

# Dokumentácia k riadeniu projektu

Textový editor obohatený o grafické prvky

*Tímový projekt*

<b>Autor:</b>	Innovators – tím č.10
<b>Téma projektu:</b>	textový editor obohatený o grafické prvky (TrollEdit)
<b>Vytvorený:</b>	02.10. 2011
<b>Stav:</b>	finálny
<b>Vedúci projektu:</b>	Ing. Peter Drahoš, PhD.
<b>Vedúci tímu:</b>	Bc. Lukáš Turský
<b>Členovia tímu:</b>	Bc. Marek Brath Bc. Adrián Feješ Bc. Maroš Jendrej Bc. Jozef Krajčovič Bc. Ľuboš Staráček
<b>Kontakt:</b>	tp-team-10@googlegroups.com

## Obsah

1	Úvod.....	1
1.1	Prehľad dokumentu .....	1
2	Ponuka.....	2
2.1	Predstavenie členov tímu .....	2
2.2	Znalosti a zručnosti študentov (Znalosti).....	3
2.2.1	Motivácia .....	3
2.2.2	Koncept riešenia .....	4
2.3	Digitálne divadlo (Divadlo) .....	6
2.3.1	Motivácia .....	6
2.3.2	Koncept riešenia .....	6
2.4	Textový editor obohatený o grafické prvky (TextEdit) .....	7
2.4.1	Motivácia .....	7
2.4.2	Koncept riešenia .....	7
3	Zoradenie všetkých tém podľa priority .....	9
4	Rozvrh členov tímu pre zimný semester .....	10
5	Plán.....	11
5.1	Hrubý plán pre zimný semester.....	11
5.2	Aktualizovaný plán pre zimný semester .....	12
5.3	Hrubý plán pre letný semester.....	14
6	Úlohy členov tímu .....	16
6.1	Dlhodobé manažérske úlohy .....	16
6.2	Dlhodobé vývojárske úlohy .....	16
6.3	Krátkodobé úlohy.....	16
6.3.1	Autori jednotlivých častí dokumentácie riadenia .....	18
6.3.2	Autori jednotlivých častí technickej dokumentácie.....	19
7	Firemná kultúra .....	21
7.1	Použité podporné prostriedky v tíme .....	21
7.2	Manažment rozvrhu a plánovania .....	21
7.2.1	Riadenie iterácie v nástroji Redmine .....	22
7.2.2	Proces riadenia iterácie .....	22
7.2.3	Koordinovanie činností pomocou Redmine .....	22
7.2.4	Sledovanie plnenia plánu .....	24
7.3	Manažment rizík.....	25
7.3.1	Identifikácia rizík.....	25
7.3.2	Klasifikácia rizík.....	26
7.3.3	Manažment chýb.....	26

7.3.4	Proces zatvorenia chyby v Redmine .....	29
7.4	Manažment komunikácie .....	29
7.4.1	Komunikačné prostriedky .....	30
7.4.2	Komunikačný plán .....	30
7.4.3	Manažment požiadavky na zmenu (Change request) .....	32
7.4.4	Roly a zodpovednosti účastníkov .....	32
7.4.5	Životný cyklus požiadavky na zmenu .....	32
7.4.6	Metodika vykonávania jednotlivých procesov požiadavky na zmenu .....	33
7.5	Manažment podpory vývoja .....	35
7.5.1	Roly a zodpovednosti .....	35
7.5.2	Prevzatie aktuálnej verzie zdrojového kódu .....	36
7.5.3	Implementácia funkcionality .....	37
7.5.4	Zlúčenie zmien forknutých repozitárov do hlavného repozitára .....	40
7.5.5	Riešenie konfliktov v zdrojovom kóde .....	41
7.5.6	Riadenie nasadzovania softvéru .....	42
7.6	Manažment kvality .....	42
7.6.1	Refaktoring .....	43
7.6.2	Prehliadky kódu .....	43
7.6.3	Programovanie v pároch .....	43
7.7	Manažment testovania .....	43
7.7.1	Integračné testovanie .....	44
7.7.2	Testovanie pomocou unit testov .....	45
7.7.3	Konfigurácia testovacieho prostredia .....	46
7.7.4	Vytváranie testovacích scenárov .....	47
7.7.5	Vytváranie unit testov .....	48
7.7.6	Testovanie pomocou unit testov .....	50
7.7.7	Zhodnotenie výsledkov testovacích scenárov .....	51
7.8	Manažment monitorovania .....	51
7.8.1	Monitorovanie projektu v nástroji Redmine .....	51
7.8.2	Monitorovanie úloh v Redmine .....	53
7.9	Manažment tvorby dokumentácie .....	54
7.9.1	Roly a zodpovednosti .....	54
7.9.2	Základné pravidlá pri písaní dokumentácie .....	54
7.9.3	Postup tvorby dokumentácie .....	55
7.9.4	Vytváranie zápisníc zo stretnutí .....	55
7.10	Štýl programovania .....	56
7.10.1	Vytváranie názvov .....	56

7.10.2	Odsadenia .....	57
7.10.3	Písanie zátvoriek .....	57
7.10.4	Písanie komentárov pre potreby nástroja doxygen .....	58
7.10.5	Písanie metód .....	60

## Zoznam obrázkov

Obr. 1 Časová os hrubého plánu .....	14
Obr. 2 Diagram aktivít procesu riadenia iterácií .....	22
Obr. 3 Okno pre modifikáciu údajov jednotlivých úloh .....	23
Obr. 4 Okno na vytvorenie novej činnosti .....	24
Obr. 5 Stavový diagram chyby .....	27
Obr. 6 Postupnosť procesov na hornej úrovni.....	28
Obr. 7 Komunikačné kanály v tíme .....	30
Obr. 8 Životný cyklus požiadavky na zmenu (CHR).....	33
Obr. 9 Vytvorenie novej požiadavky na zmenu v Redmine .....	33
Obr. 10 Pracovný postup s integračným manažérom.....	36
Obr. 11 Stratégia vetvenia pre hlavný repozitár.....	39
Obr. 12 Stratégia vetvenia pre forknuté repozitáre .....	39
Obr. 13 Procesy manažmentu testovania .....	45
Obr. 14 Nastavenie doplnkových knižníc .....	47
Obr. 15 Prehľad stráveného času na projekte.....	52
Obr. 16 Priebežný stav úloh .....	53
Obr. 17 Sledovanie stavu úloh .....	54
Obr. 18 Proces tvorby dokumentácie .....	55

## Zoznam tabuliek

Tab. 1 Hrubý plán pre zimný semester .....	11
Tab. 2 Aktualizovaný plán na zimný semester.....	12
Tab. 3 Hrubý plán pre letný semester .....	14
Tab. 4 Dlhodobé úlohy členov tímu.....	16
Tab. 5 Dlhodobé vývojárske úlohy .....	16
Tab. 6 Krátkodobé úlohy členov tímu.....	16
Tab. 7 Autori jednotlivých častí dokumentácie riadenia.....	19
Tab. 8 Autori jednotlivých častí technickej dokumentácie .....	19
Tab.9 Nástroje použité v tíme .....	21
Tab. 10 Zoznam identifikovaných rizík .....	25
Tab. 11 Zoznam klasifikovaných rizík.....	26
Tab. 12 Role a zodpovednosti pre manažment chýb.....	26
Tab. 13 Zoznam procesov na hornej úrovni.....	27
Tab. 14. Výber komunikačných prostriedkov pre potreby tímového projektu .....	30
Tab. 15 Komunikačný plán v tíme .....	31
Tab. 16 Role a zodpovednosti účastníkov manažmentu komunikácie.....	32
Tab. 17 Roly a zodpovednosti v rámci manažmentu podpory vývoja .....	35
Tab.18 Značky používané pri písaní správ vykonaných zmien v nástroji git .....	38
Tab. 19 Tabuľka základných procesov manažmentu testovania.....	45
Tab. 20 Roly a zodpovednosti v manažmente testovania .....	46
Tab. 21 Roly a zodpovednosti manažmentu dokumentácie .....	54

## **Zoznam príloh**

Príloha A:      zápisnice zo stretnutí

Príloha B:      preberacie protokoly

Príloha C:      pravidlá dokumentácie



# **1 Úvod**

Účelom tohto dokumentu je zdokumentovať riadenie tímu v rámci projektu textový editor obohatený o grafické prvky na predmete Tímový projekt. Projekt je riešený tímom č.10 s názvom „Innovators“ počas dvoch semestrov v akademickom roku 2011/2012.

## **1.1 Prehľad dokumentu**

Na začiatku sa nachádza ponuka, ktorú sme vypracovali pri výbere témy projektu. Podarilo sa nám získať jednu z troch nami preferovaných tém. V tejto časti sú zároveň krátko predstavení členovia tímu. Nasleduje prerozdelenie rolí v rámci tímu a krátkodobé úlohy, ktoré sme doteraz riešili. Ďalšou kapitolou je plán projektu na zimný semester. Nasledujúca kapitola sa zaoberá firemnou kultúrou a nami používanými podpornými prostriedkami. Poslednou kapitolou sú kópie zápisníc zo stretnutí.

## 2 Ponuka

Nasleduje ponuka tak, ako sme ju odovzdali okrem titulnej strany:

### 2.1 Predstavenie členov tímu

#### **Bc. Jozef Krajčovič**

Absolvent odboru Informatika na FPV UCM v Trnave. Vypracoval bakalársku prácu na tému „Návrh lekárskeho informačného systému ambulancie“. Má skúsenosti s vývojom webových ako aj desktopových aplikácií. Používa väčšinu technológií a nástroje z dielne Microsoft. Zaujíma sa o tvorbu a vývoj používateľských rozhraní ako aj riadenie a motivovanie ľudí v tíme. Ovláda technológií: HTML/XHTML, PHP, JavaScript, C#, Visual Basic, C/C++, Java, Mysql, MSSQL a Oracle, .Net Framework (WPF, XAML, WCF), WindowsPowerShell, XML.

#### **Bc. Adrián Feješ**

Je absolventom študijného odboru Informatika na FIIT STU. Vo svojej bakalárskej práci sa venoval procesu refaktorizácie zdrojových kódov a jej nástrojovej podpore. Výsledkom práce bol nástroj vo forme Eclipse plug-inu, podporujúci rozpoznávanie a označovanie antivzorov v kóde. Má skúsenosti s vývojom aplikácií hlavne v programovacom jazyku Java. Svoje vedomosti ďalej rozvíja aj v praxi, kde pracuje ako Java programátor a zaoberá sa vývojom podnikových aplikácií. Ovláda technológií: C/C++, Java SE/EE, XML, XMLSchema, XPath, SQL, JavaScript,

#### **Bc. Lukáš Turský**

Vyštudoval obor Informatika na FIIT STU. Počas štúdia sa zamerával najmä na vývoj aplikácií pre platformu Java SE a FX. V rámci bakalárskej práce analyzoval využitie Modelom riadenej architektúry pri tvorbe softvéru, pričom výstupom bolo úplné namodelovanie web aplikácie a jej následne implementovanie pre platformu Java EE (využitie Spring, Struts, Hibernate). Popri štúdiu získal skúsenosti v oblasti analýzy rizík a administrácie bezpečnosti bankových aplikácií. V dohľadnej dobe by sa chcel ďalej zamerať na vývoj webových aplikácií a rozšíriť znalosť databáz v rámci predmetu Pokročilé Databázové technológií.

#### **Bc. Luboš Staraček**

Absolvent študijného odboru Informatika na STU FIIT v Bratislave, vypracoval bakalársku prácu na tému „Štúdia realizácie zmien aspektovo-orientovaným spôsobom na úrovni modelu“. Za najpodstatnejšie získané zručnosti považuje osvojenie si objektovo a aspektovo

orientovaného vývoja softvéru, metódy paralelného programovania a princípy umelej inteligencie. V rámci mimoškolskej činnosti vytvoril funkčnú web aplikáciu v jazyku JavaFX.

**Bc. Maroš Jendrej**

Absolvent študijného odboru Informatika na STU FIIT v Bratislave, vypracoval bakalársku prácu na tému „Manažovanie dokumentov“. Má skúsenosti s vývojom desktopových aplikácií pre platformu JAVA SE. Počas bakalárskeho štúdia si osvojil základy programovania v rôznych programovacích jazykoch a tiež získal znalosti o tvorbe softvérových systémoch. Po ukončení bakalárskeho štúdia sa zamestnal na pozícii QA/Tester v spoločnosti zaoberajúcej sa vývojom počítačových hier. V dohľadnej dobe by sa chcel hlbšie oboznámiť s počítačovou grafikou a dizajnom používateľského rozhrania. Ovláda technológie: HTML/XHTML, XML, JAVA SE, C/C++, Assembler, UML, CUDA, MPI, OMP

**Bc. Marek Brath**

Absolvent študijného odboru Informatika na FPV UCM v Trnave, vypracoval bakalársku prácu na tému „Programovanie v Jave“. Používa hlavne prostredie Eclipse na vytváranie desktopových aplikácií. Ovláda technológie: Java, C++, C#, PHP, HTML, XHTML, CSS, PHP, SQL

## **2.2 Znalosti a zručnosti študentov (Znalosti)**

### **2.2.1 Motivácia**

V dnešnom svete plnom informácií je nájdenie a zostavenie tímu ľudí, obzvlášť takých ktorí sa takmer nepoznajú, často veľmi ťažko riešiteľný problém. Vidíme to aj teraz na nás, študentoch, že problémom je nedostatok a roztrúsenosť informácií o našich kolegoch. Veríme tomu, že my sami si možno časom začneme hovoriť, že zadelenie v rámci daného tímu nie je najideálnejšie.

Práve preto nás nadchla myšlienka vytvorenia centrálnej databázy schopností a znalostí jednotlivých študentov, ktorej plné využitie by mohlo siahať aj ďaleko do komerčnej sféry. Ved' pokiaľ by bol takýto systém dostatočne používateľsky prívetivý a interaktívny, mohol by uľahčiť prácu nielen učiteľom, ale určite aj neskôr študentom napr. pri hľadaní zamestnania.

Veľkú výhodu vidíme najmä v tom, že sami by sme boli motivovaný zlepšovať sa a týmto spôsobom ovplyvňovať svoje ohodnotenie v rámci systému.

Na druhej strane nielen pre profesorov ale aj vedúcich prác je to spôsob ako efektívne zostaviť tím podľa jeho preferencií a teda si môže rozhodnúť aké kvality by mal takýto tím, prípadne aj jednotliviec spĺňať. Taktiež je to informačný spôsob ako efektívne využiť potenciál každého jednotlivca v rámci vytváraného tímu a touto cestou aj zvýšenie miery na jeho budúci úspech.

Rozhodne by sme chceli stáť u zrodu takéhoto systému, lebo veríme tomu, že na to máme ako tím všetky predpoklady a bolo by veľmi zaujímavé pokiaľ by sa takýto systém podarilo reálne nasadiť do prevádzky.

### **2.2.2 Koncept riešenia**

Vzhľadom na to, že väčšina nášho tímu má väčšie či menšie skúsenosti s konceptom a využívaním Java EE technológií, tak by sme chceli práve túto platformu využiť pre vytvorenie požadovanej client-server webovej aplikácie, ku ktorej budú môcť používatelia voľne pristupovať.

V rámci riešenia vidíme viacero hľadísk na ktoré bude potrebné sa zamerať. V rámci prezentačnej vrstvy je to nutnosť využiť interaktívne zaujímavý framework, ktorý by sa použil pre vytvorenie používateľského prostredia, a ktorý bude na použitie dostatočne priateľský. Dobré vieme, že je to jediná časť s ktorou pracuje používateľ priamo a mnohokrát rozhoduje o zániku či úspechu systému. V tomto smere ešte nemáme jasno, o aký framework by šlo a teda by bolo nutné spraviť krátku analýzu.

Pre jednoduchšiu orientáciu, by mohlo vyhľadávanie a najmä pridávanie znalostí mať v hlavnej časti u každého študenta len nejaké zhrnutie jeho znalostí – správne zvolené väčšie celky, ktoré by zoskupovali podobné znalosti, napr. aká je miera technických znalostí, spoločenských schopností, využívanie určitých typov nástrojov, takisto by bolo zaujímavé mať aj indikátor ako sa študentovi darí v škole.

