvárajú podmienky pre jednoduchší manažment projektu. Ďalšou úlohou je správa používateľských účtov a práv k použitým nástrojom a ich funkcionalitám.

### 3.5.1 Nástroje pre podporu vývoja a manažmentu

#### *3.5.1.1 Verziovací systém, repozitáre a pridružené nástroje*

V tíme používame Git ako systém na správu repozitárov a verzií. Tieto repozitáre sme integrovali do aplikácie Atlassian Stash, ktorá ponúka webové rozhranie s prehľadným zobrazením repozitáru a so ochudobnenou paletou operácií vykonávaných nad repozitárom. Ako offline náhradu za Git sme ponúkli používanie grafickej aplikácie Atlassian SourceTree, ktorá ponúka obohatenú paletu operácií nad repozitármi. K aplikácií SourceTree a spôsobu tvorby zmien v nej bola vytvorená základná metodika pre vývojárov v tíme.

Tieto nástroje majú pomôcť v tíme s kolaboratívnou prácou na vývoji systému so zreteľom na sledovateľnosť zmien, prácu s aktuálnymi súbormi a dátami, čím by mali výrazne znížiť časové náklady na vývoj a zjednodušiť plánovanie.

Aplikácia Stash interne využíva Git ako verziovací systém a je nainštalovaná na mimoškolskom serveri vedúcej projektu. Prístup k aplikácií je cez webové rozhranie aplikácie Stash.

#### *3.5.1.2 Code-reviews*

Na podporu code-review bol nainštalovaný, nakonfigurovaný nástroj Atlassian Crucible + FishEye, ktorý podobne ako Stash ponúka prezeranie repozitáru, stromu verzií, súbor v repozitári, commitov obohatených o funkcionalitu tvorby code reviews. Táto aplikácia je tak isto ako Stash nainštalovaný na mimoškolskom serveri vedúcej projektu. Prístup k aplikácií je cez webové rozhranie aplikácie Crucible + FishEye.

#### *3.5.1.3 Podpora manažmentu*

Na mimoškolský server vedúcej projektu boli nainštalované ďalšie 2 nástroje spoločnosti Atlassian: Jira a Confluence, ktoré budú bližšie popísané v nasledujúcej časti. Všetky na server nasadené nástroje (Jira, Confluence, Crucible + FishEye, Stash) sú navzájom integrované tak, aby ich používanie bolo čo najjednoduchšie a intuitívne. Taktiež bol nakonfigurovaný SMTP server, ktorý je využitý v týchto nástrojoch na odosielanie mailových notifikácií.

Ako podporné nástroje pre manažment, kolaboráciu a komunikáciu sme zvolili už spomenuté Atlassian produkty: Jira s webovým rozhraním, ktorá slúži primárne na manažment úloh. Pre využitie metodiky scrum

bol do Jira nainštalovaný plugin Jira-Agile, ktorý ponúka zjednodušený nástroj pre projekty organizované touto metodikou. Ako wiki projektu slúži webová aplikácia Confluence.

#### *3.5.1.4 Integrácia produktu*

Pre podporu manažmentu integrácie a samotnej integrácie systému plánujeme využiť aplikáciu Atlassian Bamboo, ktorá by mala slúžiť ako nástroj na pokročilú integráciu počnúc kompiláciou cez automatizáciu testov až po nasadenie na server, čím chceme zjednotiť a urýchliť proces tvorby softvérového produktu ako celku. Tento nástroj sme chceli podobne ako ostatné nainštalovať na mimoškolský server vedúcej projektu, ale doposiaľ sme nezískali akademickú licenciu a zvažujeme možnosť integrácie s existujúcou inštanciou aplikácie na fakultnom serveri.

## 3.6 Manažment dokumentácie

Zodpovedná osoba : Bc. Michal Walder
Bc. Daniel Kĺč

### 3.6.1 Konvencie dokumentovania zdrojových kódov v Jave

Pri softvérových projektoch je dôležité určiť vhodné, spoločné konvencie dokumentovania zdrojových kódov. Je potrebné klásť dôraz na vhodné písanie komentárov v zdrojových kódoch pre jazyk Java tak, aby mohla byť jednoducho vygenerovaná programátorská dokumentácia pomocou nástroja JavaDoc.

JavaDoc je určený na generovanie štruktúrovanej programátorskej dokumentácie zo zdrojových súborova komentárov v Jave. Dokumentácia generovaná nástrojom JavaDoc je vo forme štruktúry HTML stránok. Samotný zápis inštrukcií pre JavaDoc, ktoré sa majú spracovať sa umiestňuje do špeciálneho komentáru začínajúceho /** a uzavretého */. Tento zápis sa umiestňuje pred deklaráciu súboru, triedy či metódy.

Viac o konvenciách dokumentácie zdrojových kódov v Jave v časti - "Manažment zdrojových kódov. Dokumentovanie zdrojových kódov pomocou JavaDoc".

### 3.6.2 Tvorba dokumentácie

V rámci tímového projektu sme vytvoriť tri tipy dokumentov:

- Dukumentácie k projektu (produkt a riadenie)
- Metodiky
- Zápisy zo stretnutí

Všetky dokumenty majú byť napísané v programe Microsoft Office 2010 poprípade OpenOffice.

Pri vytváraní dokumentácie k riadeniu ako aj k inžinierskemu dielu je potrebné jednotný formát písania. Preto sa v nasšich dokumentoch použijú takéto nastavenia písma:

**Fonty**

Font bežného písma: Arial

Veľkosť bežného písma: 11pt

Farba bežného písma: čierna

Na zvýraznenie sa použije buď **hrubé** alebo *šikmé* písmo.

Nadpisy nepoužívame, pretože sú nastavované priamo v dokumente k dokumentácií.

**Tabuľky**

Hrúbka okrajov tabuľky: 1pt

Farba hlavičky tabuľky: svetlo modrá

**Odrážky**

Typ nečíslovanej odrážky: plný krúžok

Typ nečíslovanej odrážky - 2 úroveň: prázdny krúžok

Typ číslovanej odrážky: číslo s bodkou

**Popisy obrázkov**

Umiestnenie: pod obrázkom

Zarovnanie: na stred

Začiatok popisu: Obr. x. (kde x je číslo), nasleduje popis obrázku s prvým veľkým písmenom

Pri dokumentácii k inžinierskemu dielu je štruktúra (podľa možnosti) nasledovná:

- Úvod
- Analýza
- Špecifikácia požiadaviek
- Návrh
- Implementácia
- Šprinty

Názov dokumentácie má formát Dokumentacia-typdokumentacie-verzia.doc

Príklad : Dokumentacia-riadenia-1.1.doc

## 3.6.3 Tvorba zápisnice zo stretnutia

Pre vytvorenie zápisnice zo spoločného stretnutia sa používajú rovnaké konvencie písania dokumentov ako v dokumentáciách. Tá je uvedená v časti - “Tvorba dokumentácie”.

**Nadpis zápisu zo stretnutia má formát :**

Zápisnica - Stretnutie č.X

Font : Arial

Veľkosť : 18pt (tučné)

Farba : Modrá

**Dokument zápisnice zo stretnutia má štruktúru v tvare :**

- Téma stretnutia - stručný opis témy stretnutia (jedna až dve vety)
- Dátum stretnutia
- Čas stretnutia
- Miesto stretnutia
- Stretnutie viedol - Člen tímu, ktorý viedol stretnutie
- Prítomní - Zoznam prítomných
- Neprítomní - Zoznam neprítomných
- Zapisovateľ - Zapisovateľ stretnutia
- Obsah stretnutia - Obsah stretnutia štruktúrovaný do maximálne 2-úrovňových odrážok
- Úlohy z predchádzajúceho šprintu
- Ostávajúce úlohy
- Úlohy ďalšieho šprintu

**Príklad tabuliek s úlohami:**

| ID | Popis úlohy | Zodpovednosť | Stav |
|---|---|---|---|
| 3.1 | Doplniť obsah, členov tímu do webovej prezentácie tímu | Bc.Michael Gloger | 50% |
| 3.2 | Návrh nových spôsobov efektívnejšieho párovania | Bc. Daniel KĺČ, | OK |

| | | | |
|---|---|---|---|
| | | Bc.Tomáš Jánošík | |
| 3.3 | Mapovanie XML CREPČ a návrh dátového modelu | Bc.Rastislav Kostrab,<br>Bc.Stanislav Kubica | 50% |
| 3.4 | Vylepšenie prototypu crawler/parser nad portálom Google Scholar | Bc.Michal Walder | OK |
| 3.5 | Konfigurácia nástrojov pre podporu vývoja, SMTP server a pre rekvizít | Bc.Šimon Kompas,<br>Bc.Michael Gloger | OK |

## 3.7 Manažment kvality

Zodpovedná osoba : Daniel KĺČ

### 3.7.1 Určenie štandardov a konvencií

Na začiatku si náš tím určil formát zápisnice zo stretnutia. Ďalej v niekoľkých iteráciach sa vytvoril dokument, ktorý popisuje základné princípy konvencií písania a udržovania zdrojového kódu. Tento dokument kopíruje všeobecne trendy a zásady programovania. Obsahuje všeobecné zásady na udržovanie:

- čitateľnosti kódu
- granularity kódu
- písania dobre štruktúrovaného a ľahko testovateľného kódu
- zásady udržovania modularity
- pokyny k unifikácií riešenia chybových stavov
- základné pokyny k testovaniu

### 3.7.2 Skrátená verzia dokumentu o udržovaní konvencií kódu

#### *3.7.2.1 Čitateľnosť kódu*

Čitateľnosť kódu je intuitívna predstava o význame kódu po prvom prečítaní človekom s dostatočnou kvalifikáciou. Závisí od čitateľného zápisu a zrozumiteľnom zápise entít kódu. Zápis entít kódu je štýl zápisu jazykových konštrukcií vybraného programovacieho jazyka, komentárov a identifikátorov. Z týchto je predovšetkým pre tímovú prácu zaujímavý zápis komentárov a identifikátorov.

Obsah komentárov je v kompetencií manažmentu dokumentácie. Z hľadiska čitateľnosti je dôležitý komentár všade tam, kde je dôležité vysvetliť určitý aspekt implementácie alebo zdôrazniť určitú dôležitú vlastnosť. Taktiež komentáre označujú etapy algoritmov, pre lepšiu oddelenosť logických krokov.

Zápis identifikátorov pozostáva z popisu a mena. Meno odpovedá funkcií a je vytvorené určitým procesom konkretizácie, ktorý je popísaný v metodike vybraných procesov čitateľnosti a konvencií kódu. Komentár je nutný tam, kde je potrebné zdôrazniť vlastnosť alebo je to z globálneho hľadiska nutné. Popis tohto procesu je taktiež v metodike vybraných procesov čitateľnosti a konvencií kódu.

#### *3.7.2.2 Udržovanie granularity kódu*

Ide o správne rozčlenenie algoritmu na logické kroky vykonávania. Podrobný popis je obsiahnutý v metodike vybraných procesov čitateľnosti a konvencií kódu.

#### *3.7.2.3 Písania dobre štruktúrovaného a ľahko testovateľného kódu*

Ide o logické rozčlenenie implementácie na producentov funkcionality a mediatorov funkcionality. Producenti produkujú podľa parametrov a stavu systému funkcionalitu. Vracajú buďto produkt alebo bližšie nešpecifikovanú chybu. Mediator je funkcionalita, ktorá volá producentov v sequencií, ktorá produkuje ich výstup. Výstupom je produkt alebo špecifikovaná chyba, ktorá je určitým spôsobom interpretovaná do zrozumiteľnej reči.

Takáto štruktúra kódu umožňuje jednoduché štrukturálne testy funkcionality. Spravidla producenti môžu byť otestovaný pre širokú množinu vstupov, kde sa testuje len to, či vráti chybu alebo produkt. Takto otestovaný producenti môžu byť prehlásený za funkčných v bežných situáciách. Na to nadväzuje kontrola mediátorov, pri ktorých sa testuje vrátenie produktu alebo vrátenie konkrétneho chybného hlásenia. Keďže po prvej úrovni testov vieme modelovať správanie producentov, testy mediátorov sú špeciálne dizajnované na vrátenie špecifickej chyby, zlyhanie špecifického producenta volaného daným mediátorom, alebo vrátenie produktu v konkrétnej konfigurácií.

Zatiaľ čo pri producentoch je možné v testovanie zahrňujúce v podstate všetky bežné chyby a výstupy, pri mediátoroch to spravidla nie je možné. Snaha pri mediátoroch je otestovať čo najväčší počet “bežných” situácií.

#### *3.7.2.4 Zásady udržovania modularity*

V podstate ide o dodržovanie špecifikácie modulu. V prípade nedodržania ide o procesy popisujúce rozdelenie modulu, posunutie nešpecifikovanej funkcionality do modulu, ktorý daný modul volá alebo rozšírenie špecifikácie modulu. Vzhľadom na to, že v prvých týždňoch sa vyvíjali komponenty riešenia menšie samostatné bloky a teda analýzy modularity zatiaľ neboli nutné, tak sa bližšie týmto procesom bude venovať druhá metodika z predmetu MSI.

#### *3.7.2.5 Pokyny k unifikácií riešenia chybových stavov*

Ide o definíciu toho, čo je chybový stav, čo je chybová hláška a čo je ošetrenie chybového stavu. V rámci projektu je pre nás

- chybový stav: Kód označujúci buďto zlyhanie komponentu alebo označujúci konkrétne zlyhanie súčasti funkcionality
- chybová hláška: Je hláška v jazyku používateľa, ktorá oznamuje zlyhanie zrozumiteľným textom, ktorý má zároveň uspokojiť používateľa a zároveň má vývojárovi povie, čo zlyhalo.
- Ošetrenie chybového stavu: Je to proces, pri ktorom sa programom na príslušnom mieste detekuje zlyhanie, program následne vypíše zmysluplným spôsobom danú chybu a nakoniec pokračuje vo vykonávaní predpripraveným spôsobom definovaným pre danú chybu.

#### *3.7.2.6 Základné pokyny k testovaniu*

Testovanie prebieha za účasti aspoň dvoch osôb na dvoch úrovniach. Testovania sa zúčastní aspoň manažér kvality a vývojár ktorý funkcionalitu implementoval. Prvá úroveň je štrukturálne testovanie funkcionality dvoma sériami vstupov. Prvá séria vstupov je séria definovaná daným vývojárom, druhá séria je upravená a rozšírená prvá verzia vstupov o vstupy definované manažérom kvality. Druhá úroveň je testovanie na reálnych vstupoch.

### 3.7.3 Kontrola zápisníc zo stretnutí

Kontrola zápisníc prebehla a zápisnice boli označené za vyhovujúce manažérom kvality.

(http://team10-13.ucebne.fiit.stuba.sk/dokumenty.html)

### 3.7.4 Kontrola dokumentov

Dokumenty sú priebežne kontrolované členmi tímu a aj manažérom kvality. Vzhľadom na momentálnu prezenciu čiastkových dokumentácií (keďže mnoho z práce, ktorá doteraz bola dokončená boli experimenty, alebo samostatné programové súčasti) tak najintenzívnejšia kontrola je realizovaná tímovým kolegom, ktorý na danej úlohe pracuje spolu s autorom dokumentácie (keďže všetky úlohy sú momentálne realizované v pároch).

### 3.7.5 Kontrola kódov

Vzhľadom na malý počet kódov, ktoré sú bežnými metódami kontrolovateľné (väčšina práce je momentálne v oblasti databáz, webu a sťahovania údajov z webu), tak kontrola prebehla len nad komparátormi publikácií, ktoré po určitej diskusií v tíme (a menších úpravách) boli akceptované.

# 4 Metodiky

## 4.1 Metodika tvorby používateľských príbehov

**Autor:** Michael Gloger

Táto kapitola obsahuje podrobne opísaný proces získavania a formulácie používateľských príbehov. Tento proces je rozšírený o rozbitie user story na podúlohy. Obsahuje tiež opis jednotlivých rolí a zodpovedností ktoré v tomto procese vystupujú.

### 4.1.1 Vstup procesu

- Požiadavky od budúceho používateľa systému / vlastníka produktu

### 4.1.2 Výstup procesu

- Naformulované používateľské príbehy v štandardizovanom tvare
- Zoznam podúloh, ktoré musia byť splnené pre dokončenie user story

### 4.1.3 Role a zodpovednosti

| **Rola** | **Zodpovednosť** |
|---|---|
| Používateľ, Vlastník produktu | Naformulovať svoje požiadavky na systém, ktoré nemusia byť ešte presne formulované |
| Vedúci tímu | Manažovanie komunikácie so zákazníkom, zabezpečenie komunikačných kanálov a získanie zoznamu požiadaviek od klienta |
| Vývojár, Konzultant | Formulácia používateľských požiadaviek do príbehov, ich korekcia, interpretácia, analýza a následne identifikácia podúloh spojených so splnením celej user story |

### 4.1.4 Proces tvorby používateľských príbehov

| **Krok** | **Názov** |
|---|---|
| 1. | Spracovanie vstupu |
| 2. | Formulácia |
| 3. | Vytvorenie podúloh |
| 4. | Zoradenie podľa priority |
| 5. | Uloženie výstupu |

#### *4.1.4.1 Spracovanie vstupu*

Vstup od klienta je vyjadrený voľnou rečou a je nutné ho previesť na bodový zápis, ktorého časti sú zoradené podľa okruhu, ktorého sa týkajú.

Pri získavaní a validácií požiadaviek sú nutné nasledovné kroky:

- Ak je požiadavka nepresná alebo moc všeobecná treba klienta požiadať o upresnenie
- Požiadavky, ktoré sú mimo dohodnutého scope treba filtrovať a ďalej diskutovať
- Z bodového zápisu treba podobné požiadavky spojiť do jednej

***Výstup:*** *Bodový zápis požiadaviek*

***Zodpovední:*** *Vedúci tímu, Vlastník produktu*

#### *4.1.4.2 Formulácia*

Z bodového zápisu určíme pre každú požiadavku:

- role
- ciele/túžby
- dôvody/benefity

Následne tieto požiadavky preformulujeme podľa jedného z nasledujúcich štandardizovaných tvarov:

1. “Ako <rola> chcem <cieľ/túžba>, aby <dôvod>”
2. “Ako <rola> chcem <cieľ/túžba>
3. “Aby som vedel <dôvod/benefit> ako <rola>, chcem <cieľ/túžba>”
4. “Ako <rola> <kedy>, chcem <cieľ/túžba>, aby <dôvod/benefit>”

Príklady:

1. “Ako zamestnanec chcem vidieť svoje odpracované hodiny, aby som si mohol viesť lepšie evidenciu”
2. “Ako projektový manažér chcem vedieť robiť reporty práce zamestnancov”
3. “Aby som vedel lepšie vyhľadávať zamestnancov ako vedúci, chcem ich vedieť vyhľadávať podľa krstného mena a priezviska”
4. “Ako vedúci počas letných prázdnin, chcem vedieť si zobraziť plán dovoleniek mojich zamestnancov aby som vedel lepšie plánovať projekty”

Ak sú požiadavky nie sú opísané veľmi detailne tak ich preformulujeme podľa jednoduchšej user story, ako je napríklad možnosť č. 2. V opačnom prípade si vyberieme jednu z ostatných možností.

***Výstup:*** *Sformulované používateľské príbehy*

***Zodpovední:*** *Vedúci tímu, Vývojár, Konzultant (asistencia vlastníka produktu)*

### *4.1.4.3 Vytvorenie podúloh*

Na user story môže v jednom čase pracovať viac členov tímu. Pre lepšiu evidenciu práce a úloh je ich teda nutné rozdeliť na podúlohy a to tak, aby na každej podúlohe pracoval maximálne jeden člen tímu.

*Príklad dekompozície user story:*

**User story:** "Ako výskumný pracovník chcem zoradiť svoje publikácie podľa počtu ohlasov, aby som si vedel lepšie robiť evidenciu citácií"

**Podúlohy:**

- Pridanie ikonky filtra zoradenia do stránky
- Vytvorenie dopytu do databázy
- Pridanie dopytu do databázy do controllera

Dekompozícia úloh je závislá na súčasnom stave projektu. Ak neexistuje žiadna stránka zobrazenia publikácií je nutné do podúloh zaradiť aj jej vytvorenie a pridanie.

***Výstup:*** *Podúlohy vychádzajúce z používateľských príbehov*

***Zodpovední:*** *Vedúci tímu, Manažér plánovania, Vývojár, Konzultant*

### *4.1.4.4 Zoradenie podľa priority*

Určenie priority používateľských príbehov a podúloh je dôležité aby sme najprv splnili tie úlohy, ktoré sú pre klienta najviac dôležité. Zoradenie priority príbehov ma na starosti vlastník produktu a zoradenie podúloh manažér plánovania a vedúci tímu.

Proces zoradenie používateľských príbehov podľa priority prebieha nasledovne:

1. Vlastníkovi produktu sa predložia všetky príbehy napísane na papieriky
2. Vlastník produktu tieto papieriky zoradí podľa priority
3. Pod každú user story následne zoradí manažér plánovania a vedúci tímu s asistenciou ostatných členov papieriky s podúlohami.

***Výstup:*** *Zoradené príbehy a podúlohy podľa priority*

***Zodpovedný:*** *Vlastník produktu, Vedúci tímu*

### *4.1.4.5 Uloženie výstupu*

Zoradené používateľské príbehy a ich podúlohy, je nutné kvôli lepšej evidencii uložiť do nástroja na projektový manažment. Postup sa môže mierne líšiť vzhľadom na diverzitu týchto nástrojov.

**Pridanie user story do JIRA:**

1. V hornom menu vyberieme možnosť *“Create issue”*
2. Do *“Issue type”* vložíme typ úlohy *“Epic”*
3. Do “Summary” vyberieme názov user story (respektíve podúlohy)
4. Do *“Priority”* vyberieme jednu z priorít. Napríklad: *“Minor, Major, …”*
5. Do *“Assignee”* vyberieme zodpovedného člena tímu
6. Do *“Description”* pridáme podrobný popis

**Pridanie podúloh k user story v JIRA:**

Postup je rovnaký ako pri pridaní user story, ale s nasledovnými zmenami:

- V kroku č. 2. *“Issue type”* nebude *“Epic”* ale akýkoľvek iný z dostupných. Napríklad *“Bug, Task”*
- Pribudne ešte jeden krok a to:
    - Do *“Epic link”* sa zo zoznamu vyberie user story, ku ktorej daná úloha patrí

***Výstup:*** *Používateľské príbehy a úlohy evidované v nástroji na projektový manažment (napríklad JIRA)*

***Zodpovedný:*** *Vedúci tímu*

## 4.2 Metodika manažmentu zberu požiadaviek

**Autor:** Michael Gloger

Zber požiadaviek a všeobecne manažment komunikácie so zákazníkom tvorí dôležitú rolu v rámci vývoja softvérových systémov. Zlyhanie tejto komunikácie, respektíve jej zanedbanie, má vážny a negatívny vplyv na výslednú kvalitu vyvíjaného produktu.

Táto kapitola obsahuje dokumentáciu k hornej metodike manažmentu zberu požiadaviek a z neho vyplývajúcich procesov. Na začiatku práce sú vymenované všetky role a zodpovednosti, prislúchajúce k jednotlivým procesom. Každý proces je opísaný z hornej (manažérskej) úrovne a detailnejšie pomocou scenára postupnosti krokov. K procesom sú priradené vstupy a výstupy. Na konci práce sú tieto procesy poprepájané pomocou diagramu aktivít na základe ich vstupov, výstupov, štartovacích pravidiel a rozhodnutí.

### 4.2.1 Role a zodpovednosti

Táto časť obsahuje tabuľku jednotlivých rolí a zodpovedností, priradených k jednotlivým procesom.

| Rola | Proces | Zodpovednosť |
|---|---|---|
| Vedúci tímu | 3.1 Zriadenie komunikačného kanálu | Registrácia komunikačného nástroja |
| | | Vybavenie licencie |
| | | Vytvorenie používateľských účtov |
| | 3.2 Zber požiadaviek na primárny systém | Naplánovanie osobného stretnutia |
| | | Spracovanie zápisu zo stretnutia |
| | 3.3 Vytvorenie používateľských príbehov | Analýza štrukturovaných požiadaviek |
| | | Návrh používateľských príbehov |
| | | Konzultácia používateľských príbehov a náčrtu rozhrania |
| | 3.4 Tvorba modelu prípadov použitia | Spracovanie používateľských príbehov |
| | | Konzultácia prípadov použitia |
| | 3.5 Prezentácia iterácie produktu zákazníkovi | Napísanie dokumentácie o výsledku práce |
| | | Pripravenie prezentácie produktu |
| | | Odprezentovanie iterácie produktu |
| Manažér podpory vývoja | 3.1 Zriadenie komunikačného kanálu | Inštalácia a konfigurácia nástroja |
| | | Vytvorenie základného návodu na používanie |
| Vlastník projektu | 3.2 Zber požiadaviek na primárny systém | Opis požiadaviek na primárny systém |
| | | Zoradenie požiadaviek podľa priority |
| | 3.3 Vytvorenie používateľských príbehov | Schválenie príbehov a rozhrania |
| | 3.4 Tvorba modelu prípadov použitia | Schválenie modelu prípadov použitia |
| | 3.5 Prezentácia iterácie produktu zákazníkovi | Odovzdanie ďalších požiadaviek |
| Systémový dizajnér | 3.3 Vytvorenie používateľských príbehov | Vytvorenie náčrtu používateľského rozhrania |
| | 3.4 Tvorba modelu prípadov použitia | Vytvorenie diagramu prípadov použitia |
| Celý tím | 3.1 Zriadenie komunikačného kanálu | Otestovanie nástroja |

### 4.2.2 Proces zriadenia komunikačného kanálu

Cieľom tohto procesu je zriadenie nového vybraného komunikačného kanálu pomocou komunikačného nástroja. Ku komunikačným nástrojom patria napríklad nástroje na zdieľanie súborov, požiadaviek, úloh a spoločné elektronické kalendáre. V tomto procese nie je opísaný výber nástroja, ale spustenie procesu je podmienené už vybraným nástrojom, ktorý je vstupom pre štart procesu.

**Role** Vedúci tímu, vlastník projektu, manažér podpory vývoja

**Vstup** Vybraný nový komunikačný nástroj

**Výstup** Nakonfigurovaný komunikačný nástroj a pripravené používateľské účty pre členov tímu a vlastníka projektu

**Scenár**

1. **Registrácia komunikačného nástroja**
   Vedúci tímu kontaktuje prevádzkovateľov nástroja a registruje tím.
2. **Vybavenie licencie**
   Vedúci tímu po potvrdení registrácie tímu predloží prevádzkovateľom informácie o firme, prípadne o akademickej inštitúcii, ktorej je členom. Ak sa jedná o komerčnú licenciu, tak vedúci tímu zabezpečí platobnú transakciu.
3. **Inštalácia a konfigurácia nástroja**
   Vedúci tímu poverí manažéra podpory vývoja odovzdáva mu všetky potrebné informácie pre inštaláciu a konfiguráciu nástroja, ktoré mu po registrácii predložil prevádzkovateľ služby. Manažér podpory vývoja následne nainštaluje a nakonfiguruje komunikačný nástroj pre špecifikované potreby tímu.
4. **Vytvorenie základného návodu na používanie**
   Manažér podpory vývoja vytvorí základný návod na používanie nainštalovaného komunikačného nástroja.
5. **Vytvorenie používateľských účtov**
   Vedúci tímu si vypýta kontaktné informácie (ak ich ešte nemá kompletné) od vlastníka projektu a registruje jeho a všetkých členov svojho tímu do komunikačného nástroja
6. **Otestovanie nástroja**
   Vedúci tímu odovzdá vlastníkovi projektu a členom tímu prihlasovacie údaje a vydá pokyn pre otestovanie nástroja. Ak nástroj je plne funkčný tak proces končí a v opačnom prípade sa vráti na krok 3. kde manažér podpory vývoja vykoná potrebné zmeny.

### 4.2.3 Proces zberu požiadaviek na primárny systém

Proces, ktorý sa opakuje počas celého vývoja softvéru pomocou agilnej metodiky. Počas tohto procesu prebieha priama komunikácia medzi vlastníkom produktu a vedúcim projektu.

**Role** Vedúci tímu, vlastník projektu, manažér plánovania, zapisovateľ

**Vstup** Požiadavky na systém od vlastníka projektu

**Výstup** Spracované požiadavky, zoradené podľa priority

**Scenár**

1. **Naplánovanie osobného stretnutia**

   Vedúci tímu zorganizuje osobné stretnutie, na ktorom sa zúčastní systémový architekt, manažér plánovania a vlastník produktu.

2. **Opis požiadaviek na primárny systém**

   Vlastník produktu je vyzvaný k opísaniu jeho požiadaviek na primárny systém. Vlastník produktu opisuje svoje požiadavky voľnou rečou a zapisovateľ zapisuje všetky informácie.

3. **Zoradenie požiadaviek podľa priority**

   Po opise požiadaviek voľnou rečou je vlastník produktu vyzvaný k zoradeniu jeho požiadaviek podľa priorít.

4. **Spracovanie zápisu zo stretnutia**

   Zápis zo stretnutia je spracovaný vedúcim projektu a požiadavky sú prepísane do štruktúrovaného zápisu.

### 4.2.4 Proces vytvorenia používateľských príbehov

V tomto procese sa transformujú vstupné požiadavky na používateľské príbehy.

**Role** Vedúci tímu, manažér plánovania, vlastník produktu, systémový dizajnér

**Vstup** Štruktúrovaný zápis požiadaviek na systém

**Výstup** Používateľské príbehy

**Scenár**

1. **Analýza štruktúrovaných požiadaviek**

   Štruktúrované požiadavky sú normalizované vedúcim tímu a podobné úlohy, ktoré odrážajú tú istú funkcionalitu sú zlúčené.

2. **Návrh používateľských príbehov**

   Vedúci tímu v spolupráci s manažérom plánovania navrhne znenie používateľských príbehov.

3. **Vytvorenie náčrtu používateľského rozhrania**

   Systémový dizajnér vytvorí náčrt používateľského rozhrania zo vstupných požiadaviek.

4. **Konzultácia používateľských príbehov a návrhu rozhrania**

   Vedúci tímu odprezentuje vytvorené používateľské príbehy a návrh používateľského rozhrania vlastníkovi produktu a požiada o ďalšie pripomienky. Pripomienky vlastníka produktu sú zapracované.

5. **Schválenie príbehov a rozhrania**

   Ak vlastník produktu súhlasí s návrhom používateľských príbehov a rozhrania, tak sú príbehy prepísané do komunikačného nástroja správy úloh.

### 4.2.5 Proces tvorby modelu prípadov použitia

Proces v ktorom sú z používateľských príbehov vytvorené modely prípadov použitia pomocou modelovacieho nástroja.

**Role** Vedúci tímu, vlastník produktu, systémový dizajnér

**Vstup** Štruktúrovaný zápis požiadaviek na systém, používateľské príbehy

**Výstup** Model prípadov použitia

**Scenár**

1. **Spracovanie používateľských príbehov**
   Vedúci tímu spracuje používateľské príbehy a navrhne z nich zoznam prípadov použitia
2. **Konzultácia prípadov použitia**
   Vedúci tímu konzultuje navrhnuté prípady použitia s členmi tímu a s vlastníkom produktu a zapracúva pripomienky.
3. **Vytvorenie diagramu prípadov použitia**
   Systémový dizajnér vytvorí zo zoznamu prípadov použitia diagram.
4. **Schválenie modelu prípadov použitia**
   Vedúci tímu odprezentuje diagram prípadov použitia klientovi. V tejto fáze ho vyzve k zamysleniu sa nad ďalšími funkcionálnymi požiadavkami na primárny systém. Ak vlastník produktu má ďalšie požiadavky tak sa vracia k procesu číslo zberu požiadaviek.

### 4.2.6 Proces prezentácie iterácie produktu zákazníkovi

V tomto procese je výsledok šprintu odprezentovaný vlastníkovi produktu a sú od neho prijímané pripomienky. Počas tohto procesu sa na základe požiadaviek od vlastníka produktu spúšťa proces zberu požiadaviek.

**Role** Vedúci tímu, vlastník produktu

**Vstup** Požiadavky na systém

**Výstup** Vytvorená iterácia produktu na základe vstupných požiadaviek

**Scenár**

1. **Napísanie dokumentácie o výsledku práce**
   Vedúci tímu napíše dokumentáciu o práci, ktorá prebehla na aktuálnej iterácii produktu a napíše súhrn implementovaných funkcionalít. Dokument obsahuje tiež zoznam splnených a nesplnených požiadaviek zo šprintu.
2. **Pripravenie prezentácie produktu**
   Vedúci tímu si pripraví prezentáciu implementovanej funktionality v produkte.

3. **Odprezentovanie iterácie produktu**

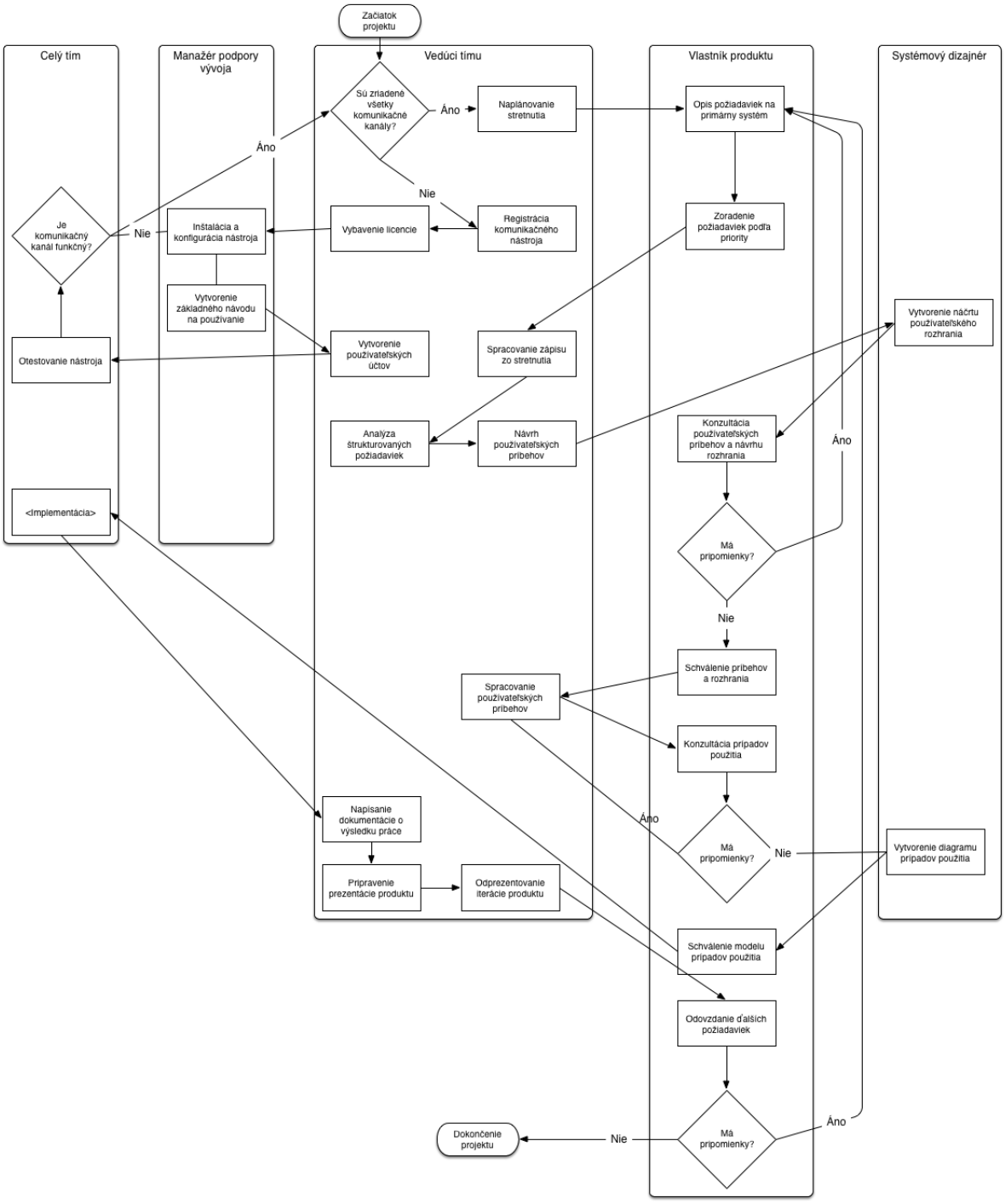
   Osobné stretnutie vedúceho tímu s vlastníkom projektu a predvedenie funkcionality.

4. **Odovzdanie ďalších požiadaviek**

   Ak vlastník produktu má ďalšie pripomienky na systém, prípadne ďalšie požiadavky tak sa spustí proces zberu požiadaviek.

## 4.2.7 Prepojenie procesov

Táto časť obsahuje diagram aktivít, v ktorom sú poprepájané scenáre opísaných procesov.



## 4.3 Metodika vybraných procesov čitateľnosti a konvencií kódu

**Autor:** Daniel KÍč

Cieľom kapitoly je popísať vybrané procesy, ktoré je treba používať na zmaximalizovanie čitateľnosti, prehľadnosti a testovateľnosti kódu ako aj na uľahčenie manipulácie. Udržovanie vnútornej a vonkajšej integrity ako aj čitateľnosti zdrojových kódov je kontinuálny proces, ktorý prebieha počas celého vývoja projektu, resp. vo všetkých častiach, ktoré v sebe zahrňujú tvorbu, opravu a refaktorizáciu zdrojových kódov.

### 4.3.1 Role zahrnuté v spomínaných vybraných procesoch

- Vývojár
    - Zodpovedný za integritu a čitateľnosť ním napísaných kódov
    - Zodpovedný za spoľahlivosť ním napísaných kódov
- Manažér kvality
    - Zodpovedný za kontrolu zdrojových kódov – čitateľnosti, integrity a manažovateľnosti

### 4.3.2 Procesy definované v oblasti čitateľnosti a testovania kódov

Procesy v tejto podkapitole sú definované v bodoch a bližší popis bodov je v ďalšej podkapitole.

#### *4.3.2.1 Proces: Pomenovanie identifikátora*

Proces je používaný pri definícií identifikátorov ako sú mená tried, premenných, funkcií, metód a podobne.

Iniciátor: Vývojár

Postup:

1. Identifikujte oblasti pôsobenia a významu v lokálnom kontexte
2. Vytvorte pomenovania
3. V prípade konfliktov s lokálnym prostredím alebo významom použite konkretizáciu
4. Skontrolujte vytvorené pomenovanie

#### *4.3.2.2 Proces: Popis identifikátora alebo modulu*

Na zaručenia čitateľnosti kódu je potrebné popísať veľký podiel identifikátorov a aj moduly, aby sa zabezpečila ich interpretovateľnosť a jasnosť.

Iniciátor: Vývojár

Postup:

1. Zvoľte identifikátor
2. Zistite všetky dostupné informácie o ňom

3. Ak je identifikátor premenná
    a. Popíšte premennú
    b. Overte či popis obsahuje všetky potrebné informácie
4. Ak je identifikátor funkcia/metóda
    a. Popíšte funkcie
    b. Popíšte vstupy
    c. Popíšte výstupy
    d. Popíšte chybové stavy
5. Ak je identifikátor trieda
    a. Identifikujte oblasti pôsobenia
    b. Identifikujte špeciálne vlastnosti
    c. Identifikujte špeciálne stavy
6. Ak popisujeme modul
    a. Identifikujte oblasti zamerania definovanej funkcionality
    b. Identifikujte autora
    c. Identifikujte vonkajšie závislosti

#### *4.3.2.3 Proces: Udržovanie granularity kódov*

Na zaručenia čitateľnosti, interpretovateľnosti a v neposlednej rade aj manažovateľnosti kódov, je treba udržovať granularitu kódov.

Iniciátor: Vývojár, manažér kvality

Postup:

1. Špecifikujte algoritmus na implementovanie
2. Dekomponujte problém na menšie samostatné bloky – etapy
3. Identifikujte redundantné bloku
4. Identifikujte algoritmy na nižšej úrovni abstrakcie
5. Pokračujte v konkretizácií až dosiahneme elementárne algoritmy
6. Určte redundantné kroky, vhodné na zovšeobecnenú implementáciu v knižniciach

a. Implementácia knižnice/pomocného modulu/triedy/funkcie

7. Implementujte pod-algoritmy
8. Implementujte samotný algoritmus

## 4.3.3 Podrobný opis krokov

### *4.3.3.1 Pomenovanie identifikátora*

#### 4.3.3.1.1 Identifikácia oblasti pôsobenia a významu v lokálnom kontexte

Identifikuje sa, o aký identifikátor ide, v akom prostredí bude fungovať a aký bude jeho význam pre program/modul/triedu/funkciu/metódu.

#### 4.3.3.1.2 Vytvorenie pomenovania

Vytvorí sa pomenovanie, ktoré odpovedá použitiu a významu premennej.

#### 4.3.3.1.3 V prípade konfliktov s lokálnym prostredím alebo významom konkretizácia

Ak premenná nedostatočne popisuje svoje význam, špecifické vlastnosti alebo je nutné zdôrazniť oblasť použitia poprípade koliduje jej pomenovanie s inou premennou, je pridané druhé pomenovanie cez podtržník. Proces konkretizácie pokračuje podľa potreby dokým je pomenovanie dostatočne konkrétne a zároveň jednoznačné.

#### 4.3.3.1.4 Kontrola vytvoreného pomenovania

Vytvorené pomenovanie je porovnané s dôležitými vlastnosťami a významom daného identifikátora a podľa potreby je zredukované alebo rozšírené ak neodpovedá účelu.

### *4.3.3.2 Popis identifikátora alebo modulu*

#### 4.3.3.2.1 Zvolíme identifikátor

Zvolíme identifikátor, pre ktorý chceme preskúmať potrebu popisu.

#### 4.3.3.2.2 Zistíme všetky dostupné informácie o ňom

Zistíme kontext identifikátora, dôležité vlastnosti, význam v rámci prostredia, funkciu a využitie identifikátora.

#### 4.3.3.2.3 Ak je identifikátor premenná

Ak je identifikátor premenná a je to lokálna premenná menšej metódy alebo premenná bez špecifických vlastností, ktorej popis vystihuje všetky detaily implementácie a logického významu, tak popis nie je nutné uviesť.

Inak ak je premenná významná lokálna premenná alebo globálna premenná, je nutné uviesť špeciálne vlastnosti a význam.

Ak je premenná externá, je treba uviesť zdroj.

#### 4.3.3.2.4 Ak je identifikátor funkcia/metóda

V prípade funkcie je nutné uviesť krátky popis zamerania.

Popis parametrov, formát, obsah.

Popis výstupu, formát obsah.

Popis chýb, spôsob indikovania chyby, formát.

#### 4.3.3.2.5 Ak je identifikátor trieda

V prípade, že je identifikátor tried je nutné uviesť krátky popis.

Popis špeciálnych vlastností.

Popis indikovania chybového stavu triedy (ak je v triede globálne riešená indikácia chýb).

#### 4.3.3.2.6 Ak popisujeme modul

Ak je popisovaný modul je treba uviesť špecializáciu modulu.

Uvedenie autora modulu.

Identifikácia vonkajších závislostí.

### *4.3.3.3 Udržovanie granularity kódov*

#### 4.3.3.3.1 Špecifikácia algoritmu na implementovanie

Je potrebné identifikovať algoritmus, ktorý sa bude implementovať.

#### 4.3.3.3.2 Dekompozícia problému na menšie samostatné bloky – etapy

Prebehne dekompozícia algoritmu na menšie, samostatne vykonateľné etapy. Tieto etapy sú logickými krokmi algoritmu a udržujú rovnakú úroveň abstrakcie.

#### 4.3.3.3.3 Identifikácia redundantných blokov

Identifikácia krokov algoritmu, ktoré sa budú vykonávať viac krát.

#### 4.3.3.3.4 Identifikácia algoritmov na nižšej úrovni abstrakcie

Identifikácie krokov algoritmu, ktoré sú samé o sebe algoritmami.

#### 4.3.3.3.5 Pokračovanie v konkretizácií až dosiahneme elementárne algoritmy

Pre algoritmy, ktoré sú súčasťou tzv. super-algoritmu je nutné znovu odštartovať tento celý proces. Proces udržovania granularity iteruje na pod-algoritmoch až dosiahne úroveň elementárnych algoritmov. Tieto elementárne algoritmy sú na požadovanej úrovni granularity.

#### 4.3.3.3.6 Určenie redundantných krokov, vhodných na zovšeobecnenú implementáciu v knižniciach

Určia sa redundantné kroky na elementárnych algoritmoch a pod-algoritmoch, ktoré sú zovšeobecniteľné a zároveň tento druh algoritmu je redundantný.

Implementácia týchto súčastí kódu do knižnice (čím sa zníži logická redundancia).

Následkom týchto krokov sa zníži logická aj faktická redundancia a zároveň sa zvýši čitateľnosť a interpretovateľnosť kódu.

#### 4.3.3.3.7 Implementácia pod-algoritmov

Implementujú sa algoritmy, ktoré sú súčasťou super-algoritmu.

#### 4.3.3.3.8 Implementácia algoritmu

Z dekomponovaných súčastí a vytvorených knižníc sa implementuje samotný algoritmus, ktorý bol objektom záujmu.

## 4.4 Metodika testovania

**Autor:** Daniel KÍč

Cieľom kapitoly je opísať procesy, ktoré je nutné vykonať pri zostavovaní testov a pri samotnom testovaní.

### 4.4.1 Potrebné vedomosti

Sekcia obsahuje zoznam okruhov tém, ktoré by čitateľ mal poznať, aby vedel metodiku vykonať.

- Testovanie, taxonómia, prístupy.
- Konvencie kódov.
- Modelovanie prípadov použitia.

### 4.4.2 Manažment udržovania testovania

Testovanie je súčasť cyklov vývoja softvéru, ktorý validuje funkčnosť softvéru z rôznych hľadísk. Kategórie sú rôzne, napríklad „works as expected“, „meets requirements“, „environment independence“, „satisfies stakeholders“ a i.

#### *4.4.2.1 Účastnícke role*

- Vývojár – podieľa sa na tvorbe testov, robí statické a dynamické testovanie, robí white box testovanie.
- Manažér kvality – robí statické testovanie, overuje výsledky dynamického testovania, robí black box testing, white box testovanie.
- Zákazník – Robí black box testovanie.

#### *4.4.2.2 Procesy definované v dokumente*

##### 4.4.2.2.1 Proces: Statické testovanie

1. Určia sa kódy na inšpekciu.
2. Kontrola čitateľnosti a konvencií kódu.
3. Kontrola dodržania špecifikácie. Tento bod zahŕňa skontrolovanie špecifikácie po stránke navrhnutej funkcionality a algoritmickej podstaty.
4. Kontrola korektnosti implementácie vonkajšieho správania funkcionality. Kontrola ošetrenia chybných stavov, dodržania špecifikácia rozhrania funkcionality.

#### 4.4.2.2.2 Proces: Dynamické testovanie

1. Určí sa kód na testovanie.
2. Dekompozícia častí kódu na testovateľné súčasti. Testovateľné súčasti sú časti kódu ktoré majú interpretovateľný výstup (napríklad sa ťažko testu rutina, ktorá zafarbí obrazovku na modro, na toto slúžia priebežné vizuálne testy) a zároveň nie sú závislé od externej entity mimo systému (napríklad je ťažko testovať internetové rozhranie, na to slúžia integračne a výkonnostné testy).
3. Vytvorenie prípadov vstupných údajov a ich výstupných údajov.
4. Formulácia testov. Pričom test je krátka rutina testujúca očakávaný výstup pri daných vstupoch.
5. Analýza výsledkov. Niekedy je výstup správny a chyba je v zlom pochopení algoritmu pri modelovaní výstupov.

#### 4.4.2.2.3 Proces: White-box testovanie

1. Určí sa kód na testovanie.
2. Určia sa jeho prípady použitia.
3. Kompetentná osoba vykoná prípady použitia. Popri vykonávaní testuje rôzne odchýlky od namodelovaných prípadov (kliknutie inam ako je očakávané, zadanie inej hodnoty a i.
4. Vytvorí sa záznam o neočakávanom chovaní funkcionality.

#### 4.4.2.2.4 Proces: Black-box testovanie

1. Určí sa funkcionalita, ktorá bude predmetom testu.
2. Vytvorí sa stručný dokument o možnostiach funkcionality.
3. Vytvoria sa prípady použitia danej funkcionality.
4. Nezaškolená osoba vykoná test pomocou dokumentu z bodu 2. Sledujú sa odchýlky od modelovaných prípadov použitia a chovanie systému na neočakávané ale aj očakávané situácie.
5. Vytvorí sa záznam o neočakávanom chovaní systému.

## 4.5 Metodika manažmentu vývoja modulov

**Autor:** Daniel KĺČ

Metodiky pokrýva úkony spojené s udržovaním striktnej modularity a poriadku v systéme. Toto je kontinuálny proces počas vývoja programu.

### 4.5.1 Nadväzujúce metodiky

- Metodika testovania
- Metodika refaktorizácie

### 4.5.2 Manažment udržovania striktnej modulárnosti

Na zaručenie znovu použiteľnosti komponentov je potrebné striktne dodržovať špecifikáciu modulov. Táto ja zvyčajne narušená chybami v špecifikácií, či už ide o zle špecifikovanú oblasť, alebo zanedbaný aspekt modulu. V týchto prípadoch udržovanie špecifikácie používa určité procesy, ktoré vzniknutú situáciu vyriešia.

#### *4.5.2.1 Role zahrnuté v spomínaných procesoch*

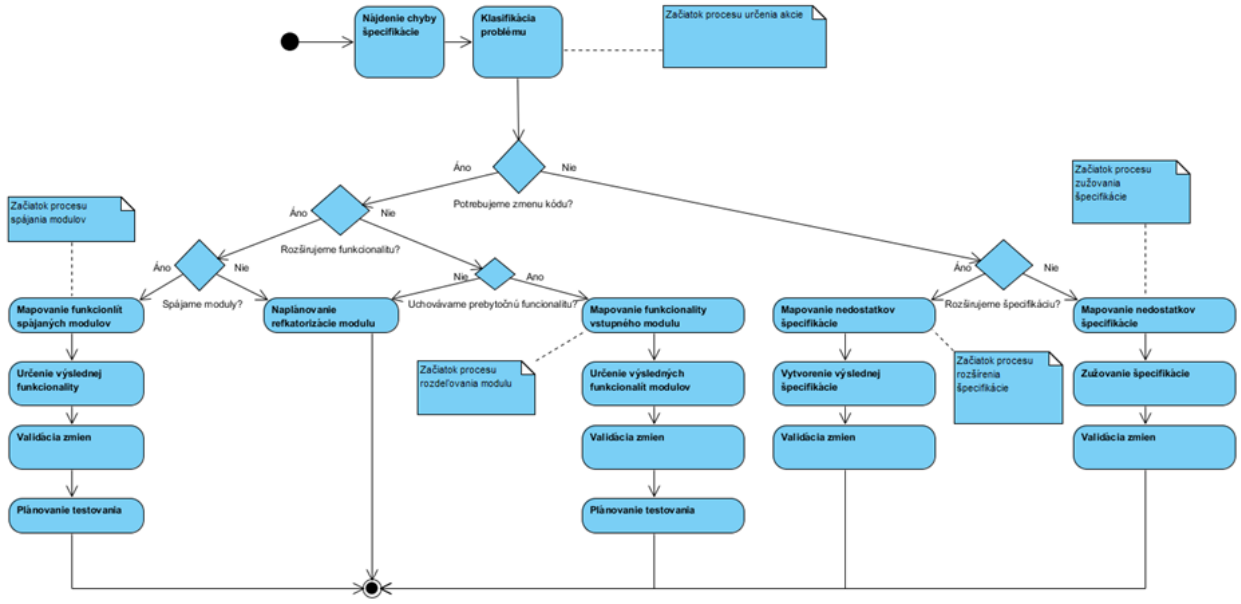
| Rola | Zodpovednosť | Proces |
|---|---|---|
| Vývojár | Upozornenie na nedodržanie špecifikácie | Proces určenia akcie (5.1) |
| Vývojár | Fyzická práca s modulom v rámci novej špecifikácie | Proces rozdelenia modulu na viac menších modulov (5.4) |
| Vývojár | Fyzická práca s modulom v rámci novej špecifikácie | Proces spájania modulov (5.5) |
| Manažér kvality | Zistenie nedodržania špecifikácie | Proces určenia akcie (5.1) |
| Manažér kvality | Kategorizácia problému a určenia akcie | Proces určenia akcie (5.1) |
| Manažér plánovania | Naplánuje refaktorizáciu | Proces určenia akcie (5.1) |

| | | |
|---|---|---|
| Manažér plánovania | Naplánuje testovanie | Proces rozdelenia modulu na viac menších modulov (5.4) |
| Manažér plánovania | Naplánuje testovanie | Proces spájania modulov (5.5) |
| Manažér rizík | Odhad dopadu zmeny funkcionality v module | Proces určenia akcie (5.1) |
| Manažér dokumentácie | Zmení špecifikáciu modulu | Proces rozšírenia špecifikácie modulu (5.2) |
| Manažér dokumentácie | Zmení špecifikáciu modulu | Proces zúženia špecifikácie modulu (5.3) |

#### *4.5.2.2 Procesy definované v oblasti udržovania striktnej modularity*

- Proces určenia akcie nad modulom(5.1).
- Proces rozšírenia špecifikácie modulu (5.2).
- Proces zúženia špecifikácie modulu (5.3).
- Proces rozdelenia modulu na viac menších modulov (5.4).
- Proces spájania modulov (5.5).

### *4.5.2.3 Hierarchia uvedených procesov*




Obrázok 1: Obrázok popisuje hierarchiu procesov opísaných v tomto dokumente a ich najdôležitejších akcií. Pomocou uvedeného rozhodovacieho stromu sa v procese určenia akcie nad modulom určí, aká je povaha problému nájdeného v danom module a následne sa určí proces, ktorý daný problém napraví. Je 5 možných akcií ktoré proces môže určiť a to 4 procesy, ktoré sú opísané v tomto dokumente a môže tiež určiť, že je potrebná rekatorizácia s ohraničeným vplyvom len na daný modul , ktorej priebeh ja definovaný v metodike refaktorizácie. Začiatky všetkých piatich procesov popísaných v tomto dokumente sú označené poznámkou v uvedenom diagrame.

## 4.5.3 Podrobný popis modulov

### *4.5.3.1 Proces určenia akcie*

Proces je vykonaný, ak sa zistí, že modul nezodpovedá špecifikácií.

#### 4.5.3.1.1 Popis procesu

Iniciátor: Vývojár, manažér kvality

Vstup: Modul, popis rozdielu špecifikácie a implementácie

Výstup: Akcia nápravy

Role: Manažér kvality, manažér rizík, manažér plánovania

##### *4.5.3.1.1.1 Postup*

1. Kategorizovanie problému

a. Ak modul obsahuje funkcionalitu navyše, kategorizujte prienik oblastí využitia.

b. Ak modul neobsahuje všetku funkcionalitu, aproximujte dopad na prípady použitia modulu.

2. V prípade ak funkcionalita navyše má silný prienik s oblasťou použitia modulu

   a. Aplikujte proces 5.2.

3. V prípade ak funkcionalita navyše má malý prienik s oblasťou použitia modulu

   a. Aplikujte proces 5.4.

4. V prípade ak chýbajúca funkcionalita spôsobuje, že špecifikácia modulu je podobná s iným modulom

   a. Aplikujte proces 5.5.

5. V prípade, že chýbajúca funkcionalita je označená za prebytočnú

   a. Aplikujte proces 5.3.

6. V ostatných prípadoch

   a. Označte modul ako nevyhovujúci.

   b. Vytvorte refaktorizačný zámer a naplánujte refaktorizáciu.

7. Naplánujte určené akcie.

#### 4.5.3.1.2 Podrobný popis krokov

##### *4.5.3.1.2.1 Kategorizovanie problému*

Využije sa tabuľka [špecifikovaná funkcionalita X dodaná funkcionalita]. Zoberú sa jednotlivé prípady použitia a vyznačí sa používanie funkcionality. Z tohto je vidieť, či funkcionalita chýba, je šitá na mieru, alebo je prebytočná. Taktiež je tu vidieť, či sa z hľadiska využívania funkcionality nerozdeľujú prípady použitia na skupiny.

##### *4.5.3.1.2.2 V prípade ak funkcionalita navyše má silný prienik s oblasťou použitia modulu*

Ak sa na tabuľke z kroku 5.1.2.1 ukáže, že funkcionalita má naozaj opodstatnenie a že sa výrazne neseparuje od zvyšku, aplikujeme proces 5.2.

##### *4.5.3.1.2.3 V prípade ak funkcionalita navyše má malý prienik s oblasťou použitia modulu*

Ak sa na tabuľke z kroku 5.1.2.1 ukáže, že funkcionalita má naozaj opodstatnenie ale že sa výrazne separuje od zvyšku, aplikujeme proces 5.4.

#### *4.5.3.1.2.4 V prípade ak chýbajúca funkcionalita spôsobuje, že špecifikácia modulu je podobná s iným modulom*

Ak sa na tabuľke z kroku 5.1.2.1 ukáže, že špecifikovaná funkcionalita je naozaj zbytočná a zároveň že funkcionalita modulu robí modul podobný s iným modulom, aplikujeme proces 5.5.

#### *4.5.3.1.2.5 V prípade, že chýbajúca funkcionalita je označená za prebytočnú*

Ak sa na tabuľke z kroku 5.1.2.1 ukáže, že špecifikovaná funkcionalita je naozaj zbytočná a zároveň modul zostáva unikátny v dostatočnej miere, tak aplikujeme proces 5.3.

#### *4.5.3.1.2.6 V ostatných prípadoch*

Ak sa na tabuľke z kroku 5.1.2.1 ukáže, že špecifikovaná funkcionalita je potrebná a zároveň nie je implementovaná v module alebo nastane ľubovoľný iný problém, vytvorí sa plán refaktorizácie modulu, resp. zoznam akcií, ktoré je treba s modulom vykonať kým bude môcť byť znova otestovaný.

#### *4.5.3.1.2.7 Naplánujte určené akcie*

Manažér rizík odhadne riziká realizácie danej akcie a následne manažér plánovania naplánuje vykonanie týchto akcií.

### *4.5.3.2 Proces rozšírenia špecifikácie modulu*

Proces je používaný pri zistení, že modul má viac funkcionality, ako má v špecifikácií a bol určený na rozšírenie špecifikácie procesom 5.1.

#### 4.5.3.2.1 Popis procesu

Iniciátor: Manažér kvality, vývojár

Vstup: Akcia nápravy, pôvodná špecifikácia, zoznam rozdielov

Výstup: Nová špecifikácia

Role: Manažér kvality, manažér dokumentácie

##### *4.5.3.2.1.1 Popis*

1. Vykonajte kategorizáciu rozdielov.
2. Určite časti špecifikácie potrebujúce zmenu.
3. Určite korešpondenciu novej verzie špecifikácie s aktuálnou situáciou.
4. Ak nová špecifikácia nie je uspokojujúcom stave, pokračujte krokom 1.
5. Inak označte špecifikáciu za novú špecifikáciu modulu.

#### 4.5.3.2.2 Podrobný popis krokov

##### *4.5.3.2.2.1 Vykonajte kategorizáciu rozdielov*

Použije sa tabuľka z 5.1.2.1. Na nej sa vyznačí, ktorá funkcionalita potrebuje byť došpecifikovaná v špecifikácií modulu.

##### *4.5.3.2.2.2 Určite časti špecifikácie potrebujúce zmenu*

V existujúcej špecifikácií sa určia podľa zoznamu novej funkcionality časti, ktoré potrebujú byť rozšírené alebo zmenené.

##### *4.5.3.2.2.3 Určite korešpondenciu novej verzie špecifikácie s aktuálnou situáciou*

Vykoná sa zmena špecifikácie podľa určenej potreby.

##### *4.5.3.2.2.4 Ak nová špecifikácia nie je uspokojujúcom stave, pokračujte krokom 1*

Určí sa pokrytie novej špecifikácie tabuľky z 5.1.2.1. V prípade potreby ďalších úprav sa pokračuje krokom 5.2.2.1.

##### *4.5.3.2.2.5 Inak označte špecifikáciu za novú špecifikáciu modulu*

Označte špecifikáciu za vyhovujúcu.

### *4.5.3.3 Proces zúženia špecifikácie modulu*

Proces je používaný pri zistení, že modul má menej funkcionality, ako má v špecifikácií a bol určený na zúženie špecifikácie procesom 5.1.

#### 4.5.3.3.1 Popis procesu

Iniciátor: Manažér kvality, vývojár

Vstup: Akcia nápravy, pôvodná špecifikácia, zoznam rozdielov

Výstup: Nová špecifikácia

Role: Manažér kvality, manažér dokumentácie

##### *4.5.3.3.1.1 Postup*

1. Vykonajte kategorizáciu rozdielov.
2. Určite časti špecifikácie potrebujúce zmenu.
3. Určite korešpondenciu novej verzie špecifikácie s aktuálnou situáciou.
4. Ak nová špecifikácia nie je uspokojujúcom stave, pokračujte krokom 1.

5. Inak označte špecifikáciu za novú špecifikáciu modulu.

#### 4.5.3.3.2 Podrobný popis krokov

##### *4.5.3.3.2.1 Vykonajte kategorizáciu rozdielov*

Použije sa tabuľka z 5.1.2.1. Na nej sa vyznačí, ktorá funkcionalita potrebuje byť odstránená zo špecifikácií modulu.

##### *4.5.3.3.2.2 Určite časti špecifikácie potrebujúce zmenu*

V existujúcej špecifikácií sa určia podľa zoznamu zbytočnej funkcionality časti, ktoré potrebujú byť odstránené alebo zmenené.

##### *4.5.3.3.2.3 Určite korešpondenciu novej verzie špecifikácie s aktuálnou situáciou*

Vykoná sa zmena špecifikácie podľa určenej potreby.

##### *4.5.3.3.2.4 Ak nová špecifikácia nie je uspokojujúcom stave, pokračujte krokom 1*

Určí sa pokrytie novej špecifikácie tabuľky z 5.1.2.1. V prípade potreby ďalších úprav sa pokračuje krokom 5.2.2.1.

##### *4.5.3.3.2.5 Inak označte špecifikáciu za novú špecifikáciu modulu*

Označte špecifikáciu za vyhovujúcu.

### *4.5.3.4 Proces rozdelenia modulu na viac menších modulov*

Proces je používaný pri zistený, že modul má viac funkcionality, ako má v špecifikácií a bol určený na rozdelenia na viac modulov procesom 5.1.

#### 4.5.3.4.1 Popis procesu

Iniciátor: Manažér kvality, vývojár

Vstup: Akcia nápravy, pôvodná špecifikácia, zoznamy navrhovaných funkcionalít výstupných modulov

Výstup: Množina nových modulov a ich špecifikácií

Role: Vývojár, manažér plánovania, manažér kvality

##### *4.5.3.4.1.1 Postup*

1. Lokalizujte časti funkcionality v pôvodnom module.
2. Namapujte navrhované funkcionality nových modulov s existujúcou funkcionalitou.
3. Vykonajte rozdelenie funkcionality do nových modulov.

4. Vytvorte schémy závislostí nových modulov.
5. Vytvorte špecifikácie nových modulov.
6. Porovnajte vstupné požiadavky s výstupom.
7. Ak výstup nie je vyhovujúci, pokračujte krokom 3.
8. Naplánujte testovanie.

#### 4.5.3.4.2 Podrobný popis krokov

##### *4.5.3.4.2.1 Lokalizujte časti funkcionality v pôvodnom module*

Fyzicky nájdite implementáciu funkcionality v module.

##### *4.5.3.4.2.2 Namapujte navrhované funkcionality nových modulov s existujúcou funkcionalitou*

Urobte tabuľku [funkcionalita modulu n X funkcionalita pôvodného modulu] pre všetky navrhované moduly.

##### *4.5.3.4.2.3 Vykonajte rozdelenie funkcionality do nových modulov*

Fyzicky preneste prienikovú funkcionalitu z tabuliek z 5.4.2.2.

##### *4.5.3.4.2.4 Vytvorte schémy závislostí nových modulov*

Pre každý nový modul vytvorte schému závislostí odvodenú z implementovanej funkcionality.

##### *4.5.3.4.2.5 Vytvorte špecifikácie nových modulov*

Na základe schémy závislostí a implementovanej funkcionality vytvorte novú špecifikáciu pre všetky nové moduly.

##### *4.5.3.4.2.6 Porovnajte vstupné požiadavky s výstupom*

Na základe tabuľky z 5.4.2.2 porovnajte prenesenú a plánovanú funkcionalitu. Ďalej posúďte špecifikáciu.

##### *4.5.3.4.2.7 Ak výstup nie je vyhovujúci, pokračujte krokom 3*

V prípade nevyhovujúcej situácie, pokračujte v 5.4.2.3.

##### *4.5.3.4.2.8 Naplánujte testovanie*

Manažér plánovania naplánuje testovanie nových modulov.

### *4.5.3.5 Proces spájania modulov*

Proces je používaný v dvoch prípadoch:

- Pri testovaní sa zistí, že modul obsahuje menej funkcionality ako bolo špecifikované, proces 5.1 určí, že zúžená funkcionalita je v poriadku, ale že nový modul s novou funkcionalitou je podobný inému modulu natoľko, že ich je potrebné zlúčiť kvôli logickej integrite systému.
- Pri testovaní alebo dokumentovaní sa zistí, že špecifikácie dvoch modulov sú podobné natoľko, že ich je potrebné z hľadiska logickej integrity systému zlúčiť.

#### 4.5.3.5.1 Popis procesu

Iniciátor: Vývojár, manažér kvality, manažér dokumentácie

Vstup: Akcia nápravy, špecifikácie danej množiny modulov

Výstup: Implementácia a špecifikácia výstupného modulu

Role: Vývojár, manažér kvality, manažér plánovania

##### *4.5.3.5.1.1 Postup*

1. Lokalizujte časti funkcionality pôvodných modulov.
2. Krížovo namapujte funkcionalitu na seba.
3. Určenia časti funkcionality, ktorá sa zachová.
4. Extrakcia funkcionality do jedného modulu.
5. Určenie závislostí nového modulu.
6. Vytvorenie novej špecifikácie.
7. Porovnanie vstupnej a výstupnej funkcionality z hľadiska prípadov použitia.
8. Ak modul nezodpovedá po stránke použiteľnosti pôvodnému stavu, pokračujte krokom 2.
9. Naplánujte testovanie.

#### 4.5.3.5.2 Podrobný popis krokov

##### *4.5.3.5.2.1 Lokalizujte časti funkcionality pôvodných modulov*

Fyzicky lokalizujte všetku funkcionalitu vo všetkých moduloch, ktorými sa zaoberáme.

##### *4.5.3.5.2.2 Krížovo namapujte funkcionalitu na seba*

Vytvorí sa zoznam funkcionality, kde sa odstránia redundantné implementácie. Ďalej sa vytvorí tabuľka [zoznam funkcionality X zoznam modulov].

#### *4.5.3.5.2.3 Určenie časti funkcionality, ktorá sa zachová*

Použijú sa prípady použitia pre spojený výsledný modul na určenie relevantnej funkcionality. Táto sa potom zachová.

#### *4.5.3.5.2.4 Extrakcia funkcionality do jedného modulu*

Extrahujú sa relevantné časti funkcionality, ktoré boli označené v bode 5.5.2.3 a ich fyzická poloha bola určená v bode 5.5.2.1.

#### *4.5.3.5.2.5 Určenie závislostí nového modulu*

Určia sa vzťahy, ktoré je nutné z pôvodných modulov zachovať na základe tabuľky [zachovaná funkcionalita X pôvodné závislosti modulov] a následnou unifikáciou výslednej množiny závislostí.

#### *4.5.3.5.2.6 Vytvorenie novej špecifikácie*

Na základe kompozitnej špecifikácie pôvodných modulov a funkcionality, ktorá bola na výsledný modul použitá sa vytvorí špecifikácia nového modulu.

#### *4.5.3.5.2.7 Porovnanie vstupnej a výstupnej funkcionality z hľadiska prípadov použitia*

Na základe prípadov použitia pre pôvodné moduly, sa určí, nakoľko dokáže nový modul uspokojiť potreby, jak po fyzickej stránke, tak po stránke rozhraní.

#### *4.5.3.5.2.8 Ak modul nezodpovedá po stránke použiteľnosti pôvodnému stavu, pokračujte krokom 2*

Ak sa ukáže, že nový modul má nedostatky v použiteľnosti v bode 5.5.2.7, tak sa pokračuje bodom 5.5.2.2.

#### *4.5.3.5.2.9 Naplánujte testovanie*

Manažér plánovania naplánuje testovanie nového modulu.

## 4.6 Metodika manažmentu chýb

**Autor:** Stanislav Kubica

### 4.6.1 Procesy evidovania a spracovania chyby

#### *4.6.1.1 Pridanie nájdenej chyby*

Proces predstavuje kroky vedúce od nájdenia chyby vo vytváranom softvérovom systéme po jej evidenciu v systéme Jira.

| | |
|---|---|
| **Stav procesu chyby po vykonaní procesu** | Nová<br>Pridelená |
| **Identifikované role** | Autor záznamu o chybe |

Postup pridania chyby:

1. Identifikovanie chyby autorom záznamu o novej chybe
2. Vyhľadanie chyby v systéme Jira podľa jej názvu, prípadne identifikátora
3. Ak chyba neexistuje vytvoriť novú (stav „Nová“), inak vložiť doplňujúce informácie k existujúcej (stav sa nemení)

#### *4.6.1.2 Vyhodnotenie chyby*

Popisuje kroky, ktoré je potrebné vykonať pred priradením chyby vývojárovi (analytikovi, návrhárovi a pod).

| | |
|---|---|
| **Stav procesu chyby pred vykonaním procesu** | Nová |
| **Stav procesu chyby po vykonaní procesu** | Zrušená<br>Duplikovaná<br>Pridelená |
| **Identifikované role** | Projektový manažér |

Postup vyhodnotenia chyby:

1. Projektový manažér si zobrazí nové (nepriradené) chyby v systéme Jira

2. Zanalyzuje, či je chyba relevantná, ak je pokračuje sa krokom 3. V opačnom prípade sa chyba označí ako „Zrušená“

3. Skontroluje či podobná/rovnaká chyba už v systéme neexistuje. Ak existuje priradí ju k existujúcej (čím sa označí ako „Duplikovaná“). V opačnom prípade ju pridelí ako úlohu na vypracovanie niekomu z analytikov, návrhárov alebo vývojárov (podľa povahy úlohy) – úloha prechádza do stavu „Pridelená“

#### *4.6.1.3 Vyriešenie chyby*

Popisuje kroky vedúce k vypracovaniu priradenej chyby.

| | |
|---|---|
| **Stav procesu chyby pred vykonaním procesu** | Pridelená |
| **Stav chyby počas vykonávania procesu** | Vypracovávaná |
| **Stav procesu chyby po vykonaní procesu** | Vypracovaná |
| **Identifikované role** | Analytik/Návrhár/Vývojár |

Postup vyriešenia chyby:

1. Vykonávateľ opravy (Analytik/Návrhár/Vývojár) označí chybu ako „Vypracovávanú“ a začne s jej opravou.

2. Po opravení chyby spíše potrebnú dokumentáciu a označí ju ako „Vypracovanú“

#### *4.6.1.4 Otestovanie vypracovanej chyby*

Popisuje kroky, ktoré musí vykonať tester po opravení chyby jej vykonávateľom.

| | |
|---|---|
| **Stav procesu chyby pred vykonaním procesu** | Vypracovaná |
| **Stav procesu chyby po vykonaní procesu** | Zatvorená<br>Pridelená |
| **Identifikované role** | Tester |

Postup otestovania opravenej chyby:

1. Tester si zobrazí vypracovanú chybu v systéme Jira
2. Po oboznámení sa s typom chyby, jej výskytom a pod. otestuje danú súčasť systému, aby zistil, či bol daný problém vyriešený. Ak bol problém vyriešený, tak chybu zatvorí. V opačnom prípade ju opäť pridelí pôvodnému vykonávateľovi.

4.6.2

## 4.6.3 Pridanie nájdenej chyby

Táto časť obsahuje detailný popis pridania nájdenej chyby na najnižšej úrovni. Chyby sú evidované v systéme Jira. Rolu autora záznamu môže zastávať každý člen tímu, ktorý nejakú chybu identifikuje.

### *4.6.3.1 Identifikovanie chyby autorom nového záznamu o chybe*

1. Po nájdení chyby zistiť o akú chybu sa jedná (chyba analýzy, návrhu alebo vývoja) a ktorého podsystému (alebo časti systému) sa týka (napr. frontend, comparator, crawlery a pod.).
2. V prípade chyby týkajúcej sa zdrojového kódu (behu programu) sa pokúsiť o jej detailnejšiu analýzu:
    1. Za akých podmienok chyba nastala?
    2. Ak sa chyba vyskytla na frontend-e, aký operačný systém a webový prehliadač bol použitý
    3. Ak je to možné, získať snímky obrazovky popisujúce chybu
3. Po získaní potrebných informácií pokračovať krokom 2

### *4.6.3.2 Vyhľadanie chyby v systéme Jira podľa jej názvu, prípadne identifikátora*

Vyhľadanie chyby v systéme Jira prebieha nasledovne:

1. Autor chyby sa musí do systému prihlásiť so svojím platným prihlasovacím menom a heslom. Následne uvidí svoju nástenku s informáciami o pridelených úlohách (záleží od individuálnych nastavení)
2. Kliknutím na „Issues“ => „Search“ for issues si autor zobrazí všetky úlohy. Pre pohodlnejšie vyhľadávanie je možné nastaviť filter, aby zobrazoval iba chyby. Ten je možné nastaviť po kliknutí na „Type: All“ a zaškrtnutí možnosti „Bug“. Následne je možné vyhľadávať chyby napísaním jej názvu (prípadne identifikátora) do poľa „Contains text“

### *4.6.3.3 a) Vytvorenie novej úlohy*

V prípade, že sa autorovi chyby nepodarí danú chybu v systéme nájsť, je potrebné vytvoriť záznam o novej chybe a to nasledovne:

1. Autor na ľubovoľnej obrazovke systému Jira klikne na tlačidlo „Create Issue“. Následne sa zobrazí formulár pre vytvorenie nového záznamu

   Formulár je potrebné vyplniť podľa nasledujúceho príkladu:

Chyba: autor našiel chybu v prihlasovaní používateľov

| **Názov položky formulára** | **Hodnota (popis hodnoty)** |
|---|---|
| Project | Tímový Projekt |
| Issue Type | Bug |
| Summary | [FRONTEND] Nefunkčné prihlasovanie používateľov ([identifikator_casti_systemu] Zhrnutie chyby) |
| Priority | Major (je potrebné vyhodnotiť prioritu chyby) |
| Due Date | (nechať prázdne) |
| Assignee | Unassigned (vyplní project manager pri vyhodnocovaní chýb) |
| Environment | (nechať prázdne) |
| Description | Po kliknutí na tlačidlo „Prihlásiť“ v okne so zoznamom publikácií určitého vedeckého pracovníka sa zobrazí chyba 500. V iných oknách táto funkcionalita funguje bez problémov<br><br>OS: Windows 8<br>Browser: Google Chrome 10<br><br>(napísať podrobný opis chyby spolu so všetkými informáciami, ktoré |

| | |
|---|---|
| | má autor k dispozícii) |
| Original Estimate | (nechať prázdne – vypĺňa project manager) |
| Remaining Estimate | (nechať prázdne) |
| Attachment | (vložiť všetky súbory týkajúce sa danej chyby, ktoré má autor k dispozícii) napr. login_screenshot.jpg, mail_od_zakaznika.txt |
| Labels | Frontend (označenie podsystému, ktorého sa chyba týka) |
| Epic Link | (nechať prázdne – epicy v Jire nepoužívame) |

### *4.6.3.4 b) Pridanie doplňujúcich informácií k existujúcej úlohe*

Ak sa autorovi podarí nájsť existujúcu chybu, ktorá je rovnaká ako identifikovaná, otvorí ju kliknutím na ňu v zozname chýb (ten si zobrazí podľa návodu vyššie). V pravom paneli sa zobrazia informácie o danej chybe.

Ak sa jedná o chybu, ktorá je v jednom zo stavov zrušená alebo zatvorená, je potrebné pred pridaním zmien kliknúť na tlačidlo „Reopen Issue“ s ponechaním pôvodného vykonávateľa. Po znovuotvorení chyby napísať do komentára dôvod znovuotvorenia a kliknutím na tlačidlo „Edit“ zobraziť okno na upravovanie chyby. Do neho autor vloží doplňujúce informácie o chybe podľa 3. a).

Keďže sa editovaná chyba môže nachádzať v rôznych stavoch, je potrebné ponechať polia, ktoré sa nevypĺňajú (podľa tabuľky vyššie), s pôvodnými hodnotami.

## 4.7 Metodika evidencie úloh

Autor: Rastislav Kostrab

### 4.7.1 Atribúty problémov a ich hodnoty

### 4.7.2 Atribúty problému a ich popis

| **atribút** | **vysvetlenie** |
|---|---|
| Projekt | Rodičovský projekt, do ktorého problém patrí |
| Kľúč | Unikátny identifikátor, pod ktorým je problém uvedený |
| Názov | Jednoriadkový popis problému |
| Typ | Enumerácia (tabuľka č.3) |
| Stav | Enumerácia (tabuľka č.4) |
| Priorita | Enumerácia (tabuľka č.5) |
| Riešenie | Enumerácia (tabuľka č.6) |
| Ovplyvnené verzie | Verzie projektu, ktoré sú daným riešením ovplyvnené |
| Opravené verzie | Verzie projektu, ktoré boli alebo budú na základe daného problému opravené |
| Komponenty | Komponenty projektu, s ktorými daný problém súvisí |
| Kľúčové slová | Kľúčové slová, ktoré popisujú daný problém |
| Prostredie | Hardvérové alebo softvérové prostredie, s ktorým daný problém súvisí |
| Popis | Popis daného problému |
| Odkazy | Odkazy na súvisiace problémy |

| | |
|---|---|
| Pridelený | Pridelený pracovník pre riešenie daného problému |
| Reportér | Pracovník, ktorý vytvoril problém |
| Hlasy | Číslo, ktoré vyjadruje, koľko pracovníkov zahlasovalo za daný problém |
| Pozorovatelia | Číslo vyjadrujúce, koľko pracovníkov pozoruje riešenie daného problému |
| Do | Dátum, vyjadrujúci deň, kedy problém má byť vyriešený |
| Vytvorené | Dátum a čas, kedy bol problém vytvorený |
| Aktualizované | Dátum a čas, kedy bol problém aktualizovaný |
| Vyriešené | Dátum a čas, kedy bol problém vyriešený |
| Časový predpoklad | Predpokladaný čas, ktorý je pridelený na vyriešenie daného problému |
| Ostáva | Vyjadrenie, koľko času do vyriešenia daného problému ostáva od predpokladaného času riešenia |
| Vykázané | Výkaz od pracovníka, ktorý na danom probléme pracuje |
| Vývoj | Ak sa používa Stash alebo BitBucket, tento atribút obsahuje odkaz na vetvu, v ktorej sa daný problém rieši |

#### *4.7.2.1 Enumerácie*

Nižšie sú v tabuľkách vypísané všetky enumerácie pre vybrané atribúty.

##### 4.7.2.1.1 Enumerácie atribútu typ

| **Enumerácia** | **Typ** |
|---|---|
| Chyba | Daný problém je chybou v produkte |

| | |
|---|---|
| Vylepšenie | Problém sa venuje zlepšeniu funkcionality |
| Nová funkcionalita | Problém sa venuje vývoju novej funkcionality |
| Úloha | Problém je úloha, ktorá sa musí splniť |

#### 4.7.2.1.2 Enumerácie atribútu priorita

| **Enumerácia** | **Priorita** |
|---|---|
| Blokujúca | Najvyššia priorita, zabraňuje plneniu ostatných problémov |
| Kritická | Ide o prioritu, pri ktorej je potrebné problém urgentne vyriešiť |
| Majoritná | Závažný problém |
| Minoritná | Nízka priorita |
| Triviálny | Najnižšia priorita |

#### 4.7.2.1.3 Enumerácie atribútu stav

| **Enumerácia** | **Priorita** |
|---|---|
| Otvorený | Problém je otvorený a čaká na pridelenie pracovníka, ktorý na ňom bude pracovať |
| Pracuje sa na tom | Na probléme sa pracuje |
| Vyriešený | Problém je vyriešený (nie ešte uzavretý) |
| Znovu otvorený | Problém je znovu otvorený z dôvodu nekompletného riešenia |
| Uzatvorený | Problém je kompletne vyriešený a uzavretý |

#### 4.7.2.1.4 Enumerácie atribútu riešenie

| Enumerácia | Riešenie |
|---|---|
| Vyriešené | Problém bol vyriešený |
| Nebude vyriešené | Problém nebude vyriešený |
| Duplikované | Problém je zadaný duplicitne |
| Nekompletné | Nedostatok informácii pre správane vyriešenie problému |

## 4.7.3 Procesy evidencie úloh

### *4.7.3.1 Vytvorenie problému*

##### 4.7.3.1.1 Postup vytvorenia problému

Nasledujúci postup popisuje kroky, po dosiahnutí ktorých v systéme JIRA vznikne problém:

1. Analyzovanie problému.
2. Skontrolovanie, či už v systéme podobný problém existuje.
3. Ak problém existuje, nevytvára sa ďalší, ale upraví sa nájdený podľa analýzy z bodu 1. Ak problém neexistuje, ide sa na bod 4.
4. Vytvorí sa problém v systéme JIRA

##### 4.7.3.1.2 Stav a riešenie po procese vytvorenia problému

| | |
|---|---|
| **Stav** | ● Otvorený<br>● Pracuje sa na tom<br>● Znovu otvorený |
| **Riešenie** | nezadané |
| **Role** | ● zákazník<br>● vývojár |

| | |
|---|---|
| | • tester<br>• reportér chyby |

### *4.7.3.2 Vykázanie práce*

#### 4.7.3.2.1 Postup vykázania práce

Nasledujúci postup popisuje kroky pri vykázaní práce v rámci daného problému:

1. V systéme JIRA sa nájde problém pre ktorý sa bude vykazovať.
2. V probléme sa vytvorí výkaz práce.
3. Výkaz sa potvrdí a skontroluje, prípadne opraví.

#### 4.7.3.2.2 Stav a riešenie po procese vykázania práce

| | |
|---|---|
| **Stav** | • Pracuje sa na tom |
| **Riešenie** | nezadané |
| **Role** | • Vývojár |

### *4.7.3.3 Vyriešenie problému*

#### 4.7.3.3.1 Postup vyriešenia problému

Nasledujúci postup popisuje kroky pri označení problému za vyriešený. Riešenie nemusí však znamenať, že funkcionalita, v rámci ktorej problém vznikol, je opravená.

1. Analyzovanie riešenia.
2. V systéme JIRA sa nájde problém pre ktorý sa bude označovať za vyriešený.
3. Problém sa označí za vyriešený a zvolí sa jeho riešenie.

#### 4.7.3.3.2 Stav a riešenie po procese vyriešenia problému

| | |
|---|---|
| **Stav** | • Vyriešený<br>• Uzatvorený |
| **Riešenie** | • Vyriešené |

| | |
|---|---|
| | ● Nebude vyriešené<br>● Duplikované<br>● Nekompletné |
| **Role** | ● Vývojár<br>● Projektový manažér |

## 4.7.4 Proces: Vykázanie práce

### *4.7.4.1 V systéme JIRA sa nájde problém pre ktorý sa bude vykazovať*

1. Po ukončení čiastkovej práce (čiastkovej v čase, nie celého problému) v rámci zadaného problému pre daného pracovníka, sa zadá do vyhľadávania problém, na ktorom pracoval. Vyhľadávanie problému je možné cez:

- rýchle hľadanie – „Quick Search“
- hľadanie problému - „Search for Issues“

2. Po zobrazení výsledkov vyhľadávania sa zvolí hľadaný problém. Po jeho zvolení sa problém zobrazuje používateľovi

### *4.7.4.2 V probléme sa vytvorí výkaz práce*

1. Zistia sa potrebné údaje o ukončenej čiastkovej práci:
    - čas, za ktorý sa čiastková práca vykonala,
    - čomu sa čiastková práca venovala,
    - čo sa počas čiastkovej práce vyriešilo.
2. Ak je splnený predošlý čiastkový bod č.2, môže sa pokračovať na bod č. 3.
3. Klikneme na tlačidlo vykazovania - „Log Work“.
4. Po vykonaní bodu č. 3 sa zobrazí okno pre vpisovanie údajov o výkaze práce.
5. Vypíše sa systémovo povinný údaj o čase strávenom počas čiastkovej práci na probléme („Time Spent“) a popis vykonanej práce („Work Description“). Ostatné polia sa nevypisujú.

### *4.7.4.3 Výkaz sa potvrdí a skontroluje, prípadne opraví*

1. Výkaz sa potvrdí tlačidlom potvrdenia - „Log“.

2. Po vykonaní bodu č. 1 sa zobrazí vo výkazoch daný potvrdený výkaz. Skontrolujú sa údaje vpísané do okna vykázania práce. Ak nie je potrebné vo výkaze niečo meniť, proces sa ukončí. V opačnom prípade sa pokračuje na bod č.3.
3. Klikne sa tlačidlo editovania a pokračuje sa bodom 1.

## 4.8 Metodika dokumentácie zdrojových súborov

### 4.8.1 Role a ich zodpovednosti

Vo vývojárskom tíme je niekoľko rolí, ktoré majú dočinenia so správnym komentovaním zdrojových kódov. Ich zoznam a popis zodpovedností je uvedený v tabuľke nižšie.

| Rola | Zodpovednosť |
|---|---|
| Manažér kvality | ● Zodpovedný za správne okomentovanie zdrojových kódov podľa daných pravidiel<br>● Kontroluje správnosť zápisu komentárov zdrojových kódov<br>● Ak zistí nedodržiavanie daného spôsobu komentovania programátorom, je v jeho kompetencii napomenúť ho |
| Programátor | ● Vytvára a komentuje zdrojové kódy |

### 4.8.2 Komentovanie zdrojových kódov

Existencia internej kultúry komentovania zdrojových kódov je dôležité hlavne z dôvodu konzistentnosti zdrojových súborov a znovu použiteľnosti. S touto internou kultúrou musia byť oboznámení všetci programátori, ktorí sa na vývoji podieľajú. Manažér kvality musí mať stanovené možné postihy pre programátorov v prípade nedodržiavania tejto internej kultúry komentovania zdrojových kódov.

### 4.8.3 JavaDoc

JavaDoc je nástroj, ktorý umožňuje vytvárať programátorskú dokumentáciu k vašim projektom v rovnakom štýle ako je samotná dokumentácia k Jave. Z toho vyplýva, že samotné vytváranie takej dokumentácie sa musí vykonávať cez štandardný nástroj, v našom prípade JavaDoc. Samotný zápis inštrukcií pre JavaDoc, ktoré sa majú spracovať sa umiestňuje do špeciálneho komentáru začínajúceho /** a uzavretého */. Tento zápis sa umiestňuje pred deklaráciu súboru, triedy či metódy.

| Názov | Popis |
|---|---|
| @author | Meno autora |
| @since | Verzia od ktorej je daný blok dostupný |

| | |
|---|---|
| @exception | Popis výnimky |
| @param | Popis jednotlivých parametrov predávaných metóde. |
| @return | Komentár k tomu čo vracia metóda |
| @see | Popis referencii |
| @version | Aktuálna verzia |

### 4.8.4 Proces komentovania zdrojových kódov

Samotný proces komentovania zdrojových kódov je možné rozdeliť do viacerých krokov. Po vykonaní prvého kroku (Vytvorenie elementu do ktorého sa bude písať komentár) sa môže programátor rozhodnúť, ktorým krokom bude pokračovať. Môže ako prvý okomentovať súbor, potom triedy a ich metódy ale aj naopak. Všetky komentáre sa píšu v slovenčine bez diakritiky.

| Poradie | Názov | Kapitola |
|---|---|---|
| 1 | Vytvorenie elementu do ktorého sa bude písať komentár | 5.1 |
| 2 | Vytvorenie komentáru ku súboru | 5.2 |
| 3 | Vytvorenie komentáru k triede | 5.3 |
| 4 | Vytvorenie komentáru k metóde | 5.4 |

### 4.8.5 Vytvorenie elementu do ktorého sa bude písať komentár

Pred samotným písaním opisu (komentáru) danej programovej štruktúry je potrebné vytvoriť blok, do ktorého sa bude komentár vypisovať.

**Vstup :** Neokomentovaný súbor, trieda alebo metóda

**Výstup :** Okomentovaný súbor, trieda alebo metóda

**Zodpovedný :** Programátor

- Programátor vytvorí súbor, triedu alebo metódu
- Na začiatok vloží element do ktorého sa bude písať komentár

**Eclipse IDE :**

- Eclipse automaticky generuje JavaDoc komentár s príslušnou značkou
    - Na začiatku súboru a pred triedami sa generuje autor (názov PC) komentáru
    - Pred metódami sa generuje zoznam parametrov a return
- Napíšte /** + Enter

```
1  package my.pckg;
2  /**
3   *
4   * @author Michal-PC
5   *
6   */
7
8  public class Person {
9      private String name;
10     private String surname;
11     private int age;
```

Obr. Príklad vytvorenia elementu pre súbor/triedu

## 4.8.6 Vytvorenie komentáru ku súboru

Každý súbor projektu bude zdokumentovaný a bude obsahovať nasledovné položky.

- Stručný popis obsahu súboru, štruktúrovaný do viet, umiestnený pred JavaDoc značkami.
- @author - Ak je autorov súboru viac, každý si pridá svoju značku @author do popisu súboru

**Vstup :** Existujúci súbor a element do ktorého sa má vpísať komentár pre súbor

**Výstup :** Vytvorený komentár pre súbor

**Zodpovedný :** Programátor

**Eclipse IDE :**

```
package my.pckg;
/**
 * Tento subor obsahuje triedu Person.
 * @author Michal-PC
 * @author Simon-PC
 *
 */

public class Person {
    /**
```

Obr. Príklad vytvorenia komentáru k súboru

## 4.8.7 Vytvorenie komentáru k triede

Každá trieda bude zdokumentovaná a bude obsahovať nasledovné položky.

- Detailný popis triedy a jej účelu, štruktúrovaný do viet, umiestnený pred JavaDoc značkami.

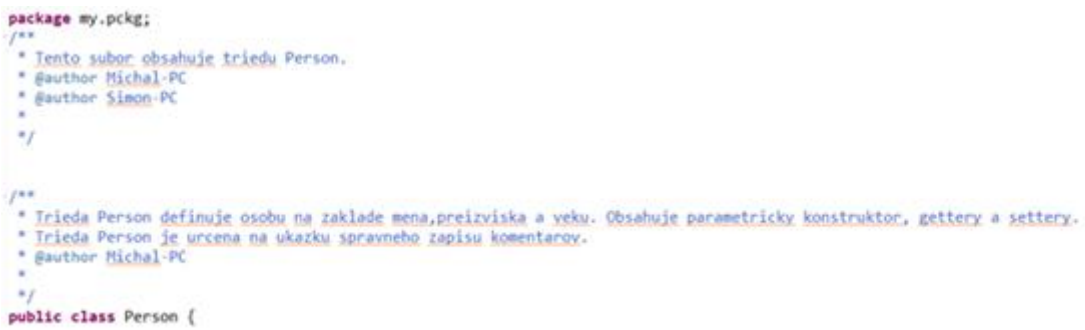
- @author - Ak je autorov súboru viac, každý si pridá svoju značku @author do popisu metódy

**Vstup :** Existujúca trieda a element pre komentár

**Výstup :** Vytvorený komentár pre triedu

**Zodpovedný :** Programátor

**Eclipse IDE :**




Obr. Príklad vytvorenia komentáru k triede

## 4.8.8 Vytvorenie komentáru k metóde

Každá metóda bude zdokumentovaná a bude obsahovať nasledovné položky.

- Detailný popis metódy a jej účelu, štruktúrovaný do viet, umiestnený pred JavaDoc značkami.
- @param - Názov parametra a stručný popis k nemu
    - 1..N
- @return - Definovanie návratovej hodnoty a stručný popis k nej
- @see - Metóda, ktorá danú metódu volá alebo metódy a triedy, ktoré majú s metódou súvis a programátor by ich mal vidieť pre dobré porozumenie danej metódy
    - 1..N v tvare : #nazovMetody(Dátové typy parametrov oddelene čiarkou)
    - Príklad : @see *#createRecord(int, String, String)*
- Gettery a settery sa nekomentujú
- Ak je konštruktor preťažený, komentuje sa vždy len jeden (Prvý v zdrojovom kóde)

**Vstup :** Existujúca metóda a element pre komentár

**Výstup :** Vytvorený komentár pre metódu

**Zodpovedný :** Programátor

**Eclipse IDE :**

```
    }
    /**
     * Metoda, ktora vracia priezvisko velkymi pismenami. Priezvisko je parameter metody.
     * @param surname Priezvisko osoby
     * @return surname - Priezvisko osoby velkymi pismenami
     * @see #print()
     */
    public String getSurnameUpperCase(String surname){
        return surname.toUpperCase();
    }

    public void print(){
        System.out.println(getName()+" "+getSurnameUpperCase(getSurname())+" ["+getAge()+"]");

    }
}
```

Obr. Príklad vytvorenia komentáru k metóde

## 4.8.9 Generovanie JavaDoc

Po správnom okomentovaní súborov, tried a metód si môže programátor vygenerovať programátorskú dokumentáciu vo forme štruktúry HTML stránok.

**Postup :**

1. Project -> Generate Javadoc
2. Nastavenie cesty k javadoc.exe, výber projektov a definovanie miesta uloženia.
3. Finish

**Generované stránky :**

- allclasses-noframe.html - Obdoba predchádzajúceho, pre prehliadače, ktoré nepodporujú framy
- allclasses-frame.html - Zoznam všetkých tried a všetkých balíkov (je to ľavý frame)
- deprecated-list.html - Zoznam deprecated API všetkých balíkov
- help-doc.html - Help pre užívateľov, popis obecnej hierarchie
- index.html - Hlavná stránka dokumentácie
- index-all.html - Defaultný index všetkého (tried, metód...)
- overview-tree.html - Hierarchia tried všetkých balíkov
- package-list.html - Zoznam mien jednotlivých balíkov
- serialized-form.html - Zoznam serialized vo všetkých balíkoch
- stylesheet.css - CSS (definícia fontov a farieb...) pre dokumentáciu

## 4.9 Metodika verziovania

### 4.9.1 Použitý nástroj

Na demonštráciu tvorby zmien v centralizovanom systéme je použitý nástroj SourceTree, ktorý je nahraditeľný za iné nástroje (git, nástroje integrované do IDE), ktoré obsahujú rovnakú funkcionalitu ako ponúka nástroj SourceTree.

### 4.9.2 Role procesu

| Rola | Popis |
|---|---|
| vývojár | ● v kontexte tvorby zmien táto rola zahŕňa všetkých používateľov repozitára, ktorý majú prístup k centrálnemu repozitáru s právami čítania a zápisu.<br>● v procese tvorby zmien je vývojár primárna rola, ktorá proces iniciuje a vo väčšine prípadov tento proces aj terminuje. Výnimkou môže byť riešenie nejednoznačných konfliktov. |
| správca repozitára | ● v kontexte tvorby zmien je táto rola sekundárna a vystupuje tu pri riešení nejednoznačných konfliktov. Konflikty by sa pri správnom rozdelení a naplánovaní úloh nemali stávať, ale aj napriek tomu môže takáto situácia nastať.<br>● táto rola môže byť nahradená rolou správca modulu, ktorá rieši nejednoznačné konflikty na úrovni modulov, kde každý modul má svojho vlastného správcu. |

### 4.9.3 Prerekvizity a nadväzné procesy

Pre tvorbu zmien nad repozitárom projektu je potrebné, aby pred tvorbou zmien boli vykonané nasledovné kroky:

1. správca repozitáru vytvorí a inicializuje centrálny repozitár
2. každý vývojár si vytvorí lokálnu kópiu centrálneho repozitáru
3. nasleduje proces tvorby zmien a verziovania, ktorý je detailne popísaný v tejto metodike
4. po vytvorení zmien nasleduje proces spájania vytvorenej funkcionality s hlavnou vývojovou vetvou

Procesy 1, 2 a 4 sú hlbšie popísané v dokumente *Manažment projektového repozitáru*.

## 4.9.4 Proces: verziovanie

Proces tvorby zmien a verziovania je rozdelený do 3 krokov (podprocesov), ktoré sú na seba nadväzné.

### *4.9.4.1 Aktualizácia repozitára*

Popis: pre potreby práce s aktuálnou verziou vyvíjaného softvéru je potrebná pravidelná aktualizácia, ktorá sa musí vykonávať vždy pred začatím tvorby zmien.

Postup:

1. V ľavej časti hlavného okna (zoznam repozitárov) dvojklikom zvolíme repozitár nad ktorým chceme vyvíjať. Ak tento repozitár už máme otvorený môžeme si ho zvoliť v lište s otvorenými repozitármi v pracovnom paneli.

2. Po zvolení repozitáru nasleduje výber typu aktualizácie repozitára

- Nezlučovacia aktualizácia (Fetch)
- Zlučovacia aktualizácia (Pull = Fetch + Merge)

#### 4.9.4.1.1 Nezlučovacia aktualizácia

Použitie: nezlučovacia aktualizácia sa používa, ak je potrebné stiahnuť zmeny z centrálneho repozitára a zároveň tieto zmeny zachovať v tvare akom sú v centrálnom repozitári. Je to presná kópia vetiev s odovzdanými súbormi pričom sú zachované aj lokálne zmeny, čim môžu vzniknúť konfliktné súbory.

Postup:

1. V hornej lište zoznam operácií vývojár zvolí operáciu Fetch.
2. Následne označí oba checkboxy, čo spôsobí aktualizovanie zoznamu repozitárov a zároveň odstránenie lokálnych vetiev, ktoré sú v centrálnom repozitári odstránené.
3. Operácia sa potvrdí pomocou tlačidla OK.

#### 4.9.4.1.2 Zlučovacia aktualizácia

Použitie: zlučovacia aktualizácia sa používa, ak je potrebné stiahnuť zmeny z centrálneho repozitára a zároveň tieto zmeny zlúčiť s vývojovou vetvou.

Postup:

1. V hornej lište zoznam operácií vývojár zvolí operáciu Pull.
2. V rámci projektu sú všetky repozitáre nastavené tak, aby obsahovali práve 1 vzdialený repozitár obvykle nazývaný „origin“. Pokiaľ bude zachovaná táto konvencia bude v zozname vzdialených repozitárov práve 1 možnosť s názvom „origin“, ktorú treba zvoliť.

3. Na rozdiel od nezlučovacej aktualizácie, ktorá aktualizuje všetky zmeny na všetkých vetvách, zlučovacia aktualizácia spája práve 2 vetvy, jednu vzdialenú vetvu z centrálneho repozitára a zvolenú „checkout-nutú“ lokálnu vetvu. Je nutné aby polia názvy týchto polí boli zhodné. Ak by neboli zhodné nešlo by o aktualizáciu, ale o spájanie vývojových vetiev, čo je samostatný proces popísaný v dokumente Manažment projektového repozitáru.

4. V možnostiach je potrebné označiť prvé 3 checkboxy. Prvý checkbox „Commit merged changes immediately“ zaručuje že súbory, ktoré sa podarilo zlúčiť budú odovzdané do centrálneho repozitára. Druhý checkbox „Include messages from commits being merged in merge commit“ spojí informácie o odovzdaniach a pri úspešnej aktualizácií sa tieto informácie zobrazia pri zlučovacom uzle v strome verzií. Tretí checkbox „Create new commit even if fast-forward is possible“ vytvorí samostatné odovzdanie.

### *4.9.4.2 Zvolenie pracovnej verzie*

Popis: každá vykonaná zmena vychádza z určitej verzie, ktorá sa nachádza v istej vývojovej vetve. Preto je potrebné vedieť ako zvoliť verziu z ktorej bude vychádzať nasledovná zmena.

Koncept udržateľného a efekivného repozitára


Obr. Model udržateľného a efektívneho repozitára

| Vetva | Popis |
|---|---|

| | |
|---|---|
| master | slúži na uchovávanie produkčných verzií softvéru |
| hotfix | táto vetva sa využíva v situáciách, kedy je potrebná rýchla oprava chyby produkčnej verzie, ktorá je závažného typu |
| release | táto vetva sa využíva pri tvorbe novej produkčnej verzie. Hlavným cieľom je zparaleizovať vývoj s tvorbou novej verzie |
| develop | vetva v ktorej sa má robiť vývoj. Do tejto vetvy by sa mali odovzdávať také súbory, ktoré nevytvárajú novú pridanú funkcionalitu, napr.: opravy chýb, zmeny existujúcich funkcionalít |
| feature | vetva určená na vývoj nových funkcionalít. Každá funkcionalita bude mať svoju vlastnú feature vetvu |

Postup:

1. Výber vetvy do ktorej má byť zmena zaradená.
2. Ak vetva neexistuje (môže sa stať pri vytváraní novej funkcionality), tak vývojár vytvorí vetvu s názvom jej názvom. Výnimkou je vetva funkcionalít (feature) kedy sa je konvenciou nazývať feature-<<názov_funkcionality>>. Napr.: je potrebné vytvoriť nové prihlasovacie okno, tak sa vytvorí vetva s názvom feature-login_page_v2
    a. Ak nová vetva vychádza z momentálnej pracovnej verzie, tak sa zvolí „working copy parent“. Nová vetva bude vychádzať z vetvy, ktorá je práve používaná.
    b. Ak nová vetva vychádza z inej ako poslednej revízie, tak je potrebné zvoliť „specified commit“ a následné pomocou pickera (tlačidlo s označením „...“) sa otvorí okno strom verzí. Tento prístup sa používa, keď je potrebné, aby nová funkcionalita vychádzala z istej produkčnej verzie, a tým sa predišlo obsahu nežiaducich súborov, alebo zmien v pracovnej verzií.
3. Ak vetva existuje, vývojár zvolí poslednú odovzdanú verziu v strome verzí, čím sa vo väčšine prípadov predíde vytváraniu enormného počtu vetiev. Ale môžu nastať prípady, kedy sa viacnásobnému vetveniu nedá predísť, napr.: pri prototypovaní a skúšaní viacerých prístupov na vyriešenie rovnakého problému.
4. Následné si voľbu verzie potvrdíme operáciou checkout zo zoznamu operácií, ktorá nám otvorí nové okno, ktoré je iba informatívne a stačí výber potvrdiť tlačidlom „OK“.

### *4.9.4.3 Tvorba a odovzdanie zmien*

Popis: tvorba zmien je najzakladanejšou funkcionalitou nad repozitárom. Repozitár je integrovaný s ďalšími nástroji pre podporu vývoja (Jira, Crucible + FishEye, Stash), ktoré uľahčujú vývoj, ale aj reportovanie a manažérske oblasti projektu preto je potrebné vytvárať odovzdania správne.

Postup:

1. V aktualizovanom repozitári môže teraz vývojár vytvárať, editovať a vymazávať súbory tak, ako predurčuje dokument Manažment kvality projektu.

2. V pracovnom paneli je umožnené pomocou drag-n-drop, alebo pomocou operácií zobrazenými medzi oknami s lokálnymi zmenami a oknom súborov zaradených do nasledovného odovzdania (commitu). Ako už bolo spomenuté „Staged files“ obsahujú súbory, ktoré budú zaradené do ďalšieho odovzdania a „Working copy files“ je zoznam súborov s lokálnymi zmenami oproti aktualizovanej zvolenej verzií.

3. Zoznam súborov zaradených do nasledovného odovzdania by mal mať formu najmenšej funkčnej jednotky, čo znamená, že zmenené súbory nespôsobia chyboví stav ostatným vývojárom a zároveň obsahuje len tie súbory, ktoré navzájom so sebou súvisia.

4. Odovzdanie súborov zahájime pomocou operácie commit, kde je potrebné správne nazvať odovzdanie. Názov odovzdania by mal vyzerať nasledovne TP-<<číslo_úlohy_v_JIRA>>: <<názov_úlohy>>: <<vykonaná_zmena>>. Napr.: TP-14: Naplnenie DB s dátami z XML Crepč: Hibernate configuration.

5. Checkbox „Create Pull request“ nie je potrebné používať a „Amend latest commit“ je zakázané používať, lebo ovplyvňuje odovzdané súbory bez vytvorenia samostatného odovzdania, čím sa stráca prehľadnosť. Checkbox „Push commits immediatly to“ umožňuje automatické odovzdanie zmien na server okamžite po odovzdaní na lokálnej úrovni. Tu je potrebné vybrať zo zoznamu vzdialených repozitárov práve „origin“.

6. Ak v kroku 5 neboli zmeny odovzdané na server je potrebné tieto zmeny odovzdať čím skôr, ale s ohľadom na funkčnosť projektu pomocou operácie Push. Otvorí sa okno, v ktorom stačí iba potvrdiť odovzdanie.

## 4.10Metodika prípravy podkladov na stretnutie so zákazníkom

Táto metodika je súčasťou širšieho okruhu manažmentu monitorovania projektu, kde sa nachádzajú rôzne iné metodiky a postupy, ktoré táto oblasť manažmentu pokrýva. Identifikovanými oblasťami sú:

- monitorovanie projektu počas aktuálneho šprintu
- monitorovanie projektu pred pravidelným stretnutím so zákazníkom
- monitorovanie projektu počas pravidelných stretnutí so zákazníkom
- monitorovanie projektu počas plánovania nasledujúceho týždňa/šprintu
- monitorovanie projektu realizované každým členom zvlášť v kontexte svojich priradených úloh
- monitorovanie projektu počas retrospektívy a sledovanie procesu monitorovania
- monitorovanie transparentnosti projektu tak, aby ho bolo možné monitorovať

V tejto metodike sa budeme venovať príprave podkladov na stretnutie so zákazníkom. Je zaradená v širšom kontexte monitorovania stavu projektu pred pravidelným stretnutím so zákazníkom.

### 4.10.1 Dôležitosť

Táto metodika je základným predpokladom schopnosti odkomunikovať stav projektu zákazníkovi, pretože pomáha utvárať si členom tímu, avšak najmä lídrovi tímu a manažérovi monitorovania projektu, predstavu o tom, v akom stave sa ich projekt nachádza, čo sa podarilo urobiť, či sa to podarilo urobiť tak, aby bol zákazník spokojný, či boli splnené všetky úlohy, ktoré sú pre zákazníka dôležité, a ktoré úlohy a vlastnosti softvéru ešte treba urobiť.

### 4.10.2 Účastníci

Role zúčastnené v tejto metodike sú:

- líder tímu – zodpovedný za komunikáciu so zákazníkom, za postup v riešení projektu a za splnenie požiadaviek zákazníka
- manažér monitorovania projektu – zodpovedný za kontrolu nad stavom projektu, jeho monitorovanie, musí mať vedomosť o tom, aký je stav projektu a zabezpečovať jeho transparentnosť
- ostatní členovia tímu – zodpovední za vhodné vykazovanie práce na projekte, postupu na riešení úloh

### 4.10.3 Nástroje

JIRA – nástroj na manažment projektu

Confluence – nástroj na evidovanie výsledkov práce, resp. výstupov čiastkových i úplných

Slack – komunikačný kanál na podporu komunikácie v tíme, je oddelený od ostatných sociálnych kanálov a určený len pre potreby internej komunikácie v tíme.

### 4.10.4 Rozhrania s inými metodikami

- metodika rozdeľovania úloh
- metodika zadávania úloh do JIRY (aj odhady trvania úloh)
- metodika evidovania čiastkových výsledkov práce do Confluence
- metodika evidencie práce v JIRE

### 4.10.5 Príprava na stretnutie

**Vedúci tímu**

- upovedomiť členov tímu, aby si doplnili do JIRY to, čo majú urobené
- vyzvať členov tímu a zozbierať od nich otázky na zákazníka
- spísať splnené a nesplnené úlohy

Povinnosťou vedúceho tímu je, aby bolo na stretnutie všetko pripravené, pretože zodpovedá za celý tím a jeho výsledky. Nasleduje preto tieto kroky:

1. Deň pred stretnutím so zákazníkom v rozmedzí 16. a 20. hodiny pošle všetkým členom tímu správu cez spoločný komunikačný nástroj Slack. V tomto nástroji sú viaceré kanály, z ktorých vyberie #general, aby túto správu dostal každý. Odoslaná správa musí obsahovať pokyn k skontrolovaniu a prípadnému aktualizovaniu záznamu práce, pokyn k zadaniu odrobených hodín práce. Tieto pokyny musia byť v rozkazovacom tvare, aby členovia tímu chápali dôležitosť tejto úlohy. Členovia tímu majú tendenciu na túto činnosť zabúdať, preto nemôže byť vynechaná.
2. Na splnenie týchto úloh nechá členom tímu dostatok času, čo je približne 1 – 2 hodiny.
3. Potom vyzve členov tímu, aby vedúcemu tímu poslali zoznam otázok, ktoré majú na zákazníka a ktoré sa nezodpovedali počas týždňa. Takýmito otázkami sú také, ktoré vznikli maximálne 2 dni pred stretnutím alebo také, ktoré potrebujú dôkladné prerokovanie, pretože sa týkajú zásadného vývoja projektu. Správu formátuje ako vyzvanie, aby nebol na členov tímu vyvíjaný nátlak. Podstatné je opakovane pripomínať, že sa nemusia hanbiť za svoje otázky a zároveň nesmú podceňovať svoje úlohy, aby projekt nezlyhal kvôli nedostatočnej komunikácii.
4. Následne si spíše v bodoch najdôležitejšie pokroky v projekte, ktoré zahŕňajú opravené chyby, vyriešené úlohy označené ako „Blockers“, novo pridané vlastnosti a funkcie. Tieto body bude prezentovať zákazníkovi, aby videl, ako napreduje vývoj projektu.
5. Po bodoch pokroku si spíše vedúci tímu na záchytný zoznam tých vecí, ktoré bude prezentovať zákazníkovi aj úlohy, ktoré neboli splnené. Zákazníkovi prezentujeme aj to, čo nie je tak, ako bolo

dohodnuté, aby sme zachovali transparentnosť projektu a korektnosť prístupu k zákazníkovi. Podklady pre tento krok vytvára manažér monitorovania projektu.

6. Po bodoch pokroku i nesplnených bodoch si pridá do záchytného zoznamu otázky zozbierané od členov tímu.

Výstupom vedúceho tímu v rámci tejto metodiky je záchytný zoznam obsahujúci nasledovné veci rozpísané v bodoch:

- zoznam splnených dôležitých úloh
- zoznam nesplnených úloh
- otázky na zákazníka

**Manažér monitorovania projektu**

- vyrobiť grafy dokumentujúce stav projektu
- pripraviť filtre na prezentáciu stavu úloh

Manažér monitorovania projektu je v rámci tejto metodiky zodpovedný za prípravu dokumentov potrebných na stretnutí a dokumentov tvoriacich podklady pre tohto manažéra, vedúceho projektu a v konečnom dôsledku i pre každého člena tímu.

Manažér monitorovania projektu musí pripraviť grafy znázorňujúce pokrok na projekte. Najdôležitejší je koláčový graf rozdelenia vyriešených úloh podľa priority. Tento graf nájde manažér v sekcii „Reports“. Najkratšia cesta vedie cez nasledovný link:

- http://arl6.library.sk:8080/browse/TP#selectedTab=com.atlassian.jira.plugin.system.project%3Areports-panel (Obrázok 1)

Tu má na výber mnoho typov grafov a sumárov, z nich vyberie „Pie Chart Report“, následne sa mu zobrazí ponuka, podľa čoho majú byť úlohy projektu rozdelené. V tejto sekcii vyberie voľbu „Priority“. Ďalej sa mu zobrazí samotný graf (Obrázok 2), ktorý je potrebné vytlačiť, pretože nie je možné obmedziť časový rozsah úloh, ktoré majú byť do grafu zahrnuté. Ďalej na vyššie uvedenom linku nájde taktiež sumár úloh zoskupený taktiež podľa priority jednotlivých úloh. K tomuto sumáru sa dostane kliknutím na „Single Level Group By Report“. Následne sú požadované dve veci:

- Filter – kde treba vybrať filter projektu, ktorý sa má zobraziť
- Statistic Type – kde treba vybrať kategóriu, podľa ktorej budú úlohy zoskupené

V časti Filter vyberie manažér filter pre aktuálny projekt a v druhej časti vyberie opäť kategóriu „Priority“, pretože táto kategória najviac zaujíma zákazníka. Ak ešte nie je žiadny filter pre projekt vytvorený, musí ho manažér vytvoriť nasledovným spôsobom:

1. Najprv klikne na horizontálnom menu na záložku „Projects“.
2. Potom vyberie aktuálny projekt v zozname (môže vybrať aj viac, ale takýto scenár nepredpokladáme v rámci Tímového projektu).
3. Takto vytvorený filter uloží pod názvom „Filter_monitorovanie_projektu“.

Vo vytvorenom sumári (Obrázok 3) sú teda zoskupené úlohy podľa priority, pričom pri každej úlohe je uvedený stav úlohy. Pre každú kategóriu je vo vrchnej časti taktiež zobrazené, koľko percent vytvorených úloh tvoria splnené úlohy. To pomáha zákazníkovi, ale aj iným členom tímu urobiť si predstavu o stave projektu, v akom sa nachádza (koľko úloh zostalo nesplnených, koľko úloh sa nezačalo ešte riešiť, koľko úloh už bolo vyriešených...). Ďalej tento sumár slúži vedúcemu tímu ako podklad pre tvorbu svojho záchytného zoznamu.

Tento sumár si pripraví manažér monitorovania projektu pred stretnutím na svojom počítači, aby ho potom mohol ukázať zákazníkovi. Takisto si pripraví aj daný graf, ktorý však musí vytlačiť, pretože na stretnutie musí priniesť tlačenú podobu grafu z minulého stretnutia. Aby sme však predišli komplikáciám s technikou a problémom s tým spojenými, musí manažér mať pripravené v zálohe aj tlačenú podobu týchto podkladov. Následne je na zákazníkovi, či preferuje tlačenú alebo elektronickú verziu. Vedúcemu projektu posiela tieto grafy vyexportované pomocou vstavanej funkcie „Export“ v pravom rohu obrazovky sumárov a grafov.

**Každý člen**

- skontrolovať aktuálnosť svojich informácií na Confluence
- pozrieť aktuálny stav ostatných úloh
- prichystať otázky na zákazníka a poslať ich vedúcemu

Pretože je každý člen súčasťou tímu, je v rámci zásad Scrum-u potrebné, aby mal prehľad o projekte, aby poznal jeho ciele a víziu projektu. V súlade s tým je povinný priebežne kontrolovať priebeh projektu a činnosti ostatných členov tímu. Vykonáva teda nasledovné činnosti:

- priebežne kontroluje obsah stránok na Confluence, o zmene ktorých dostáva aj notifikácie
- priebežne kontroluje stav úloh priradených k „epic“ úlohe, v rámci ktorej vykonáva prácu na projekte
- venuje sa otázkam, ktoré majú ostatní členovia tímu v snahe pomôcť im

Deň pred stretnutím nezávisle od pokynu vedúceho tímu ku skontrolovaniu obsahu stránok v Confluence skontroluje aktuálnosť obsahu stránky, v rámci ktorej vykonáva prácu na projekte. Drží sa nasledovného postupu:

1. Skontroluje, či stránka obsahuje všetky doposiaľ dosiahnuté výsledky práce.
2. Ak rieši v čase prípravy na stretnutie nejaký problém, ktorý doposiaľ nevyriešil, pridá čiastkové výsledky práce na tomto probléme.

3. Skontroluje, či stránka obsahuje odôvodnenia rozhodnutí učinených v rámci časti projektu, na ktorom pracuje.
4. Skontroluje formát stránky, aby bola prehľadná, čitateľná a informácie podávala stručne, jasne a transparentne.
5. Po ukončení všetkých kontrol, vyzve náhodného člena tímu, aby si prečítal túto stránku a zhodnotil jej vlastnosti, najmä prehľadnosť, zrozumiteľnosť a úplnosť.

Po vyzvaní od vedúceho tímu, aby predložil svoje otázky na zákazníka tak urobí, iba ak by nemal žiadne nejasnosti ohľadne postupu ďalšej práce. V rámci tvorby otázok si kladie tieto otázky, aby nič nepodcenil, ani neprecenil:

- Je moja nejasnosť spôsobená nepochopením zadania alebo nedeterministickým vyjadrením zadania?
    - Túto otázky vyrieši jednoduchá konzultácia s kolegom.
- Je táto otázka zodpovedateľná zákazníkom alebo je skôr technického rázu?
- Je táto otázka riešiteľná kreatívnym prístupom alebo je to otázka potrebujúca zásah zákazníka?

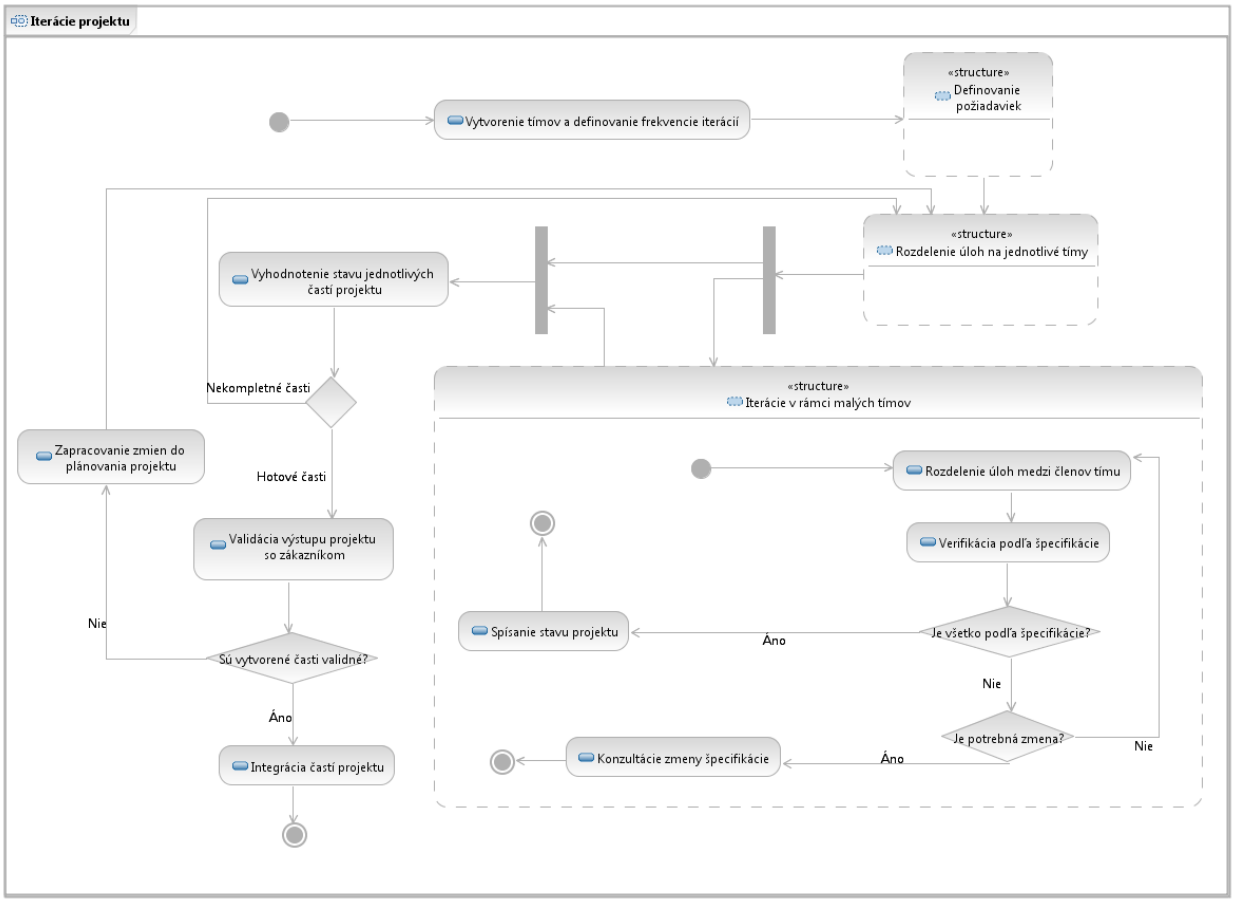
Výstupom prípravy každého člena tímu je kompletná stránka v Confluence, otázky pripravené pre zákazníka a komplexný obraz o stave projektu.

## 4.11Metodika iterácií projektu

Autor: Tomáš Jánošík

Táto metodika je určená pre veľké firmy na manažovanie procesov spojených s iteráciami projektov. Zahŕňa proces určenia nastavení tímu, ako je počet členov v tíme a počet tímov, opisuje procesy, ktoré by mali prebiehať v tíme, aby bola zabezpečený úspešný vývoj produktu.

Na úvod prikladáme procesný model, ktorý zobrazuje tok riadenia medzi procesmi a následnosť jednotlivých procesov. Opis tohto modelu je uvedený v nasledujúcej časti.




### 4.11.1 Opis procesného modelu metodiky

Metodika iterácií projektu má ako vstup zazmluvnený projekt, ktorý bol firme pridelený na riešenie. Na začiatku je potrebné definovať veľkosť a zloženie tímov a zároveň aj frekvenciu iterácií, v ktorých sa budú zástupcovia (vedúci) týchto tímov stretávať, aby prerokovali stav projektu a nasledujúci postup v ňom. Následne určení vedúci malých tímov určia pre svoj tím frekvenciu stretávania sa v rámci malého tímu.

Potom nasleduje získavanie a špecifikovanie požiadaviek od zákazníka. Túto časť pokrýva metodika plánovania úloh a metodika zberu požiadaviek. Ďalej nasleduje Rozdelenie úloh medzi jednotlivé tímy, túto časť pokrýva metodika plánovania úloh.

Ďalej nasleduje samotné riešenie projektu. Je rozdelené na iterácie v rámci menších tímov a iterácie celého tímu, ktorý rieši daný projekt.

Iterácie v rámci menších tímov začínajú vždy rozdelením úloh definovaných pre daný tím na jednotlivých členov tímu. Tento proces patrí pod manažment plánovania. Po uplynutí stanoveného času nasleduje verifikácia hotových úloh, či sú výsledky také, ako sa špecifikovalo. Ak výsledky práce zodpovedajú špecifikácii, členovia tímu spíšu svoje výsledky a pripravia ich na prezentáciu projektovému manažérovi a zákazníkovi.