Keďže bolo spomenuté, že na takomto projekte sa už v minulosti pracovalo, tak by sme chceli využiť niektoré časti tohto riešenia, ktoré už sú dostatočne vyriešené a sústrediť sa na podstatnejšie veci, ktoré sú spomenuté nižšie.

Z pohľadu aplikačnej logiky chceme venovať úsilie vytvoreniu mechanizmu, ktorý by vedel na základe daných schopností používateľa ďalej odvodiť bázu znalostí, ktorá by určite zlepšila šance pri filtrovaní a výbere. Reprezentácia znalostí by mala spĺňať požiadavky ako

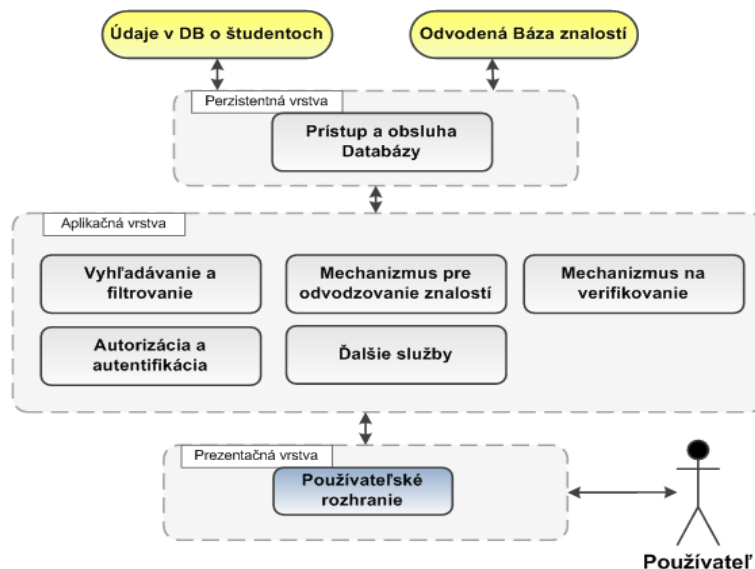
je rýchle vyhľadávanie a porovnávanie údajov, a preto spôsob reprezentácie musí byť jednoduchý a jednotný pre rôzne typy poznatkov a zručností.

Chceli by sme implementovať schopnosť automatizovane zadávať jednotlivé znalosti študentov, napr. ich hromadné pridávanie. V tomto smere by sa pre ich získavanie vo veľkej miere dalo využiť aj bodové hodnotenie v AIS pri jednotlivých zadaniach v rámci predmetov.

Vzhľadom na to, že pôjde o systém do ktorého bude mať prístup viacero skupín ľudí, navrhujeme vytvorenie viacerých úrovní prístupov (rolí) a k nim priradiť možné akcie, prípadne časti systému na ktoré by im tieto role dávali prístup. Teda logicky podľa toho do akej kategórie daný používateľ patrí, také akcie budú môcť vykonávať v systéme. Tu by bolo vhodné implementovať jednoduché pridávanie právomocí v rámci jednotlivých rolí, napr. odklikávanie akcií, alebo výber z listu.

Plánujeme implementovať spôsob overovania a kontroly študentmi zadávaných schopností, tak čo sa týka určitej miery verifikovateľnosti zadávaných schopností študentov, jeden spôsob vidíme v možnosti nechať zaslať požiadavku oprávnenej osobe na overenie.

Rozhodnutie koho potrebujeme zohnať a aké by mali byť požiadavky na študenta/tým by mali byť ponechané čisto na zadávateľa.



## 2.3 Digitálne divadlo (Divadlo)

### 2.3.1 Motivácia

Ovládanie softvéru pomocou ľudských pohybov a gest, bez nutnosti použitia klávesnice alebo myši, je samo o sebe veľmi zaujímavá a aktuálna téma. Obzvlášť, keď je tento projekt zameraný na tvorbu umeleckého diela, kde je zároveň výsledok tejto tvorby premietaný na plátno v reálnom čase. Myslíme si, že práca na takomto projekte bude pre nás zaujímavá, bude nás baviť, a tiež, v neposlednom rade, získame množstvo užitočných skúseností v zaujímavej oblasti IT.

Najmä kvôli týmto dôvodom náš tím zaujala táto téma, a chceli by sme sa podrobnejšie oboznámiť s možnosťami, ako využiť senzor Kinect na rozpoznávanie obrazovej informácie, pohybov a gest. Pokúsili by sme sa o vytvorenie originálneho riešenia, v ktorom by umelec pred plátnom pomocou svojho vlastného tela vytváral obraz. Ponúkli by sme mu na tvorbu diela nástroje, ktoré sú bežne v štandardných programoch na PC (Skicár, Adobe Photoshop, MyPaint...). Išlo by o vnorenie umelca do počítačovej reality, kde by aj bez tableta, či myšky mohol maľovať obraz.

### 2.3.2 Koncept riešenia

Standalone aplikácia pre platformu Windows XP, Vista a 7. Využívali by sme existujúce knižnice na detekciu pohybov a gest človeka. V rámci prípravy na vytvorenie tejto ponuky sme tiež vykonali analýzu niekoľkých video prezentácií umiestnených na portáli youtube, napríklad o používaní senzora Kinect na ovládanie konzoly X-Box 360 a podobne, čo nám môže poskytnúť veľa inšpirácie pri navrhovaní riešenia pre potreby tohto projektu.

Naše navrhované riešenie by mohlo byť akýmsi wrapperom na ľudské telo, pomocou ktorého sa bude vytvárať obraz, v prípade požiadavky na stereo projekciu sme pripravení pokúsiť sa o vytvorenie výstupného obrazu v troch dimenziách. Využili by sme pri tom rozpoznávanie hĺbky obrazu, ktorá nám je týmto senzorom ponúknutá. Prípadne, ak senzor Kinect umožňuje aj rozoznávanie hlasu, mohlo by stať za zváženie umožniť aj ovládanie kombináciou ľudských gest a hlasu. Tu by ale bolo dôležité zabezpečiť, aby bolo možné nastaviť ovládanie hlasom tak, že by príkazy hlasom mohla dávať iba oprávnená osoba, a nie ktokoľvek. Inak by mohli vznikať komplikácie, kde by počas používania tohto softvéru napríklad na prezentáciu mohol do tejto prezentácie vstupovať ktokoľvek z publika, čo je nežiaduce.

Jednou z alternatív pre overenia riešenia by mohlo ísť o vytvorenie prívetivého ovládania pre existujúcu open source aplikáciu MyPaint, slúžiacu na tvorbu obrázkov. Jej výhodou je jednoduché a minimalistické používateľské rozhranie, neobmedzený canvas bez nutnosti zmeny jeho rozmerov a schopnosť využívania grafického tabletu. Rovnako ako pri kreslení keď využívame grafický tablet by sme mohli využiť aj senzor Kinect, ktorý by za pomoci hĺbky obrazu dokázal určiť kedy umelec naťahuje ruku a teda snaží sa v obraze kresliť. Intenzitu kreslenia by sme určovali ako hlboko umelec ponorí svoju ruku do obrazu, je to podobne ako sa na grafickom pere určuje stupeň prítlaku. Rozlišovali by sme tiež pravú a ľavú ruku, jedna by bola ako štetec a za pomoci druhej ruky by umelec vytváral gestá takto by prepínal medzi typmi štetcov, nastavoval farbu alebo inými funkciami. Trup umelca bude dynamickým stredom a maximálne natiahnutá ruka dopredu bude zaznamenané ako maximálna intenzita prítlaku štetca, nemôže sa tu stať niečo také, že štetec bude reagovať neprimerane.

Taktiež by mohlo byť zaujímavé implementovať ovládanie gestami do softvéru na tvorbu, respektíve spúšťanie prezentácií. Napríklad do open source programu OpenLP, ktorý okrem spúšťania prezentácií umožňuje aj prehrávanie videí, vytváranie a zobrazovanie galérií obrázkov a ďalšie. <http://openlp.org/en/features>.

## **2.4 Textový editor obohatený o grafické prvky (TextEdit)**

### **2.4.1 Motivácia**

Tato téma nás predovšetkým zaujala svojou myšlienkou vytvoriť akýsi multiplatformový grafický editor, ktorý využije grafické prvky na zvýraznenie štruktúr textu pomocou grafických blokov a tým podporili myšlienku „literate programming“, čo v súčasnosti veľa podobných riešení dosiaľ neexistuje a taktiež fakt, že práca na editore je z 50% už hotová. Ďalšou motiváciou pre nás je, že sa pri tomto projekte môžeme rozšíriť svoje znalosti a zručnosti o nové technológie a postupy v danej doméne, ktorá je pre nás zaujímavá. Uvedomujeme si, že s danou doménou nemáme veľa praktických skúseností čo sa môže zdať ako nevýhoda, ale opak je však pravdou a o to viac to bude pre nás väčšia výzva, aby sme vytvorili kvalitný produkt, ktorý bude úspešný a mohol by presadiť aj v praxi.

### **2.4.2 Koncept riešenia**

Cieľom tohto projektu bude pokračovať vo vývoji existujúceho multiplatformového editora (TrollEdit), ktorý bol vytvorený predchádzajúcim tímom „UFOPAK“. Naším zameraním pre

editor bude rozšírenie stavajúcej funkcionality pre reálne nasadenie editora do praxe. Najväčšiu zmenou bude vylepšenie používateľského rozhrania, ktoré v súčasnom editore nie je tak ako u podobných editor čo sa týka dizajnu nezaujímaví t.j. klasický dizajn „*ala notepad*“.

Pri implementácii budeme predovšetkým vychádzať z už použitých technológií ako knižnica Qt, skriptovací jazyk Lua a podobne plus niektoré nami zvolené technológie, ktoré sa rozhodneme použiť po podrobnej analýze súčasného editora.

Čo sa týka rozšírenia funkcionality plánujeme implementovať tieto vylepšenia:

- Možnosti „undo“/ „redo“.
- Detekcia pachov kódu.
- Možnosť rozšírených nastavení priamo v editore
- Určitý druh fulltextového vyhľadávania s prípadnou optimalizáciou na najčastejšie vyhľadávané výrazy.
- Možnosť exportovania súboru do iných formátov (XML, WORD)
- Schopnosť detegovať určité ukazovatele v zdrojovo kóde ako index udržiavateľnosti, cyklomatická zložitosť, hodnoty fan in a fan aut, ktoré by boli zobrazené v určitej tabuľke.

Taktiež plánujeme čo najvhodnejšie použiť známe návrhové vzory, aby sme zabezpečili vysokú modularitu systému a tým umožnili neskoršie pridávanie a modifikovanie funkcionality.

Ohľadom spomínaného dizajnu používateľského rozhrania plánujeme vďaka podpore Qt modulu pre vývojové prostredie Visual Studio použiť najmodernejšie technológie ako WPF (Windows presentation foundations), XML.

Tieto technológie nám umožnia navrhnuť si dizajn podľa vlastnej fantázie bez zdĺhavého programovania pri ktorom by sme museli použiť rôzne grafické knižnice čo v tomto prípade odpadá. Plánujme návrh dizajnu používateľského rozhrania v štýle „Office“ t.j. použiť dobre známí „*Ribbon*“, ktorý je stále častejšie používaní v desktopových aplikáciách.

Veríme, že nami navrhnuté riešenie vo finálnej verzii bude kvalitný produkt, ktorý nájde uplatnenie v praxi.



### 3 Zoradenie všetkých tém podľa priority

Priorita	Názov témy	Číslo témy
1.	Znalosti a zručnosti študentov (Znalosti)	13
2.	Digitálne divadlo (Divadlo)	3
3.	Textový editor obohatený o grafické prvky (TrollEdit)	11
4.	Štatistický preklad voľného textu (Preklad)	9
5.	Inteligentná hra pre mobilné zariadenia (MobHra)	8
6.	Rozvrhový systém novej FIIT (Rozvrhy)	12
7.	Plagiáty na webe (Plagiáty)	4
8.	Simulácia davu (Dav)	15
9.	Personalizované odporúčanie (Odporúčanie)	5
10.	Osobný manažment fyzickej aktivity pomocou mobilných zariadení (Aktivita)	2
11.	Editovanie viacrozmerného grafu prepojenia informácií v dokumentoch (Dokumenty)	16a
12.	Virtuálna FIIT (VirtFIIT)	14
13.	RoboCup - tretí rozmer (RoboCup)	7
14.	Webový editor pre TeX (WebEdit)	10
15.	Tvorba "ľahko" sémantického obsahu pre adaptívny webový (výučbový) portál (ALEF)	6
16.	Imagine Cup 2012: Game Design (ICup2012) - pridelená	1
17.	3D UML (3D UML)	16b

#### 4 Rozvrh členov tímu pre zimný semester

[illegible]

## 5 Plán

Po dôkladnej analýze dostupných metodík sme sa rozhodli, že budeme vyvíjať inkrementálnym a iteratívnym spôsobom. Naše rozhodnutie ovplyvnili najmä výhody takéhoto prístupu k vývoju. Plán projektu samozrejme musíme prispôbiť vlastnostiam inkrementálneho a iteratívneho vývoja. Celý projekt sa rozloží na dobre definované a použiteľné časti (inkreментy), ktoré postupne integrujeme do celku. Získame tak prehľadnejší a ľahšie manažovateľný vývojový proces. Jednotlivé časti budeme iteratívne vyvíjať, čo môže vo veľkej miere zvýšiť kvalitu výsledkov.

### 5.1 Hrubý plán pre zimný semester

Tab. 1 Hrubý plán pre zimný semester

Týždeň	Úlohy
1.	Vytvorenie tímu Rozdelenie rolí v tíme
2.	Výber preferovaných tém Vypracovanie a odovzdanie ponúk
3.	Vytvorenie webovej stránky, plagátu a loga tímu Analýza a výber podporných prostriedkov Analýza stavu predošlého projektu (preštudovanie technickej dokumentácie a dokumentácie riadenia)
4.	Špecifikácia požiadaviek Analýza použitých technológií a nástrojov
5.	Analýza zdrojových kódov aplikácie TrollEdit Analýza použitých technológií Vytvorenie predbežnej verzie technickej dokumentácie a dokumentácie riadenia
6.	<b>1. kontrolný bod</b> Prepracovanie špecifikácie požiadaviek Určenie priority jednotlivých požiadaviek Diskusia o možnostiach implementácie jednotlivých funkcionalít
7.	Analýza implementácie určených funkcionalít
8.	Návrh implementácie funkcionality Návrh GUI Odovzdanie dokumentácie analýzy, špecifikácie a návrhu riešenia
9.	Implementácia prototypu fáza I.
10.	<b>2. kontrolný bod</b> Testovanie a oprava chýb fázy I Implementácia prototypu fáza II Kontrola stavu technickej dokumentácie a dokumentácie riadenia
11.	Implementácia prototypu fáza III Testovanie a oprava chýb fázy II, III Vypracovanie finálnej verzie technickej dokumentácie a dokumentácie riadenia

12.	<b>3. kontrolný bod</b> Odovzdanie prototypu spolu s dokumentáciou
13.	Prezentácia výsledkov práce. Vypracovanie priebežnej správy pre TP Cup

## 5.2 Aktualizovaný plán pre zimný semester

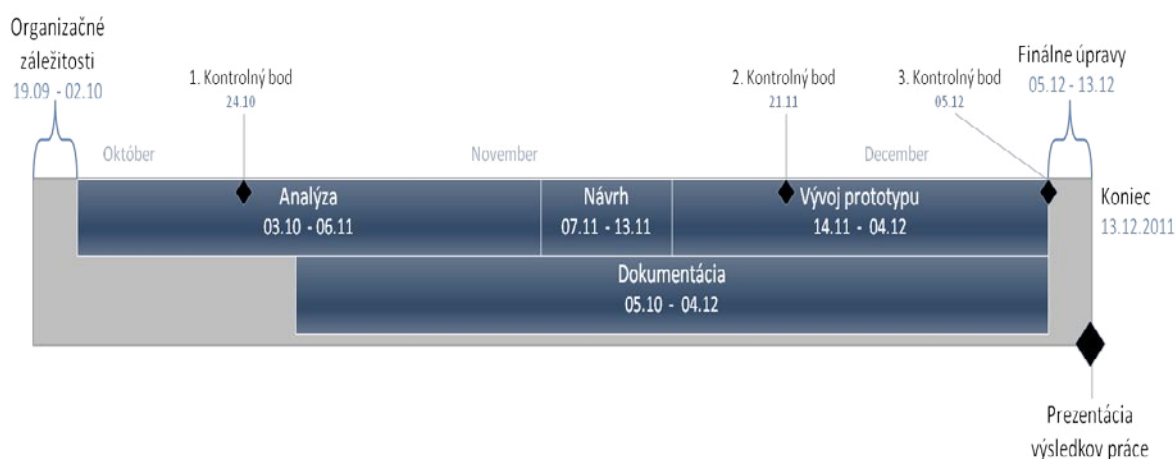
Prvá verzia plánu bola vytvorená v 3. týždni semestra bez podrobnejšej analýzy súčasného stavu riešenia. Postupne sme zistili, že niektoré existujúce funkcionality aplikácie nefungujú správne a pre implementáciu nových funkcionalít budú potrebné zmeny v existujúcich riešeniach. Kvôli uvedeným skutočnostiam v 6. týždni semestra bola potrebná aktualizácia plánu. Podrobnejšia analýza stavu projektu nám umožnila identifikovať jednotlivé úlohy, preto aktualizovaný plán je už podrobnejší a obsahuje aj zodpovedných za vykonanie úloh.