Ak však výsledky špecifikácie nezodpovedajú špecifikácii, musí vedúci tímu rozhodnúť, či je potrebné prepracovať výsledky práce, alebo prepracovať špecifikáciu. V prípade, že je potrebné prepracovať špecifikáciu projektu, túto skutočnosť konzultuje vedúci tímu s manažérom rizík a s projektovým manažérom.

Po uplynutí stanoveného času na iteráciu celého projektu sa stretnú zástupcovia jednotlivých tímov (najlepšie vedúci tímov) a prejdú si stav projektu. V tomto procese zohráva dôležitú úlohu aj manažér monitorovania projektu, ktorý musí mať prehľad o stave projektu, čiže o postupe jednotlivých tímov v projekte a hotových, resp. nekompletných častiach projektu.

Nekompletné úlohy znova prejdú procesom rozdeľovania úloh na tímy, pretože niektoré tímy mohli ukončiť svoju časť práce a mohli by sa zapojiť do riešenia ostatných častí projektu. Tu zohráva dôležitú úlohu manažér plánovania projektu, ktorý musí vedieť, ktoré časti projektu sú dôležité a musí do procesu rozdeľovania zahrnúť aj nové požiadavky od zákazníka.

Hotové úlohy sú následne prezentované zákazníkovi. Tento proces zahŕňa v sebe aj otestovanie výsledkov práce podľa plánu testovania a akceptačných testov špecializovanými testermi, ktorí môžu byť zamestnancami zákazníka. Testovanie je popísané v metodike plánovania úloh. Zákazník po vzhliadnutí a otestovaní hotových častí projektu môže mať pripomienky k týmto častiam, prípadne môže zadefinovať nové požiadavky na zmenu.

Nové a bližšie špecifikované požiadavky na projekt sú zapracované do plánovania úloh a následne opätovne rozplánované na jednotlivé tímy.

Hotové časti projektu akceptované zákazníkom sú integrované do existujúceho systému, prípadne do už vytvorených častí systému (v prípade novo vytváraného systému alebo podsystému). Následne sú tieto integrované časti otestované v testovacej prevádzke a neskôr v ostrej prevádzke.

| Rola | Zodpovednosť | Proces |
|---|---|---|

| | | |
|---|---|---|
| Projektový manažér | Určenie počtu tímov, frekvencie iterácii | Vytvorenie tímov a definovanie frekvencie iterácií projektu |
| Člen tímu | Skontrolovanie, či sú splnené všetky body požiadaviek k danej úlohe | Verifikácia podľa špecifikácie |
| Vedúci tímu | Urobenie ešte jednej kontroly požiadaviek, najmä nefunkcionálnych | Verifikácia podľa špecifikácie |
| Člen tímu | Spísať výsledky kompletne tak, ako sú implementované, pripraviť zdrojové kódy na integráciu | Spísanie stavu projektu |
| Vedúci tímu | Spísanie akceptačného protokolu, kontrola úplnosti spísaných vecí | Spísanie stavu projektu |
| Vedúci tímu | Odprezentovanie výsledkov riešenia úloh | Vyhodnotenie stavu jednotlivých častí projektu |
| Manažér monitorovania projektu | Vytvorenie komplexnej správy zahŕňajúcej všetky splnené a nesplnené úlohy | Vyhodnotenie stavu jednotlivých častí projektu |
| Projektový manažér | Zhodnotenie stavu projektu a vyvodenie dôsledkov zaň | Vyhodnotenie stavu jednotlivých častí projektu |
| Projektový manažér | Odprezentovanie výsledkov riešenia úloh zákazníkovi | Validácia výstupu projektu so zákazníkom |
| Analytik | Získať doplňujúce požiadavky, ktoré má zákazník na výsledky riešenia úloh | Validácia výstupu projektu so zákazníkom |

| | | |
|---|---|---|
| Administrátor | Pripraviť prostredie na integráciu, naplánovať integráciu | Integrácia častí projektu |
| Členovia tímu | Kontrola integrovaných častí s ohľadom na ich funkčnosť | Integrácia častí projektu |

### 4.11.2 Projektové role

- Manažér projektu – je zodpovedný za celý projekt, za jeho smerovanie a výsledky, vyhodnocuje stav celého projektu, rozhoduje o dôležitých otázkach projektu, ako aj o počte tímov potrebných na splnenie projektu
- Manažér monitorovania projektu – je zodpovedný za monitorovanie stavu projektu, sumarizuje hotové a nevyriešené úlohy do jednotného výpisu, pripravuje podklady pre stretnutie so zákazníkom
- Vedúci tímu – je zodpovedný za tím ľudí, za výsledky úloh im pridelených a motiváciu tímu, kontroluje stav úloh, pokrok v ich riešení, komunikuje s manažérom projektu s ohľadom úloh
- Člen tímu – je zodpovedný za úlohy jemu pridelené, za ich vyriešenie a pripravenie na integráciu
- Analytik – je zodpovedný za získavanie požiadaviek a analýzu systému
- Administrátor – je zodpovedný za integráciu, vykonáva činnosti spojené s integráciou

### 4.11.3 Opis procesov

Nasleduje opis jednotlivých procesov spolu s vysvetlením ich dôležitosti, vstupmi, výstupmi a scenárom.

### 4.11.4 Vytvorenie tímov a definovanie frekvencií iterácií

Tento proces je dôležitý pre nastavenie parametrov riešenia projektu. Vytvorenie tímov je základom pre úspešné riešenie projektu, pretože tímy by mali byť zohrané a fungujúce. Ak zostavujeme nový tím, musí prejsť viacerými fázami, počas ktorých sa spoznáva, kým začne fungovať naplno. Ďalej je dôležité rozhodnúť o veľkosti tímu, ktorá by mala byť primeraná. Potom je potrebné zadefinovať ako často sa majú tímy stretávať a ako často sa bude vyhodnocovať riešenie projektu zástupcami všetkých tímov spolu s projektovým manažérom.

| | |
|---|---|
| **Vstupy** | Projekt vybraný na riešenie, zadefinovaná ideálna veľkosť tímu |
| **Výstupy** | Zadefinované tímy a iterácie v rámci tímu ako aj v rámci všetkých tímov spolu |
| **Role** | Projektový manažér |

Scenár:

1. Projektový manažér určí náročnosť daného projektu.

2. Podľa zadefinovanej ideálnej veľkosti tímu projektový manažér určí, koľko tímov je potrebných na riešenie projektu, a to nasledovne.

3. Podľa zazmluvnených MD a podľa veľkosti tímu určí, koľko tímov je potrebných a to podľa priemerného počtu MD, ktorý tím s danou veľkosťou dokáže urobiť.

4. Následne určí, či je stanovený počet tímov primeraný. V niektorých prípadoch môže byť žiaduce mať radšej väčšie tímy, napríklad preto, že máme časť projektu, ktorá vyžaduje viac práce ako ostatné a činnosti v rámci tejto časti úzko súvisia.

5. Ku každému tímu priradí vedúceho tímu, ktorý bude zodpovedať za prácu v tíme.

6. Projektový manažér určí, ako dlho budú trvať iterácie v rámci všetkých tímov spoločne. Túto hodnotu nastaví podľa požiadaviek zákazníka ako často chce mať dodávaný prototyp. Každopádne by nemal prekročiť dĺžku jeden mesiac, pretože chceme prototyp dodávať čo najčastejšie. Pri prekročení limitu musí tento krok projektový manažér zdôvodniť v dokumentácii.

7. Vedúci tímu určí, ako často budú prebiehať iterácie v rámci tímu jemu prideleného. V rámci malého tímu by sa mali stretnúť aspoň 2-krát za jednu iteráciu projektu.

### 4.11.5 Verifikácia podľa špecifikácie

Tento proces sa odohráva po uplynutí „malej iterácie“ v rámci tímu. Začína tesne pred stretnutím, keď je každý člen tímu povinný skontrolovať svoju prácu. Je to preto, aby v každom okamihu vývoja produktu bolo jasné, že jeho vývoj prebieha podľa špecifikácie. A keďže vedúci tímu je zodpovedný za svoj tím a jeho prácu, musí skontrolovať, či práca na projekte prebieha podľa špecifikácie.

| | |
|---|---|
| **Vstupy** | Úlohy rozdelené na jednotlivých členov tímu, výsledky práce na úlohách |
| **Výstupy** | Dokument popisujúci, či sú úlohy vyriešené podľa špecifikácie |
| **Role** | Vedúci tímu, členovia tímu |

Scenár:

1. Každý člen tímu sa pozrie znovu na svoje výsledky riešenia a skontroluje ich podľa špecifikácie úloh, ktoré mu boli dodané. Kontroluje hlavne, či sú splnené všetky funkcionálne požiadavky a či je jeho riešenie funkčné. Vypracuje aj dokument, v ktorom sa vyjadrí ku každej funkcionálnej požiadavke či a ako je splnená.

2. Celý tím sa stretne so svojim vedúcim, aby vedúci tímu ešte raz prešiel dokumenty vypracované svojim tímom. Skontroluje postupne s každým členom jednotlivé body, ktoré prekonzultujú.

3. Ak nastane možnosť, že funkcionálna požiadavka nebola splnená, konzultuje vedúci s daným členom tímu dôvod, pre ktorý nebola splnená. Môžu nastať dve možnosti:

    a. nebola splnená kvôli tomu, že ju nebolo možné v daných podmienkach splniť a potrebuje zmenu špecifikácie alebo inú zmenu týkajúcu sa nastavenia projektu a technológii použitých na jeho riešenie; táto zmena je konzultovaná s manažérom rizík a s projektovým manažérom

    b. nebola splnená z iného dôvodu, potom sa pokračuje v riešení danej úlohy dokým nebude zodpovedať špecifikácii

### 4.11.6 Spísanie stavu projektu

Tento proces nasleduje po verifikácii výsledkov riešenia projektu a má za úlohu pripraviť podklady pre kontrolu v rámci všetkých tímov a pripraviť riešenia úloh na integráciu. Je dôležitý, aby boli zachytené výsledky kontroly vedúceho tímu a aby tieto výsledky kontroly boli spísané jednotne a pospolu. Ďalej pripravuje hotové úlohy na integráciu a testovanie. Na tieto činnosti je potrebné, aby súbory so zdrojovými kódmi boli v určitom formáte a aby testeri mali jasno v tom, ako funguje daná časť projektu.

| | |
|---|---|
| **Vstupy** | Dokument popisujúci, či sú úlohy vyriešené podľa špecifikácie, zdrojové kódy riešenia úloh |
| **Výstupy** | Dokument zhrňujúci hotové a nedokončené úlohy s obsahujúci stručný popis ich riešenia, zdrojové kódy pripravené na integráciu a testovanie |
| **Role** | Vedúci tímu, členovia tímu |

Scenár:

1. Každý člen tímu pripraví svoje zdrojové kódy podľa konvencie tak, aby neobsahovali debugové výpisy a nepotrebné časti kódov.

2. Zdrojové kódy umiestni na určené miesto, kde ich administrátor vykonávajúci integráciu ľahko nájde.

3. Ďalej zadefinuje v akom poradí sa majú zdrojové kódy integrovať, pretože niektoré časti kódu budú závisieť od iných (napríklad pri databázach).

4. Člen tímu zadefinuje aj možné vstupy pre svoje riešenie, implementovanú funkcionalitu a očakávané výstupy, aby tester vedel otestovať funkčnosť integrovaných častí.

5. Vedúci tímu zosumarizuje hotové a nedokončené úlohy do zoznamu, z ktorého bude jasné, ktoré časti sú hotové a ktoré nie, a pošle ich manažérovi monitorovania projektu

6. Ku každej riešenej úlohe s pripraví do zoznamu stručný opis riešenia.

7. Taktiež si k zoznamu nedokončených úloh napíše dôvod, pre ktorý neboli dokončené v prípade, že už mali byť hotové.

### 4.11.7 Vyhodnotenie stavu jednotlivých častí projektu

Tento proces sa odohráva po uplynutí času stanoveného pre „veľkú iteráciu“ platnú pre všetky projekty. Je dôležitý, aby vrcholový manažment mal predstavu o riešení a stave projektu, aby tento stav mohli prezentovať zákazníkovi a dohadovať termíny, prípadne ich posunutie.

Taktiež je to dôležité, aby sa celý tím riešiaci daný projekt dohodol na integrácii určitých častí projektu a na prezentácii hotových častí zákazníkovi, pretože niektoré časti spolu súvisia, a tým pádom by mali byť prezentované spolu.

| | |
|---|---|
| **Vstupy** | Dokument popisujúci, či sú úlohy vyriešené podľa špecifikácie, rozdelenie úloh na jednotlivé tímy |
| **Výstupy** | Scenár prezentovania prototypu zákazníkovi, rozhodnutia urobené projektovým manažérom, zoznam úloh pripravených na testovanie |
| **Role** | Projektový manažér, vedúci tímov, manažér monitorovania projektu |

Scenár:

1. Projektový manažér si prejde pripravené dokumenty od každého tímu a s vedúcim tímu (prípadne s povereným členom tímu).

2. Pri nekompletných úlohách si vypočuje zdôvodnenie, prečo nie sú hotové.

3. Na základe vysvetlenia, prečo úlohy nie sú hotové vyvodí dôsledky pre projekt

    a. Poverí manažéra plánovania úloh, aby prepracoval plán riešenia úloh tak, aby tímy neboli zahltené prácou a zároveň bolo vyvinuté maximálne úsilie splniť projekt

    b. Udelí tresty za nesplnené úlohy

    c. Prekonzultuje so zákazníkom posunutie termínu kvôli závažným okolnostiam brániacim vyriešeniu projektu načas.

4. Projektový manažér spíše výsledky kontroly a rozhodnutia, ktoré boli uskutočnené po kontrole.

5. Manažér monitorovania projektu určí, ktoré hotové úlohy sú pripravené na prezentovanie zákazníkovi vzhľadom na nadväznosti jednotlivých úloh na seba.

6. Manažér monitorovania projektu alebo ním poverený človek pripraví scenár prezentovania prototypu zákazníkovi. Tento scenár popisuje funkcionalitu, ktorá bola implementovaná, spôsob vyriešenia úloh spojených s funkcionalitou a postupnosť prezentovania funkcionalít.

7. Vedúci tímov spíšu hotové úlohy pripravené na integráciu a odovzdajú príslušný zoznam administrátorovi a testovaciemu tímu.

### 4.11.8 Validácia výstupu projektu so zákazníkom

Tento proces sa odohráva u zákazníka a zahŕňa prezentovanie vyriešených častí projektu zákazníkovi, aby stanovil, či je riešenie použiteľné a či zodpovedá jeho predstavám o výsledku. Je dôležité, aby pri vývoji produktu boli časti samotného produktu validované zákazníkom, teda tým, ktorý bude pravdepodobne tento produkt využívať.

| | |
|---|---|
| **Vstupy** | Scenár prezentovania prototypu zákazníkovi, otestované hotové časti projektu |
| **Výstupy** | Dokumenty obsahujúce poznámky zákazníka k hotovým častiam projektu |
| **Role** | Projektový manažér, vedúci tímov, manažér monitorovania projektu |

Scenár:

1. Zákazník si pozrie zoznam vyriešených úloh, teda hotových častí projektu spolu s ich otestovanou verziou.

2. Zákazník dostane možnosť otestovať si sám implementovanú funkcionalitu v testovacom prostredí.

3. Projektový manažér asistuje pri testovaní zákazníkom a poskytuje vysvetľujúce informácie na základe dokumentu popisujúceho implementovanú funkcionalitu. Túto asistenciu môže vykonávať pri zložitejších funkcionalitách aj iný určený člen tímu.

4. Po skončení testovania zákazníkom položí projektový manažér (alebo človek ním určený) zákazníkovi doplňujúce otázky, ktorými zistí, ktoré časti projektu neboli dostatočne pochopiteľne spracované alebo ktoré by potrebovali dlhý čas na adaptovanie sa.

5. Projektový manažér (alebo človek ním určený) zapíše všetky pripomienky zákazníka.

6. Projektový manažér spolu so zákazníkom zhodnotia a prerokujú jednotlivé poznámky k riešeniu a implementácii prezentovaných častí projektu.

7. Projektový manažér rozdelí úlohy na tie, ktoré potrebujú upraviť a nie sú v súlade s predstavami zákazníka, a na tie, ktoré sú pripravené na integráciu do existujúceho systému, prípadne do iných hotových častí projektu.

### 4.11.9 Integrácia častí projektu

Tento proces sa odohráva po tom, ako sú vyriešené úlohy schválené projektovým manažérom, ako aj zákazníkom. Nachádza sa v metodike iterácií projektu preto, že ukončuje iteráciu daných úloh a vytvára ucelený výsledný produkt. Z tohto pohľadu je dôležité zašpecifikovať procesy spojené s integráciou riešení úloh. Tohto procesu sa zúčastňujú aj členovia tímu zodpovední za dané úlohy, aby kontrolovali funkčnosť integrovaných častí.

| | |
|---|---|
| **Vstupy** | Zoznam úloh pripravených na testovanie |
| **Výstupy** | Protokol o integrovaných úlohách |
| **Role** | Administrátor, členovia tímu |

Scenár:

1. Administrátor pripraví dokument integrácie, ktorý špecifikuje poradie integrácie riešenia úloh. Zároveň určí čas, kedy dôjde k tejto integrácii.

2. Administrátor pripraví prostredie na integráciu.

3. Postupne integruje riešenia úloh, ktoré mu boli dodané v zozname spolu s ich zdrojovými kódmi.

4. Administrátor zapisuje chyby a výpisy, ktoré nevie sám odstrániť.

5. Po úspešnej integrácii skontrolujú funkčnosť integrovaných úloh a taktiež funkčnosť úlohy v rámci integrovaného prostredia a funkčnosť rozhraní.

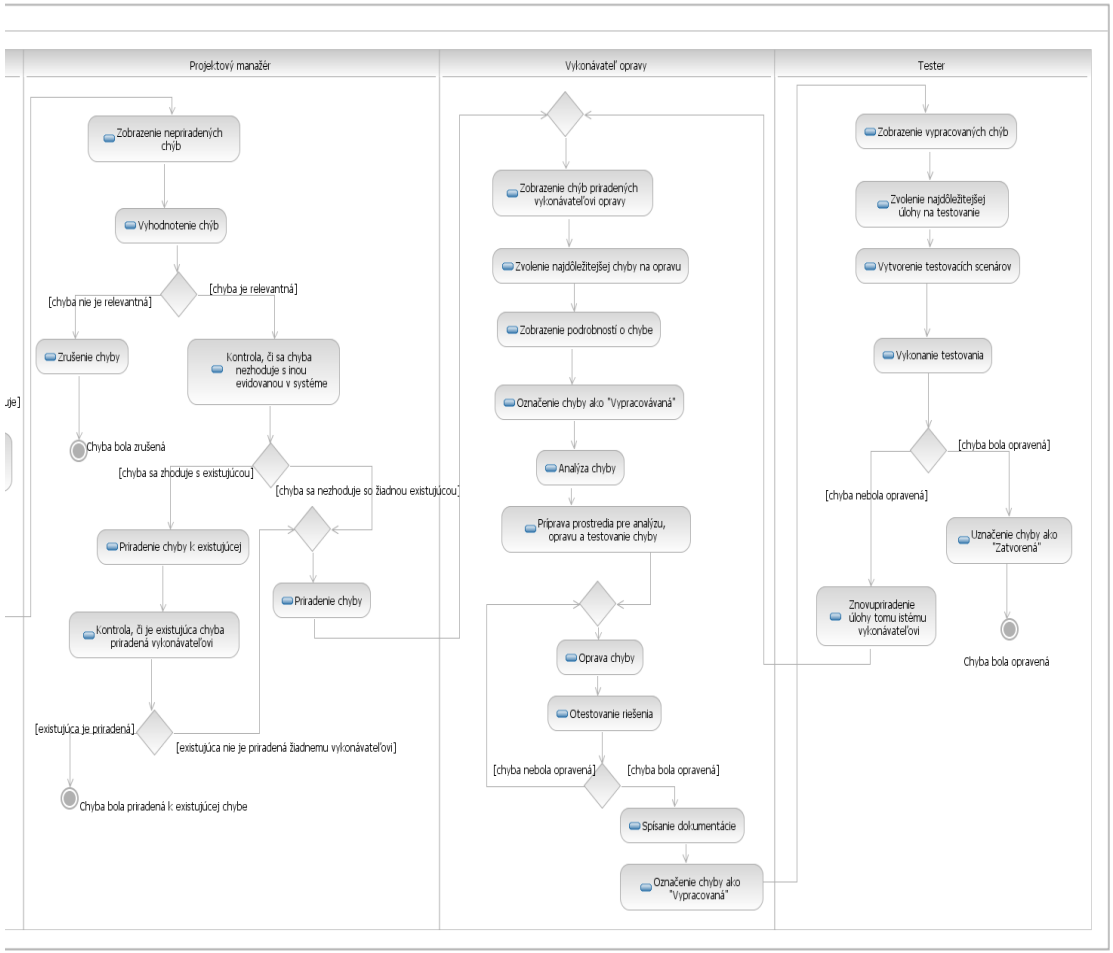
6. Pri zistení chyby alebo narušenej funkcionality zapíšu túto skutočnosť do záznamu integrácie.

7. Členovia tímu okamžite začnú pracovať na odstránení chýb.

## 4.12 Metodika manažmentu chýb

**Autor:** Stanislav Kubica

Nasledujúci diagram aktivít znázorňuje priebeh procesov, ktoré menia stav chyby. Znázorňuje tiež role, ktoré dané procesy vykonávajú (alebo sú inak do procesu zahrnuté).


Projektový manažér
Vykonávateľ opravy
Tester
Zobrazenie nepriradených chýb
Vyhodnotenie chýb
[chyba je relevantná]
[chyba nie je relevantná]
Zrušenie chyby
Kontrola, či sa chyba nezhoduje s inou evidovanou v systéme
Chyba bola zrušená
[chyba sa zhoduje s existujúcou]
[chyba sa nezhoduje so žiadnou existujúcou]
Priradenie chyby k existujúcej
Priradenie chyby
kontrola, či je existujúca chyba priradená vykonávateľovi
[existujúca je priradená]
[existujúca nie je priradená žiadnemu vykonávateľovi]
Chyba bola priradená k existujúcej chybe
Zobrazenie chýb priradených vykonávateľovi opravy
Zvolenie najdôležitejšej chyby na opravu
Zobrazenie podrobností o chybe
Označenie chyby ako "Vypracovávaná"
Analýza chyby
Príprava prostredia pre analýzu, opravu a testovanie chyby
Oprava chyby
Otestovanie riešenia
[chyba nebola opravená]
[chyba bola opravená]
Spísanie dokumentácie
Označenie chyby ako "Vypracovaná"
Zobrazenie vypracovaných chýb
Zvolenie najdôležitejšej úlohy na testovanie
Vytvorenie testovacích scenárov
Vykonanie testovania
[chyba bola opravená]
[chyba nebola opravená]
Uznačenie chyby ako "Zatvorená"
Znovupriradenie úlohy tomu istému vykonávateľovi
Chyba bola opravená

### 4.12.1 Proces 1: Vytvorenie záznamu o chybe

Proces predstavuje kroky pracovníka technickej podpory vedúce od nájdenia chyby vo vytváranom/udržiavanom softvérovom systéme po jej evidenciu v systéme pre evidenciu chýb.

| | |
|---|---|
| Stav chyby po vykonaní procesu | Nová |
| Zodpovedná osoba | Pracovník technickej podpory |

Postup pridania chyby:

1. Identifikovanie chyby autorom záznamu o novej chybe – autor záznamu zistí čo najviac doplňujúcich informácií, ktoré budú neskôr užitočné pre realizátora opravy.
2. Vyhľadanie chyby v systéme pre evidenciu chýb – autor zistí, či sa identifikovaná chyba už nachádza v systéme .

**Ak bola chyba už zaevidovaná:**

a. Vytvorenie nového záznamu o chybe – autor vytvorí nový záznam o chybe v systéme pre evidenciu chýb.

**Inak**

b. Vloženie doplňujúcich informácií k existujúcej chybe – autor vloží doplňujúce informácie k už existujúcej správe o chybe.

Bližšie podrobnosti o vykonaní tohoto procesu sú zahrnuté v Metodike pre vytvorenie záznamu o chybe.

### 4.12.2 Proces 2: Vyhodnotenie chyby

Popisuje kroky projektového manažéra, ktoré je potrebné vykonať pred priradením chyby realizátorovi opravy alebo jej zrušením, či priradením k existujúcej chybe.

| | |
|---|---|
| Stav chyby pred vykonaním procesu | Nová |
| Stav chyby po vykonaní procesu | Zrušená<br>Duplikovaná<br>Pridelená |
| Zodpovedná osoba | Projektový manažér |

Postup vyhodnotenia chyby

1. Zobrazenie nepriradených chýb – projektový manažér si v systéme pre evidenciu chýb zobrazí všetky nové chyby.
2. Vyhodnotenie chýb – na základe kritérií popísaných v dolnej úrovni metodiky uvedeného procesu vyhodnotenia určí, či je daná chyba relevantná a či je potrebné ju opravovať.

**Ak je relevantná:**

a. Kontrola, či sa chyba nezhoduje s inou už evidovanou v systéme – projektový manažér skontroluje v systéme pre evidenciu chýb, či sa daná chyba zhoduje s niektorou z existujúcich chýb.

- **Ak sa chyba zhoduje s existujúcou:**

i. Priradenie chyby k existujúcej – projektový manažér priradí chybu k už existujúcej chybe. Ak existujúca nie je priradená, pokračuje krokom Priradenie chyby.

- **Inak:**

ii. Priradenie chyby – projektový manažér na základe kritérií popísaných v dolnej úrovni metodiky uvedeného procesu vyhodnotenia určí vhodného kandidáta na vykonávateľa opravy a priradí mu ju.

- **Inak:**

b. Zrušenie chyby – projektový manažér označí chybu ako zrušenú.
Bližšie podrobnosti o vykonaní tohoto procesu sú zahrnuté v Metodike vyhodnotenie chyby.

### 4.12.3 Proces 3: Zahájenie práce na oprave chyby

Popisuje kroky vykonávateľa opravy, ktoré sa musia vykonať pred zahájením a počas samotnej opravy uvedenej chyby.

| | |
|---|---|
| Stav chyby pred vykonaním procesu | Pridelená |
| Stav chyby po vykonaní procesu | Vypracovávaná |
| Zodpovedná osoba | Analytik/Návrhár/Vývojár |

Postup zahájenia práce na úlohe:

1. Zobrazenie priradených chýb vykonávateľa opravy – vykonávateľ opravy si v systéme pre evidenciu chýb zobrazí zoznam všetkých jemu priradených chýb (v podobe úloh).
2. Zvolenie najdôležitejšej chyby na opravu – zvolenie najdôležitejšej chyby zo zoznamu priradených chýb (tie sú zoradené podľa dôležitosti)
3. Zobrazenie podrobností o chybe – otvorením chyby si zobrazí jej popis a pokračuje ďalším krokom.
4. Označenie chyby ako „Vypracovávaná“ - po otvorení chyby ju označí ako „Vypracovávanú“. V tomto stave sa bude chyba nachádzať až do jej opravenia.
5. Analýza chyby - zobrazí si podrobnosti a doplnkové informácie uvedenej chyby, zadané autorom záznamu o chybe. Na základe týchto informácií vypracuje analýzu pre ďalší postup.
6. Príprava prostredia pre analýzu, opravu a testovanie chyby – na základe predchádzajúcej analýzy záznamu o chybe vytvorí rovnaké podmienky, aké boli v čase výskytu chyby. Tento proces je bližšie opísaný v metodike dolnej úrovne uvedeného procesu.

Bližšie podrobnosti o vykonaní tohoto procesu sú zahrnuté v Metodike zahájenia práce na oprave chyby.

### 4.12.4 Proces 4: Ukončenie práce na oprave chyby

Popisuje kroky vykonávateľa opravy vedúce k vypracovaniu priradenej chyby zodpovednou osobou.

| | |
|---|---|
| Stav chyby pred vykonaním procesu | Vypracovávaná |
| Stav chyby po vykonaní procesu | Vypracovaná |
| Zodpovedná osoba | Analytik/Návrhár/Vývojár |

Postup vyriešenia chyby:

1. Opravenie chyby – vykonávateľ opravy opraví chybu na základe analýzy v procese Zahájenie práce na úlohe
2. Otestovanie riešenia vo vlastnej réžii – na základe scenárov opísaných v podrobnostiach chyby (popis, za akých podmienok chyba vznikla) otestuje opravenú chybu.

- **Ak testovanie preukáže že chyba bola opravená:**

a. Spísanie dokumentácie – vytvorenie dokumentu s popisom chyby, jej príčinou a spôsobom opravy. Ak je to potrebné, upravenie existujúcej projektovej dokumentácie (bližšie popísané v metodike dolnej úrovne daného procesu).
b. Označenie chyby ako „Vypracovaná“ - označenie chyby v systéme pre evidenciu chýb ako „Vypracovaná“ a zapísanie počtu hodín strávených na jej oprave.

- **Inak:**

c. Návrat ku kroku 1. - chyba nie je opravená, je potrebné ju opraviť.

Bližšie podrobnosti o vykonaní tohoto procesu sú zahrnuté v Metodike ukončenia práce na oprave chyby.

### 4.12.5 Proces 5: Testovanie vypracovanej chyby

Popisuje kroky, ktoré musí vykonať tester po opravení chyby jej vykonávateľom.

| | |
|---|---|
| Stav chyby pred vykonaním procesu | Vypracovaná |
| Stav chyby po vykonaní procesu | Zatvorená<br>Pridelená |
| Zodpovedná osoba | Tester |

Postup otestovania opravenej chyby:

1. Zobrazenie vypracovaných chýb v systéme pre evidenciu chýb – tester si v systéme pre evidenciu chýb zobrazí všetky vypracované chyby.
2. Zvolenie najdôležitejšej úlohy na testovanie – na testovanie zvolí najdôležitejšiu úlohu zo zoznamu vypracovaných úloh (tie sú zoradené podľa dôležitosti)
3. Vytvorenie testovacích scenárov – otvorením chyby si zobrazí jej podrobnosti a vypracuje testovacie scenáre (ich tvorba je bližšie popísaná v Metodike tvorby testovacích scenárov).
4. Vykonanie testovania – na základe vytvorených testovacích scenárov vykoná testovanie.

- **Ak testovanie preukáže že chyba bola opravená:**

a. Zatvorenie vyriešenej chyby – označí vypracovanú chybu ako zatvorenú.

- **Inak:**

b. Znovupriradenie neopravenej chyby jej pôvodnému vykonávateľovi – chybu je potrebné opraviť. Tester ju priradí jej pôvodnému vykonávateľovi. Bližšie podrobnosti o vykonaní tohoto procesu sú zahrnuté v Metodike testovania vypracovanej chyby.

## 4.13 Metodika plánovania úloh

### 4.13.1 Plánovanie úloh

Pre plánovanie úloh pri vývoji softvérového produktu je dôležitý manažment a definované procesy, ktorými sa manažment riadi. V rámci manažmentu existuje delenie, ktoré je v procesoch plánovania úloh jasne definované.

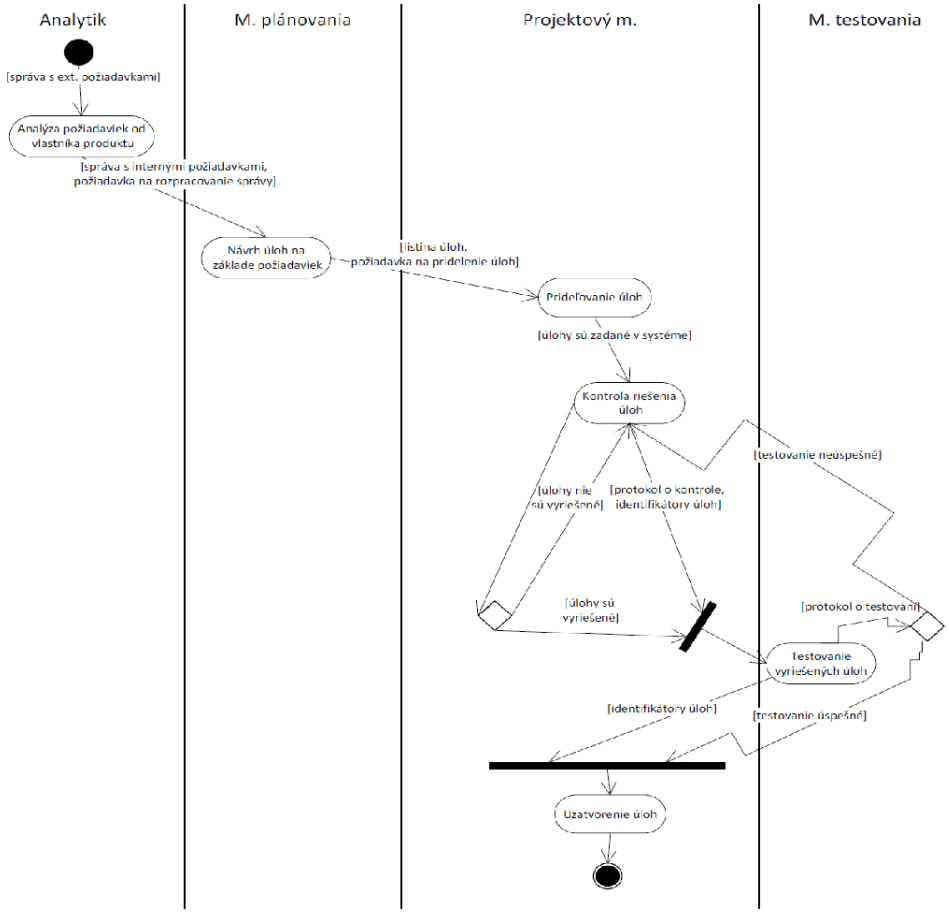
#### *4.13.1.1 Deklarácia procesov*

Nasledujúca tabuľka deklaruje procesy v manažmente plánovania, vystupujúce role a zodpovednú rolu, ktoré sú zahrnuté v rámci každého z deklarovaných procesov.

| Proces | Zodpovedná rola | Zahrnuté role |
|---|---|---|
| Analýza požiadaviek od vlastníka produktu | Analytik | Analytik,<br>manažér komunikácie,<br>projektový manažér,<br>manažér rizík,<br>manažér plánovania |
| Návrh úloh na základe požiadaviek | Manažér plánovania | Manažér plánovania,<br>analytik, |

| | | |
|---|---|---|
| | | manažér rizík |
| Prideľovanie úloh | Projektový manažér | Projektový manažér,<br>manažér plánovania<br>tím,<br>analytik |
| Kontrola riešenia úlohy | Projektový manažér | Projektový manažér |
| Testovanie vyriešenej úlohy | Manažér testovania | Manažér testovania,<br>tester |
| Uzatvorenie úlohy | Projektový manažér | Projektový manažér |

Deklarované procesy so medzi sebou úzko súvisia. Na obrázku nižšie je diagram aktivít, kde sú znázornené všetky relácie medzi procesmi, sekvencia v akej po sebe nasledujú, podmienky, ktoré musia byť v toku aktivít splnené a plavecké dráhy, ktoré definujú zodpovednú rolu pre procesy, ktoré každá z plaveckých dráh obsahuje. Diagram je vytvorený notáciou UML.


Analytik
M. plánovania
Projektový m.
M. testovania
[správa s ext. požiadavkami]
Analýza požiadaviek od vlastníka produktu
[správa s internými požiadavkami, požiadavka na rozpracovanie správy]
Návrh úloh na základe požiadaviek
[listina úloh, požiadavka na pridelenie úloh]
Prideľovanie úloh
[úlohy sú zadané v systéme]
Kontrola riešenia úloh
[testovanie neúspešné]
[úlohy nie sú vyriešené]
[protokol o kontrole, identifikátory úloh]
[úlohy sú vyriešené]
[protokol o testovaní]
Testovanie vyriešených úloh
[identifikátory úloh]
[testovanie úspešné]
Uzatvorenie úloh

### 4.13.2 Popis procesov

V tejto časti metodiky sú rozpísané všetky procesy deklarované z tabuľky č.1. Každý z procesov má popísaný vstup a výstup. Potom nasleduje stručný popis, scenár a zoznam rolí, ktoré v scenári vystupujú.

#### *4.13.2.1 Analýza požiadaviek od vlastníka produktu*

**Vstup:**

- správa s externými požiadavkami (tzv. „user stories“)

**Výstup**:

- správa s internými požiadavkami
- požiadavka na rozpracovanie správy

**Popis**:

Pred tým, ako sa súbor požiadaviek od vlastníka produktu rozdelí na úlohy, ktoré sa zadajú do systému a ktoré bude potrebné implementovať, je potrebné tento súbor požiadaviek skontrolovať na základe istých kritérií (v scenári: bod 2 a bod 3). Ďalej je potrebné súboru požiadaviek identifikovať typ, či ide o hlásenie chýb, vylepšenie funkcionality alebo o novú funkcionalitu. Nakoniec sa overí dĺžka trvania práce na požiadavkách, či je v súlade rámca šprintu, v ktorom požiadavky majú byť implementované a spíšu sa možné riziká, ktoré môžu ovplyvniť cestu k dosiahnutiu splnenia požiadaviek.

**Scenár**:

1. Analytik, ktorý je zodpovedný za vykonanie tohto procesu, príjme správu s externými požiadavkami od manažéra komunikácie.
2. Vyberie sa zo zoznamu požiadaviek jedna požiadavka a pokračuje bodom 3, ak zoznam požiadaviek je už prázdny pokračuje na bod 7.
3. Projektový manažér skontroluje, či podobná požiadavka už existuje. Ak áno, pokračuje bodom 2.
4. Analytik overí, či je požiadavka splniteľná.
5. Analytik definuje typ požiadavky.
6. Požiadavka sa vloží do správy s internými požiadavkami. Prechádza sa na bod 2.
7. Manažér rizík identifikuje riziká pre splnenie požiadaviek.

8. Manažér plánovania overí dĺžku trvania práce na splnení požiadaviek.
9. Projektový manažér vytvorí požiadavku na spracovanie správy s internými požiadavkami.

**Zahrnuté role:**

- analytik
- manažér komunikácie
- projektový manažér
- manažér rizík
- manažér plánovania

#### *4.13.2.2 Návrh úloh na základe požiadaviek*

**Vstup:**

- správa s internými požiadavkami
- požiadavka na rozpracovanie správy

**Výstup:**

- listina úloh
- požiadavka na pridelenie úloh

**Popis:**

V tomto procese sa správa s internými požiadavkami rozpracuje na úlohy, na základe možného blokovania jednotlivých úloh sa im priradí poradové číslo. Takmer celý tento proces prebieha kolaboratívne.

**Scenár:**

1. Manažér plánovania, ktorý je zodpovedný za vykonanie tohto procesu, príjme správu s internými požiadavkami.
2. Analytik roztriedi do skupín požiadavky, ktoré so sebou úzko súvisia takým spôsobom, že sa dajú zahrnúť do rámca čiastkovej implementácie.

3. Analytik a manažér plánovania kolaboratívne zoradia úlohy sekvenčne za sebou na základe identifikovaní možných blokovaní počas paralelnej práce na viacerých úlohách.
4. Manažér plánovania spíše úlohy do listiny úloh a vytvorí požiadavku na pridelenie úloh.

**Zahrnuté role:**

- analytik
- manažér plánovania

### *4.13.2.3 Prideľovanie úloh*

**Vstup:**

- listina úloh
- požiadavka na pridelenie úloh

**Výstup:**

- úlohy sú zadané v systéme

**Popis:**

V tomto procese sa kolaboratívne v tíme pre každú úlohu z listiny úloh určí dĺžka trvania plnenia úlohy, určí sa riešiteľ a zadá sa do systému.

**Scenár:**

1. Projektový manažér, ktorý je zodpovedný za vykonanie tohto procesu, na základe požiadavky na pridelenie úloh príjme listina úloh.
2. Vyberie sa z listiny úloh jedna z nich a pokračuje bodom 3, ak listina požiadaviek je už prázdna scenár sa ukončí.
3. Analytik, manažér plánovania a tím kolaboratívne identifikujú dĺžku trvania plnenia vybranej úlohy na základe metódy Planning poker.
4. Tím kolaboratívne zvolí medzi sebou zodpovedného riešiteľa za danú úlohu.
5. Projektový manažér danú úlohu vloží do systému.
6. Pokračuje sa bodom 2.

**Zahrnuté role:**

- projektový manažér
- manažér plánovania
- tím
- analytik

### *4.13.2.4 Kontrola riešenia úlohy*

**Vstup:**

- úlohy sú zadané v systéme

**Výstup:**

- protokol o kontrole
- identifikátory úloh

**Popis:**

Tento proces je určený iba pre projektového manažéra. Ten v ňom kontroluje, či niektoré úlohy sú vypracované. Ak áno, v protokole o kontrole definuje požiadavku na otestovanie čiastkových implementácií, ktorým zodpovedajú určité identifikátory úloh zo systému.

**Scenár:**

1. Projektový manažér, ktorý je zodpovedný za vykonanie tohto procesu, prezerá stav úloh zadaných v systéme.
2. Projektový manažér vyberie úlohu zadanú v systéme a pokračuje bodom 3, ak vybral už všetky úlohy, pokračuje sa bodom 6.
3. Projektový manažér skontroluje stav úlohy. Ak je v stave, že je vyriešená, pokračuje bodom 4, ináč pokračuje bodom 2.
4. Projektový manažér priradí identifikátor vybranej úlohy medzi identifikátory úloh, ktoré sú výstupom tohto procesu.
5. Projektový manažér pokračuje bodom 2.
6. Projektový manažér vypracuje protokol o kontrole na základe kontroly úloh.

**Zahrnuté role:**

- projektový manažér

### *4.13.2.5 Testovanie vyriešenej úlohy*

**Vstup:**

- protokol o kontrole
- identifikátory úloh

**Výstup:**

- protokol o testovaní
- identifikátory úloh

**Popis:**

V tomto procese sa otestujú všetky čiastkové implementácie zodpovedajúce identifikátorom úloh zo vstupu.

**Scenár:**

1. Manažér testovania, ktorý je zodpovedný za vykonanie tohto procesu, príjme protokol o kontrole a identifikátory úloh.
2. Manažér testovania vytvorí protokol testovania, ktorý bude doplnený výsledkami z testov.
3. Manažér testovania vyberie z pomedzi identifikátorov jeden z nich a pokračuje bodom 4, ak vybral už všetky identifikátory, scenár pokračuje bodom 8.
4. Manažér testovania na základe vybraného identifikátora nájde úlohu v systéme.
5. Na základe nájdenej úlohy vytvorí testovací scenár.
6. Tester príjme testovací scenár a podľa pokynov v ňom testuje čiastkovú implementáciu softvérového produktu a testy dokumentuje do protokolu.
7. Pokračuje sa bodom 3.

8. Manažér testovania potvrdí protokol, vloží ho do systému.

**Zahrnuté role:**

- manažér testovania
- tester

### *4.13.2.6 Uzatvorenie úlohy*

**Vstup:**

- identifikátory úloh

**Výstup:**

- uzatvorenie úlohy v systéme

**Popis:**

V systéme je obmedzenie, ktoré spočíva v tom, že úlohu môže uzatvoriť iba projektový manažér. V tomto procese je popísaný postup pri uzatváraní úlohy v systéme na základe vstupu.

**Scenár:**

1. Projektový manažér, ktorý je zodpovedný za vykonanie tohto procesu, príjme identifikátory úloh, ktoré sa majú uzatvoriť.
2. Projektový manažér vyberie z pomedzi identifikátorov jeden z nich a pokračuje bodom 3, ak vybral už všetky identifikátory, scenár končí.
3. Projektový manažér na základe vybraného identifikátora úlohy nájde v systéme úlohu.
4. Projektový manažér nájdenú úlohu označí ako uzatvorenú.
5. Pokračuje sa bodom 2.

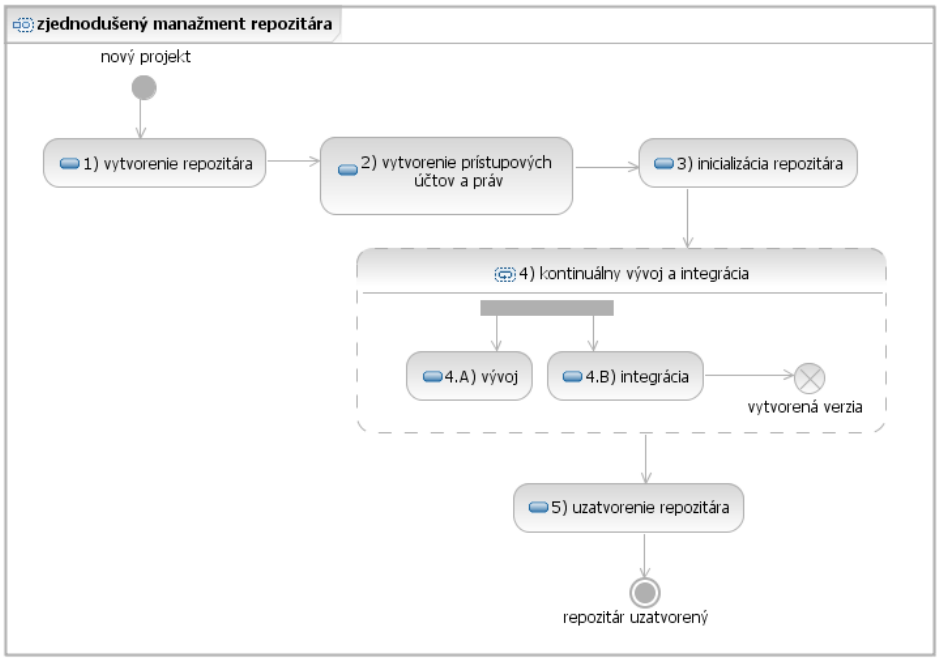
**Zahrnuté role:**

- projektový manažér

## 4.14 Metodika manažmentu projektového repozitára a verzií

Metodika je určená pre korporátne projekty na zjednotenie procesov súvisiacich s projektovými repozitármi, verziovaním, integráciou a nasadzovaním systému.

### 4.14.1 Zjednodušený procesný model



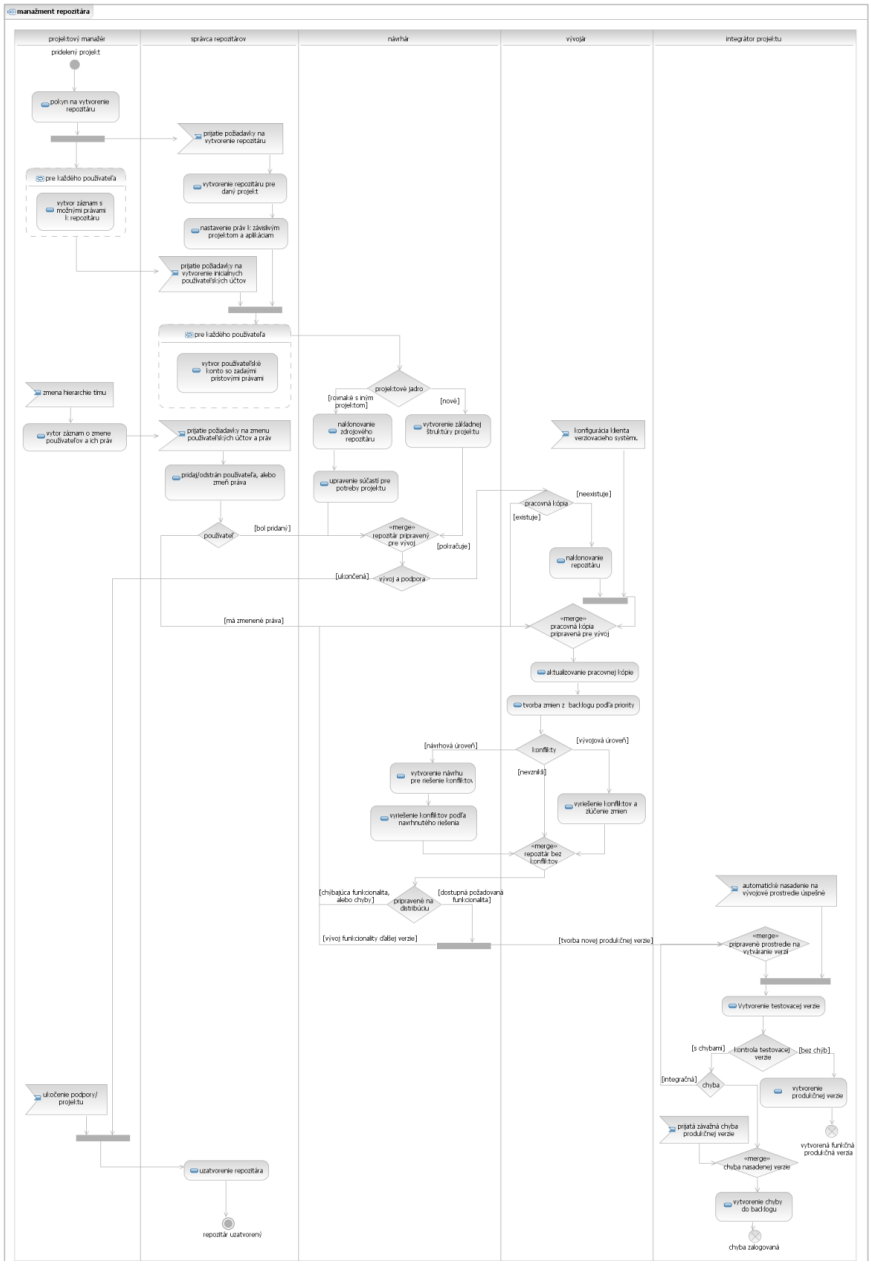
### 4.14.2 Role a ich zodpovednosti v jednotlivých procesoch

| Rola | Proces | Zodpovednosť |
|---|---|---|
| Projektový manažér | 1) Vytvorenie repozitára | Vydáva pokyn na vytvorenie repozitára |
| Projektový manažér | 2) Vytvorenie prístupových účtov a práv | Vytvára zoznam používateľov a ich práv k repozitára |
| Projektový manažér | 3) Inicializácia repozitára | Vydáva pokyn na inicializovanie základnej štruktúry repozitára |
| Projektový manažér | 4.A) Vývoj | Po dohode s návrhárom iniciuje začiatok vývojovej fázy |
| Projektový manažér | 4.B) Integrácia | Prijíma stav aktuálnej integrácie<br>Konzultuje nový možný termín nasadenia |
| Projektový manažér | 5) Uzatvorenie repozitára | Vydáva pokyn na uzatvorenie repozitára |
| Správca repozitárov | 1) Vytvorenie repozitára | Fyzicky vytvára repozitár |
| Správca repozitárov | 2) Vytvorenie prístupových účtov a práv | Spravuje prístupové účty a práva jednotlivých účtov k repozitáru |

| | | |
|---|---|---|
| Správca repozitárov | 3) Inicializácia repozitára | Povoľuje prístup návrhára ako aj samotného repozitára k existujúcim projektovým repozitárom z ktorých sa budú čerpať isté zdroje |
| Správca repozitárov | 5) Uzatvorenie repozitára | Fyzicky uzatvára repozitár na zmeny |
| Návrhár | 3) Inicializácia repozitára | Vytvára prvotný návrh ako aj samotnú štruktúru projektové repozitára a jeho komponentov |
| Návrhár | 4.A) Vývoj | Umožňuje začiatok vývojovej fázy správnou inicializáciou repozitára<br>Rieši návrhové konflikty |
| Návrhár | 4.B) Integrácia | Predpripravuje repozitár na vytváranie novej verzie<br>Umožňuje začiatok tvorby novej verzie |
| Vývojár | 3) Inicializácia repozitára | Môže návrhárovi pomôcť riešiť základnú iniciálnu štruktúru repozitára |
| Vývojár | 4.A) Vývoj | Vyvíja a testuje novú funkcionalitu spôsobom branch per feature, alebo branch per issue (závisí od potrieb projektu)<br>Rieši chyby rôznych úrovní spôsobom branch per bug<br>Rieši konflikty na vývojovej úrovni |
| Vývojár | 4.B) Integrácia | S integrátorom konzultuje a rieši nedorobenie funkcionality do termínu |

| | | |
|---|---|---|
| | | S integrátorom konzultuje a rieši chyby, ktoré spôsobil na vývojovom prostredí |
| Projektový integrátor | 4.A) Vývoj | Vytvára funkcionálne aj nefunkcionálne požiadavky na systém s ohľadom na splnenie zákazníkových potrieb |
| Projektový integrátor | 4.B) Integrácia | Vytvára novú verziu na testovacom a produkčnom prostredí<br>Loguje a rieši chyby vývojového, testovacieho a produkčného prostredia |

4.


manažment repozitára
projektový manažér
správca repozitárov
návrhár
vývojár
integrátor projektu
prideleni projekt
pokyn na vytvorenie repozitára
prijatie požiadavky na vytvorenie repozitára
pre každého používateľa
vytvor záznam s možnými právami k repozitáru
vytvorenie repozitára pre daný projekt
nastavenie práv k závislým projektom a aplikáciam
prijatie požiadaviek na vytvorenie lokálnych používateľských účtov
pre každého používateľa
vytvor používateľské konto so sadomi prístupovými právami
zmena hierarchie tímu
vytvor záznam o zmene používateľov a ich práv
prijatie požiadavky na zmenu používateľských účtov a práv
pridaj/odstráň používateľa, alebo zmeň práva
používateľ
[bol pridaný]
projektové jadro
[rovnaké s iným projektom]
[nové]
naklonovanie zdrojového repozitára
vytvorenie základnej štruktúry projektu
upravenie súčastí pre potreby projektu
«merge» repozitár pripravený pre vývoj
[pokračuje]
vývoj a podpora
[ukončená]
konfigurácia klienta verzovacieho systému
pracovná kópia
[neexistuje]
[existuje]
naklonovanie repozitára
[má zmenené práva]
«merge» pracovná kópia pripravená pre vývoj
aktualizovanie pracovnej kópie
tvorba zmien z backlogu podľa priority
konflikty
[návrhová úroveň]
[vývojová úroveň]
[nevyrieší]
vytvorenie návrhu pre riešenie konfliktov
vyriešenie konfliktov podľa navrhnutého riešenia
vyriešenie konfliktov a úpravenie zmien
«merge» repozitár bez konfliktov
[chybajúca funkcionalita, alebo chyby]
pripravenie na distribúciu
[dostupná požadovaná funkcionalita]
[vývoj funkcionality ďalšej verzie]
[tvorba novej produkčnej verzie]
automatické nasadenie na vývojové prostredie uzpešné
«merge» pripravenie prostredia na vytváranie verzií
Vytvorenie testovacej verzie
kontrola testovacej verzie
[s chybami]
[bez chýb]
[integračná]
chyba
vytvorenie produkčnej verzie
prijatá závažná chyba produkčnej verzie
vytvorená funkčná produkčná verzia
«merge» chyba nasadenej verzie
vytvorenie chyby do backlogu
chyba zalogovaná
uloženie podporu/ projektu
uzatvorenie repozitára
repozitár uzatvorený

### 4.14.4 Proces 1: Vytvorenie repozitára

| | |
|---|---|
| Časovanie | - Okamžite po získaní projektu<br>- Najneskôr pred začatím vývojovej fázy |
| Vstup | Požiadavky na prostredia (vývojové, testovacie, produkčné) |
| Výstup | Vytvorený a nakonfigurovaný repozitár podľa požiadaviek |
| Iniciačný účastník | Projektový manažér |
| Zodpovedný účastník | Správca repozitárov |
| Participujúci účastníci | - |
| Popis | Pre jednoduchšiu prácu vývojárov a zjednodušenie plánovania a vykonávania integrácie sa vytvára repozitár pre každý projekt. Repozitár sa nakonfiguruje a vytvoria sa práva k prístupu k projektom od, ktorých závisí |

#### 4.14.4.1.1 Procesný tok:

1. Po získaní a pridelení projektu, projektový manažér pošle požiadavku správcovi repozitárov o vytvorení repozitára s požadovanou konfiguráciou.
2. Správca repozitárov prijme požiadavku na vytvorenie repozitára.
3. Správca repozitárov vytvorí nový repozitár.
4. Správca repozitárov nakonfiguruje repozitár a nastaví práva k projektom od ktorých je závislý.

### 4.14.5 Proces 2: Vytvorenie prístupových účtov a práv

| | |
|---|---|
| Časovanie | - Po vytvorení repozitára<br>- Pred začatím vývoja musí mať návrhár systému vytvorené konto<br>- Prístupové účty vývojárov je najlepšie mať vytvorené tiež pred začiatkom vývojovej fázy<br>- Prístupové účty nových pracovníkov sa vytvárajú počas behu projektu |
| Vstup | Zoznam pracovníkov s ich právami |
| Výstup | Vytvorené jednotlivé prístupové účty |
| Iniciačný účastník | Projektový manažér |
| Zodpovedný účastník | Správca repozitárov |
| Participujúci účastníci | Pracovník na projektu |
| Popis | Z dôvodu bezpečnosti sa nevytvárajú projekty otvorené. Preto treba pre každého pracovníka projektu vytvoriť samostatný účet a prideliť mu príslušné práva. Požiadavku na vytvorenie prístupového účtu zasiela projektový manažér. Vo výnimočných prípadoch môže požiadavku poslať aj samotný pracovník, ale táto požiadavka musí byť dodatočne schválená. |

#### 4.14.5.1.1 Procesný tok:

1. Správca repozitárov prijme zoznam pracovníkov s prístupom k repozitáru s ich možnými právami.
2. Správca repozitára vytvorí prístupové konto pre každého pracovníka zo zoznamu.
3. Správca repozitára pridelí požadované práva zo zoznamu jednotlivým prístupovým účtom.

### 4.14.6 Proces 3: Inicializácia repozitára

| | |
|---|---|
| Časovanie | - Po vytvorení prístupových práv (aspoň pre návrhára)<br>- Po vytvorení základného návrhu riešenia |
| Vstup | Požiadavka na inicializovanie základnej štruktúry repozitára |
| Výstup | Repozitár pripravený na kontinuálny vývoj |
| Iniciačný účastník | Projektový manažér |
| Zodpovedný účastník | Návrhár |
| Participujúci účastníci | Správca repozitára<br>Vývojár |
| Popis | Projektový manažér pošle požiadavku na návrhára systému. Návrhár podľa analýzy uváži akým spôsobom inicializuje repozitár. |

#### 4.14.6.1.1 Procesný tok:

1. Návrhár prijme požiadavku na vytvorenie iniciálnej štruktúry repozitára od projektového manažéra.
2. Návrhár analyzuje iné projekty na možné prepoužitie zdrojov.

a. Ak je možné prepoužiť isté zdroje z iných projektov.

i. Návrhár naklonuje zdrojový repozitár do projektového repozitára.

ii. Návrhár upraví repozitár pre potreby projekta.

b. Ak nie je možné prepoužiť zdroje z iných projektov.

i. Návrhár vytvorí základnú štruktúru repozitára.

3. Návrhár oznámi projektovému manažérovi úspešné inicializovanie repozitára.

### 4.14.7 Proces 4: Kontinuálny vývoj a integrácia

Kontinuálny vývoj a integrácia je kontinuálny proces, ktorý je najdôležitejší pre fázy vývoja a testovania a asi aj najdôležitejší z pohľadu prínosu k ukončeniu projektu. V rámci týchto fáz by mal produkt nadobudnúť formu a funkcionalitu požadovanú zákazníkom, preto je potrebné, aby spomenuté podprocesy prebiehali paralelne a neblokujúco.

Pod vývojom sa v kontexte manažmentu repozitára a verzií rozumie akákoľvek aktivita, ktorá spôsobuje pridávanie, zmenu, alebo odstraňovanie súborov obsiahnutých v repozitári. Preto sa tu nebude rozlišovať oprava chýb a vývoj nových súčastí.

### 4.14.8 Podproces 4.A: Vývoj

| | |
|---|---|
| Časovanie | - Začiatok fázy vývoja<br>- Môže začať aj skôr ako istý vývoj v predstihu |
| Vstup | - Funkčný repozitár inicializovaný do požadovanej štruktúry<br>- Vytvorené prístupové kontá pre vývojárov<br>- Vytvorený backlog |

| | |
|---|---|
| Výstup | Kontinuálne funkčná jednotka možná nasadenia |
| Iniciačný účastník | Návrhár<br>Projektový manažér |
| Zodpovedný účastník | Vývojár |
| Participujúci účastníci | Návrhár<br>Projektový integrátor |
| Popis | Začiatok tohto procesu je možný po dohode projektového manažéra a návrhára, aby sa predišlo zapracovávaniu zmien do nesprávne iniciovaného repozitára, ale s ohľadom na dohodnutý termín začiatku. V tejto fáze sa vytvárajú všetky požiadavky na systém. Tento proces sa ukončuje rovnako ako aj samotný repozitár, teda po ukončení podpory, alebo samotného projektu.<br>Reálny vývoj sa začína, keď si každý člen vývojového tímu naklonuje projektový repozitár a je schopný vykonávať zmeny nad ním. |

#### 4.14.8.1.1 Procesný tok:

##### *4.14.8.1.1.1 Označené rímskymi číslicami sa vykonávajú len pred prvým použitím projektového repozitára.*

I. Každý člen vývojového tímu si nakonfiguruje klienta pre používanie distribuovaného systému verzií.

II. Každý člen vývojového tímu si naklonuje aktuálnu verziu repozitára.

1. Každý člen si pre vykonávaním zmien aktualizuje pracovnú kópiu repozitára.
2. Pri tvorbe zmien vývojár skontroluje projektový backlog a zvolí si jednu z prioritných úloh.
3. Vývojár vykoná zmeny a priebežne ich odovzdáva do repozitára.

a. Ak v repozitári nevznikli konflikty.
b. Ak v repozitári vznikli konflikty na vývojárskej úrovni.
i. Vývojár vyrieši konflikty.
ii. Vývojár zlúči zmeny.
c. Ak v repozitári vznikli konflikty na návrhárskej.
i. Návrhár vytvorí návrh riešenia konfliktov.
ii. Návrhár vyrieši konflikty podľa navrhnutého riešenia.
iii. Návrhár zlúči zmeny.
4. Vývojár označí úlohu za dokončenú.
5. Vývojár pokračuje vo vývoji na ďalších úlohách z backlogu krokom 1.