Tab. 2 Aktualizovaný plán na zimný semester

Týždeň	Úlohy	Zodpovedný
1.	Vytvorenie tímu (celý tím) Rozdelenie rolí v tíme	Všetci Všetci
2.	Výber preferovaných tém Vypracovanie a odovzdanie ponúk	Všetci Všetci
3.	Vytvorenie webovej stránky tímu Vytvorenie plagátu tímu Analýza a výber nástrojov na manažovanie projektu a zdrojových kódov Analýza stavu predošlého projektu (preštudovanie technickej dokumentácie a dokumentácie riadenia) Prvé neoficiálne stretnutie s vedúcim projekt	Lukáš Jozef Všetci Všetci Všetci
4.	Získavanie požiadaviek vedúceho projektu Analýza získaných požiadaviek Špecifikácia použitých technológií a nástrojov Preštudovanie dokumentácií a zdrojových kódov aplikácie TrollEdit Analýza Qt frameworku Analýza programovacieho jazyka Lua a možnosti využitia LuaJIT	Všetci Všetci Všetci Všetci Adrián, Jozef, Lukáš Luboš, Maroš, Marek
5.	Špecifikácia požiadaviek Pokračovanie v analýze a študovaní zdrojových kódov aplikácie Pokračovanie v analýze a študovaní Qt frameworku Pokračovanie v analýze a študovaní jazyka Lua Vytvorenie predbežnej verzie technickej dokumentácie a dokumentácie riadenia Vytvorenie jednoduchkej statickej stránky v anglickom jazyku pre potreby prezentovania projektu na GitHub-e	Všetci Všetci Všetci Adrián, Jozef, Lukáš Luboš, Maroš, Marek Všetci Adrián
6.	<b>1. kontrolný bod</b> Analýza a vyhodnotenie súčasného stavu projektu Kontrola stavu webovej stránky a repozitára tímu	Všetci Všetci

	Kontrola stavu úloh, bugov a termínov v Redmine a GitHub Kontrola špecifikácie požiadaviek Určenie priority jednotlivých požiadaviek Diskusia o možnostiach implementácie jednotlivých funkcionalít Definovanie a rozdeľovanie úloh Prvotný návrh implementácie určených funkcionalít	Všetci Všetci Všetci Všetci  Všetci Všetci
7.	Analýza a návrh paralelného spracovania syntaxe v QT Analýza a návrh funkcionalít UNDO, REDO Návrh spracovania syntaktického stromu v jazyku LUA Návrh a implementácia klávesových skratiek Analýza jazyka QML pre integráciu používateľského rozhrania Konzultácia s vedúcim projektu o návrhu a prípadná ukážka implementácie	Lukáš Adrián Ľuboš, Maroš Marek Jozef  Všetci
8.	Experimentovanie s paralelizmom v Qt Experimentovanie s funkcionalitou UNDO/REDO Experimentovanie s QML Experimentovanie s funkcionalitou pre shortcuts Experimentovanie so spracovaním syntaktického stromu v jazyku LUA Konzultácia s vedúcim projektu o dosiahnutých výsledkoch Odovzdanie dokumentácie analýzy, špecifikácie a návrhu riešenia	Lukáš Adrián Jozef Marek Ľuboš, Maroš  Všetci Všetci
9.	Analýza a vyhodnotenie súčasného stavu projektu Kontrola stavu a aktualizácia webovej stránky a repozitára tímu Kontrola stavu a aktualizácia úloh, bugov a termínov v Redmine a GitHub Ukážka výsledkov experimentovania Pokračovanie v experimentovaní Finalizácia a odovzdanie prihlášky na TP Cup	Všetci Všetci  Všetci  Všetci Všetci Všetci
10.	<b>2. kontrolný bod</b> Implementácia paralelného spracovania syntaxe v QT Implementácia funkcionality UNDO/REDO Implementácia shortcuts Implementácia spracovania syntaktického stromu v jazyku LUA Predvedenie implementovaných funkcionalít vedúcemu Analýza možných vylepšení implementovaných funkcionalít	Lukáš Adrián Marek Ľuboš, Maroš  Všetci Všetci
11.	Implementácia a testovanie paralelného spracovania syntaxe v QT Implementácia a testovanie funkcionality UNDO/REDO Implementácia a testovanie shortcuts Implementácia a testovanie spracovania syntaktického stromu v jazyku LUA Predvedenie implementovaných funkcionalít vedúcemu	Lukáš Adrián Marek  Ľuboš, Maroš  Všetci

12.	<b>3. Kontrolný bod</b>	
	Kontrola stavu webovej stránky a repozitára tímu	Všetci
	Kontrola stavu úloh, bugov a termínov v Redmine a GitHub	Všetci
	Finalizácia a integrácia technickej dokumentácie a dokumentácie riadenia	Všetci
	Finalizácia a integrácia implementovaných funkcionalít	Všetci
13.	Odovzdanie produktu spolu s dokumentáciou	Všetci
	Prezentácia výsledkov semestra	Všetci
	Vypracovanie priebežnej správy pre TP Cup	Všetci
		Všetci



Obr. 1 Časová os hrubého plánu

## 5.3 Hrubý plán pre letný semester

Tab. 3 Hrubý plán pre letný semester

Týždeň	Úlohy
1.	Zistenie aktuálneho stavu projektu Integrácia existujúcich riešení Testovanie aplikácie po integrácii
2.	Identifikovanie nových požadovaných funkcionalít
3.	Analýza možnosti riešenia nových funkcionalít
4.	Návrh riešenia nových funkcionalít
5.	Implementácia nových funkcionalít
6.	<b>1. kontrolný bod</b> Implementácia nových funkcionalít Analýza možnosti vylepšenia implementovaných funkcionalít Kontrola stavu technickej dokumentácie a dokumentácie riadenia Kontrola stavu repozitára na GitHubu Kontrola stavu projektu v Redmine
7.	Implementácia a testovanie nových funkcionalít Prezentácia dosiahnutých výsledkov vedúcemu Príprava na prezentáciu projektu na IIT.SRC 2012
8.	Testovanie a integrácia nových funkcionalít

<b>9.</b>	Analýza súčasného stavu projektu a identifikovanie nových funkcionalít prípadne vylepšenie existujúcich
<b>10.</b>	<b>2. kontrolný bod</b> Kontrola stavu technickej dokumentácie a dokumentácie riadenia Kontrola stavu repozitára na GitHube Kontrola stavu projektu v Redmine
<b>11.</b>	Dopracovanie chýbajúcich funkcionalít a oprava prípadných „bugov“ Testovanie a dokumentácia aplikácie
<b>12.</b>	<b>3. kontrolný bod</b> Odovzdanie aplikácie spolu s dokumentáciou Prezentácia výsledkov práce

## 6 Úlohy členov tímu

Táto kapitola obsahuje informácie o rolách jednotlivých členov tímu a krátkodobých úlohách, ktoré sme riešili pri tvorbe projektu v zimnom semestri.

### 6.1 Dlhodobé manažérske úlohy pre ZS

Jednotliví členovia tímu zastávajú nasledujúce dlhodobé úlohy na projekte

Tab. 4 Dlhodobé úlohy členov tímu

Člen tímu	Zodpovednosti
<b>Bc. Lukáš Turský</b>	Manažér tímu Manažér komunikácie Kontrolór dokumentácie Správca webového sídla
<b>Bc. Jozef Krajčovič</b>	Zástupca vedúceho tímu Manažér podpory vývoja Manažér tvorby dokumentácie
<b>Bc. Adrián Feješ</b>	Manažér rozvrhu a plánovania
<b>Bc. Maroš Jendrej</b>	Manažér kvality a testovania
<b>Bc. Ľuboš Staráček</b>	Manažér rizík
<b>Bc. Marek Brath</b>	Manažér monitorovania

### 6.2 Dlhodobé vývojárske úlohy pre ZS

Tab. 5 Dlhodobé vývojárske úlohy

Člen tímu		Zodpovednosti
<b>Bc. Lukáš Turský</b>	<b>Qt</b>	Integrácia paralelizmu do nástroja
<b>Bc. Jozef Krajčovič</b>		Používateľské rozhranie GUI
<b>Bc. Adrián Feješ</b>		Textové operácie – Undo/Redo, Copy/Paste
<b>Bc. Maroš Jendrej</b>	<b>Lua</b>	Práca nad AST pomocou C API
<b>Bc. Ľuboš Staráček</b>		Práca nad AST pomocou C API
<b>Bc. Marek Brath</b>		Zabudovanie Shortcuts

### 6.3 Krátkodobé úlohy

Rozpis krátkodobých úloh, ktoré boli riešené v zimnom semestri sú popísané v nasledujúcej tabuľke.

Tab. 6 Krátkodobé úlohy členov tímu

ID	Popis úlohy	Zodpov. osoba	Dátum vzniku	Dátum ukončenia	Stav
01.	Vytvorenie webovej stránky pre	Lukáš	28.09.2011	29.09.2011	dokončená



	prezentáciu tímu				
02.	Vybrať podporné nástroje pre vývoj	Všetci	28.09.2011	30.09.2011	dokončená
03.	Preskúmanie nástroja TrollEdit a porovnanie ho s ďalšími nástrojmi a navrhnutie zmeny funkcionality	Všetci	28.09.2011	30.09.2011	dokončená
04.	Vytvorenie loga a plagátu tímu	Jozef	28.09.2011	29.09.2011	dokončená
05.	Spojzadenie virtuálneho stroja umiestneného	Lukáš	29.09.2011	30.09.2011	dokončená
06.	Podrobná analýza možnosti, ktoré ponúkajú technológie QT,Lua	Všetci	12.10.2011	18.10.2011	dokončená
07.	Vytvorenie dokumentácie štýlu programovania	Jozef	12.10.2011	16.10.2011	dokončená
08.	Analýza možnosti ako implementovať funkcionality (2 módy, ShortCuts, Undo/Redo)	Všetci	12.10.2011	-	rozpracovaná
09.	Urobiť redesign používateľského rozhrania nástroja TrollEdit	Jozef	12.10.2011	-	odložená
10.	Vytvoriť logo pre TrollEdit	Jozef	12.10.2011	22.10.2011	dokončená
11.	Uppdate webovej stránky (doplnenie technológií, fotky vedúceho)	Lukáš	19.10.2011	23.10.2011	dokončená
13.	Prepojenie GitHub z Redmine	Marek	19.10.2011	23.10.2011	dokončená
14.	Analýza doxygen	Lukáš	19.10.2011	25.10.2011	dokončená
15.	Vytvorenie prihlášky na TP CUP	Jozef	19.10.2011	25.10.2011	dokončená
16.	Vytvorenie predbežnej verzie technickej dokumentácie	Jozef	19.10.2011	25.10.2011	dokončená
17.	Vytvorenie predbežnej dokumentácie riadenia	Jozef	19.10.2011	25.10.2011	dokončená
18.	Prenesenie súborov z SVN na GitHub	Jozef	19.10.2011	23.10.2011	dokončená
19.	Vytvorenie statickej web stránky pre TrollEdit	Adrián	19.10.2011	25.10.2011	dokončená
20.	Podrobná analýza možnosti, ktoré ponúkajú technológie QT	Jozef, Lukáš, Adrián	19.10.2011	25.10.2011	dokončená
21.	Podrobná analýza možnosti, ktoré ponúkajú technológie Lua	Marek, Ľuboš, Maroš	19.10.2011	25.10.2011	dokončená
22.	Zrušenie starého repozitára a presun taskov	Marek	26.10.2011	26.10.2011	dokončená
23.	RefaktORIZÁCIA zdrojového kódu podľa nami zadefinovaného štýlu programovania	Adrián	26.10.2011	31.10.2011	dokončená
24.	Podrobná analýza súčasného stavu TrollEditu	Lukáš	26.10.2011	2.11.2011	dokončená
25.	Špecifikácia požiadaviek	Jozef	26.10.2011	28.10.2011	dokončená
26.	Návrh funkcionality pre UNDO/REDO	Adrián	26.10.2011	01.11.2011	dokončená

27.	Analýza a návrh paralelného spracovania syntaxu v Qt	Lukáš	-	-	rozpracovaná
28.	Analýza spracovania syntaktického stromu v jazyku LUA	Ľuboš, Maroš	26.10.2011	02.11.2011	dokončená
29.	Analýza vytvárania shortcutov	Jozef	26.10.2011	01.11.2011	dokončená
30.	Doplnenie štýlu programovania o syntax Doxygen a jeho použitie	Lukáš	26.10.2011	29.10.2011	dokončená
31.	Analýza a návrh mapovania objektov z C++ do jazyka LUA	Ľuboš, Maroš	26.10.2011	29.10.2011	dokončená
32.	Napísať návod buildovania TrollEditu	Marek	26.10.2011	27.10.2011	dokončená
33.	Analýza a návrh paralelného spracovania syntaxe v QT	Lukáš	02.11.2011	09.11.2011	dokončená
34.	Návrh funkcionality pre UNDO/REDO	Adrián	02.11.2011	09.11.2011	dokončená
35.	Návrh spracovania syntaktického stromu v jazyku LUA	Ľuboš, Maroš	02.11.2011	09.11.2011	dokončená
36.	Návrh funkcionality pre shortcuts	Marek	02.11.2011	09.11.2011	dokončená
37.	Vytvorenie Use Case diagramov	Jozef	02.11.2011	05.11.2011	dokončená
38.	Analýza jazyka QML pre integráciu používateľského rozhrania	Jozef	02.11.2011	05.11.2011	dokončená
39.	Experimentovanie s paralelizmom v Qt	Lukáš	09.11.2011	16.11.2011	dokončená
40.	Experimentovanie s funkcionalitou UNDO/REDO	Adrián	09.11.2011	16.11.2011	dokončená
41.	Experimentovanie s QML,	Jozef	09.11.2011	16.11.2011	dokončená
42.	Experimentovanie s funkcionalitou pre shortcuts	Marek	09.11.2011	16.11.2011	dokončená
43.	Experimentovanie s vytvorením C štruktúry v Lua a s prácou nad ňou	Maroš, Ľuboš	09.11.2011	16.11.2011	dokončená
44.	Finalizácia dokumentácii pre odovzdanie	Všetci	09.11.2011	16.11.2011	dokončená
45.	Implementácia paralelizmu	Lukáš	23.11.2011	12.12.2011	rozpracovaná
46.	Implementácia funkcionality UNDO/REDO	Adrián	23.11.2011	12.12.2011	rozpracovaná
47.	Implementácia GUI pomocou QML	Jozef	23.11.2011	12.12.2011	rozpracovaná
48.	Implementácia funkcionality shortcuts	Marek	23.11.2011	12.12.2011	rozpracovaná
49.	Implementácia práce nad AST pomocou Lua C API	Maroš, Ľuboš	23.11.2011	12.12.2011	rozpracovaná

### 6.3.1 Autori jednotlivých častí dokumentácie riadenia

Nasledujúca tabuľka zobrazuje príspevky jednotlivých členov tímu k dokumentácii riadenia v zimnom semestri.