### 4.14.9 Podproces 4.B: Integrácia

| | |
|---|---|
| Časovanie | - Pred nasadením systému do prostredia podľa dohodnutých, alebo interných míľnikov (opakujúca sa aktivita) |
| Vstup | - Repozitár obsahujúci požadovanú funkcionalitu<br>- Úspešné automatické nasadenie na vývojové prostredie<br>- Požiadavka na vytvorenie novej verzie (explicitná, alebo z harmonogramu) |
| Výstup | Funkčný systém nasadení do prostredia |
| Iniciačný účastník | Projektový integrátor |
| Zodpovedný účastník | Projektový integrátor |

| | |
|---|---|
| Participujúci účastníci | Vývojár |
| Popis | Každé prostredie má svoje špecifikácie a systémy s ktorými treba vyvíjaný systém integrovať, preto je potrebné mať viac prostredí (vývojové, testovacie, produkčné), ktorými sa zabezpečí, že k zákazníkovi sa vždy dostaví produkt na čas a bez chýb.<br><br>Integrácia prebieha paralelne s vývojom a jej cieľ je nasadiť vyvíjaný systém do produkčného prostredia. |

#### 4.14.9.1.1 Procesný tok:

1. Integrátor prijme požiadavka, alebo má naplánovanú tvorbu novej produkčnej verzie.
2. Integrátor skontroluje, či sú všetky požadované súčasti verzie v stave na nasadenie.

a. Ak repozitár neobsahuje všetky požadované funkcionality, alebo automatické nasadenie a testovanie bolo neúspešné.

i. Integrátor konzultuje s vývojárom, ktorý spôsobil meškanie náročnosť opravy.

a. Ak chybovú časť je možné urýchlene opraviť.

i. Integrátor počká na ukončenie zmien vývojárom

b. Ak chybu nie je možné urýchlene opraviť.

i. Integrátor ohlási projektovému manažérovi meškanie integrácie.

ii. Integrátor po konzultácií s vývojármi naplánuje nový termín integrácie.

iii. Procesný tok končí.

b. Ak repozitár obsahuje všetky požadované funkcionality a automatické

i. Integrátor vytvorí a nasadí systém na testovacie prostredie.

a. Ak nasadenie prešlo bez chýb a testovacie prostredie je funkčné.

i. Integrátor nasadí systém na produkčné prostredie.

ii. Integrátor ohlási projektovému manažérovi meškanie integrácie.

b. Ak nasadenie na testovacie prostredie neprešlo, alebo bola prijatá závažná chyba na produkčnom prostredí.

i. Integrátor vloží do projektového backlogu nový chybový záznam s najvyššou prioritou riešenia.

ii. Integrátor ohlási stav projektovému manažérovi.

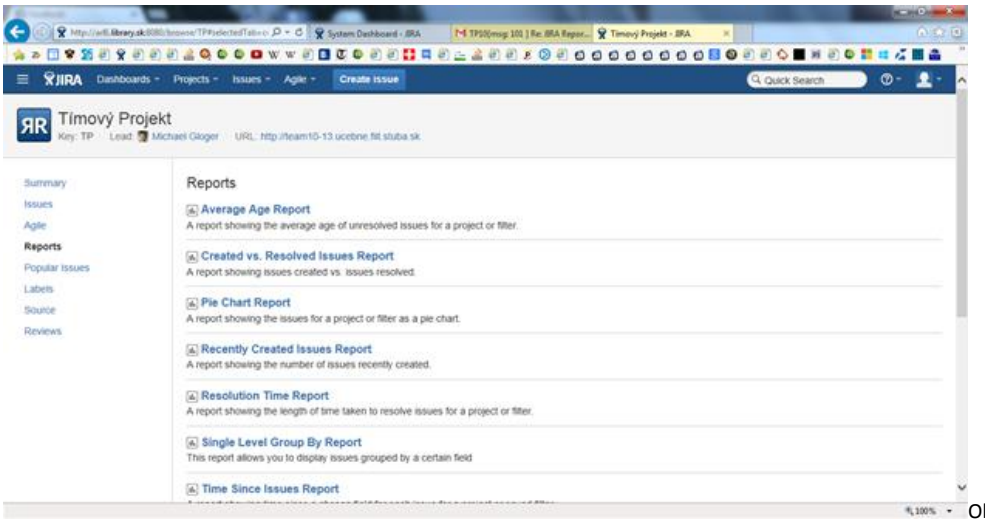
### 4.14.10 Proces 5: Uzatvorenie repozitára

| | |
|---|---|
| Časovanie | Po skončení projektu, alebo ukončení podpory |
| Vstup | Požiadavka na uzatvorenie repozitára |
| Výstup | Repozitár uzatvorený na zmeny |
| Iniciačný účastník | Projektový manažér |
| Zodpovedný účastník | Správca repozitára |
| Participujúci účastníci | - |
| Popis | Po ukončení projektu bez dohodnutej podpory, alebo ukončení podpory v rámci zmluvy sa repozitár nevymazáva, lebo jeho súčasti, alebo celý projekt môže byť využiteľný. Preto sa repozitáre nikdy nevymazávajú, iba sa uzatvoria na operáciu zmeny. |

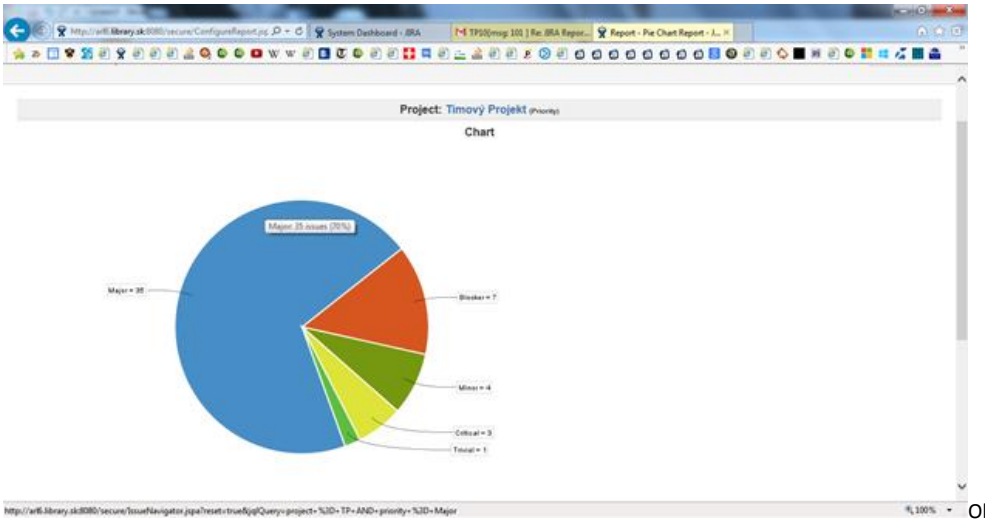
#### 4.14.10.1.1 Procesný tok:

1. Projektový manažér pošle požiadavku na uzatvorenie repozitára.
2. Správca repozitára po prijatí požiadavky uzavrie repozitár na zmenu

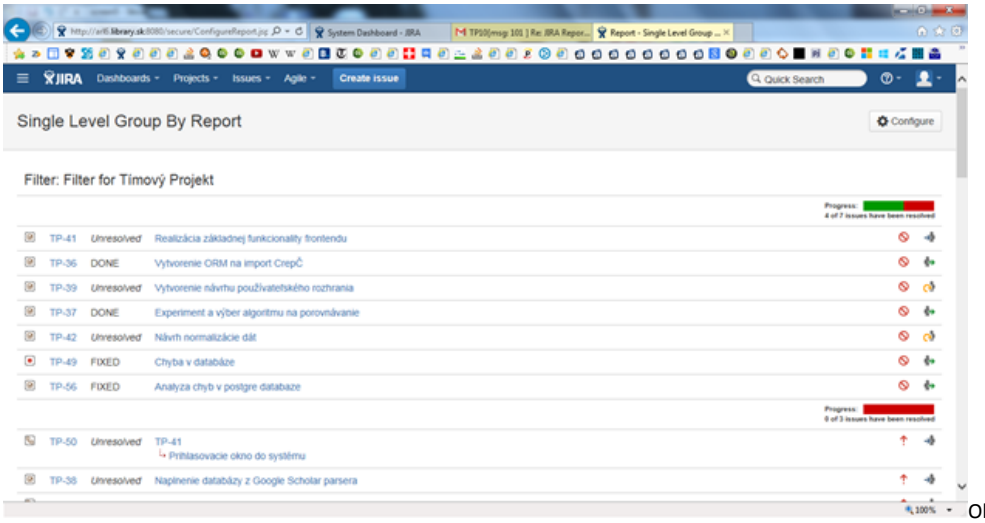
# 5 Prílohy



Obr. - Výber sumárov a grafov



Obr. - Koláčový graf



Obr. - Sumár úloh podľa priority

# Dokumentácia riadenia - letný semester

Nasledujúci dokument obsahuje popis zmien aplikovaných jednotlivými manažérmi poas letného semestra. Dokumentujú zmenu vzhadom k zimnému semestru.

## Zoznam manažérov a ich postov

Bc. Tomáš Jánošík - Manažér komunikácie

Bc. Daniel K - Manažér dokumentácie

Bc. Rastislav Kostráb - Manažér plánovania

Bc. Stanislav Kubica - Manažér rizík

Bc. Michael Gloger - Manažér kontroly stavu projektu

Bc. Šimon Kompas - Manažér podpory vývoja

## Manažment dokumentácie

Manažment dokumentácie prešiel poas zimných prázdnin významnou reštrukturalizáciou. Pri písaní dokumentácie sa vytvorilo viacej kontrolných bodov a dokumentácia podliehala prísnej kontrole. Obsah dokumentácie po novom podlieha štruktúre priloženej šablóny pre sprehadnenie požiadaviek na komponent, návrhu komponentu a z oho vychádzal, fyzickej organizácie komponentu a nakoniec rozhraní komponentu. Niektoré komponenty obsahujú alšie informácie v prípade použitia externých technológií alebo ak sa predpokladá, že komponentu bude treba pri znovapoužití upravi.
Niektoré komponenty obsahujú len niektoré súasti šablóny aby dané sekcie dokumentácie boli zmysluplné. Bola vyvinutá snaha, aby dokumentácia o možno najviac zodpovedala obsahom aj štruktúrou šablóne pre zvýšenie itatenosti.

Poas semestra boli exportované 3 verzie priebežnej dokumentácie pre neskoršie porovnanie ak to bude potrebné.

Vyplnenie súastí dokumentácie bolo pridelené jednotlivým lenom tímu a bolo vykonaných niekoko kontrol priebehu dokumentovania.

## Šablóna dokumentácie

Šablóna popisuje vnútornú štruktúru dokumentu. Táto stránka sa berie ako stránka komponentu, ktorý má by zdokumentovaný

### Úvod

Odsek - o to je?

Odsek - aké to je?

Odsek - o je úlohou tohto komponentu?

Odsek - zasadenie do prostredia, alebo systému v rámci oho to existuje

### Organizácia dokumentu

Obsahuje opis štruktúry asti dokumentácie

#### Špecifikácia požiadaviek

Opis požiadaviek, ktoré komponent spa, resp. má spa.

#### Návrh komponentu

Konceptuálny alebo logický návrh komponentu.

#### Implementácia komponentu

Fyzický návrh komponentu.

#### Špecifikácia rozhraní

Opis rozhraní na použitie pre používatea.

### Služby komponentu

Opis služieb, ktoré komponentu používa pre zákazníka.

# Šablóna špecifikácie požiadaviek

## Úvod

Uvies o je cieom dokumentu.

### Význam

o to je a pre koho je to urené

### Rozsah

Softvérové produkty, ktoré sú vyprodukované, ciele produktu

### Definícia skratiek a termínov

Uvádza skratky a termíny špecifické pre komponent

### Referencie

Referencie ak sú potrebné

### Prehad

Vysvetlenie organizácie špecifikácie požiadaviek, organizácie špecifických požiadaviek a pod.

## Všeobecné požiadavky

### Perspektíva produktu

Umiestnenie produktu na trhu, resp. v rámci konkurenných riešení. Ak je súasou väšieho riešenia, tak by mal uvádza limitácie s tým spojené.

### Funkcie produktu

Opis funkcií aké produkt zastane, zastreší - logický pohad nie fyzický.

### Charakteristika používatea

...

### Limitácie

Obsahuje fyzické limitácie produktu ako je vekos domény, bezpenos a pod.

### Predpoklady a závislosti

Obsahuje limitácie, ktoré ovplyvujú priamo požiadavky, resp. veci, ktorých zmena ovplyvní požiadavky. Napríklad predpoklad, že na stroji, na ktorom komponent má beža bude beža operaný systém.

## Konkrétne požiadavky

Obsahuje konkrétne veci, ktoré musia by implementované spolu s opisom ich vstupov(meno, popis, zdroj, formát) a výstupov(meno, popis, miesto urenia, formát). alej výstup v prípade chyby, akcie, ktoré komponent podnikne, vlastnosti a štandardy, ktoré musí daná funkcie rešpektova. Toto je opísané organizáciou, ktorá je pre daný komponent najzrozumitenejšie - napríklad opis poda objektov komponentu, opis poda služieb komponentu a pod.

## Doplujúce informácie

Príklady vstupov, príklady riešených problémov, v podstate okovek, vaka omu má lovek získa lepšie porozumenie požiadavkám

# Šablóna návrhu

Dokument je organizovaný tak, že je opísaný poda súastí, i už sú to služby, objekty, funkcie, alebo iné vhodné rozdelenie.

### Súas 1

Odsek Logický návrh komponentu

Odsek Workflow

Odseky Použité postupy a algoritmy a preo.

### Súas 2

...

### Súas n

## Šablóna Implementácie

Fyzický návrh komponentov

### Implementácia komponentu

Komponent diagram komponentu

Opis diagramu - interakcie s prostredím, interakcia súastí

#### Opis súasti komponentu 1

Vstup, výstup, moduly, závislosti, opis funkcie v rámci komponentu

#### Opis súasti komponentu 2

...

#### Opis súasti komponentu n

### Prehad modulov komponentu

Prehad modulov, ktoré patria do riešenia ako aj struný opis toho, o sa v konkrétne tom ktorom súbore nachádza.

## Šablóna opisu rozhraní

Opis rozhrania pre používatea komponentu. Dokument je organizovaný poda služieb, ktoré daný komponent ponúka.

### Služba 1

Struný opis služby pre používetea

#### Vstup

...

#### Výstup

...

#### Error

Opis výstupu pri chybe

#### Príklad

...

### Služba 2

...

### Služba n

## Šablóna opisu služieb komponentu

Opis služieb na vysvetlenie používateovi.

### Služba 1

o to vlastne robí?

### Služba 2

...

### Služba n

# Manažment komunikácie

# Manažment kontroly stavu projektu

Pravidelná kontrola stavu projektu nám umožuje priebežne zisova stav realizácie projektu. Cieom kontroly je zisti, i poas realizácie projektu sa dodržuje asový plán projektu. Medzi to patrí napríklad sledovanie, i plneníme dohodnuté termíny a miníky a i nami vytvorený produkt spa všetky normy.

V rámci manažmentu kontroly stavu projektu sme urobili oproti zimnému semestru len zopár zmien. Medzi tieto zmeny patrili astejšie code-review, aspo oproti zimnému semestru. alej to bolo pravidelnejšie testovanie implementovaných modulov frontendu. Toto testovanie prebiehalo tak, že autor, ktorý danú funkcionalitu implementoval, jej podrobný opis správania poskytol ostatným lenom tímu a tí testovali i jednotlivé kroky fungujú správne. Autor sledoval interakciu lenov tímu s týmto modulom a následne získal od nich vstupy a pripomienky, ktoré zapracoval.

V rámci kontroly stavu projektu sme pravidelne, na každom stretnutí, tiež venovali istý as (aspo pol hodinu) zhodnoteniu práce jednotlivých lenov tímu od posledného stretnutia. Týmto spôsobom sme získali lepší prehad o tom, ktoré úlohy budú trva dlhšie, ako sme plánovali a ktoré úlohy sme stihli od posledného stretnutia úspešne zapracova.

# Manažment podpory vývoja

Manažment podpory vývoja zaha prostriedky a nástroje, ktorú sú každodenne využívané všetkými zainteresovanými projektu, od vývojárov, testerov a iných výkonných pracovníkov cez administratívnych pracovníkov a manažment až po samotného zákazníka, ktorý tak môže lepšie spolupracova na tvorbe projektu. V zimnom semestri boli pre rôzne nástroje vytvorené používateské úty s rôznymi prístupovými právami poda relevancie, ím sme zabezpeili oddelenie úloh a právomocí. Na týchto predpokladoch sme stavali v letnom semestri, kedy využívanie nástrojov (hlavne vývojových) vzrástlo a napomohlo k zjednodušeniu práce na výslednom produkte.

Použitý nástroj pre verziovanie softvéru Git bol používaný poda navrhnutej metodiky verziovania. Pravidelné dohliadanie na stav repozitárov bolo zabezpeené na každodennej kontrole pomocou nástroja SourceTree a poas semestra sa nevyskytli fatálne, ani nekonzistentné stavy, ktoré by zapríinili znemožnenie kontinuálnej práce, následnej integrácie výsledného produktu a prezentácie zakomponovaných produktových zmien zákazníkovi. V rámci projektu boli vytvorené 4 repozitáre na jednoznané oddelenie jednotlivých podprojektov, ktorých primárne využívanie je bližšie popísané v nasledovnej tabuke:

| názov repozitáru | popis |
|---|---|
| cRank | repozitár obsahujúci projekt komparátora, ktorý je implementovaný v jazyku C. |
| dbRank | repozitár urený pre skripty PostgreSQL databázy využívanej ako hlavné dátové úložisko. |
| jRank | repozitár urený pre jazyk Java, v ktorom boli implementované komponenty ORM, Frontend, Scheduler, importery ohlasov a citácií pomocou webových služieb a nástroje využívajuce komparátor. |
| xRank | je administratívny repozitár slúžiaci ako úložisko dokumentov, zápisníc a samotných projektových dokumentácií. |

Nástroje na podporu manažmentu (JIRA, Confluence, Stash) boli využívané poda potreby a nebola identifikovaná situácia pri ktorej by nastala konfliktná situácia. Nástroj JIRA bol taktiež používaný poda navrhnutej metodiky, ktorá výrazne uahila používanie a unifikovala proces vytvárania úloh, zmien a identifikovaných chýb.

Nástroj Crucible + FishEye urený na prehliadku kódu nebol využitý poda navrhovanej špecifikácie a prehliadky kódu neboli vôbec realizované, o mohla spôsobi neexistencia metodiky a nejednotný prístup k vykonávaniu prehliadok a asová náronos samotného procese.

# Manažment plánovania

Manažment plánovania umožuje využi o najefektívnejšie as a udské zdroje na dosiahnutie o najlepších výsledkov tímovej práce. Okrem toho je potrebné dba na kvalitnu plánovania v rámci tímu. V zimnom semestri sme tieto fakty nepodcenili aj napriek tomu, že sme boli v tejto sfére noví. Spoiatku sme plánovali úlohy "sedliackym" spôsobom, teda o bolo potrebné urobi, na tom sme sa dohodli, odhladli as práce na každej z úloh a rovnomerne rozdelili medzi lenov tímu poda osobitných schopností každého lena.
V zimnom semestri sme na plánovaní aj trochu popracovali. Zaali sme využíva výhody retrospektívy. alej sme plánované úlohy vždy vkladali do backlogu a potom sme ich usporiadavali poda priority, akú im v prípade user stories zadal náš zákazník (pedagóg) a v prípade technologickej analýzy kolaboratívne celý tím.
Nemožno poveda, že plánované úlohy sme stíhali plni presne poda stanovených termínov, ale aj napriek tomu sa všetko zvládalo na vemi dobrej úrovni.

# Manažment rizík

Manažment rizík nám umožuje identifikova možné riziká, ktoré sa môžu vyskytnú v priebehu vypracovávania projektu. Tieto riziká môzu negatívne ovplyvni vyvoj projektu, pripadne ohrozi jeho celkovú realizaciu. Identifikované riziká nám pomohli pri riadení projektu identifikovaním potrebných opatrení, ktoré je potrebné vykona, aby sa dané riziká nevyskytli, prípadne ako rieši prípadný výskyt niektorého z rizík.

V letnom semestri sme prácu s rizikami zmenili len minimálne, keže identifikácia a riešenie problémov výskytu rizík bola dobre podchytená už v zimnom semestri. Vždy pri plánovaní šprintu sme identifikovali možné riziká vyplývajúce z naplánovaných úloh šprintu a pokúsili sme sa eliminova možné dopady týchto rizík na projekt. V prípade výskytu niektorého z rizík sme okamžite podnikli kroky potrebné k eliminácii jeho dopadu.

V prípade opakovane sa vyskytujúcich sa rizík sme takúto situáciu riešili bu zmenou vykonávatea úlohy (problém so zameraním úlohy), komunikáciou so zákazníkom (problém so špecifikáciou) alebo prostredníctvom technologických porád (problém s implementáciou).

Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 812 19 Bratislava

# Analýza výsledkov výskumu

**Dokumentácia k inžinierskemu dielu**

**Tím 10** ResearchRank
**Vedúci projektu:** Ing. Nadežda Andrejčíková, PhD.
**Členovia tímu:** Bc. Michael Gloger
Bc. Rastislav Kostrab
Bc. Šimon Kompas
Bc. Tomáš Jánošík
Bc. Daniel Kĺč
Bc. Stanislav Kubica
Bc. Michal Walder

# 1 Úvod

Tento dokument slúži ako dokumentácia k tímovému projektu s názvom Analýza výsledkov výskumu. Ide o systém, ktorý slúži na správu publikačnej činnosti v oblasti vedy a výskumu. Systém spracúva, analyzuje a uchováva záznamy o vedeckej publikačnej činnosti. Pomáha vedeckým pracovníkom sledovať svoje publikácie a okrem ohlasov, odhaľuje aj iné zaujímavé vzťahy medzi nimi.

## 1.1 Štruktúra dokumentu

Dokument je štruktúrovaný nasledovne:

Prvá kapitola Analýza problematiky obsahuje analýzu problémovej domény. V druhej kapitole Špecifikácia požiadaviek sme zhrnuli požiadavky na prvotný systém. Tretia kapitola Návrh obsahuje opis technológií, ktoré sme sa rozhodli použiť na riešenie jednotlivých problémov, spojených s našim projektom. Štvrtá kapitola Implementácia obsahuje popis riešenia jednotlivých problémov a v závere dokumentu sú opísané naše šprinty.

Vývoj v tíme je riadený agilnou metódou SCRUM a tomu bol prispôsobený aj vývoj dokumentu. Dokument bude aktualizovaný postupne ako bude prebiehať naša práca.

## 1.2 Popis problému

Vedecké pracoviská fungujú z veľkej časti na základe príjmov z grantov, na ktoré prispieva ministerstvo školstva SR. Ak pracoviská chcú získavať tieto granty, musia sa usilovať o výskumnú činnosť a výsledky z nej publikovať. Výsledky výskumnej činnosti sa okrem vedeckých digitálnych knižníc evidujú v našej krajine do Centrálneho registra publikačnej činnosti (CREPČ). Jedným zo základných kvalitatívnych meradiel výsledkov výskumnej činnosti je počet ohlasov na publikované diela, čo znamená, že časť daného diela bola citovaná v inom novom diele, ktoré bolo publikované v rámci rovnakej výskumnej domény.

Získavanie prehľadu o ohlasoch na autorove vlastné diela je veľmi ťažkou a zdĺhavou manuálnou činnosťou, ktorá by sa mala dať zautomatizovať vytvorením systému, ktorý bude informácie o týchto dielach spracovávať automaticky a bude tak šetriť čas autorom publikovaných diel. Pri manuálnom vyhľadávaní autori diel taktiež robia chyby, ktoré vznikajú napríklad nechcenom prehliadnutí iného diela a podobne. Pre obohatenie databázy týchto ohlasov je potrebné ju naplniť dátami z nových zdrojov a identifikovať nové prepojenia za pomoci porovnávacích algoritmov. Tomuto kroku však predchádza ešte jeden ďalší fakt, že tieto diela môžu byť v CREPČ zadané chybne alebo v rôznych variantoch. Informácie o týchto dielach sa do CREPČ zadávajú manuálne a dôvodom občasnej chyby pri zadávaní je samozrejme ľudský faktor.

Získavanie informácií o výskumnej činnosti v našej krajine je teda veľmi zdĺhavý a zložitý proces.

## 1.3 Cieľ projektu

Cieľom nášho projektu je vytvoriť systém, ktorý bude automatizovane spracovávať informácie o vedeckej publikačnej činnosti. Dáta budú čerpané z rôznych zdrojov, medzi ktoré patrí napríklad: CREPČ, Google Scholar a iné. Výstupom z našej práce bude webový informačný systém rozšírený o webové služby, pomocou ktorých je možné dopytovať sa na služby, ktoré bude náš systém ponúkať

## 1.4 Použité skratky a pojmy

| Pojem / skratka | Popis |
|---|---|

| | |
|---|---|
| CREPČ | Centrálny register evidencie publikačnej činnosti |
| GWT | Google Web Toolkit |
| ORM | Objektovo - relačný mapovač |

# 2 Analýza problematiky

## 2.1 Problematika párovania publikácií

Na párovanie publikácií existuje množstvo článkov a odborných prác. Existujú prístupy, ktoré sa snažia vyvinúť unsupervised metódu bez nutnosti refaktorizácie celej databázy [1], prístupy, ktoré sa snažia párovať publikácie na základe vyššej štatistiky [2], na základe strojového učenia [3], na základe heuristík a rozšírenia existujúcich metód [4] a mnohé iné. Dodnes neexistuje algoritmus, ktorý by s určitosťou vedel spárovať všetky duplicitné publikácie iba podľa údajov uvedených v zdroji informácií.

### 2.1.1 Špecifikácia problému

Problém ako taký pozostáva z viacerých pod-problémov. Akýkoľvek algoritmus párovania musí počítať s neúplnými alebo nesprávnymi údajmi, polymorfiou mien, viac-jazyčnými čiastočnými prekladmi, podobnosťou nadväzujúcich prác. Následkom týchto faktorov sa pri párovaní používajú váhy na jednotlivé dostupné informácie.

Taktiež je mnoho problémov implicitne naviazaných na jazyk, v ktorom sú situované. Napríklad mená slovenských autorov majú iné chyby a skomoleniny, špecifické problémy a synonymá ako mená anglických autorov a podobne.

### 2.1.2 Používané prístupy

Vo všeobecnosti sa páruje podľa prvého autora a potom sa používajú iné dostupné informácie a určitou váhou na verifikáciu výsledku porovnania prvých autorov. Používajú sa co-autori, názov práce, abstrakty, ISBN, ISSN a iné informácie, avšak väčšinou iba niektoré z týchto sú dostupné a v prípade autorov je výsledok tiež podmienený nejakou pravdepodobnosťou zhody. Kvôli tomuto vznikli aj alternatívne metriky, ktoré hodnotia kontext práce, jej zameranie, zhodu zamerania co-autorov a pod. Je možné logicky rozdeliť techniky, ktoré sa používajú na techniky párovania mien, techniky porovnávania údajov a techniky porovnania kontextu.

#### 2.1.2.1 Párovanie mien

Existuje množstvo prístupov avšak študované boli predovšetkým tie najpoužívanejšie. Menovite algoritmu Jaro-Winkler, Levensteinova vzdialenosť a Damerau-Levensteinova vzdialenosť, tieto porovnávajú v podstate aké veľké úsilie je potrebné na to, aby sa prvý člen porovnania zmenil na druhý člen. Ďalej sa preštudovali algoritmy, ktoré porovnávajú priamo podiel rovnakých sekvencií v reťazci. Tu boli preštudované aspekty 2-gramov a 3-gramov.

#### 2.1.2.2 Techniky porovnávania údajov

Porovnávania údajov je vo všeobecnosti dobre preskúmaná oblasť, ktorou sa zaoberá množstvo publikácií. Pre nás boli relevantné predovšetkým techniky, ktoré boli priamo už niekým použité pri porovnávaní publikácií, alebo techniky, ktoré by mohli určité čiastkové problémy porovnávania vyriešiť. Predovšetkým problematický je nedostatok informácií.

Väčšina porovnávaných atribútov je tradične otázka identity alebo dostatočného prieniku dvoch množín údajov. Z tohto hľadiska sú zaujímavé predovšetkým porovnania plných textov a abstraktov, poprípade titulov prác. Tu sa často aplikuje heuristika, ktorá hovorí, že texty, resp. abstrakty alebo tituly musia byť identické na to, aby to bola rovnaká práca. Aj napriek tomu, že sa s tým vo všeobecnosti odborná komunita stotožňuje, problémy do tejto problematiky vnášajú rôzne súčasti, ktoré môžu byť k textu pridané zo zdroja. V tomto prípade i keď ide len malú súčasť, ktorá nemá s obsahom alebo formátom samotného textu nič spoločné, tak rozvráti túto vo všeobecnosti zaužívanú heuristiku. Existuje množstvo metód, ktoré sa na

obídenie tohto problému zameriavajú. Naše štúdium sa zameralo predovšetkým na metódy s de Bruijnovými grafmi a String grafmi. Tieto okrem porovnania samotných textov môžu podľa potreby poskytnúť veľa iných užitočných informácií.

#### 2.1.2.3 Techniky porovnávania kontextu

Na porovnávanie kontextu človek potrebuje množstvo metadát. Na určenie kontextu práce treba zistiť zameranie práce, množinu autorov, s ktorými prvý autor bežne spolupracuje a samotný zdroj práce a jeho zameranie. Napríklad sa používa budovanie odborných slovníkov a porovnávanie s množinami iných autorov na určenie zamerania konkrétnej práce. Na budovanie odborného slovníka sa používajú napríklad spomínané de Bruijnové grafy alebo String grafy (a rôzne ich adaptácie). V nich po určení prieniku a vyčiarknutí najpoužívanejších slov sa vytvoria implicitne prepojené skupiny spojení špecifické pre danú oblasť, skupinu alebo indivíduum. Touto technikou, výberom a porovnaním situovania jednotlivých entít sa zistí kontext práce. Na tento proces však je potrebné množstvo informácií predovšetkým v oblasti kontextu práce, a preto je to prakticky nevyužiteľné pre autora, ktorý je úplne nový pre databázu, pretože o ňom nie je dosť informácií. Taktiež existujú pokusy na určenie kontextu zo samotných záznamov zdroja publikácie, pracoviska autorov a podobne. Vo všeobecnosti, aj keď je tento údaj veľmi užitočný, nie vždy je možné ho získať z dostupných informácií a preto je väčšinou chápaný existujúcimi prístupmi ako menej dôležitý ako predchádzajúce dva komponenty párovania.

#### 2.1.2.4 Techniky výberu kandidátov na porovnanie

V zásade je nemožné porovnávať záznam označený na párovanie s celou databázou záznamov, pretože takéto porovnávanie by viedlo k zahlteniu systému a zbytočnému porovnávaniu takých záznamov, ktoré majú len málo spoločné. Oblasť vyberania kandidátov na porovnávanie je vcelku dobre zmapovaná, existujú dva hlavné prístupy, ktoré potom rôznymi obmenami dosahujú lepšie výsledky v určitých špecifických oblastiach. Jedným prístupom je tzv. blokový prístup (napríklad [6]), ktorý rozdelí celú množinu záznamov na bloky, v rámci ktorých potom záznamy porovnáva. Druhým prístupom je Sorted Neighbourhood Method [5], kde ide o to, že máme záznamy zoradené podľa niektorého poľa(často je to primárny kľúč alebo najdôležitejší záznam, ktorý je čo najviac jedinečný) a porovnávame len v rámci určitého okna záznamov.

### 2.1.3 Supervised vs. unsupervised metódy

Všetky techniky párovania podliehajú ešte jednému atribútu a to je supervised alebo unsupervised. Rozdiel je v tom, že prvé sú pred nasadením kalibrované na známej správnej množine údajov, aby sa určili ich parametre. Je to kvôli mnohým faktorom, ale predovšetkým pre to, že v rôznych jazykoch sa grafy efektivity hodnoty parametrov môžu líšiť. Naproti tomu unsupervised metóda je univerzálne navrhnutá tak, aby fungovala na ľubovoľných údajoch. Vo všeobecnosti však prevládajú supervised metódy a to čiastočne kvôli tomu, že konštrukcia unsupervised metódy môže byť oveľa ťažšia a čiastočne preto, že unsupervised metódy musia počítať s oveľa viacej faktormi, čo má vplyv na ich výkon.

## 2.2 Zdroje metadát publikácií

Základnú množinu metadát o vedeckých prácach získavame z portálu Centrálneho registra evidencie publikačnej činnosti (ďalej CREPČ). Na tomto portáli je možné vyhľadávať vedecké publikácie na základe názvov, mien autorov, pracovísk, hesiel, roku vydania publikácie, roku citovania, ISBN, ISSN a vydavateľstiev. CREPČ tiež poskytuje ročný export metadát publikácií daných vedeckých pracovísk vo formáte XML. Na vytvorenie úvodnej množiny údajov používame práve tieto exporty, z ktorých získavame informácie o konkrétnom diele, jeho autoroch a iné užitočné metadáta (rok vydania, vydavateľstvo, pracovisko a pod.).

Vzhľadom na neúplnosť a občasnú nepresnosť záznamov v xml exporte CREPČ, bude neskôr potrebné tieto dáta obohatiť použitím webových služieb citačných indexov ako sú WOS a SCOPUS.

Nasledujúca tabuľka obsahuje jednotlivé elementy CREPČ xml exportu, dátové typy, ktoré obsahuje, početnosti a samotné namapovanie na relačnú databázu.

| Názov atribútu | Dátový typ | Formát | Početnosť | Názov stĺpca v DB | Tabuľka v DB |
|---|---|---|---|---|---|
| Kolekcia_hlavicka | | | 1 | | |
| sigla | String | Enum | 1 | sigla | library |
| kolekcia_data | | | 1 | | |
| kolekcia_data/data_zaznam/ | | | 0..* | | |
| data_zaznam /zaznam_identifikacia | | | 1 | | |
| identifikacia_orgID(attr: id_typ) | String | | 1..* | original_id | publication |
| identifikacia_datumvytvorenia | String | datum _typ | 1 | date_make | publication |
| identifikacia_datumzmeny | String | datum _typ | 1 | date_edit | publication |
| identifikacia_druhdokumentu | String | | 0..1 | code_of_typ e | document_ty pe |
| identifikacia_ISBN | String | | 0..* | code | identification _code |
| identifikacia_ISSN | String | | 0..* | code | identification _code |
| identifikacia_MDT | String | | 0..* | code | identification _code |
| data_zaznam /zaznam_nazov(attr: format) | | | 1 | | |
| nazov_hlavny | String | | 1..* | name | name |

| | | | | | |
|---|---|---|---|---|---|
| nazov_subezny(attr: jazyk) | String | | 0..* | name | name |
| nazov_podnazov | String | | 0..* | name | name |
| nazov_preklad(attr: jazyk, doplnky, cislo_casti, nazov_casti) | String | | 0..* | name | name_translation |
| nazov_casti | String | | 0..* | name | name_of_part |
| nazov_cislocasti | String | | 0..* | number | name_number_of_part |
| nazov_autorskezahlavie(attr: typ) | String | | 0..* | heading | name_author_heading |
| | | | | | |
| data_zaznam /zaznam_autori (typ=primarna, sekundarna) | | | 0..1 | | |
| autori_osoba(attr: typ) | | | 0..* | | |
| autori_osoba/osoba_priezvisko | String | | 1 | surname | author_person |
| autori_osoba/osoba_meno | String | | 1 | name | author_person |
| autori_osoba/osoba_pracovisko(attr:pracovisko_typ) | String | Enum | 1..* | code | person_workplace |
| autori_osoba/osoba_rola alebo ezp_rola (attr: format = UNIMARC,MARC21) | String | | 1..* | role | person_role |
| autori_osoba/osoba_podiel | Integer | | 1 | contribution | author_person |
| autori_osoba/osoba_orgID(attr:id_typ) | String | | 0..* | original_id | author_person |
| autori_osoba/osoba_titul(attr: titul_typ) | String | | 0..* | title | title |

| | | | | | |
|---|---|---|---|---|---|
| autori_korporacia | | | 0..* | | |
| autori_korporacia/korporacia_nazov | String | | 1 | name | author_corporation |
| autori_korporacia/korporacia_podcastnazvu | String | | 0..* | subname | corporation_subname |
| autori_korporacia/korporacia_privlastok | String | | 0..* | attribute | corporation_attribute |
| autori_korporacia/korporacia_rola | String | | 1..* | role | corporation_role |
| autori_korporacia/korporacia_orgID | String | | 0..1 | original_id | corporation |
| autori_korporacia/korporacia_zhromazdenie_cislo | String | | 0..1 | number | convention |
| autori_korporacia/korporacia_zhromazdenie_miesto | String | | 0..1 | place | convention |
| autori_korporacia/korporacia_zhromazdenie_datum | String | | 0..1 | date | convention |
| | | | | | |
| data_zaznam /zaznam_pracoviska | | | 1 | | |
| pracoviska_pracovisko | | | 1..* | | |
| pracoviska_pracovisko/pracovisko_fakulta | String | | 1 | faculty | publication_workplace |
| pracoviska_pracovisko/pracovisko_katedra | String | | 1 | desk | publication_workplace |
| | | | | | |
| data_zaznam/zaznam_jazyk (attr: format) | | | 1 | | |

| | | | | | |
|---|---|---|---|---|---|
| jazyk_textu | String | | 1..* | language_text | publication |
| jazyk_originalu | String | | 0..* | language_original | publication |
| jazyk_resume | String | | 0..* | language_resume | publication |
| | | | | | |
| data_zaznam /zaznam_ohlasy | | | 0..1 | | |
| ohlasy_ohlas | | | 1..* | | |
| ohlasy_ohlas/ohlas_kategoria | String | | 1 | category | response |
| ohlasy_ohlas/ohlas_rok | String | | 1 | year | response |
| ohlasy_ohlas/ohlas_zapis | String | | 0..1 | entry | response |
| ohlasy_ohlas/ohlas_index | String | | 0..1 | index | response |
| | | | | | |
| data_zaznam /zaznam_vydanie | | | 0..1 | | |
| vydanie_krajina(attr: format) | String | | 0..* | name | publication_country |
| vydanie_miesto | String | | 0..* | name | publication_place |
| vydanie_vydavatel | String | | 0..* | name | publisher |
| vydanie_rok | String | | 0..* | year | publication_year |
| vydanie_cislo | String | | 0..* | number | publication_number |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| data_zaznam /zaznam_ine | | | 0..1 | | |
| ine_rozsah | String | | 0..1 | range | publication |
| ine_edicia_nazov | String | | 0..1 | name | edition |
| ine_edicia_zvazok | String | | 0..1 | edition_volume | publication |
| ine_url(attr: url_typ) | String | | 0..* | url | url |
| | | | | | |
| data_zaznam /zaznam_hesla | | | 0..1 | | |
| heslo_osoba | | | 0..* | | |
| heslo_osoba/osoba_priezvisko | String | | 1 | surname | keyword_person |
| heslo_osoba/osoba_meno | String | | 1 | name | keyword_person |
| heslo_osoba/osoba_orgID | String | | 0..1 | original_id | keyword_person |
| heslo_osoba/heslo_casti | heslo_casti | | 0..1 | keyword_of_part_id | keyword_person |
| heslo_korporacia | | | 0..* | | |
| heslo_korporacia/korporacia_nazov | String | | 1 | name | keyword_corporation |
| heslo_korporacia/korporacia_podcastnazvu | String | | 0..* | subname | keyword_corporation |
| heslo_korporacia/korporacia_privlastok | String | | 0..* | attribute | keyword_corporation |
| heslo_korporacia/korporacia_orgID | String | | 0..1 | original_id | keyword_corporation |

| | | | | | |
|---|---|---|---|---|---|
| heslo_korporacia/heslo_casti | heslo_casti | | 0..1 | keyword_of_part_id | keyword_corporation |
| heslo_predmet(attr: typ) | | | 0..* | | |
| heslo_predmet/predmet_nazov | String | | 1..* | name | keyword_subject |
| heslo_predmet/predmet_orgID | String | | 0..1 | original_id | keyword_subject |
| heslo_predmet/heslo_casti | heslo_casti | | 0..1 | keyword_of_part_id | keyword_subject |
| | | | | | |
| heslo_casti | | | 0..1 | | |
| cast_tematicka | String | | 0..1 | part_theme | keyword_of_part |
| cast_geograficka | String | | 0..1 | part_geographic | keyword_of_part |
| cast_chronologicka | String | | 0..1 | part_chronological | keyword_of_part |
| | | | | | |
| data_zaznam /zaznam_vazby | | | 0..1 | | |
| vazby_zaznam(attr: typ) | | | 1..* | | |
| zaznam_identifikacia_viazaneho | | | 0..1 | | |
| zaznam_identifikacia_viazaneho/ identifikacia_orgID | String | | 0..1 | original_id | publication |
| zaznam_identifikacia_viazaneho/ identifikacia_ISBN | String | | 0..* | code | identification_code |
| zaznam_identifikacia_viazaneho/ identifikacia_ISSN | String | | 0..* | code | identification_code |

| | | | | | |
|---|---|---|---|---|---|
| zaznam_nazov | zaznam_nazov | | 1 | POPIS VYŠŠIE | |
| zaznam_oznacenie_zvazku | String | | 0..1 | POPIS VYŠŠIE | |
| zaznam_vydanie | zaznam_vydanie | | 0..1 | POPIS VYŠŠIE | |
| zaznam_autori | zaznam_autori | | 0..1 | POPIS VYŠŠIE | |
| ohlasy_ohlas | ohlasy_ohlas | | 0..1 | POPIS VYŠŠIE | |

# 3 Špecifikácia požiadaviek

## 3.1 Nefunkcionálne požiadavky

### 3.1.1 Párovanie publikácií

Algoritmy porovnávania a párovania musia mať dostatočne malú časovú zložitosť, aby mohli byť nasadené v "real time" prostredí a aby mohli analýzy a párovania prebiehať online. V rámci požiadaviek na produkt zákazník požaduje, aby bol systém schopný dohľadať informácie k špecifickému autorovi. Táto úloha však potrebuje prebehnúť v konečnom čase tak, aby sme nového používateľa neodradili dlhotrvajúcimi výpočtami a neskorými dobami odpovede. Ďalej je potrebné výpočtový výkon rozdeliť medzi získavanie nových údajov a spracovávanie týchto nových údajov, preto musia algoritmy párovania bežať čo najrýchlejšie.

### 3.1.2 Import dát do databázy z exportu XML CREPČ

Algoritmus pre import dát z exportu z XML z Centrálneho registra evidencie publikačnej činnosti (CREPČ) musí byť efektívny. Nie je určené žiadne obmedzenie pre rýchlosť algoritmu. Za základe faktu, o aké veľké dáta pri spracovaní ide, tento algoritmus musí pracovať tak, aby mal čo najnižšie nároky na pamäť.

## 3.2 Funkcionálne požiadavky

### 3.2.1 Párovanie publikácií

Je dôležité pracovníkom poskytnúť čo možno najspoľahlivejšie informácie, a preto je dôležité, aby algoritmy párovania publikácií boli čo možno najúčinnejšie. Okrem toho z hľadiska budúceho rozšírenia programu musia poskytovať možnosť rozšírenia o zisťovanie ďalších relevantných informácií. Ďalej musia podporovať manuálne spárovanie publikácii, ak overený a zaregistrovaný výskumník, teda používateľ našej aplikácie označí dve publikácie ako totožné.

### 3.2.2 Import dát do databázy z exportu XML CREPČ

Pre fungovanie párovania publikácii je potrebné pracovať s nejakými dátami. Tieto dáta sa budú získavať z Centrálneho registra evidencie publikačnej činnosti (CREPČ) v istých dlhodobých intervaloch vo forme XML exportu. Dané XML sa bude spracovávať na základe zadanej požiadavky do systému cez webové rozhranie. XML sa bude parsovat a importovat do existujúcej databázy. Pre účel perzistencie údajov sparsovaných údajov z XML CREPČ sa vytvorí PostgreSQL databáza (relačného charakteru), ktorá bude tieto údaje uchovávať a poskytovať pre funkciu Párovania publikácií. Pri zadaní požiadavky do systému sa funkcia nespustí ihneď. Bude sa spúšťať v nočných hodinách, kedy je na server smerovaných čo najmenej požiadaviek.

### 3.2.3 Použitá databázová technológia

Pre perzistenciu spracovávaných údajov bude potrebné použitie relačnej databázy, ktorá by bola dostatočne výkonná, pretože sa jedná o spracovanie veľkého množstva údajov. Jednou z požiadaviek je možnosť pripojenia na danú databázu pomocou Java JDBC. Použitie dokumentovej databázy sa nejavilo vhodné z dôvodu potreby relačných vzťahov medzi záznamami. Požiadavkou, ktorú musí spĺňať uvedená databáza, tiež bola možnosť vytvárania databázových funkcií (v jazyku SQL, PL/SQL alebo C), ktoré by mohli zjednodušiť niektoré opakované databázové operácie.

### 3.2.4 Webové rozhranie

Webové rozhranie systému musí ponúkať jednoduchý spôsob zobrazovania nahromadených dát v databáze. Medzi hlavné entity patria autori, publikácie a vzťahy medzi nimi. Používateľské rozhranie musí ponúkať prehľadný spôsob filtrovania veľkého počtu dát na základe všetkých atribútov, ktorými dané entity disponujú.

### 3.2.5 Webové služby

Systém bude okrem webového rozhrania ponúkať tiež webové služby, cez ktoré bude možné využívať implementované algoritmy porovnávania a dopytovanie sa nad dátami v našej databáze.

### 3.2.6 Zdroje znalostí

Systém bude využívať viaceré zdroje znalostí a kombinovať údaje v nich nájdené, aby sa tak zabezpečila väčšia presnosť párovania publikácií a viac údajov o ohlasoch. Týmito zdrojmi budú CREPČ - Centrálny register evidencie publikačnej činnosti, Scopus, Web of Science a Google Scholar.

Prvý menovaný zdroj zabezpečuje údaje o publikáciách na Slovensku. Nie všetky vedecké inštitúcie využívajú tento systém. Zvyšné tri sú citačné indexy, v ktorých budeme vyhľadávať ohlasy na diela získané z CREPČ a na získavanie ďalších diel.

Preto musí systém vedieť komunikovať s týmito službami, musí vedieť spracovávať výstupy získané z týchto služieb a mapovať tieto výstupy na seba. Pod mapovaním rozumieme jednak spracovanie údajov do spoločného formátu, a potom aj párovanie publikácií.

# 4 Návrh

## 4.1 Technológie

### 4.1.1 C++

Na návrh a implementáciu párovacích algoritmov bol vybraný programovací jazyk C++ a to najmä pre svoju rýchlosť, ktorá je v tomto procese veľmi dôležitá a vzhľadom na to, že sme používame adaptáciu návrhového vzoru repository, tak nemusí byť tento komponent v Jave. Vzhľadom na to, že aj MongoDB je v naprogramované v C++, používanie C++ na párovanie je taktiež výhodné.

### 4.1.2 PostgreSQL

Na uloženie metadát získaných z exportu CREPČ a neskôr aj metadát získaných z citačných indexov sme použili databázu PostgreSQL. Riešenie minuloročného tímu s použitím MongoDB sa javilo ako nevhodné z dôvodu potreby ukladania vzťahov medzi jednotlivými publikáciami a autormi, a medzi publikáciami navzájom (v prípade že sa niektoré dielo odkazuje na iné formou citácie).

### 4.1.3 Java

Modul importovania CREPČ do PostgreSQL, modul crawlovania Google Scholar a webový frontend aplikácie bol vytvorený v programovacom jazyku Java a to hlavne kvôli objektovo orientovanému prístupu a podpory mnohých knižníc, ktoré zjednodušujú prácu s databázou (Hibernate), xml súbormi (dom4j), a umožňujú vytvárať webové aplikácie v prostredí java (GWT). Výhodou takéhoto riešenia je vytvorenie systému, ktorý bude pripravený na ďalšie rozširovanie a reálnu prevádzku.

### 4.1.4 Hibernate

Na zabezpečenie objektového prístupu k PostgreSQL databáze bola použitá technológia Hibernate. Tá predstavuje objektovo - relačný mapovač (ORM), ktorý zjednodušuje a sprehľadňuje prácu s databázou, keďže odbremeňuje programátora od písania SQL dotazov.

### 4.1.5 JSP, Servlet

Frontend aplikácie je naprogramovaný v jazyku Java a využíva Tomcat server. Vďaka tejto skutočnosti vieme jednoduchým spôsobom pristupovať k našim dátam pomocou Hibernate ORM rozhrania a nemusíme implementovať túto komunikáciu nanovo (ako by bolo nutné napríklad pri využití technológie PHP).

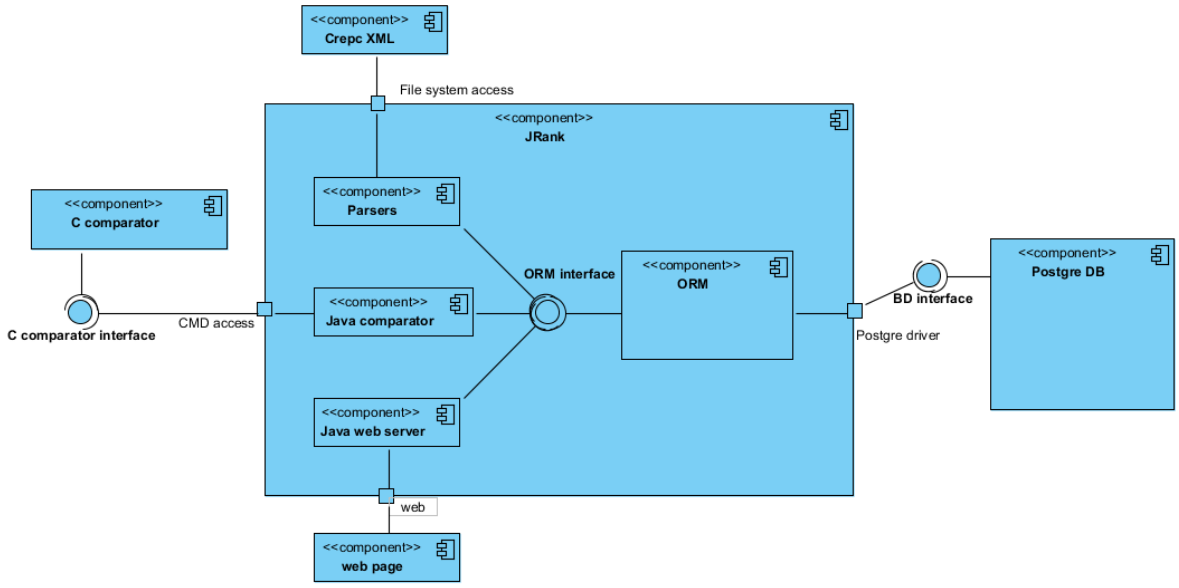
Využívame Servlety ako jednoduchý a priamočiari spôsob komunikácie typu klient-server v prostredí Java.

# 5 [1]Implementácia

Koncepcia systému vznikla predovšetkým s prihliadnutím na jasnú, jednoduchú organizáciu, slabo previazané komponenty a znovupoužiteľnosť. Taktiež jedna z najdôležitejších požiadaviek bolo zabezpečiť rozšíriteľnosť systému. Na zabezpečenie týchto požiadaviek vznikol systém opísaný na diagrame nižšie.

Research rank systém dokáže flexibilne, asynchrónne spracovávať údaje z daných zdrojov vedomostí. Nad údajmi sa ďalej asynchrónne spúšťa proces deduplikácie a proces čistenia údajov, kde sa snaží systém automaticky párovať spracované entity. Tieto procesy skvalitnia, zjednotia a sprehľadnia údaje v databáze, čím sa skvalitnia poskytované informácie. Údaje potom zobrazuje pomocou webového rozhrania.

Zatiaľ je zapojený v tomto prototype len XML Crepč. Prebieha príprava zdrojov Web of Science (WoS), Scopus a Google Scholar.




Obrázok 1: Diagram komponentov systému Research Rank.

Opis jednotlivých komponentov ďalej v podkapitolách.

## 5.1 JRank

Obsahuje komponenty systému napísane v jazyku Java.

### 5.1.1 ORM

#### 5.1.1.1 Plnenie databázy ResearchRank dátami z databázy CREPČ

Pre potreby párovania vedeckých publikovaných diel vznikla potreba vytvorenia relačnej databázy zahrnutej v rámci systému ResearchRank, ktorá by tvorila základ dát potrených pre párovanie iných diel. Údaje, ktorými

[1] Bolo zmenené formátovanie textu oproti prvému odovzdaniu v zimnom semestri, keďže sa pôvodné formátovanie vzhľadom k rozšírenému obsahu stalo neprehľadným.

je túto databázu potrebné naplniť, sú údaje z Centrálneho registra evidencie publikačnej činnosti (ďalej CREPČ). Údaje z tejto databázy sú poskytované vo forme exportu vo formáte XML. Všetky potrebné údaje o štruktúre XML súboru sú dostupné v dokumente o výsledkoch analýzy XML CREPČ.

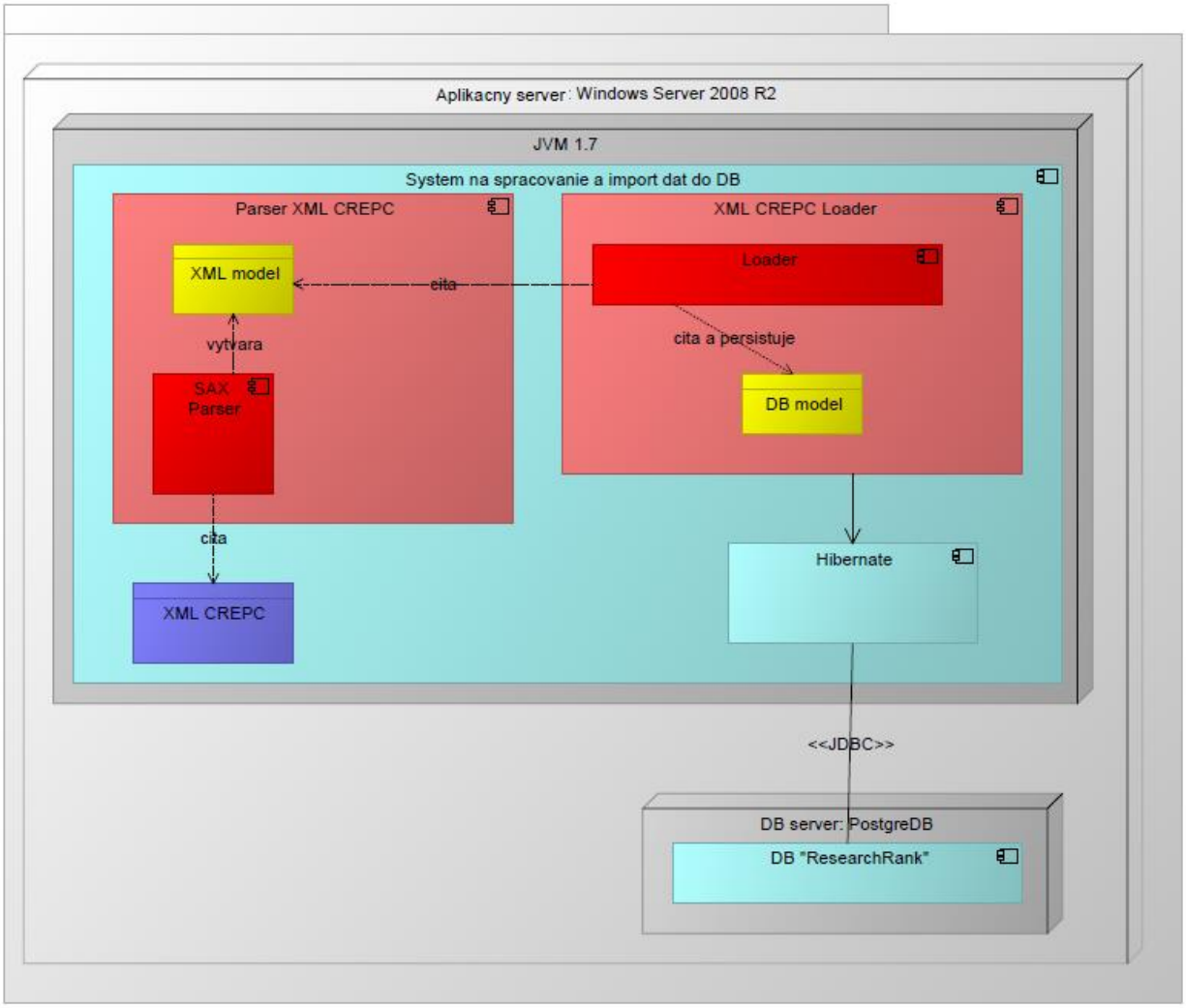
##### *5.1.1.1.1 Systém pre plnenie databázy ResearchRank relevantnými dátami*

Pre vytvorenie takejto databázy je nevyhnutné vytvoriť systém, ktorý exportovaný XML súbor prvé spracuje a potom naimportuje do relačnej databázy. Celý systém je implementovaný v programovacom jazyku Java. Systém je implementovaný v rámci prostredia JRE (Java Runtime Environment) verzie 1.7 bežiaceho v operačnom systéme Windows 7 Professional (64-bit).

Systém som rozdelil na 2 časti:

- Parser XML CREPČ
- Importer do PostgreSQL databázy

Prvá časť bude spracovávať (ďalej parsovať) XML CREPČ. Spracovávať ho bude do štruktúry objektov implementovaných podľa štruktúry XML CREPČ. Celá zaplnená štruktúra objektov XML CREPČ je postupne prepisovaná do štruktúry modelu databázy ResearchRank a počas prepisovania sa postupne ukladá do databázy. K dispozícii je dokument javadoc celého systému a dokumentácia schém DB a Hibernate.




Obrázok 2: Systém pre plnenie databázy ResearchRank relevantnými dátami

#### 5.1.1.1.1.1 Parser XML CREPC

Komponent Parser XML CREP funguje na základe metódy SAX parsovania. Čítajú sa postupne riadok po riadku a pri prečítaní xml tagu sa spustí funkcia závislá od toho, či ide o začiatočný tag alebo o koncový tag. Pri takomto postupnom čítaní tagov prebieha načítavanie textových reťazcov do XML modelu objektov.

Po skončení parsovania sa model objektov pošle do komponentu XML CREPC Loader.

##### 5.1.1.1.1.2 XML CREPC Loader

Tento komponent využíva technológiu Hibernate. Základom používania tejto technológie bolo vygenerovanie DB modelu, do ktorého sa postupne vkladali dáta z XML modelu a perzistovali priamo popri ukladaní. Implementácia systému sa niesla vo verziovaní.

Verzie systému sú:

- 1 - implementácia základnej fungujúcej verzie,
- 2 - optimalizácia - pri ktorej sa dopytom do DB kontrolovala existencia danej entity v DB namiesto vopred načítaných entít z DB, ktoré boli pamäťovo náročné.
- 3 - optimalizácia - zrušenie transakcií pri každej operácii perzistovania, ktoré boli časovo náročné,
- 4 - implementácia bezpečnostného prvku - vytvorenie jednej transakcie v rámci celého importu, čím sa zamedzilo uloženiu nekompletného importu pri nečakanom prerušení procesu,
- 5 - implementácia bezpečnostného prvku - zrušenie jednej transakcie v rámci celého importu a vytvorenie transakcie pre perzistovanie každého jedného záznamu a implementácia kontrolného zapisovania indexu naposledy uloženého záznamu, ktorý sa použije pre pokračovanie procesu importu po nečakanom zlyhaní.

### 5.1.2 Frontend

#### 5.1.2.1 Funkcia:

Sprístupňuje dáta z databázy pomocou používateľského rozhrania. Ponúka možnosti filtrovania dát a ich správu po prihlásení používateľa.

#### 5.1.2.2 Vstup:

Dáta z Postgre databázy sprístupňované pomocou modulu ORM.

#### 5.1.2.3 Výstup:

Zobrazovanie dát, ich filtrovanie a vizualizácia vzťahov.

#### 5.1.2.4 Popis:

ponúka jednoduché používateľské prostredie pre získavanie požadovaných informácií ohľadom publikácií, ich autorov a vzťahoch medzi nimi.