Tab. 7 Autori jednotlivých častí dokumentácie riadenia

Kapitola	Autor
1 Úvod	Jozef Krajčovič
2 Ponúka	
2.2 Znalosti a zručnosti študentov	Lukáš Turský & Adrián Feješ
2.3 Digitálne divadlo	Maroš Jendrej & Ľuboš Staráček
2.4 Textový editor obohatený o grafické prvky	Jozef Krajčovič
5 Plán	Adrián Feješ
6 Úlohy členov tímu	Jozef Krajčovič
7 Firemná kultúra	
7.1 Použité podporné prostriedky v tíme	Jozef Krajčovič
7.2 Manažment rozvrhu	Adrián Feješ
7.3 Manažment rizík	Ľuboš Staráček
7.4 Manažment komunikácie	Lukáš Turský
7.5 Manažment podpory vývoja	Jozef Krajčovič
7.6 Manažment kvality	Maroš Jendrej
7.7 Manažment testovania	Maroš Jendrej
7.8 Manažment monitorovania	Marek Brath
7.9 Manažment tvorby dokumentácie	Jozef Krajčovič
7.10 Štýl programovania	Jozef Krajčovič & Lukáš Turský
Príloha A – zápisnice zo stretnutia	Všetci
Príloha B – preberací protokol	Jozef Krajčovič
Príloha C – pravidlá pri tvorbe dokumentácie	Lukáš Turský

### 6.3.2 Autori jednotlivých častí technickej dokumentácie

Tab. 8 Autori jednotlivých častí technickej dokumentácie

Kapitola	Autor
1 Úvod	Jozef Krajčovič
2 Analýza	
2.1 Existujúce riešenia editorov	Marek Brath
2.2 Analýza predchádzajúceho riešenia nástroja TrolEdit	Lukáš Turský
2.3 Analýza použitých technológií	Jozef Krajčovič
2.4 Analýza spracovania syntaktického stromu	Maroš Jendrej & Ľuboš Staráček
3 Špecifikácia požiadaviek	
3.1, 3.2 Funkcionálne a nefunkcionálne požiadavky	Jozef Krajčovič + kontrola všetci
3.3 Analýza požiadaviek na paralelizmus	Lukáš Turský
4 Návrh riešenia	
4.1 Diagram prípadov použitia	Jozef Krajčovič
4.2 Architektúra programu	Jozef Krajčovič
4.3 Návrh GUI	Jozef Krajčovič
4.4 Návrh funkcionality UNDO/ REDO	Adrián Feješ
4.5 Návrh funkcionality pre shortcuts	Marek Brath
4.6 Návrh spracovania syntaktického stromu	Maroš Jendrej & Ľuboš Staráček

4.7 Návrh riešenie paralelizmu	Lukáš Turský
5 Implementácia prototypu	
5.1 Popis prototypu	-
5.2 Ďalšie experimenty	-
5.3 Experimentovanie a implementácia undo/ redo	Adrián Feješ
5.4 Experimentovanie a implementácia paralelizmu	Lukáš Turský

## 7 Firemná kultúra

Táto kapitola opisuje firemnú kultúru ktorú sme si definovali v rámci tímového projektu. Definované sú tu metodiky, ktorými by sa mal každý člen tímu riadiť počas práce na projekte.

### 7.1 Použité podporné prostriedky v tíme

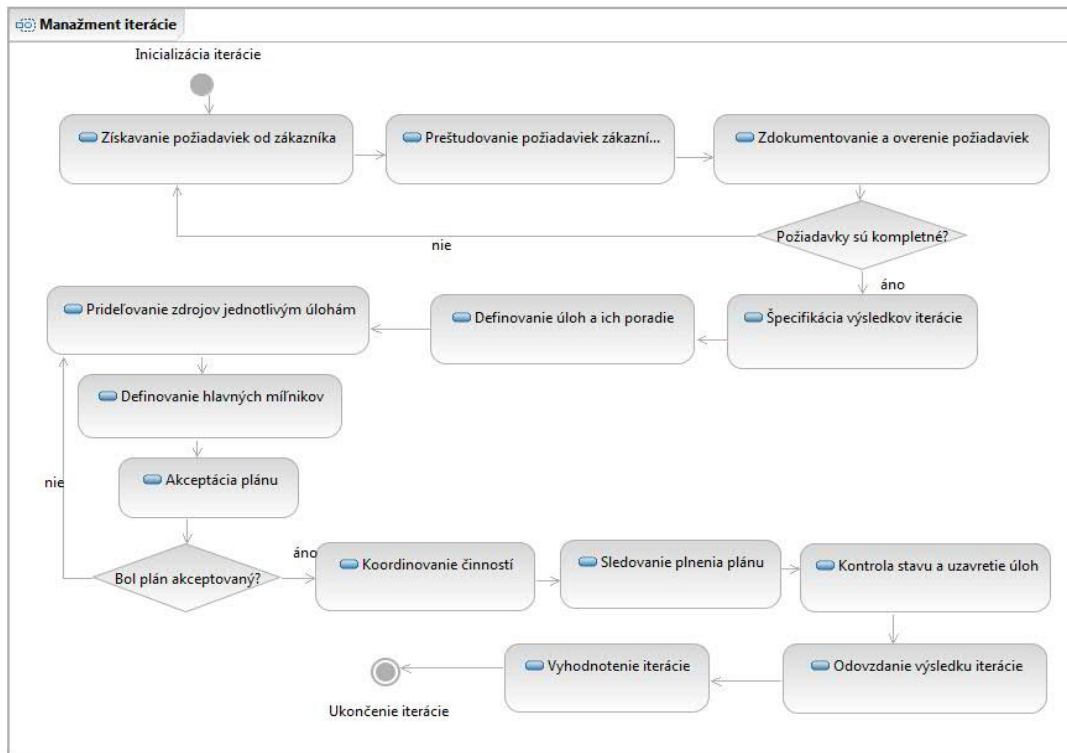
Keďže vyvíjame open-source projekt tak sme sa rozhodli použiť taktiež nástroje pre manažment a samotný vývoj v projekte z rady open-source, tieto nástroje sú popísané v Tab. 9.

Tab. 9 Nástroje použité v tíme

Manažment úloh, chýb, nápadov	Redmine (dostupné na <a href="https://redmine.fiit.stuba.sk/">https://redmine.fiit.stuba.sk/</a> )
Monitorovanie projektu	Redmine (dostupné na <a href="https://redmine.fiit.stuba.sk/">https://redmine.fiit.stuba.sk/</a> )
Manažment verzií	Git (dostupné na <a href="https://github.com/Innovators-Team10">https://github.com/Innovators-Team10</a> )
Komunikácia v tíme	Google groups, Facebook
Zostavovanie softvéru	Cmake
Vývojové prostredie	Qt creator, Qt modul pre Visual Studio
Testovanie	QtUnit

### 7.2 Manažment rozvrhu a plánovania

Náš tím sa rozhodol ísť iteratívnym a inkrementálnym spôsobom vývoja. Táto kapitola poskytuje opis procesu manažovania iterácie. Samozrejme pri manažovaní iterácií musíme zohľadniť určité obmedzenia ako je nedostatok času alebo obmedzená dostupnosť zdrojov.



Obr. 2 Diagram aktivít procesu riadenia iterácií

### 7.2.1 Riadenie iterácie v nástroji Redmine

Účelom tejto kapitoly je poskytnúť presný postup na riadenie iterácie projektu v nástroji *Redmine*. *Redmine* poskytuje funkcionality na riadenie a sledovanie projektov ako napr. vytváranie a pridelenie úloh, kalendár a *Ganttov diagram*. V opise je používaná anglická verzia nástroja, takže pre jednotlivé funkcionality použijeme anglické názvy. Pre riadenie projektov v našom tíme používame len nasledujúce typy činností: *Task*, *Bug*, *Change Request*.

### 7.2.2 Proces riadenia iterácie

Proces riadenia iterácie pozostáva z 2 hlavných krokov:

- Koordinovanie činností
- Sledovanie plnenia plánu

### 7.2.3 Koordinovanie činností pomocou Redmine

1. Manažér plánovania sa prihlási do nástroja *Redmine*
2. Vyberie príslušný projekt
3. Hneď v sekcii „Overview“ má možnosť získavať základné údaje o stave iterácie (počet Taskov/Bugov/Change Requestov)

4. Po skontrolovaní stavu projektu klikne na záložku „Activity“, kde vidí posledné činnosti členov tímu. V pravej hornej časti obrazovky si vyberie aké typy činností chce vidieť. Výber typov činností potvrdí stlačením tlačidla „Apply“. Potom sa na obrazovke zobrazia len vybrané typy činností, zoradené podľa dátumov vytvorenia.
5. V prípade ak má pripomienku k nejakej činnosti upravuje údaje danej činnosti (Obr.3)
  - a) Klikne na činnosť, ktorú chce modifikovať
  - b) Potom klikne na tlačidlo „Update“
  - c) Vykonáva požadované zmeny. Pri modifikovaní má nasledujúce možnosti: zmena riešiteľa úlohy, zmena priority úlohy, zmena stavu úlohy, zmena predpokladaného dátumu ukončenia, pridanie poznámky pre riešiteľa úlohy
  - d) Modifikáciu údajov potvrdí stlačením tlačidla „Submit“

Obr. 3 Okno pre modifikáciu údajov jednotlivých úloh

6. Po skontrolovaní nedávnych činností buď
  - a) Klikne na záložku „Issues“ ak potrebuje získať ďalšie údaje o stave projektu. Potom sa zobrazí obrazovka na nastavenie vyhľadávacích kritérií. Manažér najprv vyberie, že aký typ činností chce vidieť. Po nastavení typu vyberie aj stav požadovaných činností. Pomocou tlačidla „Apply“ dá zobrazit' výsledky. V prípade, že chce vidieť podrobnejšie informácie alebo modifikovať vybratú činnosť vykonáva postupnosť krokov č.5.
  - b) Alebo klikne na záložku „New issue“ kde má možnosť na vytváranie nových Taskov/Bugov/Change Requestov. Pri vytvorení najprv vyberie typ činností Task/Bug/Change Request. Do poľa „Subject“ zadá krátky, výstižný názov činnosti. Potom do poľa „Subject“ zadá krátky opis činnosti, ktorý jasne

definuje východiskový stav, cieľový stav a spôsob ako sa má tento cieľový stav dosiahnuť. Po špecifikovaní činnosti sa táto činnosť pridelí konkrétnemu riešiteľovi. Stav sa nastaví na „Assigned“ a nastaví sa jej priorita (preferovaná priorita je „High“). Po nastavení základných údajov sa ešte nastaví dátum predpokladaného ukončenia (Due Date). Nová činnosť sa vytvorí pomocou tlačidla „Create“ alebo „Create a continue“.

The screenshot shows the 'New Task' form in Redmine. The 'Tracker' is set to 'Bug'. The 'Subject' is 'Zle nastavena implicitna gramatika'. The 'Parent task' field is empty. The 'Description' field contains three lines of text: 'Implicitna gramatika je nastavena na jazyk XML.', 'Nastav implicitnu gramatiku na jazyk C tak, aby po otvorení kazdeho dokumentu sa pouzil tato gramatika.', and 'Po oprave implicitna gramatika bude nastavena na jazyk C'. The 'Status' is 'Assigned', 'Priority' is 'High', and 'Assignee' is 'Adrian Fejes'. The 'Start date' is '2011-11-10' and the 'Due date' is '2011-11-12'. The 'Estimated time' is '0' hours and the '% Done' is '0 %'.

Obr. 4 Okno na vytvorenie novej činnosti

7. Celý cyklus riadenia iterácie sa pravidelne opakuje počas vykonávania iterácie

## 7.2.4 Sledovanie plnenia plánu

Tento proces sa vykonáva paralelne s koordinovaním činností.

1. Manažér plánovania sa prihlási do nástroja Redmine.
2. Po získaní základných informácií o projekte prejde na sekciu „Gantt“, kde sa zobrazí Ganttov diagram na sledovanie vývoja iterácie. Vyberie príslušný typ činnosti a jej stav. V prípade potreby nastaví aj obdobie, pre ktoré chce vidieť vývoj projektu. Ganttov diagram sa zobrazí stlačením tlačidla „Apply“. V prípade ak manažér chce vidieť podrobnejšie informácie o jednotlivých činnostiach klikne na názov činnosti.
3. Po získaní informácie o vývoji iterácie prejde na záložku „Calendar“. Tu vidí jednotlivé činnosti priradené ku konkrétnemu dátumu. Pre filtrovanie informácie vykonáva tie isté kroky ako v kroku č.2.
4. Pre získanie podrobnejšieho prehľadu o aktuálnom stave iterácie v pravom hornom rohu obrazovky má k dispozícii „Details“ a „Report“.



- a) Po kliknutí na „Details“ sa zobrazí okno, v ktorom manažér vyberie príslušné obdobie vyplnením dátumov „From“ a „To“. Detail sa zobrazí pomocou tlačidla „Apply“.
- b) Po kliknutí na „Report“ sa zobrazí okno, v ktorom manažér vyberie príslušné obdobie vyplnením dátumov „From“ a „To“. Potom vyberie, či chce zobrazíť report pre činnosti, pre členov projektu alebo pre projekty. Report sa zobrazí pomocou tlačidla „Apply“.

## 7.3 Manažment rizík

Manažment rizík sa musel v tomto prípade prispôbiť špecifikám práce na projekte v malom tíme v školskom prostredí. To ovplyvňuje napríklad nemožnosť nastania rizika nedodržania rozpočtu, a podobne. Najprv sú v kapitole 7.3.1 identifikované riziká, ktoré môžu nastať. Potom sú v kapitole 7.3.2 tieto riziká (subjektívne) ohodnotené, podľa toho s akou pravdepodobnosťou môžu nastať a aké veľké dopady na úspech projektu budú mať, ak nastanú. Miera dopadu je dôležitejšia, preto viac ovplyvňuje celkové ohodnotenie rizika. Tu sú riziká zoradené zostupne podľa ich celkového ohodnotenia.

### 7.3.1 Identifikácia rizík

Tab. 10 Zoznam identifikovaných rizík

ID	Identifikované riziko	Možný spúšťač	Ošetrenie rizika
1	Odchod člena tímu	Prílišné nároky na niektorého člena tímu, nezvládnutie tlaku, strata motivácie	Rovnomerné rozkladanie úloh na všetkých členov tímu (zamedzenie rizika), prerozdelenie úloh na zvyšných členov (prijatie rizika)
2	Nedodržanie termínov	Nedisciplinovanosť členov tímu	Stanovenie dostatočne skorých termínov, prísna kontrola splnenia úloh vedúcim tímu (zamedzenie rizika)
3	Nedodržanie požiadaviek	Nesprávne pochopenie alebo neúplnosť požiadaviek zákazníka	Častá komunikácia so zákazníkom (vedúcim tímu), skoré prototypovanie (zamedzenie rizika)
4	Nezhody medzi členmi tímu	Členovia tímu budú mať rozdielne názory na určitú časť projektu, spôsob implementácie alebo riadenia	Častá komunikácia v rámci tímu, dodržiavanie firemnej kultúry a vypracovaných metodík (zamedzenie rizika)
5	Nezvládnutie novej technológie	V projekte bude nasadená technológia, s ktorou žiadny člen tímu nemá skúsenosti	Dôkladná analýza danej technológie, experimentovanie s technológiou (zamedzenie rizika), konzultácia s expertom na danú technológiu (prenos rizika)

### 7.3.2 Klasifikácia rizík

Ako vzorec na výpočet celkového ohodnotenia rizika bol použitý:

$$celkove\_ohodnotenie = pravdepodobnost * 3 + miera\_dopadu * 7$$

Tab. 11 Zoznam klasifikovaných rizík

ID	Identifikované riziko	Pravdepodobnosť nastania [0-10]	Miera dopadu [0-10]	Celkové ohodnotenie [0-100]
1	Odchod člena tímu	2,5	9	70,5
2	Nedodržanie požiadaviek	3	8,5	68,5
3	Nezvládnutie novej technológie	2	8	62
4	Nedodržanie termínov	4	5	47
5	Nezhody medzi členmi tímu	9	1,5	28,5

### 7.3.3 Manažment chýb

Manažment chýb pokrýva procesy od vzniku chyby a jej priradeniu určitej osobe, po jej vyriešenie a samotné uzatvorenie v Redmine.