### 5.1.3 Java Comparator

#### 5.1.3.1 Funkcia:

Zabezpečuje rozhranie medzi samotnými algoritmami párovania a databázou, zahŕňa algoritmus na výber kandidátov. Taktiež zabezpečuje spracovanie výstupov z C komparátora.

#### 5.1.3.2 Vstup:

Nové záznamy v databáze (prípadne iné), ktoré je potrebné spárovať s existujúcimi záznamami v databáze. Konkrétne sú to tie záznamy, ktoré majú nastavenú hodnotu checked na false.

#### 5.1.3.3 Výstup:

Spárované záznamy v databáze

#### 5.1.3.4 Popis:

Java komparátor je komponent, ktorý je spúšťaný v pravidelných intervaloch, kedy sú pripravené nové údaje. Taktiež ďalší scenár počíta s pravidelným spúšťaním tak, aby sme skontrolovali záznamy v databáze a podľa novo získaných údajov sa ich opätovne pokúsili spárovať s ďalšími záznamami.

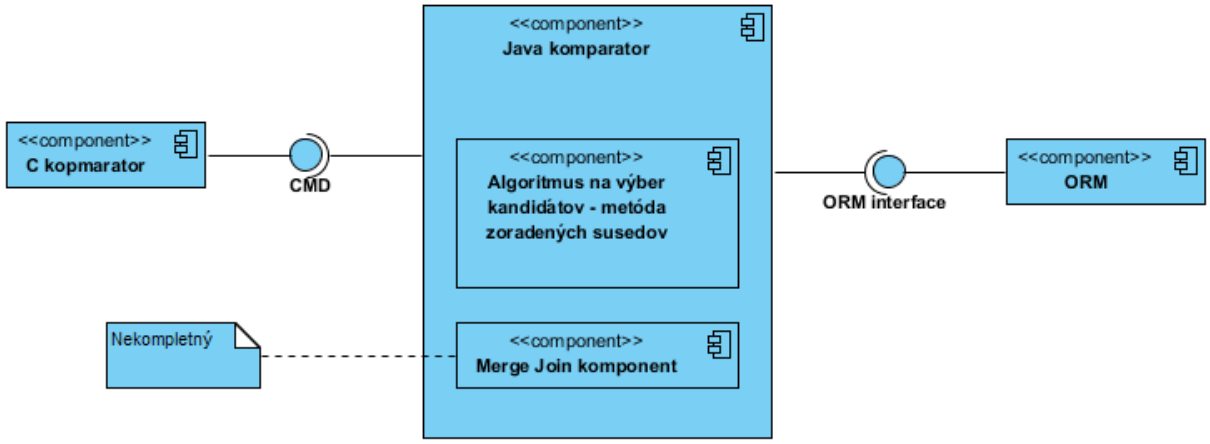
Java komparátor využíva ORM na získanie záznamov, s ktorými potom ďalej pracuje. Iteruje cez záznamy, ktoré sú označené na párovanie a ku každému získa zoznam objektov, ktoré majú najväčšiu pravdepodobnosť zhody alebo podobnosti s príslušným záznamom. Na to sa používajú algoritmy výberu kandidátov.

V tomto komponente je implementovaný algoritmus používajúci metódu zoradených susedov. Záznamy zoraďuje podľa ich názvov a na základe stanovenej hranice zoberie určitý počet záznamov nachádzajúci sa pod a nad hľadaným záznamom. Na zoradenie používa databázovú funkciu row_count.

Aby sme dokázali využiť funkcionalitu, ktorú ponúka Hibernate, vytvorili sme view, ktorý má jedinú funkciu priradiť k záznamu jeho poradie v rámci zoradeného zoznamu záznamov.

Do budúcnosti je plánované rozšíriť zoraďovanie podľa iných atribútov, napríklad podľa mien autorov alebo zdrojovej publikácie, aby sme tak zabezpečili väčšiu šancu takých záznamov, kde sa vyskytuje chyba na prvých písmenách názvu publikácie.

Podľa špecifikovaného formátu vyskladá Java komparátor argumenty C komparátora a odošle ich na samotné párovanie. Potom spracuje výsledky párovania, ktoré prijme od C komparátora. Tieto výsledky sú vo forme poradového čísla záznamu, v zozname argumentov ktorý má byť spárovaný s hľadaným záznamom. Ďalej tieto výsledky pošle Java komparátor na vykonanie merge a join operácií. Táto časť funkcionality však ešte nie je implementovaná.




Obrázok 3: Java komparátor - komponent

## 5.2 CRank

Obsahuje popis komponentov napísaných v C a C++.

### 5.2.1 C Comparator

#### 5.2.1.1 Funkcia:

Komponent zabezpečuje službu, ktorá určí, či daná publikácia ma v príslušnej množine publikácií duplikát.

#### 5.2.1.2 Vstup:

Vstupom komponentu sú argumenty z príkazového riadku. Prvý argument je počet publikácií, ktoré predávame komponentu. Druhý argument je publikácia, ktorej duplikáty hľadáme. V ostatných argumentoch predávame komparátoru množinu publikácií, kde má hľadať duplikáty. Formát publikácie je definovaný nasledujúcou tabuľkou:

| | First author | Count titles | Titles [] | count keywords | keywords[] | countco-authors | co-authors[] | Abstract_present | abstract | Year present | year | Source presents | source_name | ISBN count | ISBN[] | ISSN count | ISSN[] | workplace count | workplaces[] | publisher count | publishers[] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input string 1 | "FA T_C T1 .. Tn K_C K1 .. Kn CA_C CA1 .. CAn AP Abst. YP Year SP SN ISBN_C ISBN1 .. ISBNn ISSN_C ISSN1 .. ISSNn WP_C WP1 .. WPn P_C P1 .. Pn" | | | | | | | | | | | | | | | | | | | | |
| : | .. | | | | | | | | | | | | | | | | | | | | |
| Input string n | .. | | | | | | | | | | | | | | | | | | | | |

Vzhľadom na množstvo možných informácií je vstupný odtlačok publikácie značne zložitý, čo ilustruje tabuľka.

#### 5.2.1.3 Výstup:

Výstup komparátora je vektor čísiel. Prvé číslo určuje počet nájdených duplikátov a ostatné čísla určujú indexy duplikátov v relatívne ku vstupnej množine publikácií.

Príklad: Vo vstupnej množine je 3. publikácia duplikátom.

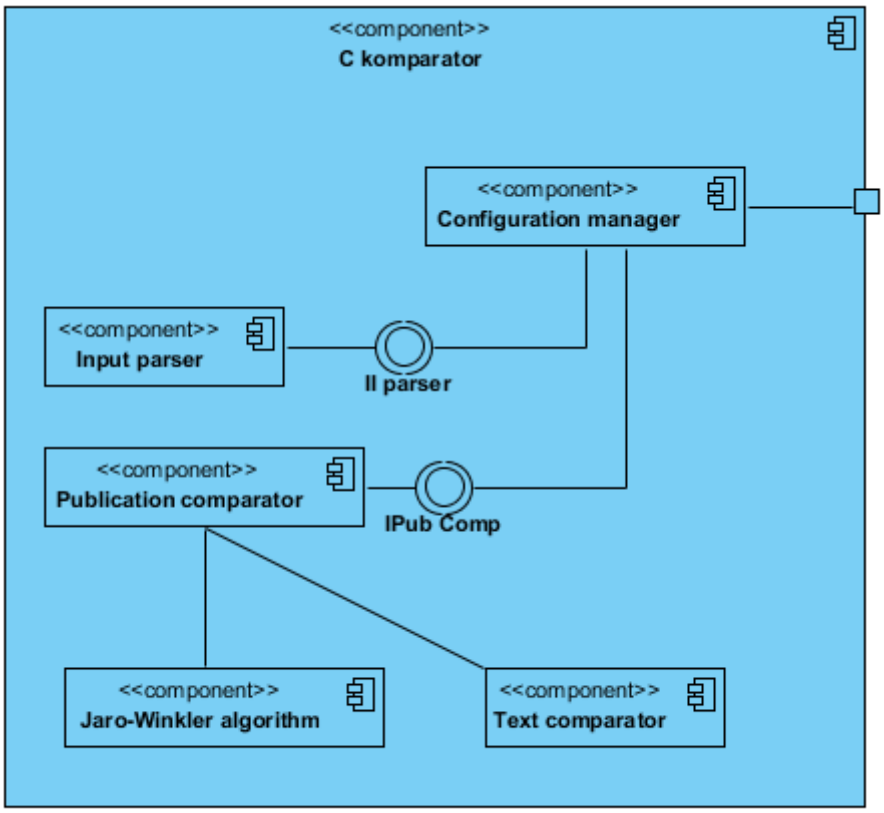
Výstup potom bude: “1 3”.

#### 5.2.1.4 Popis:

C komparátor má čo možno najspoľahlivejšie určiť podobnosť publikácií. Na to má v sebe implementovaných niekoľko komponentov. S vonkajšieho sveta ho volajú aplikácie na komparáciu pomocou príkazového riadku. V rámci architektúry programu by mal interagovať iba s Java komparátorom, ktorý zabezpečuje výber kandidátov a spúšťanie C komparátora.

Komparátor bol implementovaný vo Visual Studiu 2012 v jazyku C++. Neobsahuje žiadne “third party” komponenty.

Konkrétna implementácia C komparátora bola určená na základe porovnania niekoľkých metód párovania, z nich bol vybraný najlepší algoritmus na základe testov nad menami autorov zo slovenského prostredia. Architektúra bola koncipovaná predovšetkým s prihliadnutím na možnosť neskoršej zmeny komponentov ako aj rozšírenia funkcionality.




Obrázok 4: Vnútorná štruktúra C komparátora.

Komparátor obsahuje 5 základných komponentov. Manažér konfigurácie verifikuje správnosť vstupu, manažuje volanie parsera na vstup, volanie samotného komparátora a nakoniec výstup. Parser vstupu dostane verifikovaný vstup od manažéra konfigurácie a transformuje odtlačky vstupných publikácií do reprezentácie v pamäti kompatibilnej s komparátorom publikácii. Komparátor publikácií dostane od manažéra konfigurácie záznamy publikácií, ktoré má porovnať. Následne aplikuje na jednotlivé polia publikácie porovnanie. Podľa typu ide o porovnania pomocou Jaro-Winklerovho algoritmu na porovnávanie, porovnávanie textov (ako sú tituly, abstrakty a pod.) a nakoniec porovnania vyžadujúce presnú zhodu atribútov. Komponent Jaro-Winkler algoritmus obsahuje implementáciu rovnomenného algoritmu. Textový komparátor hľadá zhodu sekvencií tokenov v rámci stokenizovaného textu.

Ak sa komparátoru správne nastavia koeficienty, podľa ktorých určí podobnosť za zhodu, tak výkon komparátora z hľadiska odhaľovania duplicity je uspokojivý.

## 5.3 DBRank

Obsahuje popis databázy, ktorá je v systéme použitá.

### 5.3.1 Postgre DB

#### 5.3.1.1 Funkcia:

Uchovávanie dát získaných z CREPČ, a neskôr aj iných zdrojov. Slúži tiež na ukladanie informácií o používateľoch frontedu.

#### 5.3.1.2 Vstup:

Vstupné dáta sú do databázy vkladané prostredníctvom CREPČ parsera. Ten využíva ORM na prístup a úpravu databázy. Neskôr bude možné rozšíriť uvedený systém o ďalšie zdroje, akými sú WoS, Scopus a Google Scholar.

#### 5.3.1.3 Výstup:

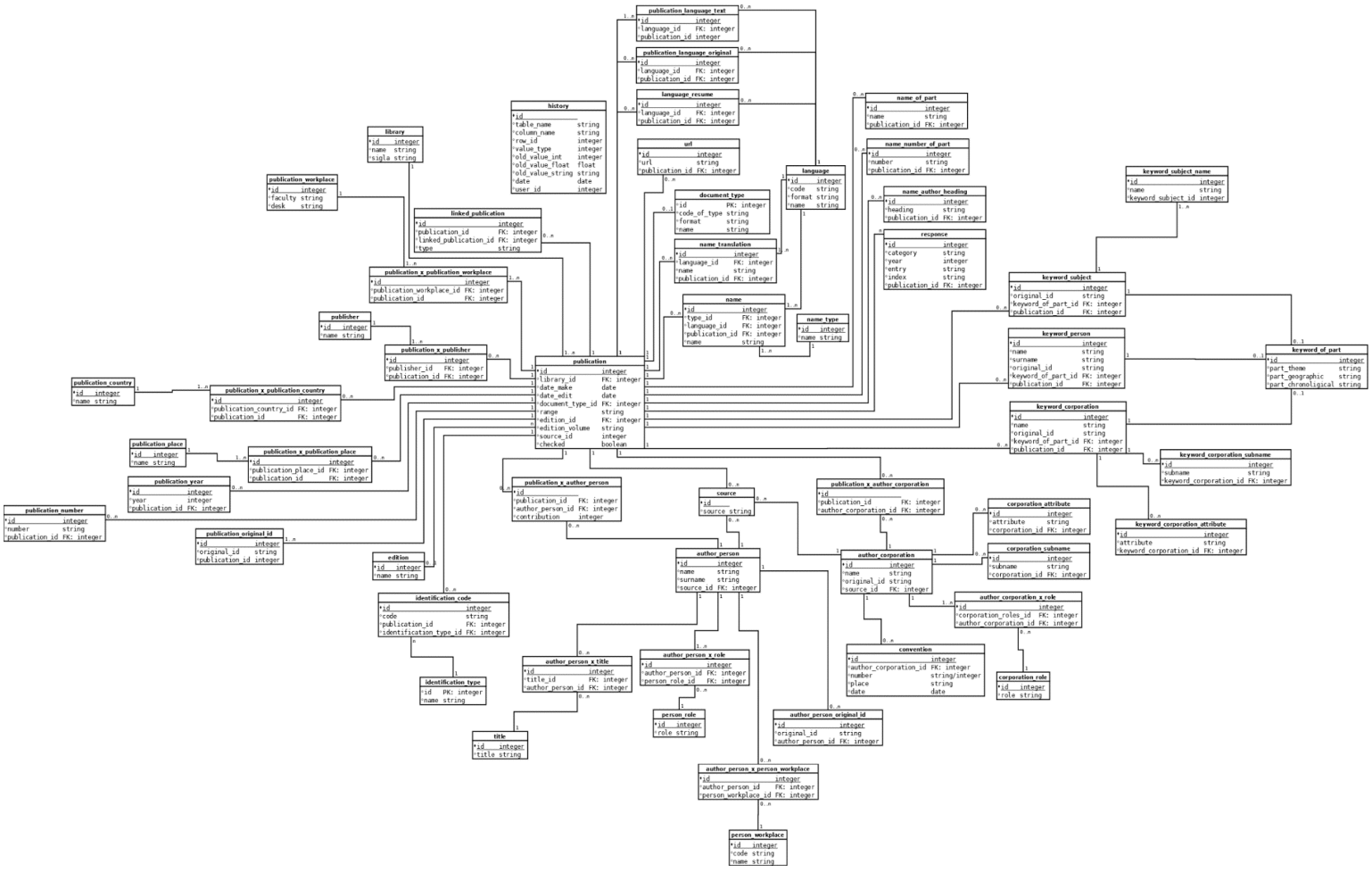
Výstupom sú uchované dáta získané v predošlých krokoch.

#### 5.3.1.4 Popis:

Technológiu Postgre sme zvolili kvôli rýchlosti, ktorou disponuje a faktu, že sa jedná o open source riešenie a pre jej použitie nie je teda potrebná platená licencia. Jednotlivé komponenty systému pracujú s databázou prostredníctvom objektovo-relačného mapovača Hibernate.

Pri návrhu dátového modelu sme primárne vychádzali z dátového modelu CREPČ-u, ktorý bol pre naše požiadavky najvhodnejší. Samotný dátový model určený pre ukladanie údajov publikácií je možné vidieť na nasledujúcom obrázku.

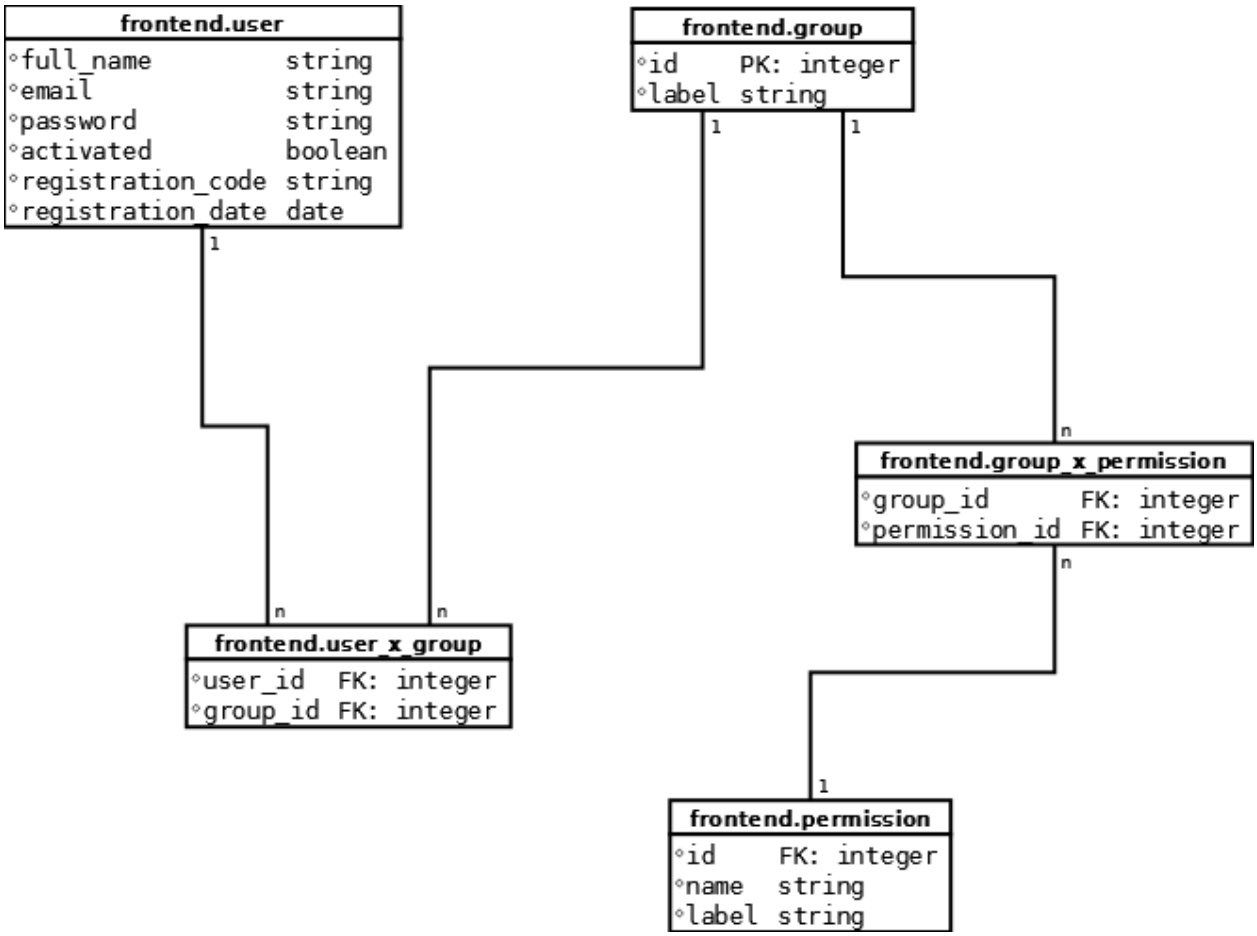
#### 5.3.1.5 Schéma databázy


publication_language_text
*id integer
*language_id FK: integer
*publication_id integer
publication_language_original
*id integer
*language_id FK: integer
*publication_id FK: integer
language_resume
*id integer
*language_id FK: integer
*publication_id FK: integer
history
*id integer
*table_name string
*column_name string
*row_id integer
*value_type integer
*old_value_int integer
*old_value_float float
*old_value_string string
*date date
*user_id integer
library
*id integer
*name string
*sigla string
publication_workplace
*id integer
*faculty string
*desk string
linked_publication
*id integer
*publication_id FK: integer
*linked_publication_id FK: integer
*type string
publication_x_publication_workplace
*id integer
*publication_workplace_id FK: integer
*publication_id FK: integer
publicher
*id integer
*name string
publication_x_publicher
*id integer
*publisher_id FK: integer
*publication_id FK: integer
publication_country
*id integer
*name string
publication_x_publication_country
*id integer
*publication_country_id FK: integer
*publication_id FK: integer
publication_place
*id integer
*name string
publication_x_publication_place
*id integer
*publication_place_id FK: integer
*publication_id FK: integer
publication_year
*id integer
*year integer
*publication_id FK: integer
publication_number
*id integer
*number string
*publication_id FK: integer
publication_original_id
*id integer
*original_id string
*publication_id integer
publication
*id integer
*library_id FK: integer
*date_make date
*date_edit date
*document_type_id FK: integer
*range string
*edition_id FK: integer
*edition_volume string
*source_id integer
*checked boolean
edition
*id integer
*name string
identification_code
*id integer
*code string
*publication_id FK: integer
*identification_type_id FK: integer
identification_type
*id PK: integer
*name string
publication_x_author_person
*id
*publication_id FK: integer
*author_person_id FK: integer
*contribution integer
author_person_x_title
*id integer
*title_id FK: integer
*author_person_id FK: integer
title
*id integer
*title string
source
*id
*source string
author_person
*id integer
*name string
*surname string
*source_id FK: integer
author_person_x_role
*id integer
*author_person_id FK: integer
*person_role_id FK: integer
person_role
*id integer
*role string
author_person_original_id
*id integer
*original_id string
*author_person_id FK: integer
author_person_x_person_workplace
*id integer
*author_person_id FK: integer
*person_workplace_id FK: integer
person_workplace
*id integer
*code string
*name string
url
*id integer
*url string
*publication_id FK: integer
document_type
*id PK: integer
*code_of_type string
*format string
*name string
name_translation
*id integer
*language_id FK: integer
*name string
*publication_id FK: integer
name
*id integer
*type_id FK: integer
*language_id FK: integer
*publication_id FK: integer
*name string
name_type
*id integer
*name string
language
*id integer
*code string
*format string
*name string
name_of_part
*id integer
*name string
*publication_id FK: integer
name_number_of_part
*id integer
*number string
*publication_id FK: integer
name_author_heading
*id integer
*heading string
*publication_id FK: integer
responce
*id integer
*category string
*year integer
*entry string
*index string
*publication_id FK: integer
keyword_subject_name
*id integer
*name string
*keyword_subject_id integer
keyword_subject
*id integer
*original_id string
*keyword_of_part_id FK: integer
*publication_id FK: integer
keyword_person
*id integer
*name string
*surname string
*original_id string
*keyword_of_part_id FK: integer
*publication_id FK: integer
keyword_of_part
*id integer
*part_theme string
*part_geographic string
*part_chronological string
keyword_corporation
*id integer
*name string
*original_id string
*keyword_of_part_id FK: integer
*publication_id FK: integer
keyword_corporation_subname
*id integer
*subname string
*keyword_corporation_id FK: integer
keyword_corporation_attribute
*id integer
*attribute string
*keyword_corporation_id FK: integer
publication_x_author_corporation
*id
*publication_id FK: integer
*author_corporation_id FK: integer
author_corporation
*id integer
*name string
*original_id string
*source_id FK: integer
corporation_attribute
*id integer
*attribute string
*corporation_id FK: integer
corporation_subname
*id integer
*subname string
*corporation_id FK: integer
author_corporation_x_role
*id integer
*corporation_roles_id FK: integer
*author_corporation_id FK: integer
convention
*id integer
*author_corporation_id FK: integer
*number string/integer
*place string
*date date
corporation_role
*id integer
*role string

Funkcie jednotlivých tabuliek sú popísané v tabuľke analýzy CREPČ uvedenej v kapitole Analýza problematiky.

Nasledujúci obrázok znázorňuje schému dátového modelu pre ukladanie informácii používateľov frontendu. Uvedený model má oproti minuloročnému riešeniu výhodu jednoduchej zmeny práv každej skupiny bez potreby zásahu do zdrojového kódu. Podobne aj v prípade pridania novej skupiny bude stačiť nastaviť jej práva v administrácii systému bez programátorského zásahu.

Registrácia používateľov je ošetrená voči spamu a automatickým registráciám pomocou emailovej aktivácie účtu, bez ktorej nie je možné účet využívať.




*5.3.1.5.1 Popis tabuliek frontendu*

**user**

Obsahuje informácie o používateľovi.

| Názov stĺpca | Význam |
|---|---|
| full_name | Celé meno používateľa (aby nebolo nutné nútiť používateľa vypisovať meno, priezvisko, tituly a pod.) |

| | |
|---|---|
| email | Emailová adresa používateľa nutná k registrácii a aktivácii konta |
| password | Zahashované heslo používateľa |
| activated | Informácia, či bol daný účet aktivovaný prostredníctvom aktivačného mailu |
| registration_code | Registračný kód vygenerovaný počas registrácie. Je potrebný na emailovú aktiváciu konta |
| registration_date | Dátum registrácie používateľa. Môže byť potrebné ak sa budú vymazávať neaktivované účty po určitej dobe. |

**group**

Zoznam skupín používateľov.

| Názov stĺpca | Význam |
|---|---|
| label | Názov skupiny, ktorý sa zobrazuje na frontende (napr. Administrátor). |

**permission**

Tabuľka jednotlivých práv, ktoré môžu používatelia nadobúdať.

| Názov stĺpca | Význam |
|---|---|
| name | Názov práva používaný v systéme (napr. showUserList). Význam jednotlivých práv je implementovaný v zdrojovom kóde frontendu. |
| label | Označenie práva, ktoré sa zobrazuje vo frontende (napr. Zobraziť zoznam používateľov). |

# 6 Šprinty zimného semestra

Táto kapitola obsahuje popis jednotlivých šprintov, ktoré sme v rámci našej práce absolvovali cez zimný semester. Šprinty sú očíslované a každý obsahuje zoznam a popis úloh, ktoré sme do neho zaradili.

## 6.1 Šprint 1 - Amstel

### 6.1.1 Analýza algoritmov párovania minuloročného riešenia, návrhy na zlepšenie

Jedna z dôležitých súčiastok projektu je párovanie publikácií. Táto téma je obsiahla a venuje sa jej veľa výskumníkov. Vzhľadom na to, že zameranie práce, na ktorú nadväzuje je do určitej miery iné, tak bolo treba zistiť, do akej miery je návrh komparátora publikácií v minuloročnom riešení dotiahnutý. V rámci tejto úlohy boli analyzované rôzne prístupy k porovnávaniu bibliografických záznamov a algoritmov na určenie podobnosti záznamov.

### 6.1.2 Analýza schémy XML CrepČ

Ako zdroj primárnych údajov, ktoré budeme obohacovať o dodatočné informácie, sme zvolili údaje z Centrálneho registra evidencie publikačnej činnosti (CREPČ). Pre použitie týchto údajov bola potrebná analýza ich formátu a obsahu. Informácie z CREPČ sú dostupné ako XML súbory so štruktúrou popísanou v oficiálnej dokumentácii [7]. Analýza formátu a obsahu nám neskôr bola užitočná pri návrhu relačnej databázy určenej na ukladanie týchto údajov. Táto databáza bude tiež slúžiť na ukladanie dodatočných informácií z iných zdrojov.

### 6.1.3 Analýza Google Scholar

Jednou z primárnych požiadaviek zadávateľa je rozšíriť dáta v systéme o ďalší zdroj - Google Scholar. Úlohou bolo zanalyzovať Google Scholar, dáta ktoré nám ponúka a možnosť ich získania a uloženia do dátového modelu pre ďalšiu prácu s nimi. V tomto šprinte bol navrhnutý spôsob ako budeme dáta získavať a spracovávať a navrhnutý základný prototyp parsera dát.

### 6.1.4 Vytvorenie webovej prezentácie tímu, nainštalovanie virtuálneho stroja

Pre potreby publikácie našej tímovej webovej stránky, sme potrebovali nainštalovať pridelený virtuálny Linux server. Po nespokojnosti s priloženou inštaláciou Fedory sme si vyžiadali distribúciu Ubuntu servera. Tento server sme úspešne nainštalovali a nakonfigurovali. Ďalej sme na neho nainštalovali webový server, kde sme uložili vytvorenú webovú prezentáciu nášho tímu. Webová stránka tímu bola aktualizovaná o informácie o projekte, ktorý riešime a o členoch nášho tímu. Stránka je pravidelne aktualizovaná a postupne do nej pribúdajú zápisnice zo stretnutí, plány projektu a iné potrebné dokumenty.

### 6.1.5 Inštalácia JIRA, Confluence, Stash, Crucible

Pre potreby tímovej komunikácie a kolaborácie sme sa rozhodli nainštalovať si Attlasian produkty. Na tieto produkty sme získali akademickú licenciu a vďaka ručnej inštaláciami sme mali plnú kontrolu nad ich konfiguráciou. Výhoda využívania týchto služieb je taká, že sú integrované, takže máme v jednom balíku nástroj na projektový manažment (JIRA), wiki stránky (Confluence), git (Stash) a review kódu (Crucible).

Inštalácia a konfigurácia týchto nástrojov prebieha iteratívne podľa potreby, čo si vyžiadalo istý čas navyše, ktorý sme do toho museli investovať, avšak po zvážení výhod, ktoré tieto nástroje ponúkajú sme sa zhodli, že to bude pre našu tímovú prácu prínosom.

## 6.2 Šprint 2 - Budweiser

### 6.2.1 Prvá implementácia a testovanie nových párovacích algoritmov

Párovanie publikácií vo všeobecnosti závisí od porovnania autorov a spoluautorov. Na základe analýzy z predchádzajúceho šprintu boli vybrané dva algoritmy existujúce prístupy na párovanie publikácií (Jaro-Winkler a Damerau-Levenstein) a jeden bol navrhnutý naším tímom. Algoritmy boli implementované, otestované a čiastočne prebehlo ich porovnanie.

### 6.2.2 Návrh dátového modelu

Na základe predchádzajúcej analýzy CREPČ sme vytvorili návrh dátového modelu pre vytváraný projekt. S použitím tohoto modelu sme následne vytvorili databázu v PostgreSQL, do ktorej sa mal v niektorom z nasledujúcich šprintov vykonať import prvých dát z CREPČ-u. Vzhľadom na rozsiahlosť a komplikovanosť navrhovaného dátového modelu sme sa niekoľkokrát stretli s komplikáciami, ktoré sa však podarilo rýchlo vyriešiť a neohrozilo to splnenie danej úlohy.

### 6.2.3 Vytvorenie parsera pre XML CrepČ

Pre potrebu vytvorenia databázy, ktorú je potrebné mať na párovanie publikácii, sa údaje získavajú z Centrálneho registra evidencie publikačnej činnosti (CREPČ). Spôsob, akým sa dáta z CREPČ získavajú je export z daného registra do súboru, ktorý je vo forme XML. Pred tým, ako sa do databázy vkladajú tieto dáta z exportu, je potrebné vytvoriť parser, ktorý bude súbor tohto typu spracovávať. Vytvorili sme teda parser, ktorý je schopný export v tvare XML spracovať a uložiť do štruktúry objektov tried, ktoré sú definované presne podľa štruktúry XML CREPČ. Parser funguje na základe parsovacieho algoritmu SAX. Implementácia tohto parsera bola prácna, avšak prebehla bez akýchkoľvek problémov.

### 6.2.4 Vytvorenie parsera pre Google Scholar

Na základe predchádzajúcej analýzy a základného návrhu parsera bola jeho funkcionalita doplnená. Parser bol schopný získať z Google Scholar o zadanom výskumníkovi názvy jeho publikácií spolu so spoluautormi, rokom vydania a zdrojového dokumentu. V rámci tohto šprintu bol následne parser doplnený o možnosť získavania informácií a full-textu článkov a publikácií, ktoré dané dielo zadaného výskumníka citovali. Tu nastal problém, že nie všetky články majú priamy odkaz na PDF. Preto nie každé dielo v databáze bude mať svoj pridružený dokument v dokumentovej databáze.

### 6.2.5 Analýza frontendu minuloročného riešenia

V rámci potreby nadviazania na minuloročný tímový projekt *Odhaľovanie a hodnotenie vzťahov v oblasti vedy a výskumu* sme potrebovali analyzovať frontend, ktorý vytvorili a museli sme určiť mieru znovupoužiteľnosti tohto riešenia. Po stretnutí s členom minuloročného tímu, ktorý sa podieľal na riešení tohto problému, sme zhodnotili, že ich riešenie by bolo znovupoužiteľné, ale len do určitej miery. Vzhľadom na to, že sme sa rozhodli nahradiť ich Mongo databázu relačnou databázou PostgreSQL, bolo by nutné ich riešenie do značnej miery refaktorovať. Rozhodli sme sa preto využiť ich riešenie len ako podklad pre používateľské rozhranie, ktoré bude implementované pomocou GWT a logika aplikácie bude na Java serveri.

## 6.3 Šprint 3 - Carlsberg

### 6.3.1 Vyriešiť blokovanie parsovania Google Scholar

Táto úloha zostala počas šprintu 3 otvorená a nevyriešená. Jej riešenie sa presúva do 4. šprintu.

### 6.3.2 Analýza citačných štýlov

Z dôvodu potreby parsovania referencií z článkov a publikácií, ktoré dané dielo citovali bolo potrebné zanalyzovať súčasný stav citačných štýlov a ich používaní výskumníkmi v rámci Google Scholar. Boli

zanalyzované citačné štýly ISO 690, ISO 690-2, APA 6th, CBE 6th a MLA. Následne bolo zanalyzované, ktoré citačné štýly implicitne Google Scholar ponúka a aké je ich zastúpenie v rôznych publikáciách. Bolo však zistené, že tieto formálne citačné štýly dodržiava len veľmi malé percento autorov. Na výsledkoch tejto analýzy bude stavať vývoj parsera pre získavanie referencií z diel.

### 6.3.3 ORM na import z XML CrepČ

Po tom, ako sme vytvorili parser pre export súboru XML z databázy CREPČ, bolo potrebné vytvoriť systém, ktorý s týmito dátami vedel pracovať. V tomto bode projektu už bola hotová databáza a pripravená pre vytvorenie objektovo-relačného mapovania. ORM sme vytvorili v programovacom jazyku Java a XML aby bol kompatibilný s Parserom XML CREPČ. Generovanie ORM sme použili technológiu Hibernate. Pri vytváraní ORM sme narazili na problémy, ktoré mali formu chýb v databáze. Kvôli tejto skutočnosti sme ORM museli viackrát generovať a v niektorých prípadoch aj manuálne modifikovať generovaný kód.

### 6.3.4 Experiment na výber algoritmu na porovnávanie

Pre dobré párovanie je potrebný správny algoritmus. Neexistuje univerzálne použiteľne porovnanie existujúcich prístupov, už len preto, že ich výkon sa môže líšiť od množiny autorov k množine. Bol uskutočnený experiment, podľa ktorého sa vybral Jaro-Winkler párovací algoritmus. Tento experiment prebehol nad syntetizovanými dátami, kde sa vybrali určité mená zo záznamov z CREPČ a boli syntetizované rôzne preklepy týchto mien. Bol implementovaný prototyp algoritmu na párovanie zahŕňajúci porovnanie viacerých atribútov. K otestovaniu tohto algoritmu už nedošlo kvôli komplikáciám s akvizíciou reálnych záznamov. Ďalej bolo riešené získavanie záznamov zo služieb WoS a Scopus, kde boli uskutočnené experimenty a analýzy existujúceho riešenia, aby sme vedeli použiť existujúce kódy na akvizíciu dát.

### 6.3.5 Vytvorenie ORM pre ukladanie získaných dát z Google Scholar do databázy.

Po úspešnom získaní dát bolo úlohou navrhnúť objektovo - relačné mapovanie do pred pripravenej databázy. Samotný Google Scholar však ponúka len malé percento údajov v porovnaní napríklad z CREPČ. Preto sa budú líšiť záznamy z rôznych zdrojov. Táto úloha však nebola v 3. šprinte úplne dokončená a jej dokončenie bude pokračovať na začiatku nasledujúceho šprintu.

### 6.3.6 Návrh frontend-u

Pre potreby naštýlovania GWT sme potrebovali v prvom rade vytvoriť návrh používateľského rozhrania v statickej forme HTML + CSS. Z tohto návrhu je hotová prihlasovacia obrazovka, zoznam autorov diel a ich filtrovanie. Vytvorenie kompletného návrhu je závislé od pripravenia prostredia pre GWT. Súčasné riešenie je postačujúce pre otestovanie GWT a ostatné časti používateľského rozhrania sa dokončia keď bude GWT nakonfigurované.

### 6.3.7 Realizácia základnej funkcionality frontend-u

Pri analýze riešenia minuloročného tímu zameraného na rovnakú problematiku sme narazili na viacero nedostatkov a jedným z nich bol frontend aplikácie. Identifikovaným nedostatkom bolo rozdelenie logiky systému do rôznych databáz, beckendu a frontendu zároveň, z čoho priamo vyplývali nedostatky pri použití návrhového vzoru MVC a to na všetkých úrovniach.

Preto sme sa rozhodli vytvoriť graficky podobný prototyp s ohľadom na odstránenie analyzovaných nedostatkov použitím technológie GWT.

Táto úloha počas šprintu nebola realizovaná a presúva sa do nasledovného šprintu.

### 6.3.8 Návrh normalizácie dát

Na normalizáciu dát existuje mnoho prístupov, ich správna selekcia pre dané prostredie je kľúčová pre spoľahlivosť párovacích rutín. Cieľom bolo vytvoriť prototyp, ktorý by bol schopný párovania záznamov aspoň v základnej funkcionalite názvu diela a autorov diela. Prototyp bol úspešne implementovaný. Zahŕňa algoritmus párovania záznamov, algoritmus porovnávania reťazcov a algoritmus výberu kandidátov. K jeho testovaniu však zatiaľ kvôli komplikáciám s akvizíciou reálnych údajov nedošlo. Problematické je aj začlenenie prototypu do architektúry systému kvôli nekompletnosti ostatných prvkov a problémov s rôznymi používanými databázami.

### 6.3.9 Správa používateľov (prihlásenia, registrácie)

Pre potrebu autorizácie používateľov, ktorí budú používať vytváraný systém, bolo potrebné analyzovať možné spôsoby ich reprezentácie v dátovom modeli. Medzi hlavnými požiadavkami správy používateľov bola ich registrácia, podpora skupín používateľov a možnosť priraďovania používateľských práv, či už používateľom alebo skupinám. Tieto požiadavky sa s použitím navrhnutého dátového modelu podarilo splniť spolu s pridaním funkcionality ľubovoľného vytvárania a upravovania skupín používateľov, ako aj priraďovaním používateľov do jednotlivých skupín administrátorom (administrátorom systému alebo konkrétnej organizácie). Použitie takéhoto modelu zjednoduší zmenu používateľských práv ako aj významu jednotlivých skupín používateľov, ak to bude v budúcnosti potrebné.

## 6.4 Šprint 4 - Duff

### 6.4.1 Pridanie stĺpca pre indikáciu nespracovaných záznamov

Na zabezpečenie možnosti asynchrónneho spracovania údajov v databáze komparátormi je nutné rozlišovať nové záznamy od už spracovaných. Táto modifikácia databázy túto možnosť zabezpečí. Úloha bola dokončená tak, že bol pridaný nový stĺpec do tabuľky “publication”, ktorý zabezpečuje túto funkcionalitu.

### 6.4.2 Pridanie tabuľky histórie zmien v DB

Na zabezpečenie vrátiteľnosti zmien v databáze je potrebné pridať tabuľku, ktorá túto možnosť zabezpečí. Úloha bola vyriešená. Táto funkcionalita je súčasťou zmien v databáze na podporu spájania publikácií následkom procesu komparácie.

### 6.4.3 Pridanie stĺpca zdroja a tabuľky obsahujúcej zdroje

Kvôli úplnosti záznamu a komparácií publikácií je potrebné evidovať aj zdroj publikácie. V databáze je potrebné cez ID odkazovať na zdroj údajov. Táto úloha bola vyriešená. Táto funkcionalita je súčasťou zmien v databáze na podporu spájania publikácií následkom procesu komparácie.

### 6.4.4 Získanie licencie k Bamboo

Bamboo je ďalší produkt od spoločnosti Atlassian. Je určený na buildovanie a testovanie, spájanie úloh, commitov, výsledkov testov, ale najmä deployovanie na server. Táto funkcionalita pre nás bude užitočná najmä v letnom semestri, kedy budeme potrebovať publikovať výsledok svojej práce veľmi často.

Úlohou v tomto šprinte bolo kontaktovanie Atlassian a získanie akademickej licencie pre tento produkt. Odpoveď nám prišla o týždeň po odoslaní požiadavky a máme k dispozícii licenciu, vďaka ktorej si nainštalujeme tento nástroj.

### 6.4.5 Vytvorenie prototypu frontendu

Pre účely vytvorenia spustiteľného prototypu frontendu bolo nutné vytvoriť prostredie, kde budú môcť prispievať všetci členovia tímu. Rozhodli sme sa pre technológiu GWT, ktorá ponúka framework pre vývoj

webových rozhraní v prostredí Java a využíva prekladanie do natívneho JavaScriptu a používa asynchrónne neblokujúce volania.

Po vytvorení prvého prototypu sme zistili, že táto technológia nie úplne vhodná pre náš typ projektu a rozhodli sme sa zameniť GWT technológiu za JSP a Servlety s použitím Bootstrap knižnice, ktorá ponúka obohatenú paletu pre návrh dizajnu.

Vytvorením prototypu frontendu pomocou JSP sme odstránili nefunkčné časti GWT prototypu a dosiahli sme jednoduchšiu možnosť vývoja a správy frontendu.

Pomocou spomínaných technológiu sme dokázali vytvoriť jednoduchý prototyp v ktorom bolo umožnená funkcionalita registrácie a prihlasovania používateľov. V prototype bol využitý aj ORM mapovač pomocou, ktorého sa vytvárajú dopyty do k databázovej schéme používateľov a ich práv.

### 6.4.6 Inštalácia Tomcat servera na arl

Pre umožnenie deployovania iterácii nášho produktu (frontendu) sme na náš server nainštalovali a nakonfigurovali Tomcat server. V súčasnosti naň publikujeme vytvorenú aplikáciu ručne, ale v budúcnosti máme v pláne na túto úlohu využívať Bamboo.

### 6.4.7 Vytvorenie serverovej časti v GWT package pre správu prihlásenia používateľov

Prvá časť prototypu frontendu odnášala funkcionalitu prihlásenia používateľov, kde sa frontend pomocou ORM prihlásil do Postgre databázy a sprostredkoval prihlásenie a registráciu používateľov.

### 6.4.8 Realizácia základnej funkcionality frontendu

Medzi základnú funkcionalitu frontendu sme zaradili zobrazenie zoznamu autorov a publikácií. Pre potreby prvého prototypu aplikácie sme túto funkcionalitu implementovali a otestovali. Po niektorých zmenách je frontend pripravený obohatiť minuloročné riešenie o dodatočnú funkcionalitu.

### 6.4.9 Zistenie možnosti volania C programov z javy

Kvôli rýchlosti boli komparačné algoritmy implementované v C++. Aby sa spojil hlavný program, ktorý je v Jave s komparátormi, je nutné zistiť najvýhodnejšiu metódu volanie C++ programov z javy. Úloha bola vykonaná. Bolo vybrané volanie pomocou CMD a to predovšetkým kvôli udržaniu striktnej samostatnosti komparátorov, veľmi dobrej regulácie používania prostriedkov a kvôli možnosti asynchrónneho spracovania (ktorá v tomto modely je implicitná).s

### 6.4.10 Definícia komunikačného rozhrania medzi komparátormi a javou

Aby časť riešenia napísaná v Jave mohla komunikovať s komparátormi je potrebné určiť komunikačný protokol, ktorý budú obe strany používať. Analýzou potrieb komparátora sa určil protokol, ktorý pracuje na úrovni publikácií a ich atribútov. Jeho bližší popis je uvedený v dokumentácií C komparátora.

### 6.4.11 Vytvoriť package v jave pre Scheduler

Súvisí to s plánovaním spúšťania párovania. Scheduler zatiaľ nebol vyriešený. Predovšetkým preto, lebo detailné testovanie komunikácie medzi Javou a Komparátormi malo vyššiu prioritu.

### 6.4.12 Návrh normalizácie údajov

Úloha bola definovaná v predchádzajúcom šprinte. Výsledkom je pár komparátorov. Prvý je definovaný v jazyku C++ a pracuje s algoritmami porovnávania. Druhý je definovaný v Jave a jeho úlohou je zabezpečiť prístup do databázy, aplikovať algoritmy na výber kandidátov a volať C++ komparátor. Úloha nebola dokončená a to hlavne z dôvodu pokračujúceho testovania, keďže najdôležitejšia vlastnosť tohto systému má byť spoľahlivosť.

# 7 Použitá literatúra

[1] CARVALHO, A. P. DE et al. 2011. Incremental Unsupervised Name Disambiguation in Cleaned Digital Libraries. In *Journal of Information and Data Management.* ISSN 2178-7107, 2011, vol. 2, no. 3, p. 289–304.

[2] BHATTACHARYA, I. et al. 2006. A Latent Dirichlet Model for Unsupervised Entity Resolution. In *Proceedings of the Sixth SIAM International Conference on Data Mining*. Bethesada: SIAM, 2006. ISBN 978-089871611-5, p. 47-58.

[3] CULOTTA, A. et al. 2007. Author Disambiguation using Error-driven Machine Learning with a Ranking Loss Function. In *AAAI Workshop - Technical Report.* Vancouver: AAAI, 2007. ISBN 978-157735341-6, p. 32-37.

[4] FERREIRA, A. A. et al. 2010. Effective Self-Training Author Name Disambiguation in Scholarly Digital Libraries. In *Proceedings of the 10th annual joint conference on Digital libraries.* New York: ACM, 2010. ISBN 978-1-4503-0085-8, p. 39-48.

[5] HERNANDEZ, M. A. 1995. The Merge/Purge Problem for Large Databases. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. New York: ACM, 1995. ISBN 0-89791-731-6, p. 127-138.

[6] VRIES, T. DE. 2011. Robust Record Linkage Blocking Using Suffix Arrays

and Bloom Filters. In *Transactions on Knowledge Discovery from Data (TKDD).* ISSN 1556-4681, 2011, roč. 5, č. 2, čl. 9

[7] Portál CREPČ - Centrálny register publikačnej činnosti a Centrálny register umeleckej činnosti.[Online] [Dátum: 5. 9 2013.]

http://cms.crepc.sk/Data/Sites/1/pdfsubory/docword2.5.pdf.

# Ciele projektu na letný semester

V zimnom semestri sme analyzovali problematiku, jednotlivé asti existujúceho projektu a na základe týchto vedomostí sme navrhli nový relaný dátový model, ku ktorému sme vytvorili zodpovedajúci objektovo relaný mapova. Kvôli zmene databázy na relanú sme museli upravi existujúce parsery, ktorých funkcionalita zostala z vekej asti zachovaná. V niektorých bodoch sme však funkcionalitu rozširovali, aby sme tak získali viac informácií, ktoré predchádzajúci tím zanedbal.

alej sme navrhli a implementovali prvú verziu vylepšeného algoritmu porovnávania, ktorý je založený na porovnávacom algoritme Jaro-Winkler a algoritme na výber kandidátov Sorted Neighbourhood Method. Okrem toho sme navrhli grafický vzhad webového rozhrania, priom sme vycházali zo skúseností predošlého tímu. Bol vytvorený funkný prototyp zobrazovania publikácií. Okrem toho sme navrhli a implementovali modelovanie používateov v systéme.

Na letný semester sme si urili nasledovné ciele:

Overi zlepšenie algoritmov porovnávania publikácií, pretože v zimnom semestri boli tieto algoritmy testované len na syntetických vstupoch a bolo zamerané na porovnávanie autorov. Navyše bolo potrebné obohati tieto algoritmy tak, aby po porovnávaní nastalo automatické spájanie zhodných publikácií.

Rozšíri funkný prototyp webového rozhrania o aktuálne požiadavky zo strany zákazníka, priom implementova minimálne zobrazovanie publikácií a autorov a príslušných informácií k nim.

Rozšíri zdroje údaajov o alšie, najmä o zdroje z citaných indexov, aby sme tak získali reálne údaje použitené na testovanie, ako aj na prezentáciu funkcionality zákazníkovi. Pri implementácií parserov z týchto zdrojov treba vychádza z úspešne overených algoritmov minuloroného tímu, priom vychádza z modelovej komunikácie so zodpovedajúcimi webovými službami.

Vykona analýzu údajov a zobrazi základné vzahy medzi publikáciami, vedeckými inštitúciami a autormi v zaujímavej forme. Odhali niektoré vzahy, ako napríklad spoloné práce niekokých inštitúcií alebo zobrazi vhodným grafom poet citácií na autora a podobne.

# Šprinty v LS

## User stories – šprint Falcon

**User story 1 – Rozšíri filter pre zobrazovanie údajov vo web rozhraní**

Používate nášho systému ho používa na rôzne úely, ktoré nemožno všetky dopredu predpoveda. Podstatné je, aby sme mu ponúkli o možno najširšiu možnú paletu možností, ktorými môže ovplyvni výsledok dopytu na zobrazenie autorov, resp. publikácií. Preto by mal filter obsahova možnos filtrova poda rôznych polí, by konfigurovatený, poskytova podporu AND a OR logických operátorov a podobne. Stránky musíme udržiava v dobrej kvalite údajov, aby sme nemiatli používateov nejasnosou výrazov.

**User story 2 – Oddeli normalizované údaje od nespracovaných**

Údaje nachádzajúce sa v databáze zobrazovanej na webovom rozhraní by mali by normalizované, aby sme ponúkali používateom informácie bez duplicít. Preto by bolo vhodné oddeli v databáze neisté údaje od istých a párovací proces ako spôsob normalizácie neistých údajov. Používate by nemal vidie nespracované záznamy, ktoré by mali predstavova rad údajov akajúcich na spracovanie.

**User story 3 – Zabezpei v projekte udržovatenos**

Produkt odovzdaný zákazníkovi by mal spa podmienku udržovatenosti, ktorá je kúová pre alší vývoj a údržbu systému. Preto je potrebné sa priebežne venova skontrolovaním tejto vlastnosti a prípadne jej zabezpeením. Jednou z oblastí udržovatenosti je dátový model a k nemu prislúchajúci objektovo relaný mapova, ktorý je predpokladom zabránenia reazeniu zmien v dôsledku zmeny dátového modelu.

**User story 4 – Používa systém môžu len autentifikovaní používatelia**

Kvôli dodržiavaniu licencií, ktoré používame pri vyhadávaní ohlasov na diela by mala by táto funkcionalita dostupná len autentifikovaným používateom, ktorých totožnos administrátor overí. Súasou registrácie preto musí by viackrokový overovací mechanizmus. Po prihlásení by sa mal používate dosta na úvodnú stránku, ktorá by ho oboznámila s možnosami a úelom nášho systému. Taktiež by mal možnos nastavi si urité atribúty jeho profilu, mal by ma možnos personalizovaného nastavenia filtrov a podobne.

## User stories – šprint Gambrinus

**User story 1 – Zobrazova percentá podobnosti pre spárované objekty v databáze a umožni na základe nich runé párovanie**

Na skontrolovanie správnosti porovnávacích algoritmov je dôležité, aby boli vypoítané percentá podobnosti ukladané do databázy a zobrazované na frontende. Je to potrebné jednak z hadiska kontroly párovania, a potom je to dôležitým podkladom pre vyhadávanie publikácií na runé párovanie. Preto budú tieto údaje ukladané po uskutonení párovania. Predpokladáme, že týchto údajov bude vemi vea, preto je dôležité s tým poíta pri návrhu doplnenia dátového modelu a pri práci s týmito údajmi. Vo frontende je potrebné implementova obrazovku pre runé párovanie, ktorá by umožovala používateovi vybra dve publikácie, ktoré by chcel spárova. Následne by sa mu zobrazili podrobnosti pre vybrané publikácie a používate by mal možnos vybra jednu z metód párovania.

**User story 2 – Implementova nástroje na získavanie ohlasov na diela z citaných indexov**

Automatické získavanie ohlasov je jedna z hlavných úloh systému. Citané indexy poskytujú webové služby na získavanie ohlasov, ktoré treba pomocou minuloroného projektu naštudova a implementova riešenie využívajúce dané služby na obohatenie naparsovaných údajov z exportov inštitúcií o ohlasy na ich diela. Zaha to vytvorenie programu na komunikáciu s webovými službami, parser na spracovanie výsledkov, ktoré nám vrátia a program ukladajúci spracované údaje do databázy.

**User story 3 – Upravi systém tak, aby podporoval alternatívne mená autorov a alternatívne názvy pracovísk**

Známym aspektom údajov z rôznych zdrojov je problém s nejednoznanosou a nepresnosou uložených údajov. Problém môže nasta chybou pri zadávaní nových údajov do databáz alebo preklepom. Vtedy nie je možné vyhadáva poda štandardných názvov pracovísk, pretože sú uložené inak, ako by sme akali. Preto je potrebné uklada názvy pracovísk, na ktoré pri vyhadávaní narazíme a aby sme ich v budúcnosti mohli zahrnú do vyhadávacích kritérií. Podobne treba pristupova k menám autorov. Tento prístup je pre publikácie už implementovaný, avšak upravi dátový model treba pre ostatné typy objektov.

# User stories – šprint Heineken

**User story 1 – Pripravi databázu, párovacie algoritmy a frontend na testovanie zákazníkom**

Keže nemáme normalizovanú testovaciu množinu, na ktorej by sme overili vlastnosti a správnos párovacích algoritmov, túto innos musí vykona zákazník (zástupca naparsovaných inštitúcií). My mu k tomu musíme pripravi podklady a poskytnú podporu v našej aplikácií. Zaha to dokoni parsovanie exportov minimálne z univerzít SPU, EU a TUKE, dokoni párovacie algoritmy, otestova ich správnu funknos a následne vyísli hodnoty párovania pre všetky záznamy v databáze. Keže zákazník nebude odborník na databázu, potrebujeme dokoni zobrazovanie podobností na frontende tak, aby si to mohol jednoducho poprezera a zisti, ktoré publikácie by mali by napárované a nie sú. Užitoné bude zisti aj štatistiky k jednotlivým algoritmom ohadne ich trvania a efektivity.

**User story 2 – Dokoni algoritmy sahovania záznamov publikácií z citaných indexov WoS a Scopus poda naparasovných inštitúcií**

Automatické získavanie ohlasov je jedna z hlavných úloh systému, preto by sme ju mali ím skôr dokoni. Pre zákazníka je podstatné vidie, koko publikácií má pokrytých v citaných indexoch a koko majú ohlasov. Implementované boli niektoré asti pre služby WoS-u a všetky pre služby Scopusu, preto treba dokoni zostávajúce asti a stiahnu záznamy publikácií poda naparsovaných inštitúcií. Treba pritom vykona rešerš, ktorá nám identifikuje alternatívne názvy inštitúcií, pod ktorými sa diela môžu nachádza. Na základe alternatívnych mien je potrebné spusti sahovanie záznamov a uklada ich do databázy. Na základe týchto záznamov údajov sme po vykonaní procesu deduplikácie schopní získa identifikátory pre dané publikácie v citaných indexoch. Identifikátory platné pre citané indexy sú potrebné na získanie ohlasov na diela, ktorých získavanie je naším cieom.

# User stories - šprint Imperial Eclipse Stout

**User story 1 – Pripravi databázu, párovacie algoritmy a frontend na testovanie zákazníkom**

Keže nemáme normalizovanú testovaciu množinu, na ktorej by sme overili vlastnosti a správnos párovacích algoritmov, túto innos musí vykona zákazník (zástupca naparsovaných inštitúcií). My mu k tomu musíme pripravi podklady a poskytnú podporu v našej aplikácií. Zaha to dokoni parsovanie exportov minimálne z univerzít SPU, EU a TUKE, dokoni párovacie algoritmy, otestova ich správnu funknos a následne vyísli hodnoty párovania pre všetky záznamy v databáze. Keže zákazník nebude odborník na databázu, potrebujeme dokoni zobrazovanie podobností na frontende tak, aby si to mohol jednoducho poprezera a zisti, ktoré publikácie by mali by napárované a nie sú. Užitoné bude zisti aj štatistiky k jednotlivým algoritmom ohadne ich trvania a efektivity.

**User story 2 – Hadanie ohlasov, aktualizácia údajov**

Vzhadom na to, že systém ma predovšetkým slúži výskumníkom na hadanie ohlasov, je potrebné pripravi zákazníkovi rozhranie na zobrazenie ohlasov, ktoré bude jednoduché a intuitívne na poskytnutie relevantnej informácie ohadom ohlasov. Na zobrazovanie reálnych ohlasov je potrebné tieto stiahnu zo zdrojov ako sú WoS a Scopus. Ohlasy je potrebné stiahnu aspo pre SPU, EU a TUKE. Používate chce tieto ohlasy ma bez autocitácií a taktiež bež duplikátov. Na to je potrebné pripravi subsystém a pripravi potrebné rutiny na deduplikáciu ohlasov ako aj rutiny na detekciu autocitácií. Je potrebné dohodnú spôsob, akým sa budú ohlasy uklada do databázy.

**User story 3 – Personalizácia rozhrania**

Zákazník by rád vedel ovplyvni spôsob, akým sa mu zobrazujú údaje a zárove chce, aby mu zobrazovalo len tie údaje, ktoré ho zaujímajú. Vzhadom na to, že používate nebude technologický expert, tak mu musí by poskytnuté jednoduché intuitívne rozhranie na selekciu toho, o si praje ma zobrazené a o nie. Na toto je potrebné pripravi rozhranie a zárove implementova uloženie do databázy. Ke sa používate pripojí, chce, aby sa mu jeho nastavenia obnovili a preto je potrebné urobi rutinu na znovu naítanie údajov z databázy.

# User stories - šprint Jai Alai IPA

**User story 1 – prezentácia potenciálnym klientom na IIT.SRC**

Zákazník chce prezentova produkt potencionálnym zákazníkom na študentskej vedeckej konferencií IIT.SRC 2014. Na to je potrebné vytvori poster a prezentáciu. Taktiež je potrebné, aby tieto zaujali formou prezentácie a obsahom. Je potrebné na prilákanie klientov aby lenovia

prezentaného tímu poznali dobre prezentované materiály, ako aj aby vedeli dopredu ktoré výhody a fakty chcú zdôrazni alebo na ne chcú poukáza. Nakoniec na alšiu prezentáciu projektu je potrebné, aby lenovia tímu nosili poas konferencie dopredu pripravené triká, ktoré majú taktiež potencionálneho klienta zauja.

**User story 2 – vizualizácia vzahov medzi údajmi v databáze**

Klient chce zisti alšie vzahy medzi publikáciami. Na to potrebujem definova aké vzahy môžu by zaujímavé. Následne potrebuje návrh vizualizácie. Tento návrh zahruje technickú realizáciu v databáze, na frontende a taktiež výpotové jadro tohto vzahu. alej musí obsahova návrh modelovania týchto vzahov. V ideálnom prípade by zákazník chcel, aby tento proces bol parametrizovatelný a vzah aby bolo možné modelova dynamicky poda požiadavky klienta.

Tento user story sme neskôr zmenili na celkové testovanie funkcionality a úpravu algoritmov poda zistení poas prezentácie na konferencii IIT.SRC. Taktiež bolo potrebné odstráni niektoré chyby, ktoré ovplyvovali funknos algoritmov.

# Research Rank

## O projekte

Research Rank je projekt ktorý je zameraný na vytvorenie systému, ktorý uahí vedeckým pracovníkom a pracoviskám vyhadáva ohlasy na ich publikácie, vzahy medzi publikáciami a podobne. Projekt je veké riešenie na analýzu údajov z viacerých zdrojov ako sú WoS, Scopus a podobne. Projekt je vyvíjaný v Jave a C++ s PostgreDB databázou.

V údajoch, ktoré Reserach Rank spracováva je veké množstvo chýb a mnohé alšie údaje sú oznaené rôznymi variantnými formami mena poprípade skratiek. Z toho vyplývajú výzvy s ktorými sa musí tento projekt vysporiada. Okrem vekého objemu údajov to sú predovšetkým výzvy týkajúce sa zbierania údajov z rôznych štandardov, služieb a stránok, jednoznaná identifikácia a deduplikácia entít. V neposlednom rade je to aj efektívna implementácia a user friendly rozhranie na prácu s naším systémom pre používateov. alšia výzva je odvodzova z údajov vzahy medzi nimi. Ide nie len o priame vzahy akými sú napríklad prehad autorov, s ktorými daný vedec najastejšie publikuje, ale aj nepriame vzahy ako je napríklad odvodenie dvojíc vedcov, ktorých publikácie majú skoro identické tematické zameranie.

Naším cieom je ponúknu o možno najspoahlivejšie údaje v itatenej forme a tým v maximálnej možnej miere šetri as vedeckým pracovníkom i už hadajú citácie na svoje diela, diela ich kolegov alebo sa zaujímajú o iný aspekt údajov, ktoré máme uložené a interpretované v databáze.