Tab. 12 Role a zodpovednosti pre manažment chýb

Rola	Zodpovednosť
Programátor	Vyvíja aplikáciu
	Opravuje reportované chyby v aplikácii
	Označuje vyriešené chyby v Redmine
Tester	Testuje aplikáciu
	Reportuje a klasifikuje chyby v Redmine
	Uzatvára chyby v Redmine

Chyba v zdrojovom kóde, ktorá sa vyskytne v prostredí nášho tímu, sa môže dostať do stavov: priradená, vyriešená alebo zatvorená. Všetky možné spôsoby dosiahnutia týchto stavov sú znázornené v diagrame Obr. 5.

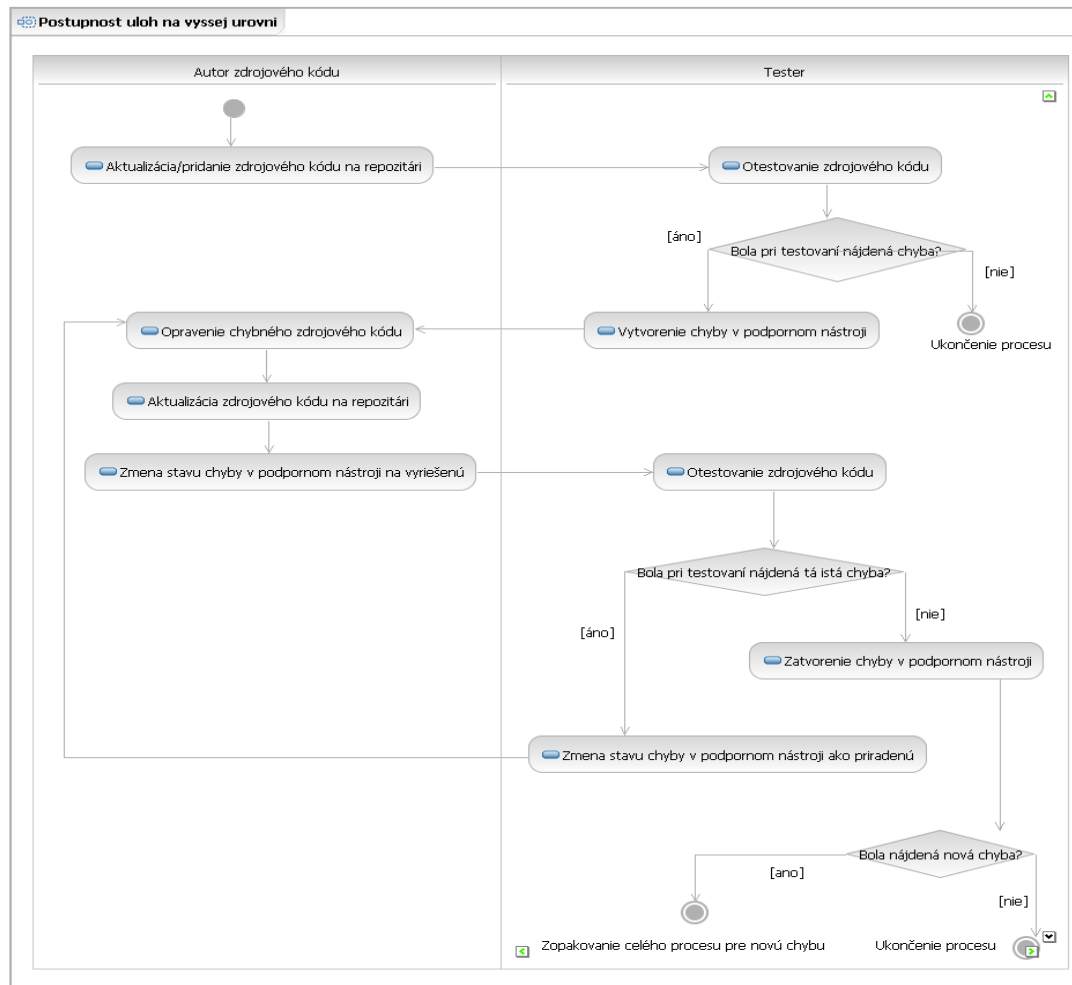


Obr. 5 Stavový diagram chyby

Jednotlivé procesy, ktorými je možné tieto definované stavy dosiahnuť, sú uvedené v tabuľke4. Presná postupnosť týchto procesov je znázornená pomocou diagramu aktivít na .

Tab. 13 Zoznam procesov na hornej úrovni

Krok	Názov
1	Aktualizácia zdrojového kódu na repozitári
2	Otestovanie zdrojového kódu
3	Vytvorenie chyby v Redmine
4	Opravenie zdrojového kódu
5	Zmena stavu chyby v Redmine na vyriešenú
6	Zmena stavu chyby v Redmine na priradenú
7	Zatvorenie chyby v Redmine



Obr. 6 Postupnosť procesov na hornej úrovni

Ako podporný nástroj na manažment práce je v našom tíme používaný nástroj Redmine, ktorý je dostupný cez webové rozhranie, teda cez internetový prehliadač. Konkrétne je používaná školská verzia dostupná na adrese <https://redmine.fiit.stuba.sk>.

### 7.3.3.1 Vyplnenie formuláru pre novú chybu

Vstup:	požiadavka na vytvorenie novej chyby
Výstup:	vytvorená chyba v nástroji Redmine, priradená autorovi chybného kódu
Zodpovedný:	tester
Dokumentácia:	žiadna

Pre zobrazenie formuláru na vytvorenie novej chyby je potrebné na stránke prehľadu aktuálneho projektu kliknúť na tab novej udalosti (New Issue) na hornej lište. Ďalej je potrebné vyplniť všetky vstupné hodnoty nasledovným spôsobom.

Typ úlohy (issue tracker) je nutné zvoliť Bug, čo v nástroji Redmine reprezentuje chybu. Pre predmet (Subject) chyby je potrebné naformulovať taký text, ktorý bude čo najlepšie

vystihovať povahu chyby v zdrojovom kóde. Formulácia textu je ponechaná na zodpovednej osobe. Do poľa popisu (Description) chyby je potrebné uviesť nasledovné údaje, ak ich bolo možné pri testovaní určiť:

- Trieda: názov triedy, v ktorej je chyba
- Riadok: číslo riadku, na ktorom sa chybný kód prejavil
- Chyba: definovaný názov chyby

Status chyby je pri vytváraní novej chyby nutné označiť ako priradenú (assigned). Nastavenie priority chyby je ponechané na zvážení zodpovednej osoby. V kolónke priradenej osoby (Assignee) je potrebné vybrať osobu, ktorá je autorom chybného kódu. Čas, do kedy je potrebné danú chybu opraviť, sa vyplní v kolónke Due Date. Čas na vyriešenie chyby je ponechané na zvážení zodpovednej osoby. Na vyplnenie tejto kolónky je odporúčané použiť formu kalendára, aby sa predišlo prípadnej chyby pri vyplňaní tejto kolónky manuálne. Nakoniec je potrebné kliknúť na tlačidlo vytvor a pokračuj (Create and continue), čím bude daná chyba vytvorená v nástroji Redmine.

### 7.3.4 Proces zatvorenia chyby v Redmine

#### 7.3.4.1 Zatvorenie chyby v Redmine

<i>Vstup:</i>	požiadavka na zatvorenie chyby
<i>Výstup:</i>	zatvorenie požadovanej chyby
<i>Zodpovedný:</i>	tester
<i>Dokumentácia:</i>	žiadna

Pre zatvorenie chyby je nutné kliknúť na text aktualizuj (Update), hneď pod hornou lištou. Týmto sa daná chyba otvorí na editovanie. V kolónke stavu (Status) je potrebné vybrať položku zatvorená (Closed). Po stlačení tlačidla potvrdenia (Submit) bude táto chyba zatvorená.

## 7.4 Manažment komunikácie

V rámci manažmentu komunikácie budú popísané dve podstatnejšie časti, ktoré je vhodné mať pre funkčnosť tímu. Prvou je komunikačný plán, ktorý popisuje bežné spôsoby komunikácie v tíme a druhým je niečo málo o požiadavkách na zmenu, ktoré budú najmä v letnom semestri bežnou súčasťou vývoja editora.

### 7.4.1 Komunikačné prostriedky

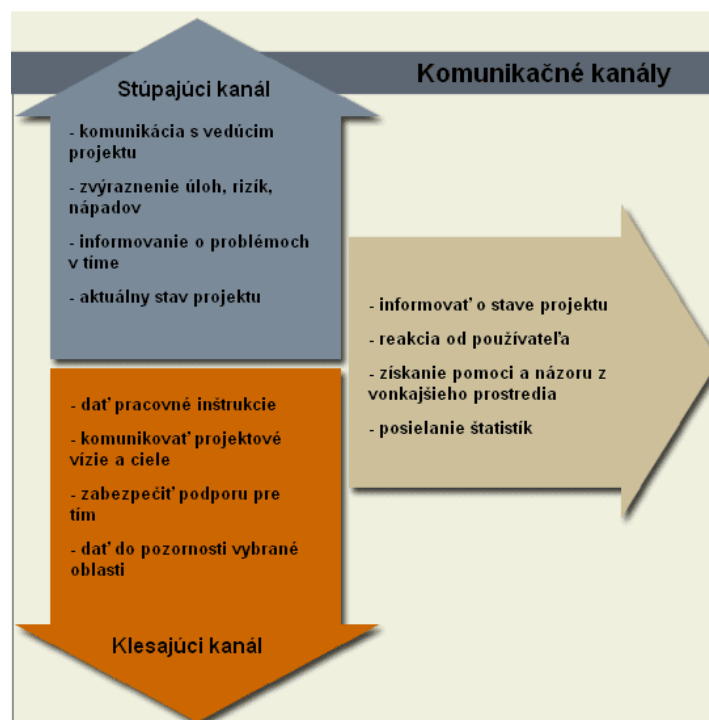
Jedna z prvých vecí, na ktorých sme sa ako tím museli dohodnúť boli použité komunikačné prostriedky. Z viacerých dôvodov sme sa zamerali predovšetkým ne využitie sociálnych sietí, ktoré sú veľmi rozšírené a splňajú potreby nášho tímu.

Tab. 14. Výber komunikačných prostriedkov pre potreby tímového projektu

Typ komunikácie	Formálna	Neformálna
Priama	Formálne stretnutia	Neformálne stretnutia
Nepriama	Google Groups	Facebook
	Redmine ako nástroj pre manažment projektu	

### 7.4.2 Komunikačný plán

V tejto časti dokumentácie sú uvedené komunikačné kanály, ktoré sme identifikovali v rámci komunikácie a pomocou ktorých môžeme zatriediť jednotlivé spôsoby komunikácie v tíme.



Obr. 7 Komunikačné kanály v tíme

Následne je uvedený komunikačný plán, ktorý opisuje spôsob ako vo všeobecnosti prebieha komunikácia v rámci nášho tímu a ktorý som sa snažil aby členovia tímu dodržiavali. Formálne zapísaný komunikačný plán vznikol až na konci semestra, dovtedy bolo všetko na čom sme sa dohodli v rámci komunikovania založené na ústnej dohode. Práve preto som cítil potrebu špecifikovať spôsoby ako by sa mali jednotlivé veci v tíme odkomunikovať.

Tab. 15 Komunikačný plán v tíme

Komunikačný prostriedok	Spôsob komunikácie	Typ komunikácie	Cieľ komunikácie	Formát komunikácie	Frekvencia	Účastníci	Záznam komunikácie
Formálne stretnutie	Face to Face	Priamo komunikácia	Formálne stretnutie s vedúcim projektu. Prezentovanie doterajších výsledkov. Prebranie bodov zápisnice. Riešia sa otázky ohľadom stavu projektu.	Vedená diskusia v tíme s vedúcim projektu	Týždenne	Všetci	Zápisnica
Neformálne stretnutie	Face to Face	Priamo komunikácia	Riešia sa potrebné otázky ohľadom stavu projektu. Všetko podstatné medzi členmi.	Diskusia v tíme	Podľa potreby	Podľa potreby, kto sa nahlási	Poznámky v projektovom denníku
Facebook - Skupina	Anketa	Nepriama neformálna komunikácia	Potreba hlasovania v tíme (týka sa celého tímu)	Vytvorenie ankety s popisom otázky a prípadné vyjadrenie k daným možnostiam hlasovania	Podľa potreby	Podľa potreby, zmeny sledujú všetci	Príspevok na Facebooku (história príspevkov)
			Vyjadrenie účasti k danej udalosti	Vybranie vyhovujúcej možnosti v ankete. Pokiaľ nevyhovuje, pridať novú možnosť. V prípade problémov pridať komentár pod anketu.			
	Príspevok		Celková neformálna komunikácia v tíme. Všetko ostatné, čoho by si mali byť ostatní členovia tímu vedomý	Treba vhodne zlučovať rovnaké kategórie správ (nový komentár k príspevku)			
			Aktuality súvisiace s tímovým projektom	Príspevok popisujúci aktuálne dianie v tíme z pohľadu člena tímu (výsledky oblasti manažmentu, novinky v repozitári)			
			Otázky na členov tímu	Príspevok by mal na začiatku obsahovať meno člena tímu, ktorého sa otázka/problém týka			
Google Goup - Team 10	Vytvorenie príspevku ako novej diskusie, alebo v rámci už začatej diskusie	Nepriama formálna komunikácia	Komunikácia s vedúcim	Vytvorený príspevok v skupine	Podľa potreby	Podľa potreby, zmeny sledujú všetci	Príspevkov v skupine (história príspevkov)
			Dohadovanie stretnutí s vedúcim				
			Kladenie otázok ohľadom problémov na projekte				
			Informovanie o stave projektu				
Redmine	Udalosť	Nepriama formálna komunikácia	V rámci udalosti stručne a výstižne komentovať čoho sa týka. Priblíženie udalosti ostatným členom tímu.	-	Pri každej udalosti	Riešiteľ udalosti	Popis pri udalosti
	Vytvorenie logu pre udalosť						

### 7.4.3 Manažment požiadavky na zmenu (Change request)

Keďže pracujeme na projekte, ktorého celkový vývoj je založený na experimentovaní s novými technológiami a návrhmi riešení, tak bude počas jeho tvorby často dochádzať k zmenám v jeho návrhu a súčasnej práci. Avšak aj tieto zmeny musia byť nejako od komunikované a práve preto sa táto časť zaoberá životným cyklom požiadavky na zmenu, ktorá predstavuje formálny návrh pre zmenu určitej časti vyvíjaného editora. *Change request* vytvára najčastejšie vedúci projektu ako požiadavku na dodatočnú zmenu alebo pridanie funkcionality k produktu voči vopred dohodnutým požiadavkám v rámci analýzy.