## Opis problému

Vedecké pracoviská fungujú z vekej asti na základe príjmov z grantov, na ktoré prispieva ministerstvo školstva SR. Ak pracoviská chcú získava tieto granty, musia sa usilova o výskumnú innos a výsledky z nej publikova. Výsledky výskumnej innosti sa okrem vedeckých digitálnych knižníc evidujú v našej krajine do Centrálneho registra publikanej innosti (CREP). Jedným zo základných kvalitatívnych meradiel výsledkov výskumnej innosti je poet ohlasov na publikované diela, o znamená, že as daného diela bola citovaná v inom novom diele, ktoré bolo publikované v rámci rovnakej výskumnej domény.

Získavanie prehadu o ohlasoch na autorove vlastné diela je vemi ažkou a zdhavou manuálnou innosou, ktorá by sa mala da zautomatizova vytvorením systému, ktorý bude informácie o týchto dielach spracováva automaticky a bude tak šetri as autorom publikovaných diel. Pri manuálnom vyhadávaní autori diel taktiež robia chyby, ktoré vznikajú napríklad nechcenom prehliadnutí iného diela a podobne. Pre obohatenie databázy týchto ohlasov je potrebné ju naplni dátami z nových zdrojov a identifikova nové prepojenia za pomoci porovnávacích algoritmov. Tomuto kroku však predchádza ešte jeden alší fakt, že tieto diela môžu by v CREP zadané chybne alebo v rôznych variantoch. Informácie o týchto dielach sa do CREP zadávajú manuálne a dôvodom obasnej chyby pri zadávaní je samozrejme udský faktor.

Získavanie informácií o výskumnej innosti v našej krajine je teda vemi zdhavý a zložitý proces.

alší zaujímavý aspekt problému sú vzahy medzi publikáciami. ktoré môžu obsahova celú škálu užitoných meta-údajov. Niekedy sa stáva, že vedec potrebuje iný typ údajov ako len citácie. Môže ís o odborné zameranie daného iného vedca, alebo astých spoluautorov daného iného vedca, potencionálnych meta-údajov je veké množstvo.

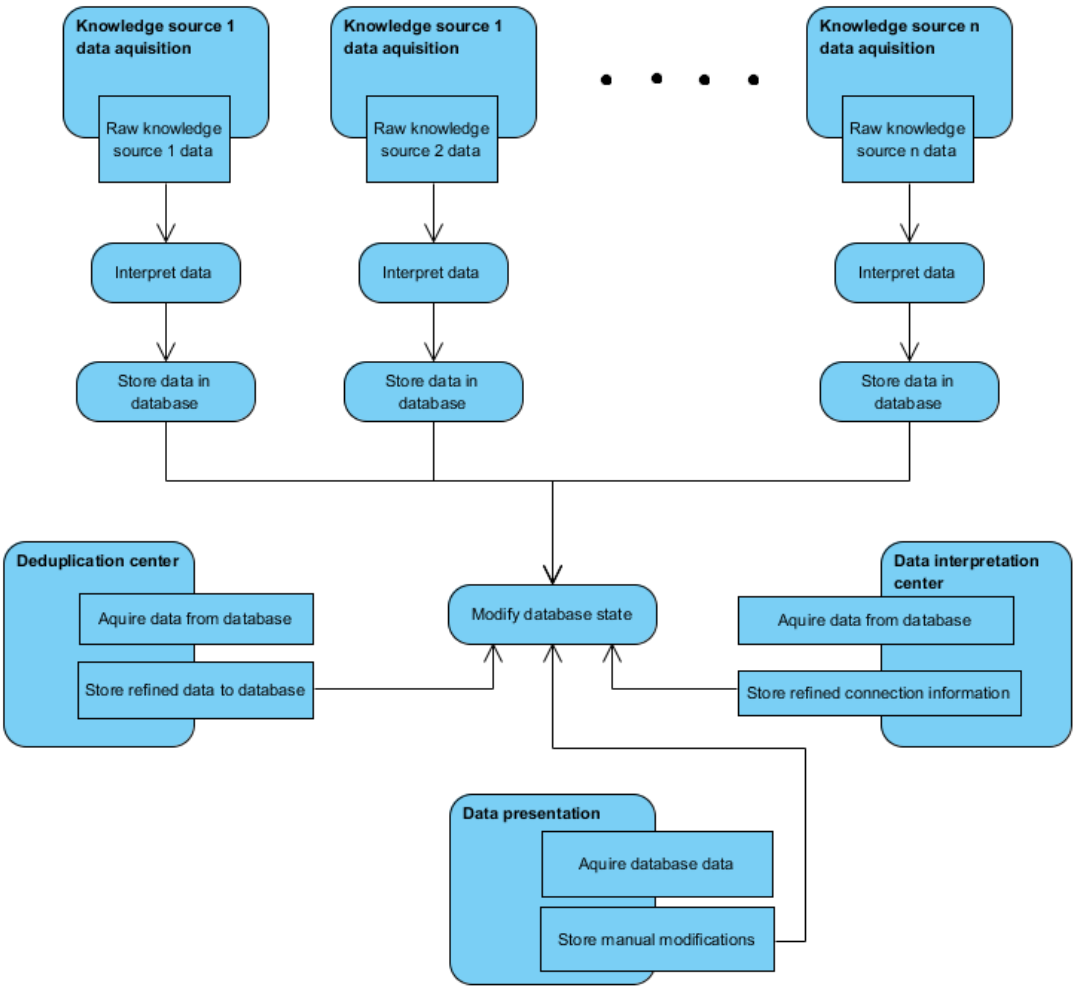
## O produkte

Výsledkom nášho snaženia bude niekoko softvérových produktov:

- PostgreDB so štruktúrou a údajmi z rôznych zdrojov, ktoré bude možné regulárne obohacova.
- Komparátor napísaný v C++, ktorý je samostatne použitený na porovnávacie služby.
- Korpus programu v Jave, ktorý zabezpeuje biznis logiky frontendu, deduplikácie a v neposlednej rade obohacovanie databázy o nové informácie zo zdrojov.

## Konceptuálny model systému

Model systému na vysokej úrovni abstrakcie.




# Definície s skratky

- RR - Research rank
- Komparátor - súiatka/komponent zabezpeujúci porovnanie autorov a publikácií
- ORM - objektovo-relaný mapova
- CREP - Centrálny register evidencie publikanej innosti
- GWT - Google Web Toolkit

# Referencie

# Organizácia dokumentu

Dokumentácia je organizovaná do niekoko za sebou nasledujúcich astí:

1. Všeobecný popis produktu, architektúra, ciele, všeobecná analýza a špecifikácia požiadaviek
2. Popis Komparátorov - požiadavky, návrh, implementácia, opis rozhrania a služieb
3. Popis Frontendu - požiadavky, návrh, implementácia, opis služieb
4. Popis komponentov spojených s parsovaním WoS - požiadavky, návrh, implementácia, opis rozhrania a služieb
5. Popis komponentov spojených s parsovaním Scopus - požiadavky, návrh, implementácia, opis rozhrania a služieb
6. Popis komponentov spojených s parsovaním Crep - požiadavky, návrh, implementácia, opis rozhrania a služieb
7. Popis nášho dátového modelu v PostgreDB - požiadavky, návrh, implementácia, údajový model
8. Popis ORM - Požiadavky, návrh, implementácia, opis rozhrania

# Analýza problematiky

## Problematika párovania publikácií

Na párovanie publikácií existuje množstvo lánkov a odborných prác. Existujú prístupy, ktoré sa snažia vyvinú unsupervised metódu bez nutnosti refaktorizácie celej databázy [1], prístupy, ktoré sa snažia párova publikácie na základe vyššej štatistiky [2], na základe strojového uenia [3], na základe heuristík a rozšírenia existujúcich metód [4] a mnohé iné. Dodnes neexistuje algoritmus, ktorý by s uritosou vedel spárova všetky duplicitné publikácie iba poda údajov uvedených v zdroji informácií.

## Špecifikácia problému

Problém ako taký pozostáva z viacerých pod-problémov. Akýkovek algoritmus párovania musí poíta s neúplnými alebo nesprávnymi údajmi, polymorfiou mien, viac-jazynými iastonými prekladmi, podobnosou nadväzujúcich prác. Následkom týchto faktorov sa pri párovaní používajú váhy na jednotlivé dostupné informácie.

Taktiež je mnoho problémov implicitne naviazaných na jazyk, v ktorom sú situované. Napríklad mená slovenských autorov majú iné chyby a skomoleniny, špecifické problémy a synonymá ako mená anglických autorov a podobne.

## Používané prístupy

Vo všeobecnosti sa páruje poda prvého autora a potom sa používajú iné dostupné informácie a uritou váhou na verifikáciu výsledku porovnania prvých autorov. Používajú sa co-autori, názov práce, abstrakty, ISBN, ISSN a iné informácie, avšak väšinou iba niektoré z týchto sú dostupné a v prípade autorov je výsledok tiež podmienený nejakou pravdepodobnosou zhody. Kvôli tomuto vznikli aj alternatívne metriky, ktoré hodnotia kontext práce, jej zameranie, zhodu zamerania co-autorov a pod. Je možné logicky rozdeli techniky, ktoré sa používajú na techniky párovania mien, techniky porovnávania údajov a techniky porovnania kontextu.

### Párovanie mien

Existuje množstvo prístupov avšak študované boli predovšetkým tie najpoužívanejšie. Menovite algoritmu Jaro-Winkler, Levensteinova vzdialenos a Damerau-Levensteinova vzdialenos, tieto porovnávajú v podstate aké veké úsilie je potrebné na to, aby sa prvý len porovnania zmenil na druhý len. alej sa preštudovali algoritmy, ktoré porovnávajú priamo podiel rovnakých sekvencií v reazci. Tu boli preštudované aspekty 2-gramov a 3-gramov.

### Techniky porovnávania údajov

Porovnávania údajov je vo všeobecnosti dobre preskúmaná oblas, ktorou sa zaoberá množstvo publikácií. Pre nás boli relevantné predovšetkým techniky, ktoré boli priamo už niekým použité pri porovnávaní publikácií, alebo techniky, ktoré by mohli urité iastkové problémy porovnávania vyrieši. Predovšetkým problematický je nedostatok informácií.

Väšina porovnávaných atribútov je tradine otázka identity alebo dostatoného prieniku dvoch množín údajov. Z tohto hadiska sú zaujímavé predovšetkým porovnania plných textov a abstraktov, poprípade titulov prác. Tu sa asto aplikuje heuristika, ktorá hovorí, že texty, resp. abstrakty alebo tituly musia by identické na to, aby to bola rovnaká práca. Aj napriek tomu, že sa s tým vo všeobecnosti odborná komunita stotožuje, problémy do tejto problematiky vnášajú rôzne súasti, ktoré môžu by k textu pridané  zo zdroja. V tomto prípade i ke ide len malú súas, ktorá nemá s obsahom alebo formátom samotného textu ni spoloné, tak rozvráti túto vo všeobecnosti zaužívanú heuristiku. Existuje množstvo metód, ktoré sa na obídenie tohto problému zameriavajú. Naše štúdium sa zameralo predovšetkým na metódy s de Bruijnovými grafmi a String grafmi. Tieto okrem porovnania samotných textov môžu poda potreby poskytnú vea iných užitoných informácií.

### Techniky porovnávania kontextu

Na porovnávanie kontextu lovek potrebuje množstvo metadát. Na urenie kontextu práce treba zisti zameranie práce, množinu autorov, s ktorými prvý autor bežne spolupracuje a samotný zdroj práce a jeho zameranie. Napríklad sa používa budovanie odborných slovníkov a porovnávanie s množinami iných autorov na urenie zamerania konkrétnej práce. Na budovanie odborného slovníka sa používajú napríklad spomínané de Bruijnové grafy alebo String grafy (a rôzne ich adaptácie). V nich po urení prieniku a vyiarknutí najpoužívanejších slov sa vytvoria implicitne prepojené skupiny spojení špecifické pre danú oblas, skupinu alebo indivíduum. Touto technikou, výberom a porovnaním situovania jednotlivých entít sa zistí kontext práce. Na tento proces však je potrebné množstvo informácií predovšetkým v oblasti kontextu práce, a preto je to prakticky nevyužitené pre autora, ktorý je úplne nový pre databázu, pretože o om nie je dos informácií. Taktiež existujú pokusy na urenie kontextu zo samotných záznamov zdroja publikácie, pracoviska autorov a podobne. Vo všeobecnosti, aj ke je tento údaj vemi užitoný, nie vždy je možné ho získa z dostupných informácií a preto je väšinou chápaný existujúcimi prístupmi ako menej dôležitý ako predchádzajúce dva komponenty párovania.

### Techniky výberu kandidátov na porovnanie

V zásade je nemožné porovnáva záznam oznaený na párovanie s celou databázou záznamov, pretože takéto porovnávanie by viedlo k zahlteniu systému a zbytonému porovnávaniu takých záznamov, ktoré majú len málo spoloné. Oblas vyberania kandidátov na porovnávanie je vcelku dobre zmapovaná, existujú dva hlavné prístupy, ktoré potom rôznymi obmenami dosahujú lepšie výsledky v uritých špecifických oblastiach. Jedným prístupom je tzv. blokový prístup (napríklad [6]), ktorý rozdelí celú množinu záznamov na bloky, v rámci ktorých potom záznamy

porovnáva. Druhým prístupom je Sorted Neighbourhood Method [5], kde ide o to, že máme záznamy zoradené poda niektorého poa(asto je to primárny kú alebo najdôležitejší záznam, ktorý je o najviac jedinený) a porovnávame len v rámci uritého okna záznamov.

### Supervised vs. unsupervised metódy

Všetky techniky párovania podliehajú ešte jednému atribútu a to je supervised alebo unsupervised. Rozdiel je v tom, že prvé sú pred nasadením kalibrované na známej správnej množine údajov, aby sa urili ich parametre. Je to kvôli mnohým faktorom, ale predovšetkým pre to, že v rôznych jazykoch sa grafy efektivity hodnoty parametrov môžu líši. Naproti tomu unsupervised metóda je univerzálne navrhnutá tak, aby fungovala na ubovoných údajoch. Vo všeobecnosti však prevládajú supervised metódy a to iastone kvôli tomu, že konštrukcia unsupervised metódy môže by ovea ažšia a iastone preto, že unsupervised metódy musia poíta s ovea viacej faktormi, o má vplyv na ich výkon.

## Zdroje metadát publikácií

Základnú množinu metadát o vedeckých prácach získavame z portálu Centrálneho registra evidencie publikanej innosti (alej CREP). Na tomto portáli je možné vyhadáva vedecké publikácie na základe názvov, mien autorov, pracovísk, hesiel, roku vydania publikácie, roku citovania, ISBN, ISSN a vydavatestiev. CREP tiež poskytuje roný export metadát publikácií daných vedeckých pracovísk vo formáte XML. Na vytvorenie úvodnej množiny údajov používame práve tieto exporty, z ktorých získavame informácie o konkrétnom diele, jeho autoroch a iné užitoné metadáta (rok vydania, vydavatestvo, pracovisko a pod.).

Vzhadom na neúplnos a obasnú nepresnos záznamov v xml exporte CREP, obohacuje tieto údaje použitím webových služieb citaných indexov ako sú WOS a SCOPUS.

## Referencie


[1] CARVALHO, A. P. DE et al. 2011. Incremental Unsupervised Name Disambiguation in Cleaned Digital Libraries. In *Journal of Information and Data Management.* ISSN 2178-7107, 2011, vol. 2, no. 3, p. 289–304.

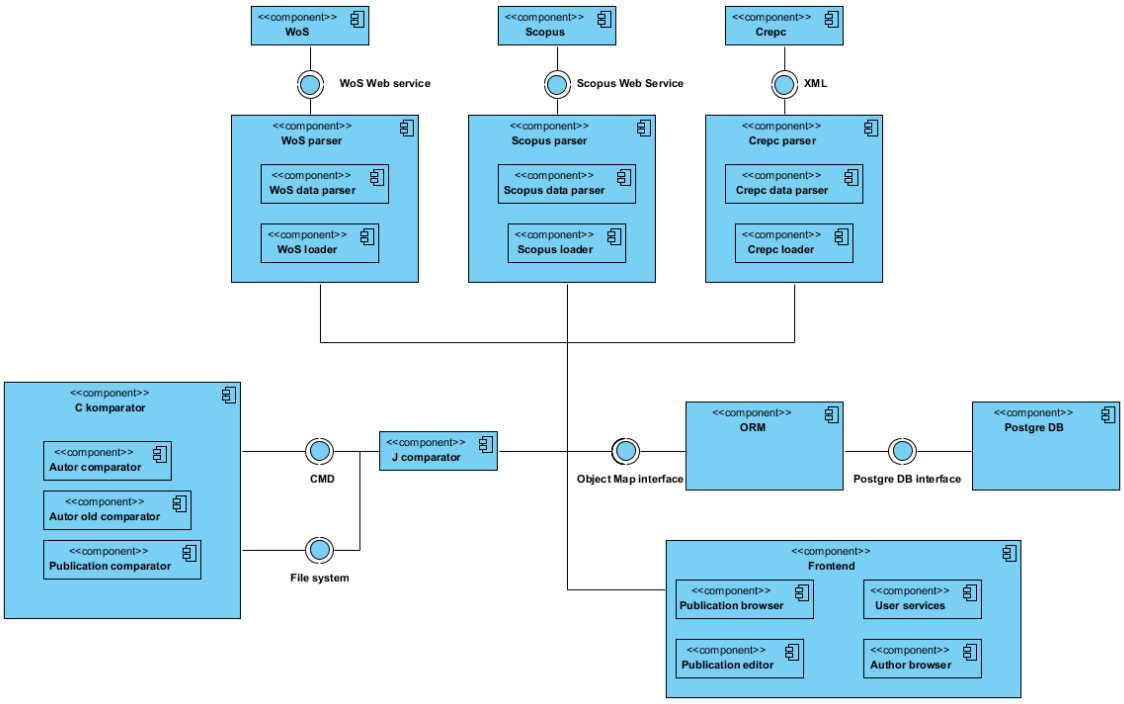
[2] BHATTACHARYA, I. et al. 2006. A Latent Dirichlet Model for Unsupervised Entity Resolution. In *Proceedings of the Sixth SIAM International Conference on Data Mining*. Bethesada: SIAM, 2006. ISBN 978-089871611-5, p. 47-58.

[3] CULOTTA, A. et al. 2007. Author Disambiguation using Error-driven Machine Learning with a Ranking Loss Function. In *AAAI Workshop - Technical Report.* Vancouver: AAAI, 2007. ISBN 978-157735341-6, p. 32-37.

[4] FERREIRA, A. A. et al. 2010. Effective Self-Training Author Name Disambiguation in Scholarly Digital Libraries. In *Proceedings of the 10th annual joint conference on Digital libraries.* New York: ACM, 2010. ISBN 978-1-4503-0085-8, p. 39-48.


# Architektúra systému

## Model architektúry



Obr.1: Model architektúry systému Research Rank

# Všeobecný popis architektúry

Architektúra sa dá popísa všeobecným architektonickým modelom úložisko. Komponent fungujú na základe údajov z PostgreDB databázy a zárove sú nezávislé.

# Všeobecný popis komponentov

Komponenty sa dajú vo všeobecnosti rozdeli do 4 skupín:

- Parsery
- Komparátor
- Databáza
- Frontend

## Parsery

Parsery sa starajú o to, aby údaje z informaných zdrojov boli interpretované a uložené do databázy.

### Crepc Parser

Paser sa stará o parsovanie XML exportov z Crepc (Centrálny register evidencie publikanej innosti). Vnútorne sa skladá z parsera a loadera. Parse interpretuje XML, loader interpretované údaje pomocou ORM uloží do databázy.

### WoS Parser

Parser sa stará o parsovanie vstupu z webovej služby WoS do databázy. Vnútorne sa skladá z parsera a loadera. Parse interpretuje informácie získané z webovej služby, loader interpretované údaje pomocou ORM uloží do databázy.

### Scopus Parser

Parser sa stará o parsovanie vstupu z webovej služby Scopus do databázy. Vnútorne sa skladá z parsera a loadera. Parse interpretuje informácie získané z webovej služby, loader interpretované údaje pomocou ORM uloží do databázy.

## Komparátor

Komponent sa stará o deduplikáciu záznamov v databáze a ich jednoznané priradenie k už existujúcim údajom v databáze.

### J komparátor

Komponent vyberá nové údaje z databázy a vyberie kandidátov, ktorý by mohli by duplikátom nového záznamu. Následne skontaktuje C komparátor, ktorý porovná nový záznam k množine kandidátov.

### C komparátor

Je to samostaný komponent, ktorý ako vstup berie publikácie alebo autorov v špecifickom formáte a porovná ich. Výsledky potom vráti do príkazového riadku.

## Databáza

Táto množina komponentov sa stará o uloženie údajov a o to, aby boli rozumným spôsobom dostupné na alšie použitie.

### Postgre DB

SQL databázy s SQL dialektom Postgre. Uchováva údajový model a spravuje údaje.

### ORM

Objektovo relaný mapova mapuje údaje z databázy na predefinované objekty, ím zabezpeuje jednoduchý prístup k údajom pre Java Corupus systému.

## Frontend

Komponent zabezpeuje interakciu s používateom a taktiež sprístupuje implementované služby.

# Špecifikácia požiadaviek

## 1.1 Nefunkcionálne požiadavky

### 1.1.1 Párovanie publikácií

Algoritmy porovnávania a párovania musia ma dostatone malú asovú zložitos, aby mohli by nasadené v “real time” prostredí a aby mohli analýzy a párovania prebieha online. V rámci požiadaviek na produkt zákazník požaduje, aby bol systém schopný dohada informácie k špecifickému autorovi. Táto úloha však potrebuje prebehnú v konenom ase tak, aby sme nového používatea neodradili dlhotrvajúcimi výpotami a neskorými dobami odpovede. alej je potrebné výpotový výkon rozdeli medzi získavanie nových údajov a spracovávanie týchto nových údajov, preto musia algoritmy párovania beža o najrýchlejšie.

### 1.1.2 Import dát do databázy z exportu XML CREP a webových služieb WoS a Scopus

Algoritmus pre import dát z exportu z XML z Centrálneho registra evidencie publikanej innosti (CREP) a webových služieb WoS a Scopus musí by efektívny. Nie je urené žiadne obmedzenie pre rýchlos algoritmu. Za základe faktu, o aké veké dáta pri spracovaní ide, tieto algoritmy musia pracova tak, aby mali o najnižšie nároky na pamä.

# Funkcionálne požiadavky

## Párovanie publikácií

Je dôležité pracovníkom poskytnú o možno najspoahlivejšie informácie, a preto je dôležité, aby algoritmy párovania publikácií boli o možno najúinnejšie. Okrem toho z hadiska budúceho rozšírenia programu musia poskytova možnos rozšírenia o zisovanie alších relevantných informácií. alej musia podporova manuálne spárovanie publikácii, ak overený a zaregistrovaný výskumník, teda používate našej aplikácie oznaí dve publikácie ako totožné.

## Import dát do databázy z exportu XML CREP a webových služieb WoS a Scopus

Pre fungovanie párovania publikácii je potrebné pracova s nejakými dátami. Tieto dáta sa budú získava z Centrálneho registra evidencie publikanej innosti (CREP) v istých dlhodobých intervaloch vo forme XML. Dané XML sa bude spracováva na základe zadanej požiadavky do systému cez webové rozhranie. XML sa bude parsovat a importovat do existujúcej databázy. Pre úel perzistencie údajov sparsovaných údajov z XML CREP sa vytvorí PostgreSQL databáza (relaného charakteru), ktorá bude tieto údaje uchováva a poskytova pre funkciu Párovania publikácií. Tieto údaje budú obohacované údajmi získanými z webových rozhraní WoS a Scopus. Pri zadaní požiadavky do systému sa funkcia nespustí ihne. Bude sa spúša v noných hodinách, kedy je na server smerovaných o najmenej požiadaviek.

## Použitá databázová technológia

Pre perzistenciu spracovávaných údajov bude potrebné použitie relanej databázy, ktorá by bola dostatone výkonná, pretože sa jedná o spracovanie vekého množstva údajov. Jednou z požiadaviek je možnos pripojenia na danú databázu pomocou Java JDBC. Použitie dokumentovej databázy sa nejavilo vhodné z dôvodu potreby relaných vzahov medzi záznamami. Požiadavkou, ktorú musí spa uvedená databáza, tiež bola možnos vytvárania databázových funkcií (v jazyku SQL, PL/SQL alebo C), ktoré by mohli zjednoduši niektoré opakované databázové operácie.

## Webové rozhranie

Webové rozhranie systému musí ponúka jednoduchý spôsob zobrazovania nahromadených dát v databáze. Medzi hlavné entity patria autori, publikácie a vzahy medzi nimi. Používateské rozhranie musí ponúka prehadný spôsob filtrovania vekého potu dát na základe všetkých atribútov, ktorými dané entity disponujú.

## Webové služby

Systém bude okrem webového rozhrania ponúka tiež webové služby, cez ktoré bude možné využíva implementované algoritmy porovnávania a dopytovanie sa nad dátami v našej databáze.

## Zdroje znalostí

Systém bude využíva viaceré zdroje znalostí a kombinova údaje v nich nájdené, aby sa tak zabezpeila väšia presnos párovania publikácií a viac údajov o ohlasoch. Týmito zdrojmi budú CREP - Centrálny register evidencie publikanej innosti, Scopus, Web of Science a Google Scholar.

Prvý menovaný zdroj zabezpeuje údaje o publikáciách na Slovensku. Nie všetky vedecké inštitúcie využívajú tento systém. Zvyšné tri sú citané indexy, v ktorých budeme vyhadáva ohlasy na diela získané z CREP a na získavanie alších diel.

Preto musí systém vedie komunikova s týmito službami, musí vedie spracováva výstupy získané z týchto služieb a mapova tieto výstupy na seba. Pod mapovaním rozumieme jednak spracovanie údajov do spoloného formátu, a potom aj párovanie publikácií.

# Algoritmy párovania

## Algoritmická as

Distance algoritmy:

- Damerau – Levenshtein
- Jaro - Winkler

Algoritmy zúženia výberu kandidátov:

- Blokový prístup
- Sorted neighborhood method [Hernandez]

Na základe testovania bol vybraný algoritmus Jaro-Winkler z dôvodov:

- Damerau-Levenstein dáva príliš rovnaké výsledky pre rôzne párovania
- KJ algoritmus navrhnutý Danielom Kom má príliš vekú zložitos
- Jaro-Winkler dáva uspokojivé výsledky pri nízkej zložitosti

Pozn.: Implementovaný algoritmus má hadanie transpozícii implementované jednoduchým štýlom ako porovnávanie posunu k zhode predchádzajúceho písmenka a aktuálneho písmenka.

## Párovanie ako podsystém v architektúre systému

Algoritmy párovania budú spúšané ako samostatný podsystém.

Jeho spustenie bude podmienené pridaním nových údajov do databázy Postgre pomocou crawlerov, spustenie nastane po uritom množstve vložených záznamov. Nové záznamy budú idetintifikované poda stpca, kde bude naznaené, že sú nové a ešte neprešli algoritmami párovania

Bude operova nad tabukami záznamov, napárované zmeny budú operáciami join a merge zapisované do tabuky záznamov, priom sa vytvorí záznam v tabuke zmien a v tabuke porovnaných záznamov.

Samotný podsystém pozostáva z dvoch astí:

- algoritmy porovnávania publikácií - implementované v C++
- algoritmus výberu kandidátov, komunikácie s databázou a plánova - implementované v Jave

Rozdelenie na dve asti bolo rozhodnuté na základe potreby jednotného prístupu do databázy a s tým spojená menšia réžia pri zmenách v dátovom modeli. alej bolo dôvodom spúšanie a plánovanie párovacích úloh, aby toto mohlo by realizované aj z frontendu bez väších problémov.

Algoritmus výberu kandidátov stavia na metóde Sorted Neighbourhood, implementované bude zoraovanie publikácií poda rôznych kritérií, momentálne funguje poda názvu publikácie.

Pripojenie do databázy je realizované využitím existujúceho ORM mapovaa využívajúceho Hibernate.

Komunikáciu s algoritmami porovnávania publikácií je realizované spustením nového procesu pomocou ProcessBuilder, priom publikácie sú podávané ako formátované reazce v argumentoch volania .exe.

## Uenie prahových hodnôt pre akceptáciu zhody

Akceptácia pre mená:

1. Výber kandidátov na porovnanie z databázy
2. Pre každý záznam spusti porovnávací algoritmus (as porovnávajúcu mená)
3. Vyfiltrova tie záznamy, ktoré majú podobnos nad 85% a zárove nie 100%
4. Prezentova tieto záznamy spolu s podobnosou testerovi
5. Uloži testerove ohodnotenie, i ide o zhodu alebo nie
6. Stanovenie hraníc nasledovne:
    a. maximálna podobnos, ktorú dosiahli záznamy, ktoré neboli zhodné
    b. minimálna podobnos, ktorú dosiahli záznamy, ktoré boli zhodné
7. Na základe týchto hodnôt spravi kontrolný test - program vypíše podobnosti alších záznamov a to, i sa rozhodol pre zhodu alebo odlišnos záznamov alebo pre pravdepodobnú podobnos, ktorú musí správca potvrdi

# C komparátor

# Úvod

C komparátor je samostatná súiastka projektu Research Rank. Úlohou tejto súiastky je ponúknu služby súvisiace s párovaním autorov a publikácií. Vaka týmto službám je možné deduplikova databázu a vytvori istú množinu údajov.

C komparátor je navrhnutý tak, aby fungoval s jeho párovou súiastkou v Jave. Úlohou tejto Java párovej súiastky je spoji sa s databázou a vytvori množinu kandidátov na porovnanie, vstup do C komparátora. Komunikácie medzi týmito dvoma súiastkami prebieha pomocou príkazového riadku (CMD). Príkazový riadok ponúka viacej výhod pre ktoré bol zvolený ako vhodná forma komunikácie.

Hlavnou výhodou komunikácie cez CMD je jednoduchos implementácie. alej tento spôsob umožuje silnú kontrolu nad potom spustených inštancií C komparátora a taktiež v prípade chyby v C komparátore nie je ovplyvnený zvyšok systému Research Rank, o prispieva k stabilite systému.

# Organizácia podradených dokumentov

V sekcií je spomenutý obsah dokumentácie poda témy ako aj struný popis ich obsahu.

## Špecifikácia požiadaviek C komparátora

Dokument špecifikácia požiadaviek obsahuje požiadavky na softvér konzistentné so štandardom IEEE 830 poda zdrojového dokumentu zverejneného v r.1998.

Je to struný popis implementovanej funkcionality, pointy programu, ponúkaných funkcií a požiadaviek na ne, popis rozhraní z konceptuálneho hadiska, použitého algoritmu a pod. Taktiež uruje audienciu, ktorej je dokument urený.

## Návrh C komparátora

Konceptuálny dokument, ktorý objasuje koncept systému ako aj programovacie paradigmy ktoré sa snaží dodržova

## Implementácia C komparátora

Popis vnútornej architektúry C komparátora. Vysvetlenie konkrétnych použitých riešení a popis jednotlivých súiastiek.

## Špecifikácia rozhrania C komparátora

Dokument obsahuje plný popis rozhraní a ich súastí, vstup, výstup ako aj ich organizáciu a podporované formáty vstupných údajov pre jednotlivé služby.

## Služby C komparátora

Obsahuje zoznam služieb a ich popis (konkrétne implementované služby).

## Doplujúce dokumenty

Obsahuje alšie dokumenty, ktoré sú zamerané na podporu práce s C komparátorom.

### Metodika: Zmena podporovaných rozhraní C komparátora

Opisuje kroky na odstránenie, pridanie alebo zmenenie podporovaných rozhraní.

### Metodika: Zmena porovnávacieho algoritmu C komparátora

Obsahuje kroky na zmenu alebo obohatenia vnútorného mechanizmu na porovnávanie.

### Metodika: Pridanie služby do C komparátora

Vyššia metodika obsahujúce kroky potrebné na to, aby sa dal komparátor rozšíri o alšiu službu.

# Špecifikácia požiadaviek C komparátora

## Úvod

Dokument obsahuje požiadavky pre C komparátor. C komparátor je prostriedok porovnávania entít napísaný v jazyku C++.

### Význam

C komparátor je súiastka, ktorá má slúži na urenie koeficientu podobnosti medzi entitou a množinou kandidátov. Pomocou tejto súiastky je možné oisti databázu publikácií a autorov od duplicitných záznamov.

Dokument je urený pre používateov komparátora ako aj pre programátorov, ktorý na om pracujú. Používateom pomôže pochopi obsah tejto verzie C komparátora a programátorom objasní tento dokument ako kontext v ktorom bol komparátor vyvinutý, tak aj akcie a rozhranie z konceptuálneho hadiska.

### Rozsah

Výsledok bude program napísaný v jazyku C++, ktorý ponúka služby prostredníctvom CMD. Služby budú porovnáva entity a vraca koeficienty podobnosti. Program nebude priamo pristupova do databázy. Program bude naprojektovaný tak, aby mohol prijíma príkazy z CMD, ím sa oblas jeho zamerania zúži isto na kvalitu a systém porovnania entít.

### Definícia termínov a skratiek

RR - Research Rank - program, súasou ktorého je C komparátor.

CMD - Štandardný príkazový riadok

C komparátor - Softvérový produkt vytvorený poda tohto dokumentu

Java komparátor - párová súiastka C komparátora, ktorá C komparátor riadi prostrednísvom CMD

TP - tímový projekt

minuloroný tím - TP tím pani doktorky Andrejíkovej 2012/2013.

### Referencie

### Prehad

Dokument je organizovaný nasledovne. Úvod obsahuje všeobecné informácie o produkte.

Všeobecné požiadavky obsahuje popis prostredia, do ktorého je C komparátor zasadený, opis všeobecných limitácií a požiadaviek a taktiež opis používateov.

Špecifiké požiadavky obsahujú rozpísané špecifické požiadavky poda ponúkanej služby. Opis služby obsahuje koncept vstupu, výstupu, akcie, ktoré sa vykonajú a vlastnosti tej ktorej služby, špecifiké limitácie služby.

Doplujúce informácie obsahujú príklady vstupu a výstupu ako aj príklad riešeného problému.

## Všeobecné požiadavky

Sekcia obsahuje všeobecné požiadavky, spomenie predchodcu a víziu C komparátora. Uvedie limitácie a požiadavky na používatea, predpoklady a taktiež spomenie funkcionalitu, ktorá je plánovaná až do alšej verzie systému.

### Perspektíva produktu

Produkt je snahou sofistikova a vylepši poprípade nahradi existujúci komparátor minuloroného tímu na TP. Ten porovnával na základe N-gramov. Produkt bude používa viac stupové porovnávanie, ím sa zlepšia výsledky procesu párovania v RR projekte.Párovanie publikácií je stále otvorená otázka a je predmetom mnohých publikácií. Neexistuje univerzálna procedúra, ktorá by dokázala odhali ubovonú duplicitu v databáze autorov a publikácii. Produkt bude snahou vytvori o možno najefektívnejšiu metódu na párovanie publikácií slovenských autorov.

### Funkcie produktu

Produkt obsahuje funkciu na párovanie publikácií. Táto funkcia je vo forme služby, ktorú ponúka v CMD. alej produkt obsahuje funkciu na párovanie autorov ktorá je taktiež vo forme CMD služby. Tretia funkcia je identická s druhou, iba že na párovanie používa algoritmus zo starého komparátora. Štvrtá funkcia je identická s druhou, ibaže ako vstup v CMD má súbor, kde sa nachádzajú ostatné údaje potrebné pre analýzu.

### Charakteristika používatea

Komparátor nie je priamo vo styku s používateom. Používate môže komparátor spusti, avšak interakcie a informácie, ktoré prúdia dnu a von sú riadené pomocou Java komparátora.

### Limitácie

Kedže produkt bude fungova na jednom stroji, tak je produkt limitovaný iba systémovými prostriedkami, ktoré mu dá k dispozícií operaný systém

daného stroja.

### Predpoklady a závislosti

Predpokladá sa, že používate služby vie preo volá C komparátor a taktiež že vie o má robi s výstupom. alej sa predpokladá, že používate služby má oprávnenie na volanie programov pomocou CMD.

Produkt nemá žiadne priame závislosti.

### Rozdelenie požiadaviek

V budúcich verzia systému sa poíta s tým, že porovnanie bude fungova paralelne a alej to, že formát a vstupné údaje testu budú definované používateom C komparátora.

## Špecifické požiadavky

Sekcia obsahuje špecifické požiadavky na produkt. Požiadavky obsahujú vstup, výstup, akcie a vlastnosti. Sekcia je organizovaná kvôli o možno najlepšej itatenosti poda služieb.

### Služba íslo 0, porovnávanie publikácií

Významom tejto služby je poskytnú koeficient podobnosti pre publikáciu porovnanú s množinou iných publikácií.

#### Vstup

Meno: Záznam o publikácií
Popis: Vstup obsahuje íslo služby, poet vstupných publikácií a potom záznamy o publikácií.  Záznam o publikácií obsahuje všetky relevantné položky, ktoré sú k dispozícií.
Zdroj: Java komparátor, používate služby služby, databáza
Formát: Vstup obsahuje íslo služby, poet vstupných publikácií a potom entice ktoré reprezentujú poet rôznych atribútov pre každú publikáciu.

#### Výstup

Meno: Záznam o porovnaní
Popis: Vektor údajov obsahujúci poet údajov a potom koeficient podobnosti pre pred jednotlivé pari. Koeficient je medzi 0 a 1.
Miesto urenia: CMD, používate služby.
Formát: Vektor obsahuje poet položiek a potom jednotlivé položky.

#### Akcie

- Prijatie vstupu z CMD
- Preloženie vstupu do vnútornej formy
- Porovnanie publikácie poda položiek
- Vypísanie výsledného vektora

#### Vlastnosti

- Rýchlos
- Spoahlivos
- Presnos

### Služba íslo 1, porovnanie autorov

Význam tejto služby je poskytnú podobnosti pre autora porovnaním s množinou autorov.

#### Vstup

Meno: Záznam o autorovi
Popis: Vstup obsahuje íslo služby, poet vstupných autorov a potom záznamy o autoroch.  Záznam o autorovi obsahuje spoluautorov a pracovisko.
Zdroj: Java komparátor, používate služby služby, databáza
Formát: Vstup obsahuje íslo služby, poet vstupných autorov a potom ntice ktoré reprezentujú poet rôznych atribútov pre každého autora.

#### Výstup

Meno: Záznam o porovnaní
Popis: Vektor údajov obsahujúci poet údajov a potom koeficient podobnosti pre pred jednotlivé pari. Koeficient je medzi 0 a 1.
Miesto urenia: CMD, používate služby.
Formát: Vektor obsahuje poet položiek a potom jednotlivé položky.

#### Akcie

- Prijatie vstupu z CMD
- Preloženie vstupu do vnútornej formy
- Porovnanie publikácie poda položiek
- Vypísanie výsledného vektora

#### Vlastnosti

- Rýchlos
- Spoahlivos
- Presnos

### Služba íslo 2, porovnanie autorov starým algoritmom

Význam tejto služby je poskytnú spätnú väzbu o starom algoritme na párovanie.

#### Vstup

Meno: Záznam o autorovi
Popis: Vstup obsahuje íslo služby, poet vstupných autorov a potom záznamy o autoroch.  Záznam o autorovi obsahuje spoluautorov a pracovisko.
Zdroj: Java komparátor, používate služby služby, databáza
Formát: Vstup obsahuje íslo služby, poet vstupných autorov a potom ntice ktoré reprezentujú poet rôznych atribútov pre každého autora.

#### Výstup

Meno: Záznam o porovnaní
Popis: Vektor údajov obsahujúci poet údajov a potom koeficient podobnosti pre pred jednotlivé pari. Koeficient je medzi 0 a 1.
Miesto urenia: CMD, používate služby.
Formát: Vektor obsahuje poet položiek a potom jednotlivé položky.

#### Akcie

- Prijatie vstupu z CMD
- Preloženie vstupu do vnútornej formy
- Porovnanie publikácie poda položiek použijúc starý algoritmus
- Vypísanie výsledného vektora

#### Vlastnosti

- Rýchlos
- Spoahlivos
- Presnos

### Služba íslo 3, porovnanie autorov so vstupom zo súbora

Význam tejto služby je obís limitáciu služby 2 a to maximálnu džku vstupu z CMD.

#### Vstup

Meno: CMD vstup do služby 3
Popis: Vtupný súbor a íslo služby.
Zdroj: Java komparátor
Formát: Obsahuje íslo služby a potom meno súbora aj s cestou, kde je ostatov vstupu

Meno: Záznam o autorovi v súbore
Popis: Obsahuje poet vstupných autorov a potom záznamy o autoroch.  Záznam o autorovi obsahuje spoluautorov a pracovisko.
Zdroj: Java komparátor, používate služby služby, databáza
Formát: Obsahuje poet vstupných autorov a potom ntice ktoré reprezentujú poet rôznych atribútov pre každého autora.

#### Výstup

Meno: Záznam o porovnaní
Popis: Vektor údajov obsahujúci poet údajov a potom koeficient podobnosti pre pred jednotlivé pari. Koeficient je medzi 0 a 1.
Miesto urenia: CMD, používate služby.
Formát: Vektor obsahuje poet položiek a potom jednotlivé položky.

#### Akcie

- Prijatie vstupu z CMD
- Otvorenie vstupného súbora
- Preloženie vstupu do vnútornej formy
- Porovnanie publikácie poda položiek
- Vypísanie výsledného vektora

#### Vlastnosti

- Rýchlos
- Spoahlivos
- Presnos

## Doplujúce informácie

Sekcia obsahuje príklady vstupu a výstupu a opis problémov, ktoré C komparátor rieši.

### Príklady vstupu a výstupu

Príklad na vstup dvoch publikácií do komparátora

Vstup do komparatora:

2 "\"Team_member_x\" 1 \"Research rank\" 0 0 0 0 1 \"2014\" 0 0 0 0 0"  "\"Team_member_x\" 1 \"Research prank\" 0 0 0 0 1 \"2014\" 0 0 0 0 0"

Vystup komparátora:

1 1,0

### Príklad riešeného problému

C komparátor dostane dvoch autorov, ktorých mená su podobné ale nie zhodne, spoluautori sú zhodný a pracovisko nie je k dispozícií.
Úloha: Urite podobnos týchto dvoch autorov objektívne.

# Návrh C komparátora

Dokument obsahuje popis služieb, ktoré majú by v C komparátora implementované a konkrétny workflow.

## Služba .0 - porovnávanie publikácií

Porovnávanie publikácií sa uskutouje na základe niekokých atribútov a to:

- Prvý autor
- Množina názvov publikácie - prvych 80 znakov
- Množina kúových slov
- Množina spoluautorov
- Abstrakt
- Rok publikácie
- Meno zdroja záznamu publikácie
- ISBN - iba ak zdroj publikacie je zhodny
- ISSN - iba ak zdroj publikacie je zhodny
- pracoviská prvéhou autora
- vydavatestvá, ktoré publikáciu vydali

Výsledky individuálnych porovnaní budú zaznamenané a nakoniec vyhodnotené kosínusovou vzdialenosou. Údaje sú rozdelené na primárne informácie, sekundárne a meta informácie. Primárne informácie sú prvý autor, titul publikácie a kúové slová. Sekundárne informácie sú spolu autor, abstrakt, rok, zdroj publikácie, ISBN, ISSN. Meta údaje sú pracoviská prvého autora a vydavatestvá. Tieto tri sú vyhodnotené individuálne kosínusovou vzdialenosou a potom výsledky sú spojene s váhou 0,75, 0,2 a 0,05.

### Workflow

1. Naítanie množiny publikácií z CMD; každá publikácia je v jednom argumente CMD
2. Interpretovanie publikácie do vnútornej formy; vnútornej reprezentácie publikácie

3. Porovnanie atribútov pre všetky dvojice, ktoré tvorí prvá publikácia v argumentoch s 2 až n-tou publikáciou :
    a. Autori a spoluautori pomocou Jaro-Winklerovho algoritmu (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    b. Abstrakt a titul pomocou rutiny na porovnávanie celých textov (hadanie exact matchu v ztokenizovanom texte)
    c. Ostatok pomocou priameho porovnania (exact match)
4. Vyhodnotenie - spojenie výsledkov troch priorít údajov.
5. Vrátenie výsledku vo forme koeficientov podobnosti (float <0,1> vyjadrujúci na koko % sa porovnávané publikácie podobajú) pre každú dvojicu.

## Služba .1 - porovnanie autorov

Porovnávanie autorov na základe atribútov:

- Meno autora
- Množina organizácií kde patrí
- Množina spoluautorov s ktorými publikuje

Porovnáva sa tak, že ak sa nenatrafí na dostatonú podobnos (aspo 0,8) v mene autora, tak sa vracia 0. Inak sa pristúpi k porovnaniu spoluautorov a publikácií. Výsledok je vyhodnotený kosínusovou vzdialenosou.

### Workflow

1. Naítanie množiny autorov z CMD; každý záznam je na 1 argumente v CMD
2. Interpretovanie autorov do vnútornej formy; vnútornej reprezentácie autora
3. Vykonanie porovnania dvojíc autorov, ktoré tvorí prvý autor v argumentoch s 2 až n-tým autorom.
    a. Porovnanie mena autorov. Ak dosahujú koeficient podobnosti >= 0,8, tak sa pokrauje alej, inak sa vráti 0. Porovnávací algoritmus: Jaro-Winkler (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    b. Porovnanie organizácií - koko organizácií množiny A sa nachádza v množine B; hadá sa exact match
    c. Porovnanie spoluautorov - koko spoluautorov množiny A sa nachádza v množine B; Porovnávací algoritmus: Jaro-Winkler (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    d. Spojenie individuálnych výsledkov do jedného pomocou kosínusovej vzdialenosti a váhou jednotlivých súasti 0,33,0,33,0,33.
4. Vyhodnotenie - spojenie výsledkov
5. Vrátenie výsledku vo forme koeficientov podobnosti (float <0,1> vyjadrujúci na koko % sa porovnávané publikácie podobajú) pre každú dvojicu.

## Služba .2 - porovnanie autorov metódou TP1213

Porovnávanie autorov na základe atribútov:

- Meno autora
- Množina organizácií kde patrí
- Množina spoluautorov s ktorými publikuje

Porovnáva sa tak, že ak sa nenatrafí na dostatonú podobnos (aspo 0,8) v mene autora, tak sa vracia 0. Inak sa pristúpi k porovnaniu spoluautorov a publikácií. Výsledok je vyhodnotený kosínusovou vzdialenosou.

### Workflow

1. Naítanie množiny autorov z CMD; každý záznam je na 1 argumente v CMD
2. Interpretovanie autorov do vnútornej formy; vnútornej reprezentácie autora
3. Vykonanie porovnania dvojíc autorov, ktoré tvorí prvý autor v argumentoch s 2 až n-tým autorom.
    a. Porovnanie mena autorov. Ak dosahujú koeficient podobnosti >= 0,8, tak sa pokrauje alej, inak sa vráti 0. Porovnávací algoritmus: ngram_last_year (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    b. Porovnanie organizácií - koko organizácií množiny A sa nachádza v množine B; hadá sa exact match
    c. Porovnanie spoluautorov - koko spoluautorov množiny A sa nachádza v množine B; Porovnávací algoritmus: ngram_last_year  (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    d. Spojenie individuálnych výsledkov do jedného pomocou kosínusovej vzdialenosti a váhou jednotlivých súasti 0,33,0,33,0,33.
4. Vyhodnotenie - spojenie výsledkov
5. Vrátenie výsledku vo forme koeficientov podobnosti (float <0,1> vyjadrujúci na koko % sa porovnávané publikácie podobajú) pre každú dvojicu

## Služba .3 - porovnanie autorov naítaných zo súbora

Porovnávanie autorov na základe atribútov:

- Meno autora
- Množina organizácií kde patrí
- Množina spoluautorov s ktorými publikuje

Porovnáva sa tak, že ak sa nenatrafí na dostatonú podobnos (aspo 0,8) v mene autora, tak sa vracia 0. Inak sa pristúpi k porovnaniu spoluautorov a publikácií. Výsledok je vyhodnotený kosínusovou vzdialenosou.

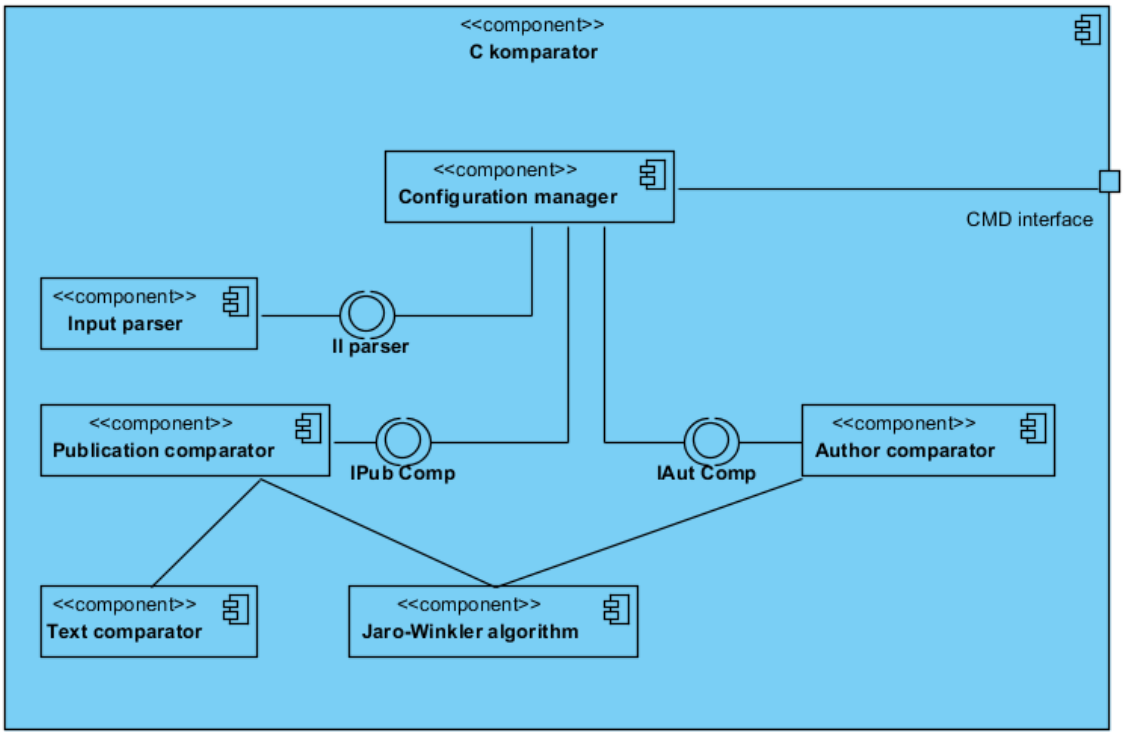
### Workflow

1. Naítanie mena vstupného súbora z CMD
2. Otvorenie a preítanie vstupného súbora
3. Interpretovanie obsahu súbora do vnútornej formy; vnútornej reprezentácie autora
4. Vykonanie porovnania dvojíc autorov, ktoré tvorí prvý autor v argumentoch s 2 až n-tým autorom.
    a. Porovnanie mena autorov. Ak dosahujú koeficient podobnosti >= 0,8, tak sa pokrauje alej, inak sa vráti 0. Porovnávací algoritmus: Jaro-Winkler (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    b. Porovnanie organizácií - koko organizácií množiny A sa nachádza v množine B; hadá sa exact match
    c. Porovnanie spoluautorov - koko spoluautorov množiny A sa nachádza v množine B; Porovnávací algoritmus: Jaro-Winkler (meno je rozkúskované na meno, priezvisko ..., a tieto sú porovnané individuálne)
    d. Spojenie individuálnych výsledkov do jedného pomocou kosínusovej vzdialenosti a váhou jednotlivých súasti 0,33,0,33,0,33.
5. Vyhodnotenie - spojenie výsledkov
6. Vrátenie výsledku vo forme koeficientov podobnosti (float <0,1> vyjadrujúci na koko % sa porovnávané publikácie podobajú) pre každú dvojicu do CMD.

# Implementácia C komparátora

## Popis implementácie

C komparátor má o možno najspoahlivejšie uri podobnos publikácií a autorov. Na to má v sebe implementovaných niekoko komponentov. S vonkajšieho sveta ho volajú aplikácie na komparáciu pomocou príkazového riadku. V rámci architektúry programu by mal interagova iba s Java komparátorom, ktorý zabezpeuje výber kandidátov a spúštanie C komparátora.
Komparátor bol implementovaný vo Visual Studiu 2012 v jazyku C++. Neobsahuje žiadne “third party” komponenty.
Konkrétna implementácia C komparátora bola urená na základe porovnania niekokých metód párovania, z nich bol vybraný najlepší algoritmus na základe testov nad menami autorov zo slovenského prostredia. Architektúra bola koncipovaná predovšetkým s prihladnitím na možnos neskoršej zmeny komponentov ako aj rozšírenia funkcionality.




Obrázok 7: Vnútorná štruktúra C komparátora.
Komparátor obsahuje 6 základných komponentov:

### Manažér konfigurácie

Manažér konfigurácie verifikuje správnos vstupu, manažuje volanie parsera na vstup, volá pomocou externej metódy obsah súbora, volanie samotného komparátora a nakoniec výstup. Priamo v manažérovi konfigurácie sa urí na základe prvého parametra predaného z CMD, o akú službu ide. Je tam implementovaný rozhodovací blok, ktorý potom na základe toho zavolá správny parser. V prípade služieb, ktoré erpajú informácie aj zo súbora, zavolá sa z knižniná funkcia, ktorá otvorí súbor, preíta jeho obsah do reazca a znova ho zavrie.

Modul: "main.cpp"
Metóda: "int main(int argc, char *argv[])"

### Parser vstupu

Parser vstupu dostane verifikovaný vstup od manažéra konfigurácie a transformuje odtlaky vstupných publikácií a autorov do reprezentácie v pamäi kompatibilnej s komparátorom publikácii i autorov. Existujú tri rôzne implementácie vstupného parsera. Prvou je implementácia urená na parsovanie záznamu publikácie, druhý je urený na spracovanie autora a tretí je urený na spracovanie autora s tým, že sa íta záznam zo súboru. Modifikácia tohto tretieho parsera spoíva v tom, že ako vstup je obsah súbora a pred tým ako zane spracovávanie ešte pretransformuje vstup do formátu totožného s CMD vstupom.

#### Parser publikácie z CMD

Modul: "main.cpp"
Metóda: "class simple_list<class publication_repository *> *parse_arguments(int argc, char *argv[])"

#### Parser autorov z CMD

Modul: "main.cpp"
Metóda: "class simple_list<class author_repository *> *parse_arguments_to_authors(int argc, char *argv[])"

#### Parser autorov zo súboru

Modul: "main.cpp"
Metóda: "class simple_list<class author_repository *> *parse_arguments_to_authors(wchar_t *file_content)"

### Komparátor publikácií

Komparátor publikácií dostane od manažéra konfigurácie záznamy publikácií, ktoré má porovna. Následne aplykuje na jednotlivé polia publikácie porovnanie. Poda typu ide o porovnania pomocou Jaro-Winklerovho algoritmu na provnávanie, porovnávanie textov (ako sú tituly, abstrakty a pod.) a nakoniec porovnania vyžadujúce presnú zhodu atribútov. Výsledok je dopoítaný pomocou kosinusovej vzdialenosti a váh jednotlivých atribútov.

Modul: "kj_publication_comparator.h"
Trieda: "publication_handler"
Metóda: "double publication_comparator(class publication_repository *pub_one, class publication_repository *pub_two)"

### Komparátor Jaro-Winkler

Komponent Jaro-Winkler algoritmus obsahuje implementáciu rovnomenného algoritmu.

Modul: "Jaro-Winkler.cpp"
Metóda: "double jw_compute(wchar_t *s1, wchar_t *s2)"

### Textový komparátor

Textový komparátor hadá zhodu sekvencií tokenov v rámci stokenizovaného textu.

Modul: "kj_text_komparator.h"
Trieda: "text_comparator"
Metóda: "float compare_texts(wchar_t *text1, wchar_t *text2)"

### Komparátor autorov

Komparátor autorov dostane z manažéra konfigurácie informácie o porovnávaných autoroch a to meno, pracoviská a co-autoroch. Na základe tejto informácie, cosínusovej vzdialenosti a jaro-winkler algoritmu odhadne podobnos dvoch autorov.

Modul: "kj_author_comparator.h"
Trieda: "author_comparator"
Metóda nová: "float compare_authors(author_repository *first, author_repository *candidate)"
Metóda TP1213: "float compare_authors_old(author_repository *first, author_repository *candidate)"

## Pohad z hadiska modulov

Sekcia obsahuje popis jednotlivých modulov

main.cpp: Obsahuje funkcionalitu konfiguraného manažéra a parsera vstupu. Okrem toho obsahuje implementáciu primitívnych porovnávacích metód pre typy bool a wchar_t.

Jaro-Winkler.cpp: Obsahuje implementaciu Jaro-Winklerovho algoritmu pre char a wchar_t.

ngram_last_year.cpp: Obsahuje implementáciu porovnávacích metód použitých v TP 2012/2013 pre wchar_t.

Jaro-Winkler.h: Hlavikový súbor na deklarácie funkcií modulu Jaro-Winkler.cpp

main_header.h: Hlavikový súbor obsahuje deklarácie primitívnych porovnávacích metód pre bool a wchar_t definovaných v main.cpp

ngram_last_year.h: Hlavikový súbor obsahujúci deklarácie funkcií modulu ngram_last_year.cpp

kj_text_comparator.h: Obsahuje definíciu triedy na porovnávanie celých textov

kj_repository_publication.h: Obsahuje definíciu triedy na uchovávanie záznamu publikácie

kj_repository_author.h: Obsahuje definíciu triedy na uchovávanie záznamu autora

kj_publication_comparator.h: Obsahuje definíciu triedy, ktorá ponúka porovnávacie služby pre publikácie

kj_author_comparator: Obsahuje definíciu triedy, ktorá ponúka porovnávacie služby pre autorov

tools_error_handler.h: Obsahuje implementáciu triedy, ktorá má na starosti automaticky logova chyby.

tools_file_IO.h: Obsahuje implementáciu tried, ktoré ponúkajú služby log a služby súvisiace s IO súboru.

tools_math.h: Ponúka implementáciu niektorých matematických funkcií, predovšetkým cosínusovej vzdialenosti

tools_simple_list.h: Ponúka triedu ktorá implementuje jednoduchý zoznam s celou funkcionalitou.

tools_string_tools.h: Ponúka triedy, ktoré implementujú rutiny súvisiace so spracovaním reazcov vo formáte char a wchar_t

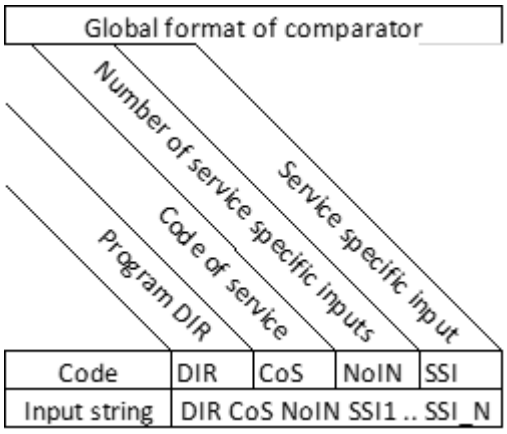
# Špecifikácia rozhrania C komparátora

## Špecifikácia rozhrania C komparátora

### Funkcia

Komponent zabezpeuje službu, ktorá urí, i daná publikácia alebo autor ma v príslušnej množine publikácií i autorov duplikát.

### Všeobecný vstup

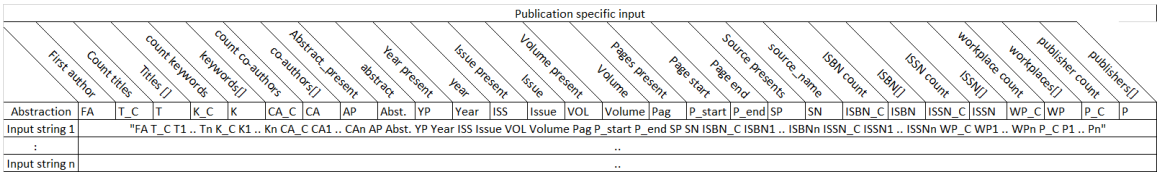



Obrázok 1: Všeobecný popis vstupu z príkazového riadku.

Kde „Program DIR“ je atribút od operaného systému, ktorý používate nenastavuje v príkazovom riadku. Výnimka je pri službách, kde vstupujú údaje zo súboru, v tomto prípade je uvedené len íslo služby a meno súboru.

### Vstup do komparátora publikácií - služba íslo 0

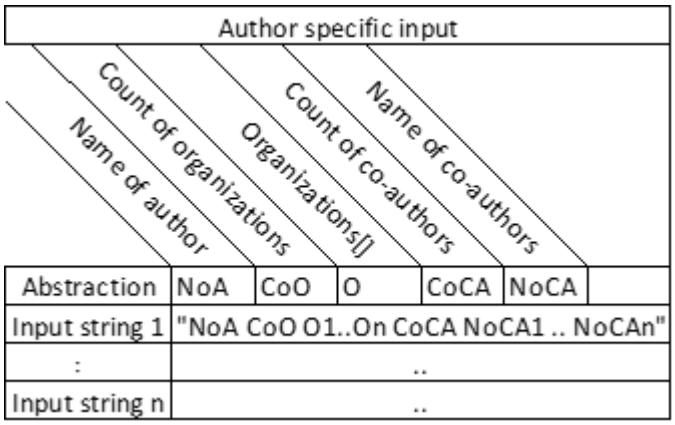
Vstupom komponentu sú argumenty z príkazového riadku. Prvý argument je poet publikácií, ktoré predávame komponentu. Druhý argument je publikácia, ktorej duplikáty hadáme. V ostatných argumentoch predávame komparátoru množinu publikácií, kde má hlada duplikáty. Formát publikácie je definovaný nasledujúcou tabukou:




Obrázok 2: Formát vstupu do komparátora publikácií.
Vzhadom na množstvo možných informácií je vstupný odtlaok publikácie znane zložitý, o ilustruje tabuka.
V rámci vstupného reazca treba informácie obali do úvodzoviek (ilustrujúci príklad nižšie). ísla vstupujú do komparátora bez úvodzoviek.