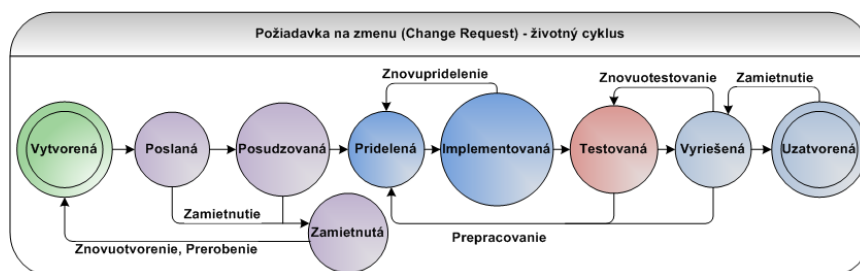
### 7.4.4 Roly a zodpovednosti účastníkov

Tab. 16 Role a zodpovednosti účastníkov manažmentu komunikácie

Rola	Zodpovednosť
<b>Vedúci projektu</b>	Odkomunikovanie požadovanej zmeny ohľadom produktu
	Vytvorenie požiadavky na zmenu
	Dohodnutie sa s adresátom na konkrétnom znení požadovanej zmeny
	Kontrola, či zmena bola zapracovaná ako bolo dohodnuté
<b>Vedúci tímu, Analytik</b>	Konzultovanie požadovanej zmeny s používateľom
	Určenie odhadu pre zapracovanie zmeny (čas, ľudské zdroje)
	Zamietnutie / povolenie zmeny
	Komunikácia s používateľom pokiaľ zmenu nie je možné vykonať
	Komunikácia s používateľom o úspešnom zavedení zmeny
<b>Vývojový tím</b>	Študovanie ako realizovať navrhovanú zmenu, čo všetko treba zmeniť
	Dať späť feedback analytikovi o možnosti vykonania takejto zmeny
	Implementovanie zmeny do existujúceho riešenia
	Posunutie implementovaných zmien ďalej na otestovanie
<b>Tester</b>	Po otestovaní zmeny oboznámiť o výsledku ostatných členov tímu

### 7.4.5 Životný cyklus požiadavky na zmenu

Požiadavka na zmenu môže vzniknúť počas celého procesu vývoja softvéru v jeho jednotlivých fázach. Jednotlivé stavy cez ktoré by mala takáto požiadavka prejsť sú zobrazené na Obr. 8.



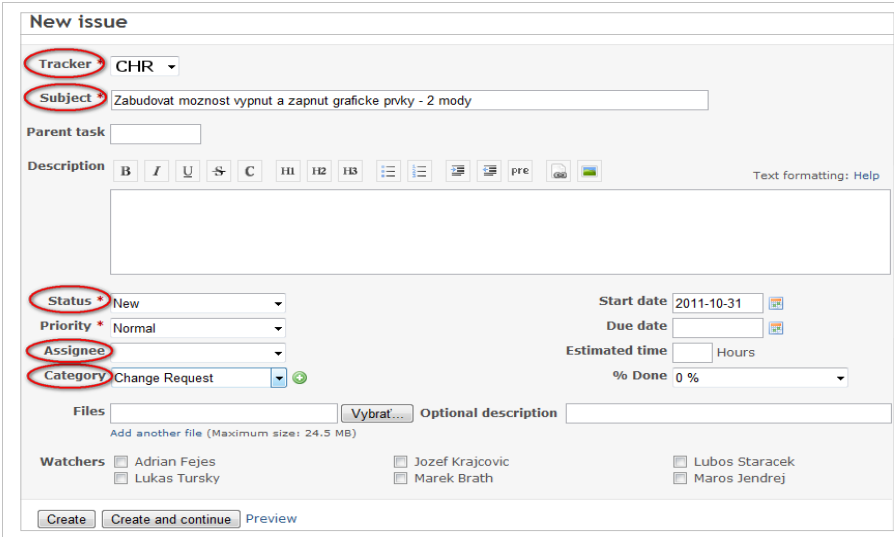


Obr. 8 Životný cyklus požiadavky na zmenu (CHR)

### 7.4.6 Metodika vykonávania jednotlivých procesov požiadavky na zmenu

Pre potreby nášho tímového projektu je nasledujúci proces na konkrétnejšej úrovni zameraný na zadanie a spracovanie požiadavky na zmenu v rámci nástroja Redmine. V tejto časti sa metodika zameriava len na jej správne zaznamenanie a aktualizovanie v rámci jednotlivých častí, ktoré už boli znázornené na Obr. 8

V rámci tímového projektu je vlastníkom projektu náš vedúci projektu, pre ktorého vyvíjame daný projekt. Za hlavnú komunikáciu s vedúcim projektu je zodpovedný vedúci tímu, ktorý predstavuje jedného zo študentov.



Obr. 9 Vytvorenie novej požiadavky na zmenu v Redmine

#### 1. Zadanie CHR na zmenu do nástroja Redmine

Vstup: Neformálna dohoda o vytvorení CHR

Výstup: Vytvorená žiadosť v Redmine

Popis: Proces pre vytvorenie novej požiadavky na zmenu (CHR):

Proces:

1. V rámci aktuálneho projektu kliknúť na *New Issue* (Nová udalosť)
2. Zo zoznamu *Tracker* vybrať *CHR*
3. Vložiť stručný ale konkrétny názov požadovanej zmeny do pola *Subject*. Z názvu musí byť jasné čoho by sa mala zmena týkať.
4. Pridať detailnejší popis problému do textového poľa *Description*. Treba popísať, čo treba zmeniť alebo dorobiť.
5. Vybrať prioritu v poli *Priority*. Vedúci projektu špecifikuje aké podstatné je spracovanie CHR.

6. Pole *Status* musí byť nastavené na *New*
7. Pole *Assignee* by malo obsahovať meno vedúceho tímu
8. Podľa zváženia a potreby môžu byť pridané aj ďalšie, nepovinné, informácie k vytváranej udalosti.
9. Pre vytvorenie a zaevidovanie CHR kliknúť na *Create*

## 2. Rozhodnutie o zamietnutí alebo sprocesovaní CHR

Proces: Rozhodnutie o zamietnutí, sprocesovaní počas spracovávania CHR

1. V rámci aktuálneho projektu z *Issues* vybrať požadovaný CHR
2. Otvorí sa obrazovka s aktuálnym stavom CHR Obr. 9 **Chyba! Nenašiel sa žiaden zdroj odkazov.**
3. Kliknúť na *Update*
4. V obrazovke pre editáciu tasku zmeniť *Status* na *Rejected* alebo *Accepted*
5. Je nutné vložiť odôvodnenie zamietnutia do poľa *Notes*
6. Kliknúť *Submit*

## 3. Pridelenie CHR na implementáciu

Vstup: Akceptovanie požiadavky

Výstup: Vytvorený *Subtask* pre implementáciu s prideleným riešiteľom

Proces:

1. V rámci aktuálneho projektu z *Issues* vybrať požadovaný CHR
2. Otvorí sa obrazovka s aktuálnym stavom CHR.
3. V rámci časti *Subtasks* kliknúť na *Add*
4. Otvorí sa obrazovka ako pre nové Issue. Vid' Obr. 9
5. Do poľa *Subject* napísať prefix [*Impl*] + pôvodný text zo *Subject*
6. Podľa potreby vyplniť ďalšie časti ako v procese 1. „Zadanie CHR na zmenu do nástroja Redmine“
7. V poli *Assignee* zvoliť člena tímu zodpovedného za implementáciu.
8. Kliknúť *Submit*

## 4. Uzatvorenie CHR v Redmine

Vstup: Zatvorený *Subtask* pre implementáciu obsahujúci zhodnotenie implementácie

Výstup: Zatvorený hlavný CHR obsahujúci záznam o výsledku

Proces:

1. V rámci aktuálneho projektu z *Issues* vybrať požadovaný CHR
2. Otvorí sa obrazovka s aktuálnym stavom CHR.

3. Kliknúť na *Update*
4. V obrazovke pre editáciu tasku zmeniť *Status* na Closed
5. Do poľa *Notes* vložiť výsledok spracovania požadovanej zmeny. Vložiť popis, prípadne aj priložiť súbor.

## 7.5 Manažment podpory vývoja

Proces manažmentu podpory vývoja softvéru v našom tíme má za cieľ udržiavať a kontrolovať správu verzií softvéru, konfiguráciu podporných prostriedkov v aktuálnom stave počas životného cyklu projektu. To zahŕňa riadenie zmien v zdrojovom kóde, vetvenie programu, zostavovanie a nasadzovanie softvérového systému do produkčnej verzie.

### 7.5.1 Roly a zodpovednosti

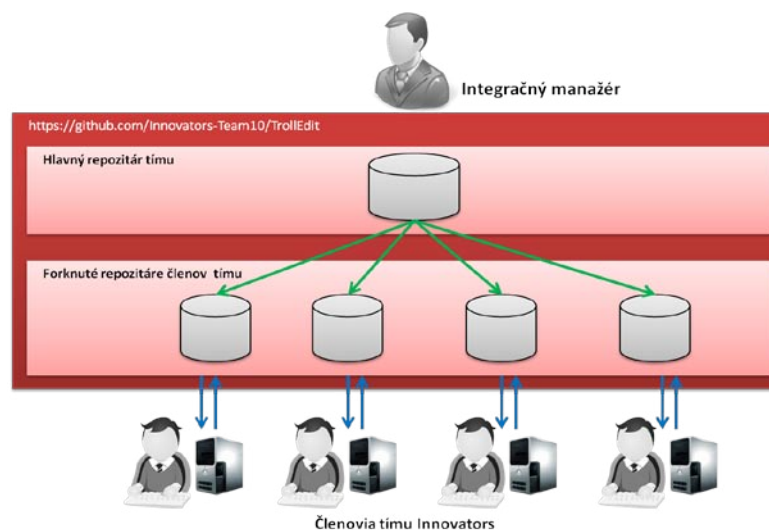
V Tab. 17 sú identifikované roly a prislúchajúce zodpovednosti členom tímu v rámci podpory vývoja softvéru.

Tab. 17 Roly a zodpovednosti v rámci manažmentu podpory vývoja

Rola	Zodpovednosť
<b>Manažér podpory vývoja = integračný manažér</b>	Konfigurácia a správa podporných prostriedkov použitých pri vývoji
	Zlučovanie forknutých repozitárov do hlavného repozitára
	Riešenie konfliktov pri odovzdávaní zmien v zdrojovom kóde
	Spojenie vývojovej vetvy s hlavnou vetvou vývoja pre nasadenie produkčne verzie softvéru
<b>Vývojár</b>	Prevzatie aktuálnej verzie zdrojového kódu
	Implementácia novej funkcionality prípadne jej modifikácia
	Odovzdanie zmien v zdrojovom kóde
	Oboznámenie členom tímu o vykonaných zmenách v zdrojovom kóde
<b>Manažér kvality</b>	Verifikácia kvality zdrojového kódu podľa interných softvérových metrik
	Zaznamenanie vyskytujúcich sa chýb do systému
	Podáva správu o nasadení produkčnej verzie softvéru
<b>Manažér dokumentácie</b>	Zapisuje vykonané zmeny v zdrojovom kóde do príslušnej dokumentácie
	Vygeneruje programátorskú príručku v nástroji doxygen
	Podáva správu o kvalite dokumentácie pre produkčnú verziu softvéru

V rámci projektu využívame distribuovaný systém pre správu verzií (DCVS), konkrétne systém Git, ktorý nám umožňuje väčšiu flexibilitu pri práci na projekte. Projekt je hostovaný na serveri <https://github.com/Innovators-Team10/TrollEdit>.

Pri práci na projekte využívame pracovný postup s integračným manažérom. Tento postup zahrnuje jeden hlavný repozitár, ktorý reprezentuje oficiálny projekt tímu. Každý člen tímu má forknutý hlavný repozitár na svojom účte, ktorý je verejný a kde odosiela svoje vykonané zmeny v zdrojovom kóde. Ak chce člen tímu odoslať zmeny, ktoré vykonal vo svojom repozitári do hlavného repozitára, odošle správcovi hlavného repozitára (integračný manažér) žiadosť, aby jeho zmeny zlúčil do hlavného repozitára. Postup práce je symbolický zobrazený na Obr. 10.



Obr. 10 Pracovný postup s integračným manažérom

Náš pracovný postup v skratke prebieha v nasledujúcich krokoch:

- Prevzatie aktuálnej verzie zdrojového kódu
- Implementácia funkcionality
- Zlúčenie zmien forknutých repozitárov do hlavného repozitára
- Riadenie nasadzovania softvéru

### 7.5.2 Prevzatie aktuálnej verzie zdrojového kódu

<i>Vstup:</i>	aktuálna verzia zdrojového kódu v hlavnom repozitári tímu
<i>Výstup:</i>	uložená verzia zdrojového kódu v repozitári člena tímu
<i>Zodpovedný:</i>	všetci členovia tímu
<i>Dokumentácia:</i>	žiadna

Vývojár si pred začatím práce prevezme aktuálnu verziu zdrojového kódu, čo vykoná nasledujúcim postupom:

1. Ubezpečí sa, že nemá neodovzdané zmeny vo svojom repozitári. V prípade, že takéto zmeny má neodovzdané, tak najskôr odovzdá zmeny
2. Prevezme si aktuálnu verziu zdrojového kódu projektu
  - a. v prípade, že po prevzatí aktuálnej verzie zdrojového kódu nastanú konflikty v zdrojovom kóde, postupuje sa podľa kapitoly 7.5.5

### 7.5.3 Implementácia funkcionality

<i>Vstup:</i>	analýza a návrh funkcionality
<i>Výstup:</i>	implementovaná funkcionality v zdrojovom kóde, popis implementácie
<i>Zodpovedný:</i>	vývojár
<i>Dokumentácia:</i>	technická dokumentácia, programátorská príručka

Vývojár implementuje funkcionality (oprava chýb, pridanie novej funkcionality) v zdrojovom kóde, podľa úloh zaznamenaných v *Redmine*. Postup implementácie funkcionality pozostáva z nasledujúcich krokov:

1. Vytvorenie novej vetvy na vývoj funkcionality v lokálnom repozitári (podľa kapitoly 7.5.3.2)
2. Implementácia funkcionality vo vytvorenej vetve
3. Zostavenie programu v nástroji *Qt Creator & Cmake*
4. Spustenie *Unit* testov pre overenie správnosti implementovanej funkcionality
5. Okomentovanie vykonaných zmien (podľa kapitoly 7.5.3.1)
6. Odoslanie vykonaných zmien (iba funkčný kód!!)
7. Zlúčenie vytvorenej vety s prípravnou vetvou
8. Zaznamenanie vykonaných úloh v nástroji *Redmine*
9. Oboznámenie integračného manažéra a členov tímu s výsledkom vykonanej práce

#### 7.5.3.1 Písanie správ pri vykonaní zmien v nástroji Git

Pri písaní správ vykonaných zmien v nástroji git je treba dodržiavať nasledujúce pravidlá pre lepšiu kooperáciu členov tímu.

Každá správa musí mať nasledujúci formát:

[značka] Stručný a výstižný názov vykonaných zmien

riadok vynechať

(detailný popis vykonaných zmien)

Na začiatok je treba uviesť v hranatých zátvorkách značku podľa Tab.18. Nasleduje stručný a výstižný názov vykonaných zmien, tento názov musí:

- Prvé slovo začínať veľkým písmenom
- V anglickom jazyku
- Kratší ako 50 znakov
- Slovesného tvaru
- Musí byť spolu so značkou v jednom riadku

Za týmto názvom je potrebné vynechať jeden prázdny riadok a za ním nasleduje podrobný popis vykonaných zmien v okrúhlych zátvorkách, kde dĺžka textu by nemala presiahnuť viac ako 5 riadkov!

Príklady písania správ

[+] Add fautures#02 undo/redo

(implementation of the functionality undo/redo according with to using Command objects. These objects are applies the changes and saved to stack Command.)

[!] Fix bug#23

(fixed bug that, caused wrong print a value in textbox form field.)

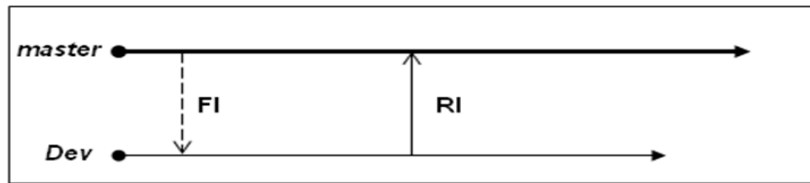
Tab.18 Značky používané pri písaní správ vykonaných zmien v nástroji git

Značka	Popis
+	Pridanie novej funkcionality
-	Zmena funkcionality
r	Refactoring kódu – realokácia kódu bez zmeny funkcionality
!	Oprava chyby
t	Zmena testov
p	Zmena podporných súborov (Cmake, knižnice a podobne)

### 7.5.3.2 Vetvenie projektu

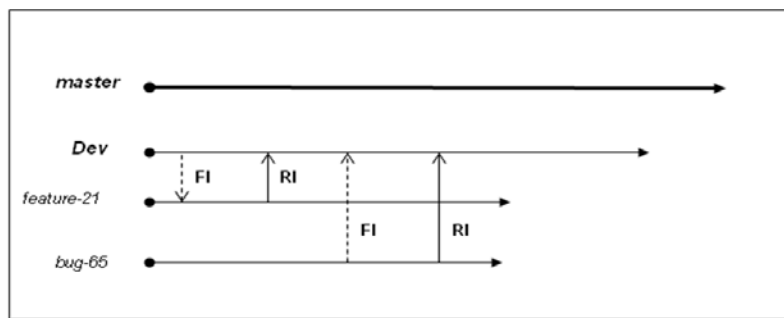
V projekte využívame dve stratégie vetvenia, pre hlavný repozitár a pre forknuté repozitáre členov tímu. Hlavný repozitár projektu obsahuje tri vetvy *master*, *dev*, *gh-pages*. Vetva *master* obsahuje verziu projektu, ktorá spĺňa požiadavky na kvalitu a je prezentovaná

zákazníkovi. Vetva *dev* obsahuje verziu projektu, ktorá je vo vývoji a nebola schválená ako produkčná verzia pre vetvu *master*. Vetva *gh-pages* obsahuje súbory pre webovú stránku projektu. Stratégiou hlavného repozitára je zlučovanie (FI, RI) iba medzi vetvami *master* a *dev* zobrazené na Obr. 11.



Obr. 11 Stratégia vetvenia pre hlavný repozitár

Stratégia pre forknuté repozitáre vychádza zo stratégie pre hlavný repozitár z tým rozdielom, že vývojár si navyše vytvára vlastné vetvy pre implementáciu novej funkcionality, opravu chyby, ktoré potom zlúči do vývojovej vetvy *dev*, nezlučuje vetvu *dev* s vetvou *master*, to robí integračný manažér Obr. 12.



Obr. 12 Stratégia vetvenia pre forknuté repozitáre

### Postup vytvorenia vetvy pre vývoj

1. Vývojár si aktualizuje vetvu *dev*
2. Vytvorí si novú vetvu pre vývoj (implementáciu novej funkcionality, oprava chyby)
3. Tuto vetvu pomenuje podľa toho či sa jedná o *feature* alebo *bug* + identifikátor úlohy, ktorá je zaznamenaná v *Redmine* (napr. *feature#23*)
4. Implementuje funkcionality v zdrojovom kóde
  - ak sa medzi časom zmenila vetva *dev* t.j. už nepracuje nad aktuálnou verziou kódu urobí FI
5. Ak je vývojár spokojný z implementáciou a kód je funkčný! urobí zlúčenie vetiev RI

- o ak nastanú chyby pri RI a chyba sa nedá vyriešiť, tak vráti zámeny na poslednú funkčnú revíziu kódu

#### 6. Danú vetvu po RI vymaže

#### Vytváranie značiek

Značiek (tag) predstavuje pomenované označenie konkrétneho verzie kódu alebo jeho časti, ktoré sú vytvárané k jednotlivým kontrolným bodom. Bude tak možné jednoduchým spôsobom sa vrátiť k stavu, aký bol v danom kontrolnom bode. V rámci projektu používame proste (*lightweight*) značky.

Názov značky bude mať nasledujúci formát:

VX.Y.Z

Kde „v“ je prefix, ktorý znamená „verzia“ a schéma x.y.z predstavuje aktuálnu verziu zdrojového kódu.

x- nová verzia produktu (od 1)  
y- pri zmene funkcionality (od 0)  
z- pri oprave chyby (od 0)

### 7.5.4 Zlúčenie zmien forknutých repozitárov do hlavného repozitára

<i>Vstup:</i>	modifikovaný kód vo forknutých repozitároch
<i>Výstup:</i>	aktualizovaný hlavný repozitár
<i>Zodpovedný:</i>	manažér podpory vývoja, manažér kvality
<i>Dokumentácia:</i>	projektový denník

Integračný manažér po správe od vývojára, ktorý vykonal zmeny vo svojom repozitári, vykoná *pull requests* pre zlúčenie zmien do hlavného repozitára. Postup odovzdávania zmien v zdrojovom kóde pozostáva z nasledujúcich krokov:

1. Vývojár odošle e-mail s žiadosťou manažérovi kvality pre verifikáciu kvality danej revízie kódu
  - o manažér kvality v prípade splnenia funkčnosti kódu, potvrdí schválenie zlúčenia kódu do hlavného repozitára a odošle správu integračnému manažérovi, a taktiež vývojárovi
  - o v prípade výskytu chýb upozorni vývojára na chyby
2. Integračný manažér si prezrie správu od manažéra kvality a urobí *pull request* pre zlúčenie vývojárovej revízie kódu do vývojovej vetvy (*dev*) v hlavnom repozitári



3. V prípade ak sa v procese zlučovania vyskytne konflikt postupuje sa podľa kapitoly 7.5.5 riešenie konfliktov
4. Oboznámi členov tímu s novou revíziou vývojovej vetvy hlavného repozitára

### 7.5.5 Riešenie konfliktov v zdrojovom kóde

<i>Vstup:</i>	konflikt medzi dvomi rôznymi zmenami toho istého súboru
<i>Výstup:</i>	vyriešený konflikt v zdrojovom kóde
<i>Zodpovedný:</i>	manažér podpory vývoja, vývojár
<i>Dokumentácia:</i>	projektový denník

Konflikt vzniká, ak bol rovnaký súbor modifikovaný dvoma rôznymi spôsobmi. Môžu vzniknúť dva typy konfliktov:

- Pri pull requestoch
- Pri zlučovaní vetiev vo vývojárovom repozitári

Konflikty prvého typu rieši integračný manažér, ktorý zodpovedá za zlučovanie repozitárov jednotlivých členov tímu, do hlavného repozitára .

Druhý typ konfliktu môže nastať v prípade nepozorného zásahu vývojára, tento typ konfliktu rieši vývojár sám vo svojom repozitári.

Postup riešenia konfliktov prvého typu pozostáva z nasledujúcich krokov:

1. Integračný manažér si prezrie históriu zmien, kto naposledy upravoval zdrojový kód
2. Integračný manažér oboznámi vývojára zodpovedného za konflikt s daným problémom
3. Vývojár analyzuje príčinu vzniku konfliktu
4. Vývojár vyrieši konflikt a oboznámi manažéra podpory vývoja z výsledkom
  - ak sa konflikt nepodarí vyriešiť, urobí sa návrat k predchádzajúcej revízii kódu
5. Manažér podpory vývoja verifikuje vyriešenie konfliktu

Postup riešenia konfliktov druhého typu pozostáva z nasledujúcich krokov:

1. Vývojár si prezrie históriu zmien zdrojového kódu
2. Porovná aktuálnu revíziu s predchádzajúcou revíziou kódu a analyzuje obsah zmien a možnú príčinu vzniku konfliktu
3. Vývojár vyrieši konflikt
  - ak sa konflikt nepodarí vyriešiť, urobí sa návrat k predchádzajúcej revízii kódu

## 7.5.6 Riadenie nasadzovania softvéru

<i>Vstup:</i>	vývojová vetva softvéru
<i>Výstup:</i>	produkčná verzia softvéru
<i>Zodpovedný:</i>	manažér podpory vývoja, manažér kvality, manažér dokumentácie
<i>Dokumentácia:</i>	technická dokumentácia, programátorská príručka

Riadenie nasadzovania softvéru slúži pre integráciu vývojovej vetvy (*dev*) s hlavnou vetvou (*master*) pre nasadenie produkčnej verzie softvéru, ktorá je úspešne otestovaná a zodpovedá stanovenej kvalite. Tato verzia bude oficiálne prezentovaná zákazníkom.

Postup nasadzovania softvéru pozostáva z nasledujúcich krokov:

1. Manažér kvality odsúhlasí, že vývojová vetva splna požiadavky na kvalitu podľa interných metrik, môže byť zlúčená do hlavnej vývojovej vetvy
  - o ak splna požiadavky na kvalitu, odošle správu o stave manažérovi podpory vývoja
  - o ak nespĺňa požiadavky na kvalitu, odošle správu o vyskytujúcich sa chybách členovi tímu, ktorý je zodpovedný za chyby
2. Manažér dokumentácie verifikuje technickú dokumentáciu a programátorskú príručku pre danú verziu softvéru
  - o ak dokumentácia splna požiadavky, označí ich ako finálnu verziu iba pre čítanie
  - o ak dokumentácia nespĺňa požiadavky, opraví príslušné nezrovnalosti a v prípade väčších opráv kontaktuje autora danej kapitoly
3. Ak manažér podpory vývoja dostane správu od manažéra kvality a manažéra dokumentácie, že všetko je v poriadku, zlúči kód z vývojovej vetvy (*dev*) do hlavnej vety (*master*) a zmení označenie vetvy podľa schémy x.y.z
4. Manažér podpory vývoja oboznámi členov tímu o novej produkčnej verzii softvéru

## 7.6 Manažment kvality

Manažment kvality sa zaoberá plánovaním, zabezpečovaním a riadením kvality v softvérovom projekte. Naším hlavným cieľom je udržiavať a zlepšovať kvalitu softvérového projektu. Snažíme sa o neustálu verifikáciu a validáciu požiadaviek pomocou rôznych metód a prostriedkov. Sú nimi hlavne testovanie, refaktoring a prehliadky kódu.

### 7.6.1 Refaktoring

Refaktoring je zmena architektúry kódu bez zmeny jeho funkčnosti s vidinou získania lepšej flexibility a udržiavateľnosti kódu. Tento projekt sme zdedili po minuloročnom tímovom projekte, po rýchlej prehliadke zdrojových kódov sme zistili, že autori nedodržovali žiadne konvencie pri tvorbe kódu. Preto sme sa na začiatku rozhodli prepísať tento kód podľa nami zadefinovaného štýlu programovania a tento štýl dodržiavať. Sľubujeme si od toho lepšiu prehľadnosť kódu a neskôr aj možnosť generovania technickej dokumentácie pomocou nástroja Doxygen. Ďalej sme sa rozhodli, že budeme na projekte aplikovať refaktoring po malých častiach pri implementovaní a vylepšovaní funkcionality. Základným princípom zmeny len malých častí kódu je to, že pri chybách je možný rýchly návrat na funkčnú verziu. Tieto postupy by mali zabezpečovať postupne zlepšovanie kvality projektu.

### 7.6.2 Prehliadky kódu

Prehliadky kódu (Code review) využívame pred integráciou novej verzie projektu. Pred každou takou integráciou sa vyvoláva požiadavka *Merge pull* v Githube následne musí nastať prehliadka kódu poverenou osobou, ktorá musí otestovať funkčnosť verzie. Ak testovanie prebehne bez komplikácií tak sa dokončí proces integrácie projektu.

### 7.6.3 Programovanie v pároch

V tímovom projekte sme sa snažili vyskúšať aj programovanie v pároch, ktorým sme chceli zabezpečiť zvýšenú kvalitu vytváraného kódu. Prebiehalo tak, že jeden z programátorov vytváral kód a druhý ho kontroloval, prípadne inak podporoval jeho činnosť. Tieto úlohy sa stále striedali.

## 7.7 Manažment testovania

Veľmi významnú časť pri zabezpečovaní kvality tvorí aj manažment testovania. Manažment testovania zabezpečuje kontrolu vytváraného softvéru, jeho cieľom je minimalizovanie šance aby obsahoval nejaké chyby. Je to neustále sa opakujúci proces a začína už po implementácii prvej iterácie až po ukončenie vývoja a nasadenie softvéru pre zákazníka. Neznamená len spúšťanie testov, ale je to predovšetkým plánovanie a riadenie procesov pred a po vykonaní týchto testov.

V ďalších častiach dokumentu sú procesy manažmentu testovania nastavené pre potreby veľkosti menšieho tímu (6-7 ľudí) a iteratívny spôsob vývoja.

Metóda biela skrinka pri tejto metóde je nám známa vnútorná reprezentácia kódu. Pri prehliadke sa kontroluje vytvorený kód a jeho interpretácia.

Metóda čierna skrinka nie je známa vnútorná reprezentácia kódu. Pri takomto testovaní sa bude tester zameriavať na faktory ako je rýchlosť aplikácie, použiteľnosť (klávesové skratky, editácia textu, gramatiky) a grafické rozhranie (menu, kontextové menu, farebné vyznačovanie elementov, štruktúrovanosť blokov).

### 7.7.1 Integračné testovanie

Integračné testovanie vykonávame pri zostavovaní novej verzie projektu (produktu). Ide predovšetkým o zlúčenie viacerých vetiev vyvíjanej funkcionality, ktorá bola implementovaná nezávisle od ostatných.

<i>Vstup:</i>	testovací plán, nová verzia projektu
<i>Výstup:</i>	výstup testovania (splnené/nespĺnené)
<i>Zodpovedný:</i>	manažér kvality, tester
<i>Dokumentácia:</i>	správa o testovanej verzii projektu

1. Vykonanie testov podľa aktuálneho testovacieho plánu.
2. Prechádzame každý bod z testovacieho scenára
  - a. Ak nachádzame chybu zapisujeme ju do Redmine.
3. Vytvárame správu o tejto verzii projektu.
  - a. Ak verzia obsahuje chyby označujeme ju za nesplnenú.
  - b. Ak verzia neobsahuje žiadne chyby označujeme ju za splnenú.
4. Správu pridávame do dokumentácie projektu a označujeme ju príslušnou verzou projektu.

#### 7.7.1.1 Testovací plán

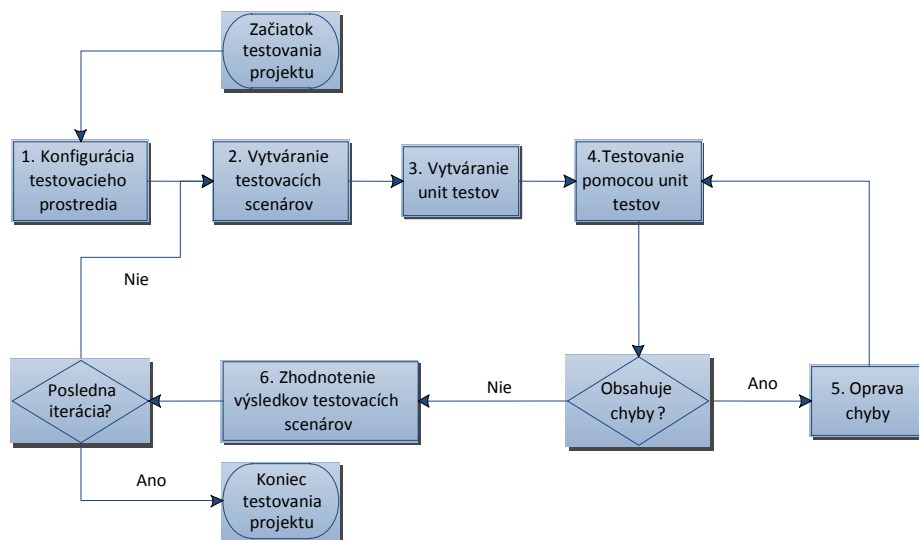
Vykonáva sa pri každej integrácii novej verzie projektu. Pozostáva z týchto stručných bodov:

- Dokument – otvor, zatvor, modifikuj, ulož
- Gramatiky – výber gramatiky, zvýraznenie textu a blokov
- Blok – štruktúrovanosť, vnáranie, drag&drop, zoom in/out
- Text – vyfarbenie syntaxe, editácia
- Klávesové skratky – definované v konfiguračnom súbore
- Undo/Redo
- Tlač do pdf

Tento plán sa priebežne dopĺňa o ďalšie body. V zásade však treba dodržiavať jeho prehľadnú a jednoduchú podobu, lebo slúži hlavne na rýchly test hneď po integrácii projektu.

## 7.7.2 Testovanie pomocou unit testov

Manažment testovania projektu treba začať po vytvorení prvej iterácií projektu. Po tejto prvej iterácii treba nastaviť testovacie prostredie. Ďalej treba vytvoriť testovacie scenáre a uložiť ich k ostatným do nástroja na správu testovacích scenárov. Pre všetky testovacie scenáre treba vytvoriť unit testy. Následne treba testovať všetky vytvorené unit testy od prvého až po posledný. Ak po testovaní v projekte nachádzame chyby, treba ich opraviť a skontrolovať. Ak projekt po testovaní neobsahuje žiadne chyby tak treba zhodnotiť výsledky testovania a zistiť či daná iterácia projektu bola posledná, ak áno, nepridávajú sa už ďalšie testovacie scenáre a nastane koniec testovania projektu. Na Obr. 13 sú znázornené procesy manažmentu testovania a v Tab. 19 je určené ich poradie a kapitola, v ktorej sa nachádzajú.