## Vstup do komparátora autorov - služby íslo 1 a 2

Služba slúži na porovnávanie autorov.




Obrázok 3: Formát vstupu do komparátora autorov

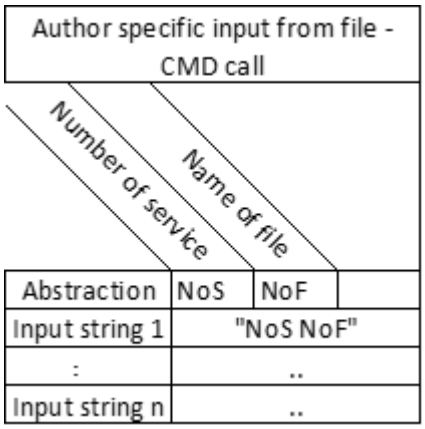
Všetky vstupy by mali by uzavreté v úvodzovkách „atribút“ aby sa povolili viac slovné zápisy.

Do C komparátora vstupujú množiny autorov na porovnanie. Formát je:

Count „Auth.1“ .. „Auth.n“

Count popisuje poet autorov v danej množine. Auth.1 – n popisujú samotných autorov.

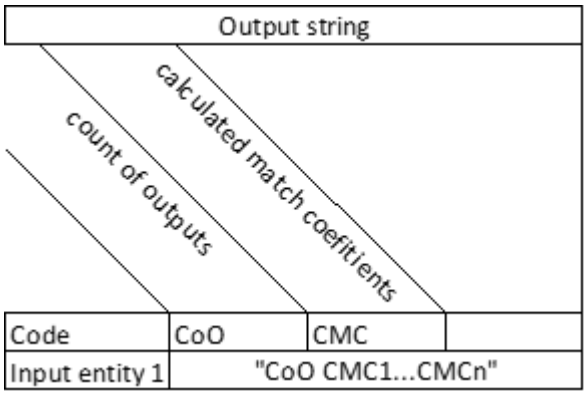
## Vstup do komparátora autorov s údajmi naítanými zo súboru - služba 3




Obrázok 4: Vstup z CMD volajúci službu na naítanie autorov z uvedeného súboru.

Vstupom je íslo servisu a meno súboru. V súbore sú údaje o autoroch tak, ako je to v prípade porovnávania autorov ako takých, resp. ten istý ako keby sa vstupovalo do porovnávania priamo z CMD s tým, že íslo servisu sa už v súbore neuvádza, keže bolo uvedené v samotnom volaní tejto služby cez CMD.

## Výstup




Obrázok 4: Výstup z komparátora pri oboch službách.

Výstup komparátora je vektor ísiel. Prvé íslo uruje poet výstupov a ostané ísla urujú koeficient zhody <0,1> vo float ísle. Tento koeficient uruje nakoko je daná položka zhodná so vstupom.

Príklad: Vo vstupnej množine je 3. publikácia duplikátom.

Výstup potom bude: “1 3”.

## Priklad:

Example of input

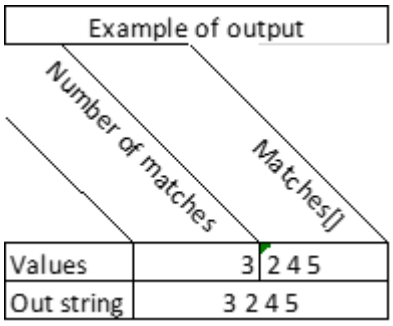
| | First author | Count titles | Titles [] | count keywords | keywords[] | count co-authors | co-authors[] | Abstract_present | abstract | Year present | year | Issue present | Issue | Volume present | Volume | Pages present | Page start | Page end | Source presents | source_name | ISBN count | ISBN[] | ISSN count | ISSN[] | workplace count | workplaces[] | publisher count | publishers[] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Attributes | "Jaro" | 1 | "ABC" | 0 | | 1 | "Juro" | 0 | | 1 | "2000" | 0 | | 0 | | 0 | | | 0 | | 0 | | 0 | | 1 | "Nowhere" | 0 | |
| String | "\"Jaro\" 1 \"ABC\" 0 1 \"Juro\" 0 1 \"2000\" 0 0 0 0 0 0 1 \"Nowhere\" 0" | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| In service | 0 1 "\"Jaro\" 1 \"ABC\" 0 1 \"Juro\" 0 1 \"2000\" 0 0 0 0 0 0 1 \"Nowhere\" 0" | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Notes: | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| - 1st number is code of service | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| - 2nd number is number of publications on input | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Obrázok 5: Príklad vstupu do komparátora.

as „In service“ tabuky špecifikuje CMD vstup pre jednu publikáciu servisu komparácie publikácií. Dôležité je, aby každá publikácia bola obalená v úvodzovkách, aby ju CMD rozparsoval správne do jednotlivých argumentov vstupu.

alej je dôležité aby ísla špecifikujúce poet entít vstupu ako je autor, rok a pod boli bez úvodzoviek. Nakoniec asti vstupu musia by v úvodzovkách a to tak, že pri príkazovom riadku je zrušený význam úvodzoviek pomocou „\“.

Ak v rámci nejakého reazca atribútu vstupu je znak „\“, treba ho vloži do CMD ako „\\\\“. Vaka tomu do programu prejde len ako „\\“ o program interpretuje ako znak „\“ na vstupe bez špeciálneho významu. V prípade znaku „“ v rámci reazca teda na vstupe bude taktiež „\\\““.




Obrázok 6: Príklad výstupu z komparátora.

as „Matches“ sú indexy vstupných údajov, ktoré sú zhodné. Oddelené sú medzerou. Napríklad íslo 3 znamená, že sa zhoda našla v troch publikáciách. íslo 2 znamená, že 1. vstupná publikácia bola zhodná s 3. vstupnou publikáciou. íslo 4 znamená, že 1. vstupná publikácia bola zhodná so 5. vstupnou publikáciou a podobne.

Publikácia: Research rank
Autor: Team_member_x
Rok: 2014

Publikácia: Research prank
Autor: Team_member_x
Rok: 2014

Vstup do komparatora:

0 2 "\"Team_member_x\" 1 \"Research rank\" 0 0 0 0 1 \"2014\" 0 0 0 0 0 0 0 0"  "\"Team_member_x\" 1 \"Research prank\" 0 0 0 0 0 0 0 1 \"2014\" 0 0 0 0 0"

Vystup komparátora:

1 1,0

# Služby C komparatora

## Služby C komparatora

Komparator ponuka niekoko služieb. Ich vstupný a výstupný formát sú popísane v asti implementácia.

1. Služba na porovnávanie publikácií - CMD - íslo 0
2. Služba na porovnávanie autorov - CMD - íslo 1
3. Služba na porovnávanie - CMD - íslo 2
4. Služba na porovnávanie autorov naítaných zo súboru - CMD - íslo 3

### Služba íslo 0

Porovnávanie publikácií kopíruje vstupný formát spomínaný v špecifikácií rozhrania. Volá sa pomocou ísla služby, ktorým je 0. Spustí sa algoritmus navrhnutý na porovnávanie publikácií.

Algoritmus porovnáva mená tak, že ich rozkúskuje na priezvisko a zvyšné mená a tie súasti potom porovnáva pomocou Jaro-Winkler algoritmu. Ak je v mene spomenutá len iniciálka, tak porovná iniciálky dodaných mien. Najväšiu a zárove rozhodujúcu váhu má priezvisko.
íselné údaje porovnáva pomocou aritmetického porovnania.
Nakoniec súvislý text porovnáva na základe presnej zhody ignorujúc medzery.

Celkovo po porovnaní všetkých položiek algoritmus vykoná pomocou kosínusovej podobnosti a urených váh konený výsledok.

#### Príklad:

Vstup do komparatora:

0 2 "\"Team_member_x\" 1 \"Research rank\" 0 0 0 0 1 \"2014\" 0 0 0 0 0"  "\"Team_member_x\" 1 \"Research prank\" 0 0 0 0 1 \"2014\" 0 0 0 0 0"

Vystup komparátora:

1 1,0

### Služba íslo 1

Porovnávanie autorov kopíruje vstupný formát spomínaný v špecifikácií rozhrania. Volá sa pomocou ísla služby, ktorým je 1. Spustí sa algoritmus navrhnutý na porovnávanie autorov.

Algoritmus porovnáva mená tak, že ich rozkúskuje na priezvisko a zvyšné mená a tie súasti potom porovnáva pomocou Jaro-Winkler algoritmu. Ak je v mene spomenutá len iniciálka, tak porovná iniciálky dodaných mien. Najväšiu a zárove rozhodujúcu váhu má priezvisko.

Celkovo po porovnaní všetkých položiek algoritmus vykoná pomocou kosínusovej podobnosti a urených váh konený výsledok. Ak sa nezhoduje prvý autor, tak sa s porovnávaním nepokrauje a vráti sa nula.

#### Príklad:

Vstup do komparátora:

1 3 "\"Zdenik Dlabáeek\" 1 \"FZU-D\" 11 \"Lubomír Jastrabík\" \"Alexandr Dejneka\" \"V. Trepakov\" \"M. Aono\" \"Y. Okawa\" \"Pavlo Bykov\" \"Jan Drahokoupil\" \"Marina Makarova\" \"Petr Sazama\" \"Jioí Franc\" \"Zdenik Dlabáeek\" " "\"B. Dkhil\"1 \"N/A\"  16 \"Milan Orlita\" \"Dmitry Nuzhnyy\" \"Jan Petzelt\" \"J.H. Lee\" \"Stanislav Kamba\" \"Veronica Goian\" \"Viktor Bovtun\" \"K.Z. Rushchanskii\" \"Martin Kempa\" \"M. Ležaie\" \"B. Dkhil\" \"P. Gemeiner\" \"Jioí Hlinka\" \"T. Birol\" \"D. G. Schlom\" \"C.J. Fennie\" " "\"Zdenik Dlabáeek\"1 \"FZU-D\"  8 \"Marian Eeroanský\" \"Marina Makarova\" \"Jan Drahokoupil\" \"Alexandr Dejneka\" \"Vladimír Trepakov\" \"Pavlo Bykov\" \"Zdenik Dlabáeek\" \"Lubomír Jastrabík\" "

Výstup komparátora:

0

Pozn. Žiaden match sa nenašiel

### Služba íslo 2

Porovnávanie autorov kopíruje vstupný formát spomínaný v špecifikácií rozhrania. Volá sa pomocou ísla služby, ktorým je 1. Spustí sa starý algoritmus navrhnutý na porovnávanie autorov, jeho efektivita je otázna.

Algoritmus porovnáva mená tak, že ich rozkúskuje na priezvisko a zvyšné mená a tie súasti potom porovnáva pomocou Jaro-Winkler algoritmu. Ak je v mene spomenutá len iniciálka, tak porovná iniciálky dodaných mien. Najväšiu a zárove rozhodujúcu váhu má priezvisko.

Celkovo po porovnaní všetkých položiek algoritmus vykoná pomocou kosínusovej podobnosti a urených váh konený výsledok. Ak sa nezhoduje prvý autor, tak sa s porovnávaním nepokrauje a vráti sa nula.

#### Príklad:

Vstup do komparátora:

2 3 "\"Zdenik Dlabáeek\" 1 \"FZU-D\" 11 \"Lubomír Jastrabík\" \"Alexandr Dejneka\" \"V. Trepakov\" \"M. Aono\" \"Y. Okawa\" \"Pavlo Bykov\" \"Jan Drahokoupil\" \"Marina Makarova\" \"Petr Sazama\" \"Jioí Franc\" \"Zdenik Dlabáeek\" " "\"B. Dkhil\"1 \"N/A\"  16 \"Milan Orlita\" \"Dmitry Nuzhnyy\" \"Jan Petzelt\" \"J.H. Lee\" \"Stanislav Kamba\" \"Veronica Goian\" \"Viktor Bovtun\" \"K.Z. Rushchanskii\" \"Martin Kempa\" \"M. Ležaie\" \"B. Dkhil\" \"P. Gemeiner\" \"Jioí Hlinka\" \"T. Birol\" \"D. G. Schlom\" \"C.J. Fennie\" " "\"Zdenik Dlabáeek\"1 \"FZU-D\"  8 \"Marian Eeroanský\" \"Marina Makarova\" \"Jan Drahokoupil\" \"Alexandr Dejneka\" \"Vladimír Trepakov\" \"Pavlo Bykov\" \"Zdenik Dlabáeek\" \"Lubomír Jastrabík\" "

Výstup komparátora:

0

Pozn. Žiaden match sa nenašiel

### Služba íslo 3

Služba kopíruje funkcionalitu služby íslo 1, avšak v CMD je vstup len íslo služby a meno súboru. V súbore je presne totožný formát vstupných údajov ako pri službe íslo 1, avšak íslo služby sa už neuvádza. Výstup je ten istý ako pri službe íslo 1 do CMD. Algoritmus porovnávania je tiež totožný.

Algoritmus porovnáva mená tak, že ich rozkúskuje na priezvisko a zvyšné mená a tie súasti potom porovnáva pomocou Jaro-Winkler algoritmu. Ak je v mene spomenutá len iniciálka, tak porovná iniciálky dodaných mien. Najväšiu a zárove rozhodujúcu váhu má priezvisko.

Celkovo po porovnaní všetkých položiek algoritmus vykoná pomocou kosínusovej podobnosti a urených váh konený výsledok. Ak sa nezhoduje prvý autor, tak sa s porovnávaním nepokrauje a vráti sa nula.

#### Príklad:

Vstup do komparátora:

3 test.txt

Súbor test.txt:

3 "\"Zdenik Dlabáeek\" 1 \"FZU-D\" 11 \"Lubomír Jastrabík\" \"Alexandr Dejneka\" \"V. Trepakov\" \"M. Aono\" \"Y. Okawa\" \"Pavlo Bykov\" \"Jan Drahokoupil\" \"Marina Makarova\" \"Petr Sazama\" \"Jioí Franc\" \"Zdenik Dlabáeek\" " "\"B. Dkhil\"1 \"N/A\"  16 \"Milan Orlita\" \"Dmitry Nuzhnyy\" \"Jan Petzelt\" \"J.H. Lee\" \"Stanislav Kamba\" \"Veronica Goian\" \"Viktor Bovtun\" \"K.Z. Rushchanskii\" \"Martin Kempa\" \"M. Ležaie\" \"B. Dkhil\" \"P. Gemeiner\" \"Jioí Hlinka\" \"T. Birol\" \"D. G. Schlom\" \"C.J. Fennie\" " "\"Zdenik Dlabáeek\"1 \"FZU-D\"  8 \"Marian Eeroanský\" \"Marina Makarova\" \"Jan Drahokoupil\" \"Alexandr Dejneka\" \"Vladimír Trepakov\" \"Pavlo Bykov\" \"Zdenik Dlabáeek\" \"Lubomír Jastrabík\" "

Výstup komparátora:

0

Pozn. Žiaden match sa nenašiel

# Súbory s meta údajmi

## Úvod

Nasledujúci dokument popisuje súbory a meta údaje ktoré sú do nich ukladané. Údaje majú predovšetkým diagnostický a plánovací význam. Ide predovšetkým o to vedie efektívne naplánova aktivity súvisiace s optimalizáciou.

# Formát údajov

Údaje sú uložené v binárnych súboroch a potom sú z nich naítané do príslušných údajových štruktúr popísaných nižšie.

## Formát údajov pri publikáciách

```
struct stats{
//Required data analysis
unsigned int num_of_required_analysis;
unsigned int num_of_microsecond_in_required_analysis;

//Optional data analysis
unsigned int num_of_optional_analysis;
unsigned int num_of_microsecond_in_optional_analysis;

//Meta data analysis
unsigned int num_of_meta_analysis;
unsigned int num_of_microsecond_in_meta_analysis;
}
```

### Popis formátu

Pre každú z troch etáp výpotu sa ukladajú údaje o tom, že analýza prebehla a koko mikrosekúnd zabrala. Tieto údaje sú pripoítané ku globálnym premenným.

Pri vykonaní asti výpotu sa ku unsigned int num_of_XX_analysis pripoíta +1 ako indikátor toho, že koko analýz zaznamenalo svoj výsledok. Potom sa do unsigned int num_of_microsecond_in_XX_analysis uloží príslušný poet milisekúnd. Za XX sa doplní meno príslušnej asti napríklad required alebo optional a podobne.

## Formát údajov pri autoroch

```
struct stats{
//Required data analysis
unsigned int num_of_first_author_analysis;
unsigned int num_of_microsecond_in_first_author_analysis;

//Optional data analysis
unsigned int num_of_co_authors_analysis;
unsigned int num_of_microsecond_in_co_authors_analysis;

//Meta data analysis
unsigned int num_of_workplaces_analysis;
unsigned int num_of_microsecond_in_workplaces_analysis;
}
```

### Popis formátu

Pre každú z troch etáp výpotu sa ukladajú údaje o tom, že analýza prebehla a koko mikrosekúnd zabrala. Tieto údaje sú pripoítané ku globálnym premenným.

Pri vykonaní asti výpotu sa ku unsigned int num_of_XX_analysis pripoíta +1 ako indikátor toho, že koko analýz zaznamenalo svoj výsledok. Potom sa do unsigned int num_of_microsecond_in_XX_analysis uloží príslušný poet milisekúnd. Za XX sa doplní meno príslušnej asti napríklad first_author alebo workplaces a podobne.

# Zoznam aktívnych súborov a popis

kj_publication_comparator.dat - Binárny záznam trvaní analýz publikácií a ich potu v horeuvedenom formáte.

kj_author_comparator.dat - Binárny záznam trvaní analýz autorov a ich potu v horeuvedenom formáte.

kj_author_comparator_old.dat - Binárny záznam trvaní analýz autorov a ich potu v horeuvedenom formáte. V tomto súbore sú záznamy údajov vyprodukovaných starou metódou porovnávania.

# Doplujúce dokumenty

Obsahuje alšie dokumenty, ktoré sú zamerané na podporu práce s C komparátorom.

### Metodika: Zmena podporovaných rozhraní C komparátora

Opisuje kroky na odstránenie, pridanie alebo zmenenie podporovaných rozhraní.

#### Metodika: Zmena porovnávacieho algoritmu C komparátora

Obsahuje kroky na zmenu alebo obohatenia vnútorného mechanizmu na porovnávanie.

#### Metodika: Pridanie služby do C komparátora

Vyššia metodika obsahujúce kroky potrebné na to, aby sa dal komparátor rozšíri o alšiu službu.

# Metodika: Zmena podporovaných rozhraní C komparátora

## Úvod

Metodika je urená na to, aby objasnila a uahila zmenu implementácie rozhraní C komparátora.

## Terminológia

C komparátor - program napísaný v jazyku C++ za úelom porovnávania autorov a publikácií prístupný z CMD.

## Nadväzujúce metodiky

- Zmena služby v C komparátore

## Manažment zmeny rozhraní C komparátora

Poas životného cyklu C komparátora sa môže sta situácia, že je potrebné prida, zmeni alebo vymaza rozhranie, aby sa tak C komparátor prispôsobil situácií. V prípade nesystematického procesu zmeny rozhraní v C komparátore, sa môže naruši vnútorná integrita programu alebo jeho funkcionalita. V týchto prípadoch dodržiavame postup, ktorý je v tomto dokumente vytýený.

Metodika je urená pre vývojárov.

Je potrebné opísa procesy:

- Proces rozšírenia rozhraní
- Proces zmeny rozhrania
- Proces vymazania rozhrania

## Podrobný popis procesov

#### Proces rozšírenia rozhraní & Proces zmeny rozhrania

1. Je potrebné vytvori novú triedu na uloženie entity? Ak ano pokraujeme bodom 1.a, inak pokraujeme bodom 2.
    a. Napíšeme reprezentanú triedu na entitu, ktorú chceme reprezentova do samostatného modulu alebo hlavikového súboru. Nazveme ju kj_repository_entita, kde slovo entita nahradíme menom entity.
    b. Prilinkujeme ju do modulu main.cpp pomocou hlavikového súboru.
2. Implementujeme v samostatnom module alebo priamo v main.cpp funkciu, ktorá sa volá parse_arguments_to_entity, kde entity nahradíme menom entity, ktorú chceme naíta. Vstupom sú informácie nutné na to, aby sa dali vstupné informácie preíta (napríklad meno súboru, alebo argumenty príkazového riadku). Výstupom je trieda reprezentujúca záznam o entite, ktorú chceme reprezentova.
3. Vyhadáme službu, ktorej rozhranie chceme rozšíri poda ísla.
4. V implementácií služby nájdeme riadok, kde sa íta vstup.
5. Nahradíme/doplníme ítaciu funkciu novou implementáciou.

```
452     //Publications service
453     class simple_list<char *> *list_res = new simple_list<char *>();
454     if(argv[0][0] == '0'){
455         //Moove program caller out of way
456         argc--;
457         argv++;
458         //Parse publications
459         class simple_list<class publication_repository *> *list;
460         list = parse_arguments(argc, argv);
461         if(list == NULL){
462             //Error parsing arguments
463             file_works->log_msg("Error while parsing arguments.");
464             return -1;
465         }
```

Obr.1 : Príklad implementácie parsovania argumentov.

#### Proces vymazania rozhrania

1. Nájdeme službu poda ísla
2. Nájdeme riadok, kde sa volá parsovanie vstupu. Ide o funkciu,ktorá dostáva ako parametre informácie nutné k naítaniu vstupu a ako výstup má záznam entity. Volá sa  kj_repository_entita, kde slovo entita je nahradené menom entity ktorú parsuje.
3. Ak chceme natrvalo vymaza entitu:
    a. Vymažeme celú funkciu.
    b. Vrátime sa k jej volaní v implementácií služby.
    c. Zistíme, i záznam entity, ktorú implementácia rozhrania vracala je referencovaný inde v kóde.
    d. Ak nie nájdeme modul, kde je záznam entity implementovaný a vymažeme ho (predpokladá sa, že modul slúži iba na úel definície tohto jedného záznamu).
4. Vymažeme riadok, kde sa volá parsovanie vstupu.

## Metodika: Zmena porovnávacieho algoritmu C komparátora

### Úvod

Metodika je urená na to, aby uahila aplikáciu zmien porovnávacích algoritmov v C komparátore.

### Terminológia

C komparátor - program napísaný v jazyku C++ za úelom porovnávania autorov a publikácií prístupný z CMD.

### Nadväzujúce metodiky

- Zmena služby v C komparátore

### Manažment zmeny implementácie porovnávania v C komparátore

Poas životného cyklu C komparátora sa môže sta situácia, že je potrebné prida, zmeni alebo vymaza niektoré rutiny na porovnávanie entít, aby sa tak C komparátor prispôsobil situácií. V prípade nesystematického procesu zmeny porovnávacieho algoritmu v C komparátore, sa môže naruši vnútorná integrita programu alebo jeho funkcionalita. V týchto prípadoch dodržiavame postup, ktorý je v tomto dokumente vytýený.

Metodika je urená pre vývojárov

Je potrebné opísa procesy:

- Proces pridania porovnávacieho algoritmu
- Proces zmeny porovnávacieho algoritmu
- Proces vymazania porovnávacieho algoritmu

### Podrobný popis procesov

### Proces pridania porovnávacieho algoritmu & proces zmeny porovnávacieho algoritmu

1. Je potrebné vytvori novú triedu na uloženie entity? Ak ano pokraujeme bodom 1.a, inak pokraujeme bodom 2.
    a. Napíšeme reprezentanú triedu na entitu, ktorú chceme reprezentova do samostatného modulu alebo hlavikového súboru. Nazveme ju kj_repository_entita, kde slovo entita nahradíme menom entity.
    b. Prilinkujeme ju do modulu main.cpp pomocou hlavikového súboru.
2. Je potrebné prida elementárnu porovnávaciu rutinu? (tj. rutinu, ktorá zoberie dva elementárne dátové typy, resp. zložky porovnávaných entít a pomocou implementovaného algoritmu ich porovná) Ak ano:
    a. Implementujeme algoritmus, ktorý je potrebný pre porovnanie do samostaného modulu.
    b. Pridáme ho do projektu
3. Vytvoríme/otvoríme modul, kde bude implementovaná funkcia, na porovnávanie zvolených entít.
4. Prilinkujeme všetky potrebné elementárne porovnávacie rutiny.
5. Implementujeme predom zvolenú logiku porovnávania daných dvoch entít. Vstupom do porovnania sú dve entity, výstupom je zoznam koeficientov podobnosti.
6. Prilinkujeme vytvorený/upravený modul do modulu main.cpp
7. Nájdeme poda ísla službu, ktorej porovnávaciu logiku chceme upravi.
8. Vymeníme v bloku, ktorý porovnáva dve entity starú volanie starej funkcie porovnávania ta novú.

### Proces vymazania porovnávacieho algoritmu

1. Nájdeme poda ísla službu, ktorej porovnávaciu funkciu chceme zmaza.
2. Nájdeme blok kódu, kde sa volá porovnávacia entita.
3. Poda mena porovnávacej funkcie nájdeme modul, kde je implementovaná.
4. Vymažeme modul (predpokladá sa, že modul slúži iba na definíciu tejto konkrétnej porovnávacej funkcie)
5. Vymažeme volanie porovnávacej entity v main.cpp

# Metodika: Pridanie služby do C komparátora

## Úvod

Metodika je urená na to, aby uahila proces pridávania a odoberania služby z C komparátora.

## Terminológia

C komparátor - program napísaný v jazyku C++ za úelom porovnávania autorov a publikácií prístupný z CMD.

## Nadväzujúce metodiky

- Zmena porovnávacieho algoritmu C komparátora
- Zmena podporovaných rozhraní C komparátora

## Manažment pridania služby do C komparátora

Poas životného cyklu C komparátora sa môže sta situácia, že je potrebné prida, zmeni alebo vymaza služby ktoré C komparátor ponúka, aby sa tak prispôsobil situácií. V prípade nesystematického procesu zmeny služieb v C komparátore, sa môže naruši vnútorná integrita programu alebo jeho funkcionalita. Služba pozostáva z naítania vstupných údajov a porovnania údajov, manipulácie s týmito dvoma aspektami sú vo svojej podstate zmenami služieb a ako také sú opísané v nadväzujúcich metodikách. V ostatných prípadoch dodržiavame postup, ktorý je v tomto dokumente vytýený.

Metodika je urená pre vývojárov

Je potrebné opísa procesy:

- Proces pridania služby
- Proces odobratia služby

## Podrobný popis procesov

Existuje niekoko podprocesov ktoré budeme potrebova. Tieto sú opísané v nadväzujúcich metodikách. Tu je zoznam:

- Proces vymazania rozhrania - Metodika Zmena podporovaných rozhraní C komparátora
- Proces pridania rozhrania - Metodika Zmena podporovaných rozhraní C komparátora
- Proces vymazania porovnávacieho algoritmu - Metodika Zmena porovnávacieho algoritmu C komparátora

- Proces pridania rozhrania - Metodika Zmena porovnávacieho algoritmu C komparátora

### Proces pridania služby

1. Nájdeme íslo služby, ktoré je volné.
2. Otvoríme main.cpp
3. Na koniec blokov, ktoré obsluhujú služby v metóde "int main(int argc, char *argv[])" pridáme blok, ítajúci z cmd íslo služby (podmienku), v danom mieste bude íslo služby na adrese argv[0][0] ako char.
4. Implementujeme rozhranie podla procesu pridania rozhrania.
5. Pridáme blok na kontrolu správneho spracovania vstupu.
6. Pridáme blok kódu na porovnanie publikácií, porovnávací algoritmus je implementovaný poda procesu pridania rozhrania. Vystupom bloku je zoznam floatov typu class simple_list<float>.
7. Skontrolujeme správnos zoznamu.

```
517
518     //Author service - old
519     if(argv[0][0] == '2'){
520         //Moove program caller out of way
521         argc--;
522         argv++;
523         //Parse publications
524         class simple_list<class author_repository *> *list;
525         list = parse_arguments_to_authors(argc, argv);
526         if(list == NULL){
527             //Error parsing arguments
528             file_works->log_msg("Error while parsing arguments.");
529             return -1;
530         }
531 
532         //Porovnanie publikacii
533         class author_comparator handler;
534         for(int i = 1; i < list->get_list_length(); i++){
535             if(PUBL_TRASHOLD <= handler.compare_authors_old(list->browse_element(0), list->browse_element(i))){
536                 char buff_work[10];
537                 itoa(i, buff_work, 10);
538                 list_res->add_element(buff_work);
539             }
540         }
541 
542         argc++;
543         argv--;
544 
545         if(list != NULL) delete list;
546     }
```

Obr.1: Príklad implementácie vzorovej služby s íslom 2.

### Proces odobratia služby

1. Otvoríme main.cpp
2. Nájdeme službu poda ísla
3. Vymažeme rozhranie služby poda procesu vymazania rozhrania
4. Vymažeme porovnávací algoritmus poda procesu vymazania porovnávacieho algoritmu
5. Vymažeme telo služby

# J comparator

# Úvod

J komparátor je súasou podsystému, ktorý obsluhuje algoritmy porovnávania. Vznikol ako odpove na potrebu jednotného prístupu k databáze cez ORM, ktoré je napísané v Jave. Je teda napísaný v Jave kvôli tomu, aby mohol efektívnejšie komunikova s databázou pomocou ORM, keže algoritmy porovnávania sú napísané v jazyku C++. Postupne k úlohe mediátora medzi C komparátorom a databázou pribudli alšie úlohy, ktoré súvisia s predspracovaním údajov pred samotným porovnávaním a so spracovaním výsledkov porovnávania.

Úelom J komparátora je komunikácia s ORM, cez ktoré pristupuje k databáze, a to jednak k novým údajom, ktoré musíme napárova, a potom k údajom, voi ktorým budeme nové záznamy porovnáva. Tieto údaje alej spracováva, priom obsahuje algoritmus na výber kandidátov na porovnanie. alej je úlohou J komparátora vysklada vstup pre C komparátor a spracova výsledky z neho. Súasou J komparátora sú taktiež algoritmy, ktoré nad spárovanými objektami vykonávajú merge a join operácie.

Okrem spomínaných úloh poskytuje aj rozhranie pre spustenie algoritmov párovania, preto je komunikanou asou podsystému párovania.

## Vstup

Ako priamy vstup nepotrebuje J komparátor žiadne argumenty a ani nie sú plánované. J komparátor však komunikuje s databázou, ktorá je zdrojom údajov, ktoré J komparátor alej spracováva

## Výstup

J komparátor taktiež nemá žiadny výstup, výstupom algoritmov porovnaní sú spárované publikácie v databáze

# Špecifikácia požiadaviek J komparátora

### Všeobecné požiadavky

J komparátor má v prvom rade spa funkciu mediátora medzi C komparátorom a ORM, preto musí by adaptovaný po každej zmene v týchto komponentoch.

Niektoré úlohy J komparátora vyžadujú atomický prístup do databázy, na o využíva J komparátor komunikáciu pomocou tried Hibernate, ako sú Session, Transaction a podobne. Pre jednu inštanciu J komparátora platí, že obsahuje jednu inštanciu triedy Session.

Pre každé porovnávanie vytvára J komparátor nový proces, ktorý obsahuje funkcionalitu C komparátora a s ktorým komunikuje spôsobmi opísanými v požiadavkách na rozhranie.

Primárnou požiadavkou je rýchlos spracovaniam ktorá musí by optimalizovaná a nesmie prekroi limit 1s na publikáciu

### Požiadavky na rozhranie

J komparátor komunikuje s ORM štandardným spôsobom cez API volania, ktoré ORM poskytuje. Takýmto spôsobom pristupuje J komparátor k databáze a preferovane nepoužíva iné formy SQL komunikácie.

Pre rozhranie s C komparátorom existuje viacero možností. Keže C komparátor je definovaný genericky, je možné komunikova prostredníctvom rôznych rozhraní, primárne rozhranie je však cez poskytovanie argumentov pri vytváraní nového procesu s C komparátorom. Záložnou možnosou je komunikácia cez súbory, priom súbor obsahuje presne taký vstup ako pri komunikácii cez CMD. Pritom predpokladáme, že na danom stroji, na ktorom funguje J komparátor, má možnos vytvára procesy a pristupova k vykonávatenému kódu C komparátora.

### Požiadavky na stratégiu výberu kandidátov na porovnanie

Výber kandidátov musí by objektívny, o znamená, že nesmie znevýhodova záznamy, ktoré obsahujú chybu v niektorom poli. Mal by uprednostova podobnejšie záznamy pred menej podobnými

Algoritmus musí prebieha rýchlo, nemal by obsahova žiadne zložité výpoty. Celkový cyklus párovania musí prebieha v reálnom ase, preto výber kandidáta by nemal zabera podstatný as voi celkovému asu porovnávania.

Je urený na efektívny výber kandidátov tak, aby sme v celkovom dôsledku porovnávali ovea menej záznamov ako pri absencii tejto stratégie. Dôsledkom má by zrýchlené porovnávanie, priom musíme zachova vysokú mieru efektivity, o znamená, že všetky záznamy, ktoré by sa mali spárova sa aj reálne spárujú

**Vstup**

- hadaný záznam, napríklad hadaná publikácia

**Výstup**

- množina záznamov, ktoré sú najpodobnejšie danému hadanému záznamu

### Požiadavky na stratégie párovania rovnakých záznamov

Po aplikovaní algoritmov porovnávania na hadaný záznam a kandidátov na párovanie s týmto záznamom, sú identifikované záznamy, ktoré boli oznaené ako rovnaké. Tieto záznamy sú následne spárované v rámci J komparátora. Oznai dva záznamy ako rovnaké môže používate aj rune, avšak je prepokladom, že má na to povolenie.

J komparátor musí obsahova implementácie dvoch typov párovaní:

- merge
- join

Pri stratégii merge ostáva po párovaní len jeden záznam, ktorý obsahuje všetky opakovatené polia, ktoré obsahovali párované záznamy, priom neopakovatené polia dedí od záznamu, ktorý je urený ako master.

Pri stratégii join ostáva taktiež len jeden záznam, ktorý je oznaený ako master, obsahuje však aj údaje chýbajúce v pôvodnom zázname doplnené zo záznamu, ktorý bol oznaený ako slave

**Vstup**

- dva záznamy, ktoré boli oznaené ako rovnaké
- stratégia párovania

**Výstup**

- žiadny priamy výstup
- nepriamym výstupom sú spárované záznamy poda urenej stratégie

# Návrh J komparátora

Poda vypracovanej špecifikácie bol J komparátor navrhnutý nasledovne

### Umiestnenie v rámci systému

J komparátor nie je samostatným komponentom systému, existuje ako prostriedok komunikácie medzi C komparátorom a prostredím, priom zabezpeuje niektoré podporné funkcie, ako je napríklad výber kandidátov na porovnávanie

### Komunikácia v rámci rozhraní

J komparátor komunikuje s C komparátorom prostredníctvom vytvorenia alšieho procesu a prostredníctvom argumentov, ktoré mu odovzdá

So zvyškom systému komunikuje J komparátor štandardným volaním funkcií a procedúr zodpovedajúceho rozhrania v Jave

### Porovnávanie

Porovnávanie dvoch záznamov môže predstavova porovnávanie nasledovných entít:

- publikácia
- autor
- publikácia a následne autori v nej

Porovnávanie publikácie a autora ako samostatných entít je realizované ako testovanie implementovaných metód pri absencii dostatoného množstva rozdielnych údajov, aby sme mohli vykonáva hlavné porovnávanie, ktorým je porovnávanie publikácií a následne porovnávanie autorov v nich.

### Workflow

V rámci podsystému na párovanie je workflow z hadiska J komparátora nasledovný (pre párovanie publikácií a potom autorov v nich):

1. Zavolanie párovania
2. Získanie alšej nespracovanej publikácie
3. Ak sme spracovali všetky publikácie, koniec
4. Získanie kandidátov pre porovnávanie pre danú publikáciu
5. Zavolanie C komparátora so vstupom zodpovedajúcim párovanej publikácii a príslušných kandidátov pre porovnávanie
6. Spracovanie výsledkov C komparátora a urenie rovnakých publikácií, ak existujú
7. Ak neexistujú žiadne zhodné publikácie pokrauj krokom 2
8. Pre dané zhodné publikácie namapuj autorov z nespracovanej publikácie na autorov spárovanej publikácie
9. Vykonaj jednu zo stratégií párovania pre danú publikáciu
10. Pokrauj krokom 2

Tok aktivít pre porovnávanie publikácií a autorov ako samostatných entít prebieha podobne s výnimkou bodu 8.

# Implementácia J komparátora

Z hadiska implementácie je J komparátor na uritej úrovni abstrakcie komponent, ktorý predpripraví a odošle údaje C komparátoru, na druhej strane prijme výsledky a urobí nutné zmeny v databáze. Na databázu je napojený pomocou ORM komponentu. Na C komparátor je napojený pomocou CMD, poprípade cez file system.

## Implementácia komponentu

Na vykonávanie svojich funkcií vykonáva J komparátor niekoko akcií. V prvom rade pomocou ORM vyberá z databázy kandidátov na párovanie. Následne pripravý spojenie s CMD a pripravý vstup do C komparátora. alej zozbiera výsledky z C komparátora. Poslednou akciou je uloženie podobnosti v databáze a ak je to potrebné, vykonanie operácie JOIN medzi dvoma publikáciami.

### Výber kandidátov

Kandidáti sa vyberajú tak, že v databáze utriedia publikácie poda mena a následne sa zoberie okolie 10 publikácií na porovnanie. Kvôli optimalizácií sa zoberie naraz okolie pre všetky publikácie, ktoré sú riešené. Toto výrazne zníži nároky na databázu, ktorá sa tým pádom nemusí preusporiada každý krát, kedy sa hadajú duplikáty riešenej publikácie. Tento prístup však prináša aj niekoko ažkostí. Hlavne treba zabezpei to, že

ak je nejaká publikácia P1 zjednotená s PX a zárove P1 je kandidátom pre publikáciu P2, tak kandidát P1 v kandidátoch P2 je nahradený PX. Tým sa zabezpeí konzistencia údajov. V tomto zmysle je tento komponent implementovaný.

Fyzicky ide o funkciu "private static List<Set<Publication>> create_candidates(List<PublicationT> query_results)" z zdrojovom súbore: Main.java, directory: sk.fiit.tp.reserchrank.comparator.

### Vstup a výstup z C komparátora

Vstupnú množinu publikácií vytiahne rutina z databázy. Následne vytvorí množinu kandidátov volaním komponentu "Výber kandidátov". Údaje jednotlivých publikácií a ich kandidátov spracuje do vstupného formátu z CMD poda rozhrania C komparátora popísanom v príslušnej podkapitole C komparátora. Takto pošle údaje C komparátoru. Následne zozbiera výsledky. Ak výsledky prekraujú prahovú hodnotu, tak pošle dané publikácie do komponentu "Join tool".Výsledky C komparátora taktiež poznaí do databázy.

Fyzicky ide o funkciu "private static void processPublication()" z zdrojovom súbore: Main.java, directory: sk.fiit.tp.reserchrank.comparator.

Pozn. Existujú metódy na porovnávanie autorov "private static void processAuthor()" a  "private static void processAuthorOld()", ktoré pracujú analogicky, avšak kandidáti sa vyberajú iným spôsobom priamo v databáze. Okrem toho neaplikujú JOIN. Inak pracujú rovnako ako tento komponent. Bližšie sa im venova nebudeme.

### Join tool

Komponent spojí fyzicky publikácie v databáze. Zmenu zaznamená v histórií zmien. Spojenie dochádza v atribútoch dátového modelu popísaného v príslušnej kapitole dokumentácie. Všetky referenciu na spájanú publikáciu prereferencuje na cieovú publikáciu a potom pôvodnú publikáciu vymaže.

Fyzicky ide o funkciu "public boolean changeRefs(Set<Object> set_to_chng_refs, Object referenced_obj, Session s)" z zdrojovom súbore: JoinTool.java, directory: sk.fiit.tp.reserchrank.comparator.

## Prehad modulov komponentu

- Main.java, directory: sk.fiit.tp.reserchrank.comparator - Komponent na výber kandidátov, Vstup a výstup z C komparátora.
- JoinTool.java, directory: sk.fiit.tp.reserchrank.comparator - Komponent Join tool

# Opis rozhraní J komparátora

Opis rozhrania pre používatea komponentu. Dokument je organizovaný poda služieb, ktoré daný komponent ponúka.

## Deduplikácie publikácií

Služby zistí, ktoré publikácie boli pridané do databázy a skúsi nájs všetky duplikáty v databáze. Zárove zaznamená podobnosti v databáze. Priamo s rozhraním neprichádza používate do styku. Na spustenie funkcie používate zavolá iba funkciu main() v main.java v balíku sk.fiit.tp.researchrank.comparator.

## Runé párovanie

Službu využíva frontend pri runom párovaní kedy volá Join Tool aby dané oznaené publikácie poda potreby spojil. Používate priamo s rozhraním nedochádza do styku. Funkcia sa spúša pomocou frontnedu, kde je implementované runé párovanie.

# Opis služieb J komparátora

Opis služieb na vysvetlenie používateovi.

## Deduplikácie publikácií

Služby zistí, ktoré publikácie boli pridané do databázy a skúsi nájs všetky duplikáty v databáze. Zárove zaznamená podobnosti v databáze. Tento proces je spúšaný automaticky v pravidelných intervaloch.

## Runé párovanie

Službu využíva frontend pri runom párovaní kedy volá Join Tool aby dané oznaené publikácie poda potreby spojil. Tento proces spúša používate prostredníctvom funkcií fronendu.

# Frontend

**Autor:** Michael Gloger

# Úvod

Frontend je komponent systému, služiaci na prezentáciu dát používateom. Obsahuje stránky pre prehadávanie autorov a publikácii a vykonávanie operácii nad týmito dátami. Umožuje tiež registráciu používateov a správu ich útov.

Je napísaný v **Jave** a spúšta sa na **Tomcat serveri**. S databázou komunikuje prostredníctvom **ORM** knižnice **postgre**. Jeho prezentaná vrstva funguje vaka **JSP** šablónam**.**

Táto stránka obsahuje kapitoly popisujúce špecifikáciu požiadaviek na systém a jednotlivé asti tohto komponentu. Opis jeho architektúry, ktorýmá služi budúcemu vývojárovi ako referenný manuál sa nachádza v asti implementácia.

## Špecifikácia požiadaviek

## Dátový model

## Implementácia

## Spustenie na localhost

## ANT Build

# Špecifikácia požiadaviek na frontend

## Úvod

Cieom tejto sekcie dokumentu je uvies funkcionálne a nefunkcionálne požiadavky, poda ktorých bol frontend implementovaný. Vzhadom na to, že existuje veké množstvo možných služieb, ktoré by sme v rámci zamerania produktu mohli prostredníctvom frontendu poskytova, tak dokumentu uruje podmnožinu týchto možných služieb, ktoré sme sa snažili implementova.

### Význam

Objektom tejto kapitoly dokumentácie je webová stránka a jej vnútorná mechanika, ktorá poskytuje služby zákazníkom prostredníctvom zvolenej vizualizácia. Dokument je smerodajný pre developerov, ktorý takto môže zisti o sme sa snažili implementova a z oho sme vychádzali. Taktiež dokument môže slúži používateom ako katalóg funkcionality, ktorá by nejakou formou mala by v momentálnej verzií frontendu implementovaná.

### Rozsah

Produktom je webová stránka, vnútorné mechaniky a procesy, ktoré poskytujú funkcionalitu.

### Definícia skratiek a termínov

Frontend - webová stránka, jej implementácia a implementácia poskytovaných funkcií

### Prehad

Požiadavky sú organizované nasledovne. V prvej kapitole sú všeobecné požiadavky a v druhej kapitole sú špecifické požiadavky organizované vo forme funkcionálnych požiadaviek

## Všeobecné požiadavky

### Perspektíva produktu

Fronend je súasou projektu Research Rank ako komponent, ktorý poskytuje vizualizáciu výsledkov a dovouje roné zmeny v databáze na miestach, kde je to žiadúce. Limitovaný je údajovým modelom, ktorý je vysvetlený v kapitole zaoberajúcej sa databázou. Používa dve iné súasti projektu Research Rank na svoju funkciu a to databázu (a jej model) a komparátor (funkcie J komparátora) na runé párovanie publikácií. Taktiež je požiadavka na to, aby prístup k implementovaným službám mali len autorizovaný používatelia a preto musí by implementovaný model používatea s prihlásením.

## Funkcie produktu

Frontend musí dovoova používateovi sa prihlási, prezera publikácie, vidie svoje publikácie a rune párova publikácie.

## Charakteristika používatea

Používate je vedec, ktorý chce vidie svoje publikácie, opravova chyby pri automatickom spracovaní jeho publikácií a vidie ohlasy na svoje publikácie.

## Limitácie

Frontend musí by optimalizovaný z hadiska používania databázy aby mohol poskytnú služby v rozumnom ase bez prílišného spomaovania.

## Predpoklady a závislosti

Pri implementácií sa oakáva, že používate pozná jednotlivé atribúty publikácií do uritého hadiska, taktiež sa predpokladá že databáza beží a funkcie komparátora sú k dispozícií.

# Konkrétne požiadavky

Obsahuje konkrétne veci, ktoré musia by implementované spolu s opisom. alej akcie, ktoré komponent podnikne, vlastnosti a štandardy, ktoré musí daná funkcie rešpektova. Toto je opísané organizáciou, ktorá je pre daný komponent najzrozumitenejšie - poda funkcionálnych požiadaviek.

## Prihlásenie

Je potrebné aby sa používate vedel prihlási.

Výstup: prístup používatea k ostatným funkciám frontendu.

Vlastnosti: Jednoducho používatené, viditené na stránke, akceptovatene rýchle, dostatone bezpené pre projekt Reseach Rank.

## Prezeranie publikácií a profilov autorov

Je potrebné aby používate vedel prezera publikácie a profily autorov.

Výstup: Zobrazenie publikácií prehadným spôsobom, poda pripomienok zákazníka. Zobrazenie profilov autorov, ich mena, publikácií a pod..

Vlastnosti: Flexibilné zobrazenia dostupných atribútov a publikácií zamerané hlavne na zrozumitenos a realizovatenos. Rozumný výkon.

## Filtrovanie zoznamov publikácií a autorov

Je potrebné aby používate bol schopný vyhadáva v prezentovaných dátach

Výstup: Vyfiltrovanie zoznam publikácií alebo autorov na základe vstupných filtraných pravidiel od používatea

Vlastnosti: Filtrané metódy su generické a flexibilne konfigurovatené

## Runé párovanie

Používate chce opravi chyby pri automatickom spracovaní.

Výstup: Možnos rune spoji dve alebo viac publikácií do jednej pomocou JOIN nástroja J komparátora.

Vlastnosti: Zobrazované údaje publikácie sú aktuálne, spájanie prebieha akceptovatenou rýchlosou.

## Konfigurácia atribútov tabuliek

Používate chce zmeni obsah dát, prezentovaných v zoznamoch autorov a publikácií.

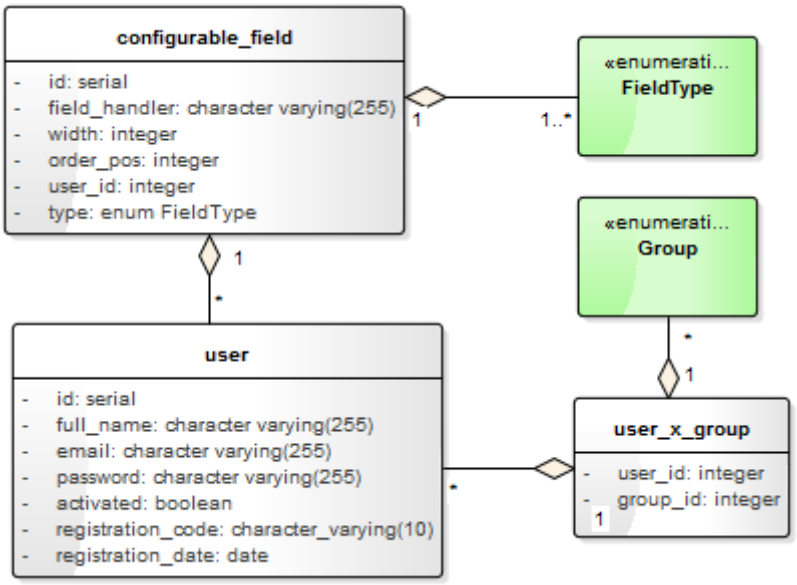
Výstup: Možnos nastavi šírku, usporiadanie a výskyt jednotlivých stpcov tabuliek po príhlásení používatea.

Vlastnosti: Usporiadané a nakonfigurované stpce tabuliek zoznamu autorov a publikácií.

# Dátový model frontendu

## Správa používateov

Na nasledujúcom obrázku je schéma dátového modelu správy používateov. Význam jednotlivých tabuliek a stpcov je uvedený nižšie.




## Význam tabuliek

### user

Obsahuje informácie o používateovi

| Názov stpca | Význam |
|---|---|
| full_name | Celé meno používatea (aby nebolo nutné núti používatea vypisova meno, priezvisko, tituly a pod.) |
| email | Emailová adresa používatea nutná k registrácii a aktivácii konta |
| password | Zahashované heslo používatea |
| activated | Informácia, i bol daný úet aktivovaný prostredníctvom aktivaného mailu. |
| registration_code | Registraný kód vygenerovaný poas registrácie. Je potrebný na emailovú aktiváciu konta |
| registration_date | Dátum registrácie používatea. Môže by potrebné ak sa budú vymazáva neaktivované úty po uritej dobe. |

### configurable_field

Obsahuje informácie o atribútoch, zobrazovaných v zoznamocha utorov a publikácií

| Názov stpca | Význam |
|---|---|
| field_handler | Názov triedy (s kompletným názvom package), ktorá je podtriedou triedy FieldHandler. |
| width | Šírka stpca (1 až 12) |
| order_pos | Poradie daného stpca na ostatné stce rovnakej kategórie. |

| | |
|---|---|
| user_id | ID používatea, ktorému patrí táto konfigurácia. Ak je NULL, tak daná konfigurácia sa považuje za defaultnú. |
| type | Typ políka z enum FieldType. Hodnoty: 0 - zoznam autorov, 1 - zoznam publikácií, 2 - runé párovanie |

### user_x_group

Väzobná tabuka, služiaca na ukladanie many-to-many vzahov medzi tabukami user a group.

# Implementácia frontendu

Táto as má slúži ako referenný manuál pre budúcu prácu s Frontendom. Obsahuje popis jednotlivých tried a ich význam.

## Balíky

Frontend je rozdelený do viacerých balíkov, ktoré obsahujú triedy spolu súvisiace. V nasledovnej tabuke ponúkame popis jednotlivých tried a balíkov. Všetky balíky sú vnorené v koreovom balíku **sk.fiit.tp.researchrank.frontend**. Pri alších opisoch, týchto vnorených balíkov používame skrátený názov balíka bez uvedeného prefixu.

| Balík | Názov triedy v balíku | Popis triedy |
|---|---|---|
| Koreový balík<br>*Triedy, ktoré sú využívané v rámci ostatných balíkov a poskytuje rôzne služby.* | FunctionsJSTL | Implementuje JSTL funkcie, ktoré sú prístupné v JSP šablónach |
| | NullObject | Implementuje vzor NullObject, ktorý slúži napríklad na spracovanie vstupov z URL parametrov |
| | RequestData | Trieda zaobaujúca výstupné dáta (napríkald error hlášky) zo servleta, ktoré sa posielajú do JSP šablóny |
| | ServerListener | Inicializuje pripojenie na databázu po spustení servera |
| | Servlet | Abstraktná trieda, ktorá obsahuje logiku, zdieanú pre všetky servlety |
| | ServletModel | Integrácia medzi Frontendom a Hibernate. Implementuje dopytovanie do databázy |
| fieldhandler<br>*Triedy, využívané pre konfiguráciu stpcov pri zoznamoch autorov a publikácií* | EmptyFieldHandler | Imlementácia triedy FieldHandler pre práznu bunku stpca |
| | FieldHandler | Abstraktná trieda, ktorej implementácie obsahujú logiku zobrazovania jednotlivých atribútov zobrazovaných dát |
| fieldhandler.*<br>*Konkrétne implementácie jednotlivých stpcov tabuliek. Týchto stpcov je väší poet a v tejto dokumentácii ich neopisujeme* | | |
| servlets<br>*Triedy, ktoré implementujú triedu Servlet a slúžia ako controller pre danú url* | AddFieldServlet | Servlet, kontrolujúci spracovanie dát po požiadavke používatea o pridanie stpca do tabuky |
| | AuthorServlet | Stránka s detailom autora |
| | AuthorsServlet | Stránka so zoznamom autorov |
| | LoginServlet | Stránka pre prihlásenie používatea |
| | LogoutServlet | Stránka odhlasovania používatea |
| | PublicationServlet | Stánka detailu publikácie |
| | PublicationsServlet | Stránka so zoznamom publikácií |
| | RegisterServlet | Stránka s registraným formulárom |

| | | |
|---|---|---|
| | SaveFieldsServlet | Implementuje logiku uloženia aktuálnej konfigurácie stpcov tabuliek |
| servlets.account<br>*Servlety implementujúce podstránky, ktoré sú*<br>*prístupné len prihlásenému používateovi* | AccountHomeServlet | Domovská stránka prihláseného používatea |
| | AccountMessagesServlet | Stránka so zoznamom nových upozornení používatea* |
| | AccountSettingsServlet | Stránka s nastavením informácií o úte používatea |
| servlets.admin<br>*Servlety implementujúce podstránky, ktoré sú*<br>*prístupné len používateom, patriacim do admin*<br>*skupiny* | AdminAuthorsServlet | Administraná stránka autorov* |
| | AdminHistoryServlet | Stránka zobrazujúca históriu zmien* |
| | AdminUsersServlet | Stránka umožujúca pridávanie používateov do skupín |
| utils<br>*Triedy implementujúce správanie filtra pre rôzne*<br>*vstupy* | CollectionSizeOperationComboBoxModel | |
| | NumericOperationComboBoxModel | |
| | OperationComboBoxModel | |

* Implementácia vyznaených podstránok nie je sfinalizovaná.

## Adresár WebContent

Tento adresár obsahuje resources v HTML, CSS, Javascript a JSP. Obsahuje viaceré podadresáre

| Názov podadresára | Popis |
|---|---|
| assets | Externé JS knižnice a štýly |
| bootstrap | Základný vzhad stránky z bootstrap |
| css | Modifikované css štýly stránky |
| fonts | Obrázkové ikony, vložené pomocou fontov |
| js | Javascriptová logika aplikanej vrstvy |
| WEB-INF | Obsahuje serverové runtime knižnice, konfiguraný súbor a jazykovú lokalizáciu |

### JSP šablóny

JSP šablóny sú uložené priamo v adresári WebContent a obsahujú HTML a Javascript pre jednotlivé podstránky. Väšina JSP šablón vykresuje malé komponenty, ktoré sú spájané do väších astí. Hlavnou šablónov je **page.jsp** do ktorej sa vnárajú ostatné šablóny na základe vstupov z poda konfigurácie aktuálnej implementácie Servletu.

# Spustenie frontendu na localhost

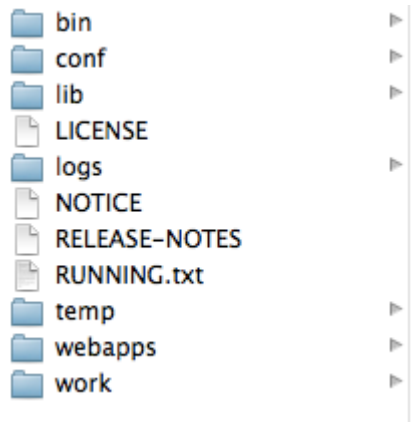
Tato kapitola obsahuje návod ako spusti frontend na lokalnom Tomcat serveri.

V tomto režime sa hibernate pripája na databázu vziadalene a odozva môže by pomalšia.

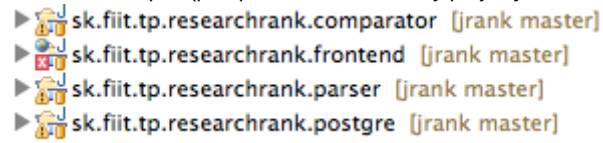
**Autor:** Michael Gloger

## Postup

1. Nainstalova (rozbali) Tomcat 7
2. Otvori si tomcat adresár. Mal by obsahova pod-adresár **lib**

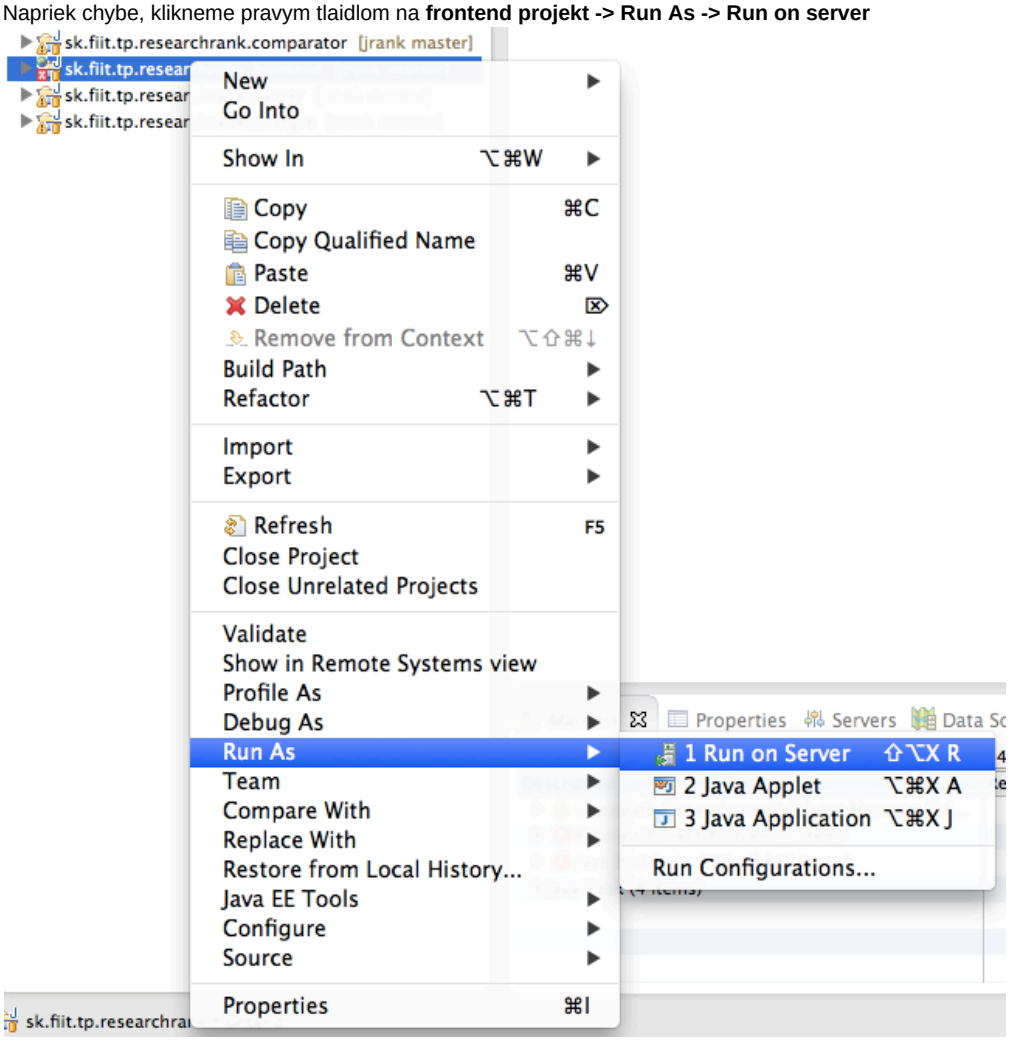



3. **lib** premenujeme na **lib-backup**
4. Stiahneme tomcat-lib.zip a **lib** adresár, ktorý sa v nom nachadza skopírujeme na tomcat
5. Nový **lib** adresár obsahuje **hibernate** knižnce a tiež aj **jrank-postgre.jar**. Je dôlezité aby to bol posledný build z **postgre** projektu, inak frontend nemusí fungova správne.
6. Otvoríme si Eclipse (predpokladame, že vsetky projekty sú už importované vo workspace). Musíte pouzi **Java EE Eclipse**

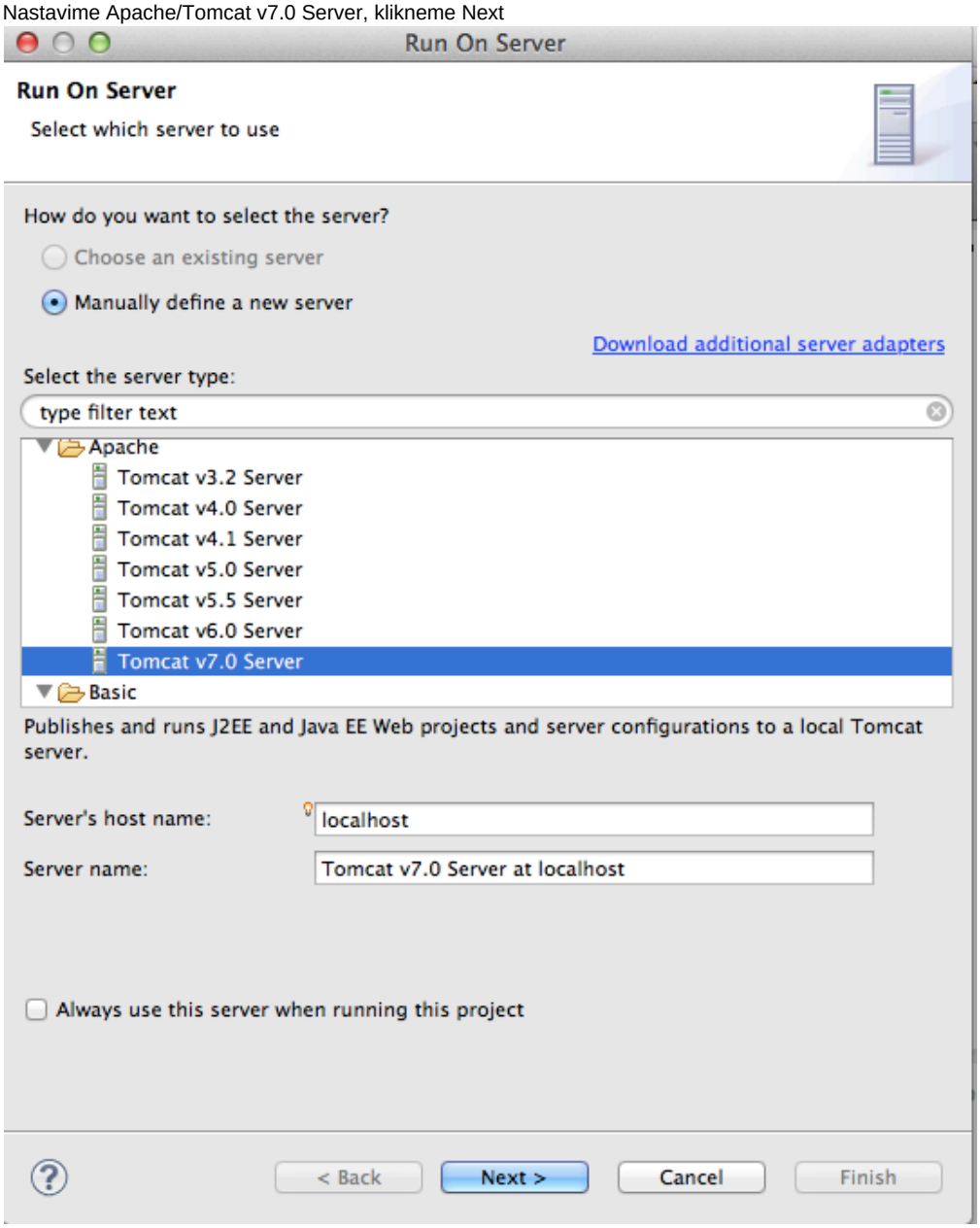



7. Ak ste ešte projekt nespúštali a nevytvárali server, tak vam **frontend** a projekt bude hlasit chybu

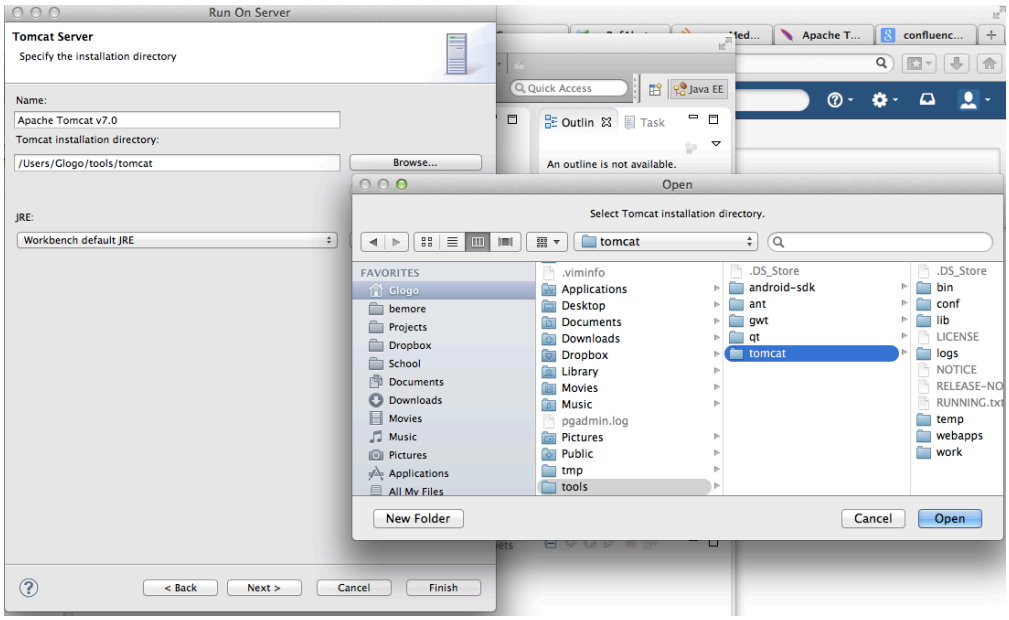
8. Napriek chybe, klikneme pravym tlaidlom na **frontend projekt -> Run As -> Run on server**



9. Nastavime Apache/Tomcat v7.0 Server, klikneme Next

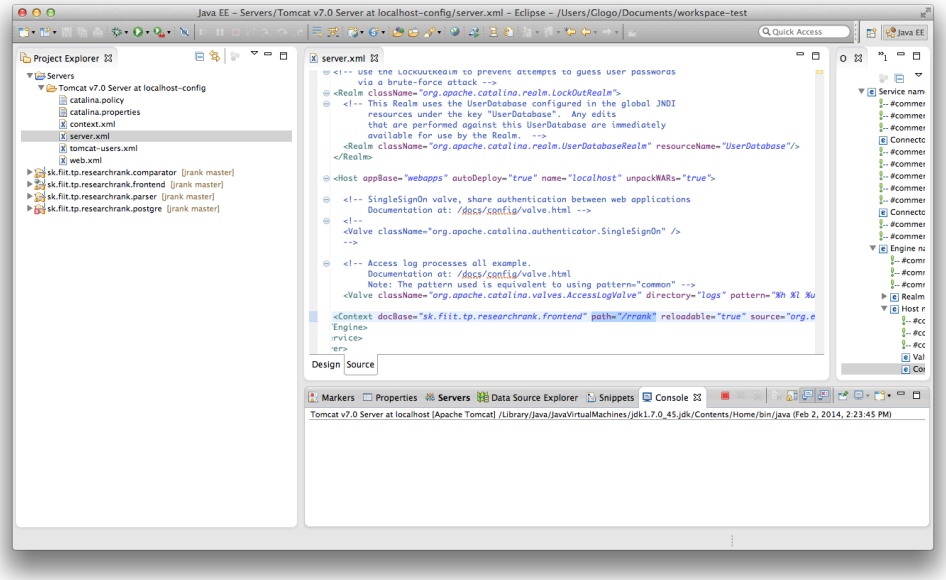



10. Klikneme Browse a vyberieme cestu k tomcat serveru.




11. Klikneme Finish a server by sa mal rovno spustit a zobrazi nám frontend. Môzeme vypnú server stlacenim tlaidla Terminate

12. Pribuol nám projekt Servers, ktorý si otvoríme a nájdeme subor **server.xml .** V om zmeníme atribút v elemente **Contextpath="/sk.fiit.tp.researchrank.frontend"** na **path="/rrank"**




13. Ak sme nemenili port, pripadne inu cestu tak by po spusteni server mal bežat na http://localhost:8080/rrank/
14. V prípade akýchkovek iných chýb/porúch sa presvedte, že ste skopírovali nove libky (hlavne erstvo vybuildeny postgre projekt) na Tomcat a reštartovali ho.

# ANT Build

**Autor:** Rastislav Kostrab

V sk.fiit.tp.researchrank.frontend sa nachadza subor build.xml. Je to ANT script, ktory zabezpeci kompletne vybuildenie celeho projektu a dokaze aj vykonat redeploy.

## Použitie:

1. klikni pravým tlaidlom na súbor build.xml
2. v kontextovej ponuke cho na Run As...> Ant build...
3. zobrazí sa okno pre výber operácii skriptu

Je nutné používa operácie iba s oznaením "Automatic operation". Tieto operácie vykonajú všetko potrebné .

Pomôcka: Vybrané operácie sa uložia a nabudúce staí len použi skratku ALT + SHIFT + X a hne na to stlai Q.

## Pred použitím operácií týkajúcich sa nasadzovania na tomcat server

Jje potrebné pripoji knižnice tomcatu pre ANT. V eclipse sa to robi nasledovne:

1. Cho na Window > Preferences > Ant > Runtime > Global Entries
2. Klikni na Add External JARs...
3. Pridaj všetky jar súbory nachádzajúce sa v adresári "lib" v adresári lokálneho tomcat servera

## Možné problém:

Pri chybe s JAVA_HOME

Riešenie:

1) In Eclipse click Run->External Tools->External Tools Configurations
2) Click JRE tab
3) Click Installed JREs... button
4) Click Add button (select Standard VM, where applicable)
5) Click Directory Button
6) Browse to your JDK version (not JRE) of your installed Java (eg: C:\Program Files\Java\jdk1.7.0_04)
7) Click Finish.

Pri iných problémoch kontaktuje rastislav.kostrab@gmail.com alebo kompas.simon@gmail.com

# WoS

V nasledujúcich kapitolách je uvedená dokumentácia parsera údajov zo systému WoS.

## Špecifikácia požiadaviek na import údajov z WoS

### Úvod

Cieom tohoto dokumentu je popis špecifikácie webovej službu WoS na získavanie ohlasov vedeckých diel.

#### Význam

Táto dokumentácia je urená pre programátorov aplikácie na získavanie ohlasov z portálu WoS.

#### Rozsah

Cieom je vytvori aplikáciu umožujúcu získavanie ohlasov vedeckých publikácií z portálu Web of science. Publikácie sú získavané tiež z tohoto portálu a vyhadávané poda názvu pracoviska.

#### Definícia skratiek a termínov

WoS - Web of knowledge (Web of science)

WS - webservis

# Všeobecné požiadavky

## Funkcie produktu

Produkt bude poskytova 2 základné funkcie:

- vyhadanie a uloženie všetkých vedeckých diel uritého pracoviska
- ku nájdeným dielam nájs všetky citácie, ktoré sa nachádzajú v portále WoS

## Predpoklady a závislosti

Predpoklady, ktoré musia by splnené pre spustenie importéra publikácií a vyhadávaa citácií sú:

- pripojenie k internetu
- IP adresa, ktorá je zaregistrovaná v portále WoS s platnou licenciou

# Konkrétne požiadavky

Vo výslednom produkte musí by imlementovaná funkcionalita vyhadávania publikácií v systéme WoS na základe názvu pracoviska. Následne musí by možné ku tímto publikáciám vyhada a uloži citované diela do existujúcej databázovej štruktúry.

## Vstupy

Vstupom pre funkciu vyhadávania bude názov publikácie, ktorý bude získaný z databázy. Nájdené publikácie sa uložia. WosId týchto publikácií bude zárove vstupom pre funkciu vyhadávania citácií.

## Výstupy

Výstupom danej funkcionality budú publikácie a k nim nájdené citácie, ktoré budú uložené v databáze v príslušnej dátovej štruktúre.

# Doplujúce informácie

Príklad vstupu: "Slov Univ of Technol"

Príklad výstupu: databázová štruktúra s nájdenými publikáciami

# Návrh komponentu na import údajov z WoS

# Search

Komponent Search slúži na získavanie publikácií z citaného indexu WoS. Samotný komponent je organizovaný v triede sk.fiit.tp.researchrank.parser.wos.WosWebserviceClient. Funkcia, ktorá vykonáva danú funkcionalitu je findPublicationsXml(), ktorá prijíma nasledujúce parametre:

1. query (String) - Text, poda ktorého sa majú vyhada publikácie vo WoS-e
2. firstRecord (int) - Poradie prvého získaného diela
3. recordCount (int) - Poet diel, ktoré sa majú vyhada

# CitingArticles

Komponent CitingArticles slúži na získavanie citácií z citaného indexu WoS. Samotný komponent je organizovaný v triede sk.fiit.tp.researchrank.parser.wos.WosWebserviceClient. Funkcia, ktorá vykonáva danú funkcionalitu je getCitationsXml(), ktorá prijíma nasledujúce parametre:

1. uid (String) - id publikácie vo WoS-e
2. dateBegin (String) - zaiatoný hraniný dátum získaných publikácií
3. dateEnd (String) - konený hraniný dátum získaných publikácií

# Opis rozhraní komponentu na import údajov z WoS

Opis rozhrania pre používatea komponentu. Dokument je organizovaný poda služieb, ktoré daný komponent ponúka.

## Stiahnutie publikácií z WoS

Táto funkcia poskytuje funkcionalitu na získavanie WoS publikácií poda názvu pracoviska.

Pri analýze publikácií v systéme WoS sme identifikovali nasledovné variácie názvov slovenských pracovísk.

| Pracovisko | Variácie názvov |
|---|---|
| Ekonomická univerzita v Bratislave | "University of Economics Bratislava" "EU Bratislava" "Univ Econ Bratislava" "Univ Econ" |
| SPU Nitra | "SPU, Nitre, Slovakia" "Slovak University of Agriculture Nitra" "Slovak Agr Univ" "Slovenska Polnohospodarska Univ" "Slovenska Polnohospodarska Umiverzita Nitre" "Slovenska Polnohospodarska Univ Nitre" |
| Slovenská akadémia vied | "Slovak Academy of Sciences" "SAV" "Slovak Acad Sci" |
| Technická univerzita vo Zvolene | "Tech Univ Zvolen" "Technical University Zvolen" |

Metóda získavania publikácií z WoS sa spúša s nasledovnými argumentmi: "oznaenie importu" "názov1" "názov2" ... "názovN"

Príklad: "WOS_2014_05_15_10_00" "Tech Univ Zvolen" "Technical University Zvolen"

## Stiahnutie citácií z WoS

Uvedená funkcia poskytuje funkcionalitu na získavanie WoS citácií poda existujúceho WoSID v zozname originálnych identifikátorov publikácií.

Metóda získavania citácií z WoS sa spúša s nasledovnými argumentmi: "oznaenie importu" parse (argument "parse" je volitený)

Príklad: "WOS_2014_05_15_10_00"

Metóda sahuje XML súbory so zoznamom citácií do prieinka "wos_temp". Poas importovania citácií z týchto súborov je možné daný proces kedykovek zastavi a znovu spusti s argumentom "parse". V tomto prípade sa opätovne nesahujú zvolené citácie z WoSu.

# Opis služieb komponentu na import údajov z WoS

WSDL súbory sú umiestnené na lokalitách:

- http://search.webofknowledge.com/esti/wokmws/ws/WOKMWSAuthenticate?wsdl
- http://search.webofknowledge.com/esti/wokmws/ws/WokSearch?wsdl

Pre používanie služieb WOS je potrebné msearch.pdfa identifikátor relácie (session). Tento získame požiadavkou typu POST na adresu: http://search.webofknowledge.com/esti/wokmws/ws/WOKMWSAuthenticate.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:auth="http://auth.cxf.wokmws.thomsonreuters.com">
  <soapenv:Header/>6<soapenv:Body>
    <auth:authenticate/>
  </soapenv:Body>
</soapenv:Envelope>
```

Získaný identifikátor je potrebné prida do hlaviky HTTP požiadavky, v tvare Cookie:
SID="session_id". Jedna relácia zvládne maximálne 2 dopyty za sekundu. Relácia sa uzatvára poslaním požiadavky na adresu http://search.webofknowledge.com/esti/wokmws/ws/WOKMWSAuthenticate. Platnos sa stratí aj uplynutím uritého asu. Telo požiadavky je nasledovné:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:auth="http://auth.cxf.wokmws.thomsonreuters.com">
  <soapenv:Header/>
  <soapenv:Body>
    <auth:closeSession/>
  </soapenv:Body>
</soapenv:Envelope>
```

Po získaní relácie treba nastavi do hlaviky "Cookie" s hodnotou SID=“session_id“.

Webová služba WoS poskytuje nasledovné služby vyhadávania:

- citedReferences,
- citedReferencesRetrieve,
- citingArticles,
- relatedRecords,
- retrieve,
- retrieveById,
- search.

Najdôležitejšie pre náš projekt sú search a citingArticles, ktoré bližšie popíšeme v nasledujúcich astiach.

## Search

Služba poskytuje vyhadávanie na základe názvov publikácií, pracovísk, autorov a podobne. Z dôvodu nesprávneho vyhadávania poda pracoviska (pracoviská sú vo WoS-e ukladané v skratkách) sme sa rozhodly vyhadáva už konkrétne publikácie. Toto vyhadávanie publikácií je dôležité pre získanie identifikaného ísla publikácie systému WoS, ktoré bude neskôr potrebné pre vyhadávanie citácií daného diela.

## CitingArticles

Služba umožuje vyhadávanie citácií diela poda jedineného WoS identifikátora. Ten je možné získa využitím predchádzajúcej služby "Search". Po úspešnom nájdení citovaných diel vracia služba ich zoznam spolu s potrebnými údajmi (akými sú zoznam autorov, názvov a pod.).

## Mapovanie atribútov WoS na relanú databázu

Webová služba vyhadávania publikácií a citácií vracia xml súbor. Hodnoty v jeho atribútoch ukadáme do relanej databázy nasledovne.

Prijatý záznam o publikácii sa najprv rozdelí na dáta týkajúce sa konkrétneho lánku a dáta týkajúce sa zborníka/knihy, v ktorom je lánok vydaný.

### Dáta lánku

| Názov atribútu WoS | Názov tabuky | Názov atribútu tabuky |
|---|---|---|
| UID | publication | wos_id |
| pub_info(attr: vol) | publication | volume |
| pub_info(attr: pubyear) | publication_year | year |
| page | publication | range |
| title(attr: type == item) | name | name (name_type="hlavny") |
| name: first_name | author_person | name |
| name: last_name | author_person | surname |
| name(attr: role) | person_role | role |
| doctype | document_type | name |
| language | languages | name |
| city, country | publication_place | name |
| subject | keyword_subject_name | name |

| | | |
|---|---|---|
| abstract | abstracts | abstract |
| bk_publisher | publisher | name |
| identifier(attr: value) | identification_code | code |
| identifier(attr: type) | identification_type | name |

### Dáta zborníku/knihy

| Názov atribútu WoS | Názov tabuky | Názov atribútu tabuky |
|---|---|---|
| title(attr: type != item) | name | name (name_type=type) |
| language | languages | name |
| bib_bagecount | publication | range |
| identifier(attr: value) | identification_code | code |
| identifier(attr: type) | identification_type | name |

# Scopus

V nasledujúcich kapitolách je uvedená dokumentácia parsera údajov zo systému Scopus.

Digitálna knižnica Scopus poskytuje webové služby iba registrovaným používateom. Ide o používateov, ktorý si za získavanie údajov platia, alebo o vzdelávacie inštitúcie. Scopus ponúka svojim klientom RESTfull webovú službu. Tá na základe vyslanej URL poskytne relevantný XML výstup. Nám sa podarilo prácu s touto knižnicou implementova aj v našom systéme. V implementácii sme nepoužili existujúcu RESTfull API urenú pre Javu. Vytvorili sme to na základe implementácie viacerých komponentov.

# Špecifikácia požiadaviek na import údajov zo Scopusu

## Úvod

Cieom tohoto dokumentu je popis špecifikácie webovej službu Scopus na získavanie ohlasov vedeckých diel.

### Význam

Táto dokumentácia je urená pre programátorov aplikácie na získavanie ohlasov z portálu Scopus.

### Rozsah

Cieom je vytvori aplikáciu umožujúcu získavanie ohlasov vedeckých publikácií z portálu Web of science. Publikácie sú získavané tiež z tohoto portálu a vyhadávané poda názvu pracoviska.

### Definícia skratiek a termínov

Scopus - abstraktová, citaná a referenná databáza

WS - webservis

## Všeobecné požiadavky

### Funkcie produktu

Produkt bude poskytova 2 základné funkcie:

- vyhadanie a uloženie všetkých vedeckých diel uritého pracoviska
- ku nájdeným dielam nájs všetky citácie, ktoré sa nachádzajú v portály Scopus

### Predpoklady a závislosti

Predpoklady, ktoré musia by splnené pre spustenie importéra publikácií a vyhadávaa citácií sú:

- pripojenie k internetu
- IP adresa, ktorá je zaregistrovaná v portály Scopus
- platny kú pre dopytovanie sa na RESTfull API portálu Scopus

# Konkrétne požiadavky

Vo výslednom produkte musí by implementovaná funkcionalita vyhadávania publikácií v systéme Scopus na základe názvu pracoviska. Následne musí by možné ku týmto publikáciám vyhada a uloži citované diela do existujúcej databázovej štruktúry.

## Doplujúce informácie

Príklad vstupu pre najdenie podla pracoviska:

"University of Economics in Bratislava"
"University of Economics Bratislava"
"Ekonomicka univerzita v Bratislave"
"Ekonomicka Univerzita Bratislave"

Príklad výstupu: databázová štruktúra s nájdenými publikáciami

## Návrh komponentu na import údajov zo Scopusu

Pri návrhu parsera pre Scopus sme sa inšpirovali už existujúcim riešením, ktorým je parser pre XML Crep. Podobne, ako XML Crep, aj XML prichádzajúce zo Scopusu sme parsovali na základe SAX parsera, ktorého vstupom boli hadania, a to:

- aká dimenzia sa má prehadáva,
- parameter dimenzie.

Okrem toho, vyslaná URL ma aj iné nutné parametre, ako sú klientsky kód a adresa servera, od ktorého má služba povolenie požiadavky poúva. Výstupom z parsera je java objekt (Feed.java), ktorý obsahuje jednotlivé nájdené publikácie(Entry.java). Java object, ktorý je výstupom z parsera pre webovú službu Scopusu, sa alej kopíroju do iného java object-u, ktorý je už inštanciou modelu databázy generovaného technológiou Hibernate. Takto vytvorený java objekt sa nakoniec ukladá do databázy za pomoci API technológie Hibernate.

### ParserScopus

Komponent slúžiaci na stiahnutie údajov z portálu Scopus. Vysiela URL request na RESTfull API, ktorú Scopus poskytuje. Maximálny poet vrátených publikácií je 25, preto sa vysiela viacero URL requestov, ktoré pokryjú získanie všetkých údajov. Každý XML response sa priamo parsuje technikou SAX Parsingu a údaje sa ukladajú do Java Objektov, ktoré sa potom perzistujú za pomoci komponentu ScopusXmlLoader.

### ScopusXmlLoader

Vstupom pre túto komponentu sú Java Objekty s údajmi získanými poas parsovania responsov zo služby RESTfull API portálu Scopus. Pre persiztenciu sa využíva technológia Hibernate, ktorá zabezpeuje objektovo-relané mapovanie s databázou PostgreDB.

## Implementácia komponentu na import údajov zo Scopusu

Pri implementáci sme narazili na viaceré problémy, ktoré nám znane predžili predpokladaný as pre vyriešenie parsovania údajov zo Scopusu. Hne zo zaiatku sme narazili na problém nefungovania RESTfull služby. Problém bol však rýchlo vyriešený, pretože išlo iba o predženie zmluvy so Scopusom. alším z problémov bolo zisovanie formy URL, ktorú je potrebné zasiela ako požiadavku o XML. Išlo napríklad o problém, kde sa namiesto medzier v URL musí zadáva znak podtržnika a mnoho alších.

Vzhadom na to, že poet výsledkov vyhadávania sa môže pohybova aj vo vysokých íslach, RESTfull API to ošetruje zasielaním iastkových výsledkov. Týmto sa musela vytvori sluka, ktorá iterovala všetkými výsledkami. Sparsované XML je potrebné uklada do databázy. Jediným problémom v tejto situácii bolo vytvorenie mapovania na jednotlivých párových tagov XMLka na tribúty tabuliek databázy ResearchRank. Celé mapovanie je popísané nižšie.

#### Mapovanie XML response na DB

- Feed
    - Entry
        - prism:url  -> url.url
        - dc:identifier -> identification_code.code
        - dc:title -> name.name
        - prism:publicationName -> name.name (priradenie k zdrojovej publikacii)
        - prism:issn -> identification_code.code(priradenie k zdrojovej publikacii)
        - prism:volume -> publication.edition_volume (priradenie k zdrojovej publikacii)
        - prism:pageRange -> publication.range (priradenie k zdrojovej publikacii)
        - prism:coverDisplayDate -> publication_year.year (priradenie k zdrojovej publikacii)
        - prism:doi -> identification_code.code (priradenie k zdrojovej publikacii)
        - prism:aggregationType -> document_type.name (priradenie k zdrojovej publikacii)
        - subtypeDescription -> document_type.subtype
- Author
    - given-name -> author_person.name
    - Surname -> author_person.surname
    - authid -> author_person_original_id.original_id
- Affiliation
    - name-variant-> workplace_name_variant.name
    - Affilname -> workplace_name.full_name
    - afid -> workplace_name.code

# Popis rozhrania komponentu na import údajov zo Scopusu

Pre ScopusParser boli vytvorené **balíky** pre parsovanie a ukladanie do DB a taktiež spustitený **JAR súbor**

## Balíky

Parser Scopusu poskytuje 3 typy parsovania:

- Hadanie poda zoznamu pracovísk (Tvar: „Pracovisko fakulty“)
- Hadanie poda mien autorov (Tvar: „Priezvisko, M“)
- Hadanie ohlasov poda zadaného SCOPUS_ID

Vstupom do parsera je teda java object ArrayList<String>, ktorý obsahuje všetky podmienky, cez ktoré chceme v Scopuse vyhadáva. Vzahom medzi danými podmienkami zoznamu je ten, že pri dopytovaní je medzi nimi operátor „OR“.

Parser inicializujeme konštruktorom:

ParserScopus(Query **query**, List<String> **params**)

Kde:

- **query** znamená to, aká je typ parsovania,
- **params** je zoznam parametrov pre typ parsovania.

Parsovanie spustíme zavolaním metódy parse() od inštancie ParserScopus.

Ukladanie do databázy vykonáme inicializáciou konštruktora:

ScopusXmlLoader(Feed **feed**)

Kde:

- feed je objekt, ktorý je výstupom parsera ParserScopus. Ten zavoláme metódou getFeed z inštancie ParserScopus.

Po inicializácii ScopusXmlLoader je ho potrebné ešte spusti. To sa robí zavolaním metódy persist(int **libraryId** ,String **libraryName**, String **library Sigla**, boolean **newRun**) od inštancie triedy  ScopusXmlLoader.

Kde:

- **libraryId** je id záznamu v tabuke library, ku ktorej sa viaže daný import dát (pri re-importe ostáva prázne)
- **libraryName** je atribút 'name' pre inštanciu z tabuky library
- **librarySigla** je atribút 'sigla' pre inštanciu z tabuky library
- **newRun** je boolean flag, ktorý uruje, i sa ide robi nový import, alebo len re-import

## JAR súbor

Pre spustenie JAR súboru je potrebné ma nainštalovanú JVM (Java Virtual Machine). MInimálna verzia JVM je verzia 1.7.

Spustenie sa robí cez systémovú konzolu zadaním:

**java -jar {JAR súbor} {parametre}**

Pred spustením je potrebné spusti súbor s parametrom "-h", ktorý vypíše na výstup presný popis parametrov pre spustenie parsovania.

# Importer XML Crep

V nasledujúcich kapitolách je uvedená dokumentácia parsera údajov z exportu XML CREP.

# Špecifikácia požiadaviek na import XML Crep

## Úvod

Cieom dokumentu je opísa špecifikova požiadavky na súas projektu Research Rank urený na analýzu a zapísanie informácií do databázy z XML CREP.

### Význam

XML CREP je dokument vydaný centrálnym registrom evidencie publikanej innosti obsahujúci informácie a publikáciách slovenských inštitúcií. Jednotlivé atribúty publikácií sú uchovávané v XML formáte. Spracovávaním tohto dokumentu sa získavajú informácie o publikanej innosti. Použitím týchto informácií s informáciami z iných informaných zdrojov akými sú Scopus alebo WoS je možné po spracovaní Komparátorom uri poet ohlasov danej publikácie. Táto sekcia dokumentácie je smerodajná pre vývojárov pri urení konkrétnej implementácie parsera a determinizácií atribútov, ktoré ukladáme do databázy.

### Rozsah

Produktom je autonómny parser, ktorý dokáže spracováva XML exporty z CREP-u. Cieom je spracováva minimálnu podmnožinu údajov na to, aby sme mohli vykona s nimi deduplikáciu a jednoznanú determináciu vzahov v databáze aj v súvislosti s údajmi z iných zdrojov znalostí.

### Definícia skratiek a termínov

CREP - Centrálny Register Evidencie Publikanej innosti

### Referencie

CREP Web - http://cms.crepc.sk/

### Prehad

Dokument je organizovaný do troch hlavných súastí a to úvod, všeobecné požiadavky a špecifické požiadavky. alej špecifické požiadavky sú organizované podla funkcionality.

## Všeobecné požiadavky

### Perspektíva produktu

Produkt by mal by pomerne nemenný z hadiska asu vzhadom na to, že sa nepredpokladajú asté zmeny v XML formáte. V súvislosti s projektom Research Rank je produkt autonómnou súiastkou s prístupom k ORM. Pomocou ORM potom údaje sú uložené v databáze.

### Funkcie produktu

Produkt implementuje interpretáciu exportov XML CREP, ich preklad do vnútornej formy a nakoniec ich uloženie do databázy pomocou ORM.

### Charakteristika používatea

Používate je technik, ktorý vie ako sa produkt spúša a ako sa používa. Nakoniec ma dostatoné technické znalosti na to, aby vedel pochopi ako produkt funguje.

### Limitácie

Produkt je limitovaný prístupom do databázy a potom údajov, ktoré sú k dispozícií.

### Predpoklady a závislosti

Predpokladá sa, že databáza je online a súbory XML CREP, ktoré chceme analyzova, sú v adresári.

## Konkrétne požiadavky

### XML CREP parser

Vstup: XML CREP
Výstup: Vnútorná forma - java objekt
Vlastnosti: úplnos, spoahlivos
Štandardy: XML 2,0

### Rozhranie ORM

Vstup: Vnútorná forma - java objekt
Výstup: Údaje zapísané v databáze pomocou ORM
Vlastnosti: spoahlivos, úplnos

# Návrh komponentu na import údajov z XML Crep

## Parser

Úlohou parsera je spracova XML export CREP a vlozi údaje do java-objektov tried vytvorených špeciálne pre komponent Parser.

## Loader

Úlohou loadera je vloži údaje z objektov tried XML CREP do objektov tried ORM.

# Implementácia komponentu na import údajov z XML Crep

Celý systém je implementovaný v programovacom jazyku Java. Systém je implementovaný v rámci prostredia JRE (Java Runtime Environment) verzie 1.7 bežiaceho v operanom systéme Windows 7 Professional (64-bit).

Systém sa delí na 2 asti:

- Parser XML CREP
- Importer do PostgreSQL databázy

## Parser

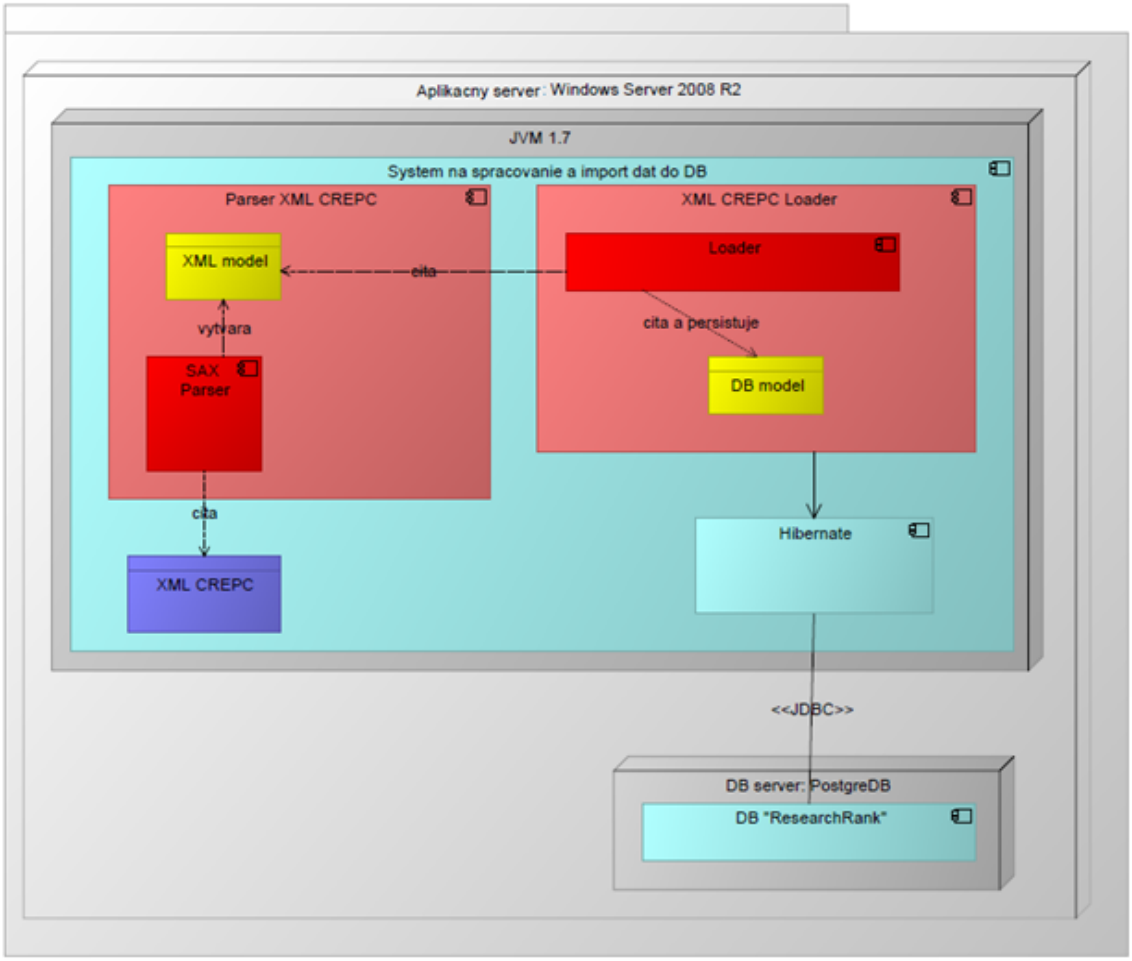
Komponent Parser XML CREP funguje na základe metódy SAX parsovania. ítajú sa postupne riadok po riadku a pri preítaní xml tagu sa spustí funkcia závislá od toho, i ide o zaiatený tag alebo o koncový tag. Pri takomto postupnom ítaní tagov prebieha naítavanie textových reazcov do XML modelu objektov.

Po skonení parsovania sa model objektov pošle do komponentu XML CREPC Loader.

## Loader

Tento komponent využíva technológiu Hibernate. Základom používania tejto technológie bolo vygenerovanie DB modelu, do ktorého sa postupne vkladali dáta z XML modelu a perzistovali priamo popri ukladaní. Implementácia systému sa niesla vo verziovaní.

## Architektúra celého komponentu




# Opis služieb komponentu na import údajov z XML Crep

Opis služieb na vysvetlenie používateovi.

## Parsovanie exportov XML CREP

Ide o spracovávanie exportov vo formáte XML z CREP. Analyzovaný súbor je otvorený, analyzovaný a jeho obsah je preložený do vnútornej formy komponentu.  V druhej etape sú atribúty z vnútornej formy programu zapísané do komponentu ORM, ktorý potom následne tento zápis uloží v databáze.

# XML Crep

tp-databaza-zaloha.dia

## Oficiálna špecifikácia

Crepc.sk

## Analýza

Poas analýzy sme analyzovali oficiálnu dokumentáciu Crep xml zo stránky www.crepc.sk. Výsledkom analýzy je nasledujúci zoznam xml tagov, ktoré parsujeme pomocou vytvoreného parsera. Tabuka obsahuje dátové typy, ktoré sa v tagu môžu vyskytnú, tabuku do ktorej je jeho obsah uložený a názov konkrétneho stpca.

| Názov atribútu | Dátový typ | Formát | Dátový typ po úprave | Poetnos | Názov stpca v DB | Tabuka v DB |
|---|---|---|---|---|---|---|
| Kolekcia_hlavicka | | | | 1 | | |
| sigla | String | Enum | | 1 | sigla | library |
| kolekcia_data | | | | 1 | | |
| kolekcia_data/data_zaznam/ | | | | 0..* | | |
| data_zaznam /zaznam_identifikacia | | | | 1 | | |
| identifikacia_orgID(attr: id_typ) | String | | | 1..* | original_id | publication |
| identifikacia_datumvytvorenia | String | datum_typ | | 1 | date_make | publication |
| identifikacia_datumzmeny | String | datum_typ | | 1 | date_edit | publication |
| identifikacia_druhdokumentu | String | | | 0..1 | code_of_type | document_type |
| identifikacia_ISBN | String | | | 0..* | code | identification_code |
| identifikacia_ISSN | String | | | 0..* | code | identification_code |
| identifikacia_MDT | String | | | 0..* | code | identification_code |
| data_zaznam /zaznam_nazov( attr: format) | | | | 1 | | |
| nazov_hlavny | String | | | 1..* | name | name |
| nazov_subezny( attr: jazyk) | String | | | 0..* | name | name |
| nazov_podnazov | String | | | 0..* | name | name |
| nazov_preklad(attr: jazyk, doplnky, cislo_casti, nazov_casti) | String | | | 0..* | name | name_translation |
| nazov_casti | String | | | 0..* | name | name_of_part |
| nazov_cislocasti | String | | | 0..* | number | name_number_of_part |
| nazov_autorskezahlavie(attr: typ) | String | | | 0..* | heading | name_author_heading |
| | | | | | | |
| data_zaznam /zaznam_autori (typ=primarna, sekundarna) | | | | 0..1 | | |
| autori_osoba(attr: typ) | | | | 0..* | | |
| autori_osoba/osoba_priezvisko | String | | | 1 | surname | author_person |

| | | | | | | |
|---|---|---|---|---|---|---|
| autori_osoba/osoba_meno | String | | | 1 | name | author_person |
| autori_osoba/osoba_pracovisko(attr:pracovisko_typ) | String | Enum | | 1..* | code | person_workplace |
| autori_osoba/osoba_rola alebo ezp_rola (attr: format = UNIMARC,MARC21) | String | | | 1..* | role | person_role |
| autori_osoba/osoba_podiel | Integer | | | 1 | contribution | author_person |
| autori_osoba/osoba_orgID(attr:id_typ) | String | | | 0..* | original_id | author_person |
| autori_osoba/osoba_titul(attr: titul_typ) | String | | | 0..* | title | title |
| autori_korporacia | | | | 0..* | | |
| autori_korporacia/korporacia_nazov | String | | | 1 | name | author_corporation |
| autori_korporacia/korporacia_podcastnazvu | String | | | 0..* | subname | corporation_subname |
| autori_korporacia/korporacia_privlastok | String | | | 0..* | attribute | corporation_attribute |
| autori_korporacia/korporacia_rola | String | | | 1..* | role | corporation_role |
| autori_korporacia/korporacia_orgID | String | | | 0..1 | original_id | corporation |
| autori_korporacia/korporacia_zhromazdenie_cislo | String | | | 0..1 | number | convention |
| autori_korporacia/korporacia_zhromazdenie_miesto | String | | | 0..1 | place | convention |
| autori_korporacia/korporacia_zhromazdenie_datum | String | | | 0..1 | date | convention |
| | | | | | | |
| data_zaznam/zaznam_pracoviska | | | | 1 | | |
| pracoviska_pracovisko | | | | 1..* | | |
| pracoviska_pracovisko/pracovisko_fakulta | String | | | 1 | faculty | publication_workplace |

| | | | | | | |
|---|---|---|---|---|---|---|
| pracoviska_pracovisko/pracovisko_katedra | String | | | 1 | desk | publication_workplace |
| | | | | | | |
| data_zaznam/zaznam_jazyk (attr: format) | | | | 1 | | |
| jazyk_textu | String | | | 1..* | language_text | publication |
| jazyk_originalu | String | | | 0..* | language_original | publication |
| jazyk_resume | String | | | 0..* | language_resume | publication |
| | | | | | | |
| data_zaznam /zaznam_ohlasy | | | | 0..1 | | |
| ohlasy_ohlas | | | | 1..* | | |
| ohlasy_ohlas/ohlas_kategoria | String | | | 1 | category | response |
| ohlasy_ohlas/ohlas_rok | String | | | 1 | year | response |
| ohlasy_ohlas/ohlas_zapis | String | | | 0..1 | entry | response |
| ohlasy_ohlas/ohlas_index | String | | | 0..1 | index | response |
| | | | | | | |
| data_zaznam /zaznam_vydanie | | | | 0..1 | | |
| vydanie_krajina(attr: format) | String | | | 0..* | name | publication_country |
| vydanie_miesto | String | | | 0..* | name | publication_place |
| vydanie_vydavatel | String | | | 0..* | name | publisher |
| vydanie_rok | String | | | 0..* | year | publication_year |
| vydanie_cislo | String | | | 0..* | number | publication_number |
| | | | | | | |
| data_zaznam /zaznam_ine | | | | 0..1 | | |
| ine_rozsah | String | | | 0..1 | range | publication |
| ine_edicia_nazov | String | | | 0..1 | name | edition |
| ine_edicia_zvazok | String | | | 0..1 | edition_volume | publication |
| ine_url(attr: url_typ) | String | | | 0..* | url | url |
| | | | | | | |
| data_zaznam /zaznam_hesla | | | | 0..1 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| heslo_osoba | | | | 0..* | | |
| heslo_osoba/osoba_priezvisko | String | | | 1 | surname | keyword_person |
| heslo_osoba/osoba_meno | String | | | 1 | name | keyword_person |
| heslo_osoba/osoba_orgID | String | | | 0..1 | original_id | keyword_person |
| heslo_osoba/heslo_casti | heslo_casti | | | 0..1 | keyword_of_part_id | keyword_person |
| heslo_korporacia | | | | 0..* | | |
| heslo_korporacia/korporacia_nazov | String | | | 1 | name | keyword_corporation |
| heslo_korporacia/korporacia_podcastnazvu | String | | | 0..* | subname | keyword_corporation |
| heslo_korporacia/korporacia_privlastok | String | | | 0..* | attribute | keyword_corporation |
| heslo_korporacia/korporacia_orgID | String | | | 0..1 | original_id | keyword_corporation |
| heslo_korporacia/heslo_casti | heslo_casti | | | 0..1 | keyword_of_part_id | keyword_corporation |
| heslo_predmet(attr: typ) | | | | 0..* | | |
| heslo_predmet/predmet_nazov | String | | | 1..* | name | keyword_subject |
| heslo_predmet/predmet_orgID | String | | | 0..1 | original_id | keyword_subject |
| heslo_predmet/heslo_casti | heslo_casti | | | 0..1 | keyword_of_part_id | keyword_subject |
| | | | | | | |
| heslo_casti | | | | 0..1 | | |
| cast_tematicka | String | | | 0..1 | part_theme | keyword_of_part |
| cast_geograficka | String | | | 0..1 | part_geographic | keyword_of_part |
| cast_chronologicka | String | | | 0..1 | part_chronological | keyword_of_part |
| | | | | | | |
| data_zaznam/zaznam_vazby | | | | 0..1 | | |
| vazby_zaznam(attr: typ) | | | | 1..* | | |
| zaznam_identifikacia_viazaneho | | | | 0..1 | | |
| zaznam_identifikacia_viazaneho/identifikacia_orgID | String | | | 0..1 | original_id | publication |

| | | | | | | |
|---|---|---|---|---|---|---|
| zaznam_identifikacia_viazaneho/identifikacia_ISBN | String | | | 0..* | code | identification_code |
| zaznam_identifikacia_viazaneho/identifikacia_ISSN | String | | | 0..* | code | identification_code |
| zaznam_nazov | zaznam_nazov | | | 1 | POPIS VYSSIE | |
| zaznam_oznacenie_zvazku | String | | | 0..1 | POPIS VYSSIE | |
| zaznam_vydanie | zaznam_vydanie | | | 0..1 | POPIS VYSSIE | |
| zaznam_autori | zaznam_autori | | | 0..1 | POPIS VYSSIE | |
| ohlasy_ohlas | ohlasy_ohlas | | | 0..1 | POPIS VYSSIE | |

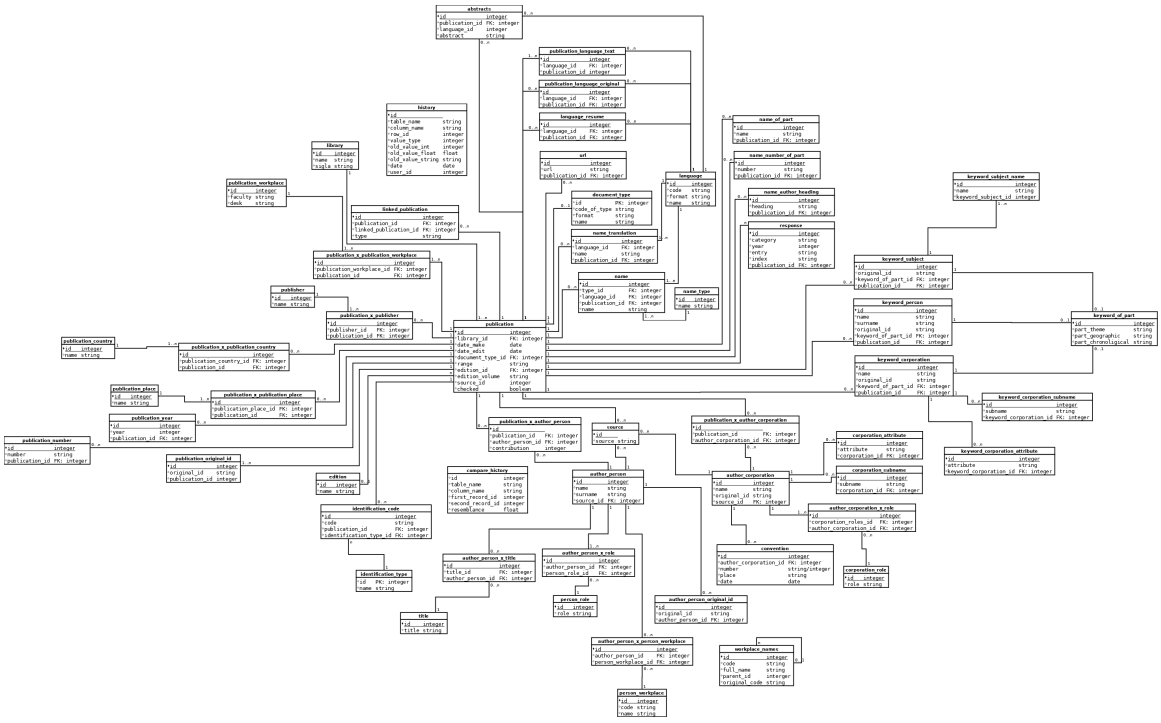
## Dátový model

Presunuté

# Databáza

## Databázový model

Databázový model vytvorený na základe analýzy XML Crep



Na vytvorenie modelu bol použitý program DIA

## Unique constraits

| Názov tabuky | Názvy zahrnutých stpcov |
|---|---|
| library | name |
| | sigla |
| publisher | name |
| publication_country | name |
| publication_place | name |
| edition | name |
| identification_type | name |
| person_role | role |
| corporation_role | role |
| document_type | code_of_type |
| | name |
| language | code |
| | name |
| name_type | name |
| publication | library_id<br>original_id<br>document_type_id<br>language_text<br>language_original<br>language_resume<br>edition_id<br>edition_volume |
| linked_publication | publication_id<br>linked_publication_id<br>type |
| publication_x_publication_workplace | publication_workplace_id<br>publication_id |
| publication_x_publisher | publisher_id<br>publication_id |
| publication_x_publication_country | publication_country_id<br>publication_id |
| publication_x_publication_place | publication_place_id<br>publication_id |
| identification_code | code<br>identification_id<br>identification_type_id |
| title | title |

| | |
|---|---|
| author_person_x_title | author_person_id<br>title_id |
| author_person_x_role | author_person_id<br>person_role_id |
| person_workplace | code<br>name |
| author_person_x_workplace | author_person_id<br>person_workplace_id |
| author_person | name<br>surname<br>original_id |
| author_corporation | name<br>original_id |
| convention | author_corporation_id<br>number<br>place<br>date |
| author_corporation_role | corporation_roles_id<br>author_corporation_id |

# Špecifikácia požiadaviek na databázu

Pri návrhu databázového modelu sme špecifikovali nasledovné požiadavky na navrhovaný model. Pri návrhu sme vychádzali z existujúceho modelu XML Crep.

- možnos uklada publikácie spolu s dodatonými informáciami akými sú viaceré názvy publikácie, preklady názvov, zdrojová knižnica, rok vydania, vydavate, pracovisko, identifikátory (ISBN, ISSN), informácie o autoroch, vedeckých konferenciách ako aj doplnkové informácie (URL adresy dokumentov, kúové slová a podobne)
- možnos uklada vzahy medzi dokumentami (viazané dokumenty, citácie)
- možnos uklada históriu zmien v databáze
- možnos uklada informácie používateov navrhnutého informaného systému ako aj ich prístupových práv

Na základe týchto špecifikácií sme navrhli dátový model.

# ORM

# Úvod

ORM (Objektovo Relaný Mapova) je modulárna súiastka projektu ResearchRank, ktorá je ponúkaná ako samostatná Java (.jar) knižnica pre objektový prístup k tabukám využívanej relanej PostgreSQL databázy.

Vytvorená knižnica **jrank-postgre.jar** je systémovým jadrom, kedže zabezpeuje komunikáciu s databázov aj pre frontend, komparátor ako aj služby na strane servera.

## Špecifikácia požiadaviek

## Návrh komponentu

## Implementácia komponentu

# Špecifikácia požiadaviek na ORM

**Modulárnos**: ORM má by modulárna súiastka, ktoré budú využíva iné moduly systému, napr.: komparátor, frontend, serverové služby, preto je potrebné, aby tento komponent bol implementovaný ako samostatný modul.

**Jednoduchá rozšíritenos**: Zárove modul musí spa požiadavky na jednoduché rozšírenie, alebo zmenu pri úprave dátového modelu.

**Prenositenos a znovupoužitenos**: ORM ako súiastka by mala by prenositená a znovupoužitená pre iné moduly, projekty a riešenia.

**Úplnos**: Taktiež dôležitou požiadavkou je úplnos a robustnos. Knižnica musí poskytova prístup k všetkým tabukám dátového modelu s výnimkou mapovacích tabuliek, ku ktorým má by prístup zamedzený.

# Návrh ORM

Návrh modulu ORM spoíva v odstránení mapovacích tabuliek ku ktorým by mal by zamedzený prístup zo strany programátora a tým zabezpeená kontinuálna konzistencia týchto údajov. alším dôvodom preo mapovacie tabuky nezapadajú do návrhu je odahenie úlohy s nadbytonými entitami, ktoré v žiadnom okamihu nie sú využívané priamo.

Je potrebné, aby ORM zárove poskytovalo prístup k databázovým *session*. Session bude ponúkaná ako klasická JDBC session. Nedostatone špecifikovanou asou je spôsob ich ponúkania. V rámci úvahy sú možné 2 prístupy. Nemenežované session, ktoré musia by spojené s prístupom *session per request*, alebo menežovaný zdroj (managed session pool), ktorý je vhodnejší pre dlhotrvajúce špecifické prístupy k databáze.

Samotné ORM je navrhnuté a implementované na využívanie špecifickej technológie Hibernate a Java Persistence API (JPA), ktorá je síce braná ako štandard, ale v niektorých prípadoch nepostauje programovým požiadavkám a neposkytuje tak robustnú paletu potrebnej funkcionality. V momentálnej implementácií sa využíva *session per request* prístup z dôvodu všeobecnosti implementácie, ale táto skutonos môže by pozmenená v hlavnom konfiguranom súbore pre ORM *hibernate.cfg.xml*.

Pre každú tabuku databázového modelu je vytvorená osobitná POJO trieda s JPA anotáciami s *property access* (anotácie nad gettermi jednotlivých atribútov tried). alším možným prístupom je *field access*, ktorý je možné implementova anotáciami nad jednotlivými atribútmi.

Pre databázové view sú taktiež vytvorené POJO triedy rovnako ako pre tabuky. Pri view ale nie je možné vklada záznamy, ak view využíva agregácie pomocou group by. Preto bolo potrebné vytvori prístup ktorým by bolo možné vklada záznamy, ale zárove ma prístup aj k agregovaným údajom.Takýmito problémovými tabukami a pohadmi (view) boli **publication** (**publication_v**) a **author_person** (**author_person_v** ) boli vytvorené samostatné entity *programované proti rozhraniam*. Rozhranie zabezpeuje jednoduché využívanie tabuky ako aj pohadu bez zmeny tried. Bez takejto úpravy by bolo potrebné implementova druhý dátový model, ktorý by využíval namiesto tabukových entít, entity pohadu.


# Implementácia ORM

Táto as slúži ako manuál pre budúcu prácu s ORM.

ORM je implementovaný ako samostatný projekt, ktorého výstupom je knižnica jrank-postgre.jar, ktorá sa využíva v iných komponentoch architektúry projektu.

## Konfigurácia ORM

konfiguraný súbor je koreovom adresári projektu `sk.fiit.tp.researchrank.postgre` pod názvom `hibernate.cfg.xml`.

| oznaenie | popis |
|---|---|
| `hibernate proerties` | obsahuje atribúty na samotnú konfiguráciu hibernate ORM ako aj prístupové credentials k DB. |
| `c3p0 - connection pool - properties` | obsahuje doplujúce atribúty konfigurácie ORM pre potreby opätovného napojenia a atribúty týkajúce sa potu session, timeout-ov, at. |
| USERS | obsahuje cesty k mapovacím triedam potrebným pre frontend a konfiguráciu používateských atribútov. |

| | |
|---|---|
| CREPC MODEL | obsahuje cesty k základným mapovacím triedam databázového modelu. |
| INSERT | obsahuje cesty k mapovacím triedam používaným na špecifické vkladanie záznamov (autorov, publikácií). |
| VIEWS | obsahuje cesty k mapovacím triedam databázových *views*. |

### Balíky

| **názov balíku** | **popis** |
|---|---|
| codelist | obsahuje potrebné triedy pre jednoduchšie využívanie íselníkov. |
| model | obsahuje potrebné triedy dátového modelu. |
| users | obsahuje triedy používané pri frontende, ako skupiny, používateov a triedy pre konfiguráciu používateských tabuliek. |
| util | obsahuje factory triedu pre generovanie session. |

# Scheduler

## Úvod

Obsahuje skrátenú dokumentáciu implementácie Quartz Schedulera do projektu Research Rank. Vnútorná štruktúra je nasledovná:

1. Par slov k implementácií Quartz Scheduleru
2. Pouzité metódy quartz Scheduleru
3. Definované Joby
4. Prehad súborov

## Pár slov k implementácií Quatz Scheduleru

Bola použitá verzia 2.2.1 Quartz Schedulera. Na tieto úely bol výtvorý konfiguraný súbor pre log4j logger v bin adresáry Scheduler projektu (log4j.properties) a konfiguraný súbor samotného schedulera v koreovom adresári projektu (quartz.properties).

## Použité metódy Quartz Schedulera

Okrem technológie log4j bol Quartz nakonfigurovaný tak, aby sa inštancia schedulera volala SchedulerConf, Id je scheduler_prop. Quartz je nastavený tak, aby používal 5 threadov a dáva okno 5 hodín pre jednotlivé joby na to, aby sa dokonili (pred tým ako ich reštartuje). Joby boli definované rozšírením rozhrania Job. Bol vytvorený job pre WoS loader, Scopus Loader a deduplikanú rutinu. Okrem toho je implementovaná HelloJob Job, ktorý je na urený na testovanie funkcie Quartz schedulera, ak to bude nutné.

Na naplánovanie Jobov sa použil štandardný Cron Trigger. Momentálne sú všetky joby naplánované na stredu o 15:00. Na ukladanie plánu jobov a recovery sa zatial používa RAM Job Store, ktorý sa nevie zotavi po páde serveru. Neskôr bude implementovaný JDBC Job Store, ktorý sa vie zotavi aj po zlyhaní serveru.

Pre viac informácií o vnútorných mechanikách Quartz Schedulera konzultujte Quartz Scheduler manuál.

## Definované Joby

### WoS Job

WoS Job je definovaný v triede WoS_loader_job. Zatria je job prázdny.

### Scopus Job

Scopus Job je definovaný v triede Scopus_loader_job. Zatia je job prázdny.

## Deduplikaný job

Je definovaný v triede BD_deduplication_job. Zatia je job prázdny.

# Prehad súborov

bin/log4j.properties - Konfigurácia log4j loggera. Výstup je nasmerovaný do súbora loging.log a na konzolu.

quartz.properties - Konfigurácie Quartz Schedulera, nastavenia boli spomenuté v sekcií "Použité metódy Quartz Schedulera"

Quartz.java - Trieda, ktorá definuje joby, triggery a štartuje scheduler

HelloJob.java - Trieda urená na testovanie funknosti schedulera

WoS_loader_job.java - Trieda definujúca WoS loader job

Scopus_loader_job.java - Trieda definujúca Scopus loader job

BD_deduplication_job.java - Trieda definujúca deduplikaný job

# o sme nestihli

Plánované bolo napríklad využitie iných zdrojov údajov okrem najznámejších citaných indexoch WoS a Scopus. Žia, tento cie sa nám nepodarilo splni najmä kvôli absencii jedného lena tímu, ktorému bola pridelená táto úloha a neskôr sme presmerovali zdroje na projekte na iné problémy.

Tento projekt slúži ako infraštruktúra pre hodnotenie vedy a výskumu, priom nebolo implementované odhaovanie zložitejších vzahov medzi bibliografickými entitami, ako sú publikácie, autori, inštitúcie, konferencie a iné. V zámere projektu na zaiatku letného semestra bolo vypracovanie riešenia schopného vykonáva hodnotenie vedy a výskumu, súasné riešenie sa však zameriava viac na automatizáciu získavania ohlasov na diela. Preto považujeme hodnotenie vedy a výskumu ako nestihnutú úlohu, ktorú je možné rieši nadstatvbou na nami vyvinutý systém.

Poslednou asou projektu, ktorú sme nestihli je export údajov, ktoré automaticky získavame, deduplikujeme a modelujeme do formy, v ktorej by bola umožnená lepšia archivácia ohlasov. V súasnej dobe je možné pristupova k údajom len prostredníctvom webového rozhrania.

# o sme sa nauili

Práca na tomto projekte obohacuje naše schopnosti a skúsenosti v mnohých smeroch. Vemi cenná je napríklad práca v širšom tíme, keže nie všetci z nás s týmto mali skúsenos. Nauilo nás to, ako robi kompromisy medzi protichodnými názormi, s ím sme sa vo viacerých príležitostiach stretli, prispôsobi náš individuálny pracovný štýl a režím požiadavkám manažerov, ale napríklad aj prijíma názory väšiny a aplikova ich do našej práce. Tímového ducha nebolo zo zaiatku pre niektorých našich lenov ahké prija, ale nakoniec tak všetci urobili, ím sa utužila výkonnos nášho tímu.

Projekt vo všeobecnosti ohohatil naše schopnosti v oblasti manažmentu lenov tímu. Niektoré z týchto lekcií neboli ahké. Ako najobtiažnejšie, ale aj najcennejšie skúsenosti vnímame skúsenosti so zavádzaním štandardov a dobrých praktík do tímu, ktoré sa mnohokrát stretli s odporom. Práve skúsenosti s odporom lenov tímu považujeme za najcennejšie, lebo nás nauili systematicky odpor zo strany kolegov v tíme rieši a nakoniec ho bez demotivaných efektov na kolegov odstráni.

Na druhej strane nás nauili naše role odloži kamarátstva bokom a vníma problémy a vyostrenú diskusiu racionálne. Tým sme sa nauili neprenáša nesúhlas a vyostrené vzahy, ktoré vznikli v procese manažmentu tímu, do naších medziudských vzahov, o považujeme za jednu z najdôležitejších vlastností dobrého manažéra.

Okrem týchto skúseností s prácou s udmi na profesionálnej úrovni nám projekt dal aj vemi vea iných vedomostí, i už ide o manažérske postupy alebo o nové technológie, s ktorými sme sa stretli.

Mnohé technológie, ktoré sme pri vypracovaní riešenia použili, boli pre nás nové a museli sme sa ich naui, ni menej to je iba zlomok toho, o sme sa nauili o cieovej oblasti projektu. Tieto znalosti z oblasti hodnotenia výskumu, spracovávania a porovnávania údajov, hadanania citácií a zdrojov znalostí presahujú tento projekt a bezpochyby niektoré z nich využijeme aj v budúcnosti.