Obr. 13 Procesy manažmentu testovania

Tab. 19 Tabuľka základných procesov manažmentu testovania

Krok	Základný proces
1	Konfigurácia testovacieho prostredia
2	Vytvorenie testovacích scenárov
3	Vytváranie unit testov
4	Testovanie pomocou unit testov
5	Oprava chyby
6	Zhodnotenie výsledkov testovacích scenárov

### 7.7.2.1 Roly a zodpovednosti

V Tab. 20 sú uvedené roly a ich zodpovednosti v rámci manažmentu testovania.

Tab. 20 Roly a zodpovednosti v manažmente testovania

Rola	Zodpovednosť
<b>Programátor</b>	Vyvíja aplikáciu
	Opravuje reportované chyby v aplikácii
<b>Tester</b>	Vytvára unit testy podľa testovacích scenárov
	Testuje aplikáciu pomocou unit testov
	Reportuje a klasifikuje chyby
	Kontroluje opravené chyby
<b>Manažér kvality</b>	Navrhne a vytvára testovacie scenáre
	Pozoruje výskyt chýb
	Kontroluje klasifikovanie chýb
	Kontroluje výsledky testovania a vyhodnocuje ich

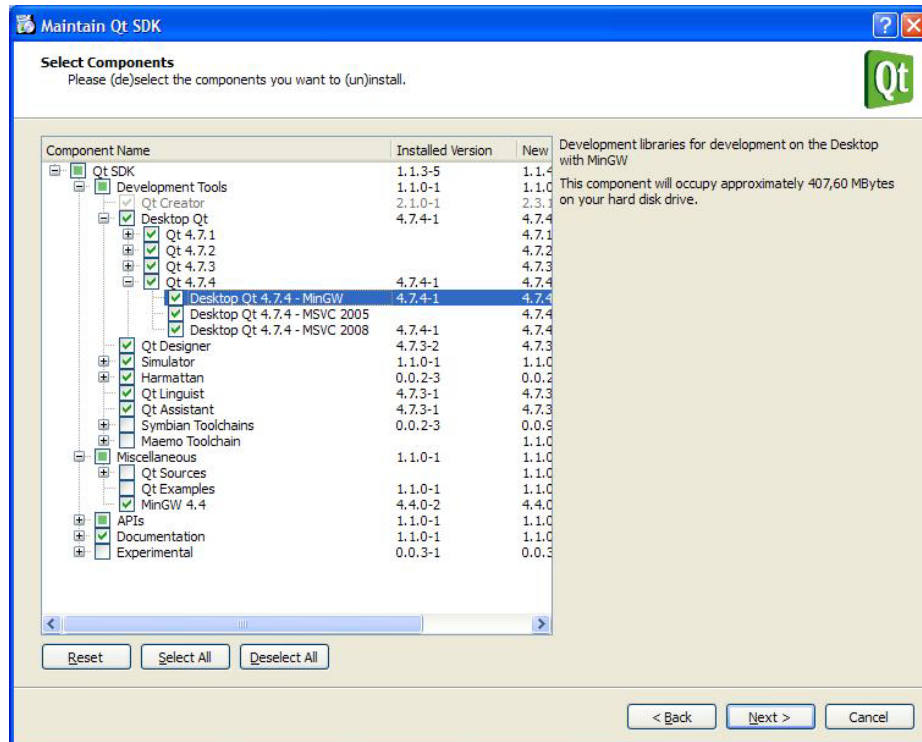
### 7.7.3 Konfigurácia testovacieho prostredia

Manažér kvality alebo tester nastaví prostredie pre testovanie. Tento proces pozostáva z nastavenia správnych knižníc a ciest, aby prostredie neskôr správne fungovalo pri vytváraní a testovaní unit testov.

<i>Vstup:</i>	prvá iterácia projektu, nenastavené testovacie prostredie
<i>Výstup:</i>	projekt, v ktorom sa dajú do testovacieho prostredia pridávať a vykonávať unit testy
<i>Zodpovedný:</i>	manažér kvality, tester
<i>Dokumentácia:</i>	žiadna

#### 7.7.3.1 Inštalácia vývojového prostredia

5. Z webovej adresy <http://qt.nokia.com/downloads/sdk-windows-cpp> stiahni QT Creator a následne ho nainštaluj do predvoleného adresára (je ním C:\QtSDK).
6. Po inštalácii spusti C:\QtSDK\SDKMaintenanceTool.exe, v okne vyber možnosť „package manager“ a vyznač všetky doplnkové knižnice pre MinGW ako je na Obr. 14.



Obr. 14 Nastavenie doplnkových knižníc

7. Ďalej stiahni gcc kompilátor MinGW a nainštaluj ho do predvoleného priečinka (C:\MinGW).

### 7.7.3.2 Importovanie projektu

1. Stiahni aktuálnu verziu projektu a to nasledovne. Otvor konzolu pre git a píš do konzoly tieto príkazy:
  - 1.1. `cd C:\<názov_projektu>\`
  - 1.2. `git clone git@github.com:<názov_organizácie>/<názov_projektu>.git`
  - 1.3. Aktuálny projekt je teraz nahratý v priečinku „C:\<názov\_projektu>“.
2. Spusti QT Creator klikni na File->OpenProjects->Load a otvor súbor C:\<názov\_projektu>\<názov\_projektu>.pro
3. Ak sa zobrazí okno, tak nastav cestu pre build do priečinka C:\<názov\_projektu>\\_build a zvol kompilátor pre MinGW.
4. Klikni tlačidlo finish. Projekt sa teraz otvorí a jeho kompilovanie je nastavené pre kompilátor MinGW.

### 7.7.4 Vytváranie testovacích scenárov

Manažér kvality podľa dokumentácie a špecifikácie softvéru vytvára testovacie scenáre. Testovacie scenáre vytvára po každej iterácii, ak boli v softvéri zmenené alebo pridané

moduly a funkcionálne/nefunkcionálne požiadavky, kontroluje či to neovplyvnilo doteraz vytvorené testovacie scenáre.

<i>Vstup:</i>	dokumentácia, špecifikácia, funkcionálne/nefunkcionálne požiadavky
<i>Výstup:</i>	testovacie scenáre
<i>Zodpovedný:</i>	manažér kvality
<i>Dokumentácia:</i>	testovacie scenáre

#### 7.7.4.1 Analýza funkcionálnych / nefunkcionálnych požiadaviek

Táto analýza zahŕňa tiež preskúmanie dokumentácie, špecifikácie a požiadaviek, ktoré má program spĺňať. Pri prvotnej iterácii musí nastať dôkladná analýza, pri ďalších iteráciách sa analyzujú predovšetkým zmeny a pridaná nová funkcionálna.

#### 7.7.4.2 Definovanie testovacích podmienok, kritérií a požiadaviek

Z predchádzajúcej analýzy vyplynuli podmienky, kritéria a požiadavky. Tieto definujeme tak aby ich bolo možné otestovať v jednoduchých „jednotkových“ testoch. Je veľmi dôležité aby testy bolo možné vykonávať nezávisle od ostatných.

#### 7.7.4.3 Návrh testovacích scenárov podľa definovaných požiadaviek

V tomto kroku navrhujeme testovacie scenáre, ktoré ukladáme k ostatným testovacím scenárom. Jednotlivé testovacie scenáre označujeme číslom iterácie, ktorá v projekte práve prebieha. Scenáre sú uložené v dokumentácii projektu.

### 7.7.5 Vytváranie unit testov

Tester zisťuje, pre ktoré testovacie scenáre ešte nie sú vytvorené unit testy. Podľa poradia vyberá testovací scenár, pre ktorý bude vytvárať unit testy.

<i>Vstup:</i>	testovací scenár
<i>Výstup:</i>	unit testy
<i>Zodpovedný:</i>	tester
<i>Dokumentácia:</i>	doxygen komentáre v unit testoch

#### 7.7.5.1 Výber konkrétneho testovacieho scenára

1. Vyber testovací scenára z Redmine podľa poradia. Ak je to prvá iterácia projektu testovací scenár ma názov SC01\_<Názov\_scen> a začni týmto scenárom. Postupne pokračuj ďalšími v poradí SC02\_<Názov\_scen>, SC03\_<Názov\_scen>..., SC99\_<Názov\_scen>.
2. Každý takýto vybraný scenár označ ako riešený v Redmine.

Ukážka číslovania scenárov z metodiky manažmentu testovacích scenárov:



SC<XX>\_<Názov\_scen> v poli <XX> je číslo scenára a v poli <Názov\_scen> je názov scenára. Ukážka názvu scenára „SC01\_TlačSúboru“.

#### 7.7.5.2 Definovanie kritérií pre splnenie a vytváranie testovacích dát

1. Podľa vybraného testovacieho scenára vytvor hlavičku pre konkrétny unit test a to nasledovne.
  2. Unit testy čísluj takto SC<XX>UT<YY>\_<Názov\_func> v poli <XX> je číslo scenára a v poli <YY> je číslo unit testu.
    - 2.1. Pole <XX> vyplň podľa toho, ktorý testovací scenár spracovávaš.
    - 2.2. Pole <YY> vyplň podľa toho, ktorý unit test v poradí vytváraš. Ak je to prvý unit test vyplň „01“ ak je to druhý „02“ a takto postupne až po „99“.
    - 2.3. Pole <Názov\_func> vyplň podľa kontextu, pre ktorý unit test ho vytváraš (konkrétny názov objektu, rozhrania, databázy).
- Ukážka názvov unit testov SC01UT01\_Súbor, SC01UT02\_InterfaceTlaciaren, ..., SC01UT99\_Blok, SC02UT01\_Element, ... SC02UT99\_Analyzator, ..., SC99UT99\_Text.
3. Z testovacieho scenára definuj kritéria pre konkrétny unit test a zaznač ich do komentára v zdrojovom kóde podľa programovacieho štýlu. Tieto kritéria sú jasné z testovacieho scenára.
    - 3.1. V komentári definuj generovanie testovacích dát.
    - 3.2. V komentári definuj stav objektu pred spustením testu.
    - 3.3. V komentári definuj stav objektu počas vykonávania testu.
    - 3.4. V komentári definuj stav objektu po skončení testu.

#### 7.7.5.3 Tvorba unit testov a pridanie k existujúcim

1. Pri vytváraní zdrojového kódu unit testu dodržuj definovaný programovací štýl, ktorý je definovaný pre vývoj aplikácie.
2. Vytvor unit test a jeho názov je uvedený vyššie (3.3.2 krok 2.).

Ukážka tela unit testu:

```
#include <QtGui>
#include <qtest.h>
class SC02UT05_Tabulka: public QObject
{
    Q_OBJECT
private slots:
    void simple_init();
    void multiple_data();
    void multiple_run();
    void series_finish();
```

```
};
```

- 2.1. Z komentára unit testu prečítaj ako sa generujú testovacie dáta a implementuj takúto funkciu. Označ ju takto `<názov_funkcie>_data()`. Názov funkcie určíš z kontextu pre ktorý je vytváraná.

Ukážka funkcie pre generovanie dát:

```
void SC02UT05_Tabulka::multiple_data()
{
    QTest::addColumn<bool>("useLocaleCompare");
    QTest::newRow("locale aware compare") << true;
    QTest::newRow("standard compare") << false;
}
```

- 2.2. Z komentára unit testu prečítaj aký stav má objekt pred spustením testu a implementuj takúto funkciu. Označ ju takto `<názov_funkcie>_init()`.
- 2.3. Z komentára unit testu prečítaj aký stav je počas vykonávania testu a implementuj takúto funkciu. Označ ju takto `<názov_funkcie>_run()`.
- 2.4. Z komentára unit testu prečítaj aký je stav po vykonaní testu a implementuj takúto funkciu. Označ ju takto `<názov_funkcie>_finish()`.
- 2.5. Vytvorené unit testy pridaj k ostatným. Tento proces je presne určený v metodike manažmentu verziovania.

### 7.7.6 Testovanie pomocou unit testov

Tester spúšťa všetky unit testy, ak nastane chyba zapisuje ju, klasifikuje ju a prideluje jej závažnosť. Tento proces je presne opísaný v metodike manažment chyby.

Ak tester vykonáva tento proces znova tak opätovne spúšťa všetky unit testy, ak sú testy úspešné zaznačí chybu ako definitívne opravenú. Ak chyba stále nie je opravená zaznačí ju ako neopravenú.

Vstup:	unit testy
Výstup:	prehľad splnených/nespĺnených unit testov, zaznamenané chyby
Zodpovedný:	tester
Dokumentácia:	žiadna

- Príprava testovacích údajov a prostredia.
  - Ak je implementovaná nová funkcionálna, pre ktorú sú potrebné testovacie údaje tak ich vytvoríme a uložíme k unit testom.
- Spustíme unit testy, v prostredí QT Creator ako run pre testy
  - Mimo prostredia QT Creator testy spustíme týmto príkazom

```
/<myTestDirectory>$ qmake -project "CONFIG += qtestlib"
```

3. Zaznačíme výsledky a chyby z testovania.
  - 3.1. Ak počas testov nastali chyby, tak do Redminu zaznačíme chybu.
  - 3.2. Opíšeme v ktorom unit teste nastala a pridáme aj ďalší opis z výsledku testovania.
  - 3.3. Ak nenastala žiadna chyba, testovanie končí úspešne bez zaznačenia chýb.
4. Ďalšie testovanie nastáva v ďalšej iterácii.

### 7.7.7 Zhodnotenie výsledkov testovacích scenárov

Po každej iterácii softvéru manažér kvality kontroluje vykonané testovacie scenáre a vytvára report. Zhodnocuje ich a určuje, ktoré testovacie scenáre podliehali najväčšiemu množstvu chýb. Snaží sa navrhnúť lepšie testovacie scenáre a pre programátora metodiku alebo štýl programovania, aby sa zlepšil proces vývoja softvéru.

<i>Vstup:</i>	iterácia softvéru, testovacie scenáre
<i>Výstup:</i>	správa zhodnocujúca úspešnosť testovacích scenárov, upravená metodika alebo štýl programovania
<i>Zodpovedný:</i>	manažér kvality
<i>Dokumentácia:</i>	zhodnocujúca správa testovania iterácie projektu

1. Analýza výsledkov testovacích scenárov a ich unit testov.
2. Určíme počet bugov v danej iterácii projektu z Redmine.
3. Zisťujeme dodatočné informácie o týchto chybách.
4. Vykonávame zhodnotenie potreby ďalších testov alebo zmenu ich existujúcich kritérií.
5. Vytvoríme správu zhodnocujúcu testovacie scenáre a ich výsledky.
6. Správu uložíme, k ostatnej dokumentácii projektu. Označíme ju číslom príslušnej iterácie projektu.

## 7.8 Manažment monitorovania

V tejto časti sú popísané spôsoby monitorovania projektu, ktoré sa uplatňujú v tíme. Na monitorovanie projektu sme využívali *Redmine* (dostupné na <https://redmine.fiit.stuba.sk/>). Manažér monitorovania kontroluje plnenie úloh členov tímu a počet zaznamenaných hodín pri jednotlivých úlohách.

### 7.8.1 Monitorovanie projektu v nástroji Redmine

Manažér monitorovania sa prihlási do nástroja Redmine.

Prejde na sekciu „Charts“, kde sa zobrazí „Burndown“, kde možné sledovať napredovanie projektov. Vyberie si projekt a obdobie za ktoré chce pozorovať výsledky a potom klikne na tlačidlo „Apply“.

Zobrazí sa mu graf so štyrmi hodnotami:

- a) Odhadovaný čas (zelená farba)
- b) Zaznamenaný čas (oranžová farba)
- c) Zostávajúci čas (žltá farba)
- d) Predpovedaný čas (modrá farba) – súčet zaznamenaného času a zostávajúceho času

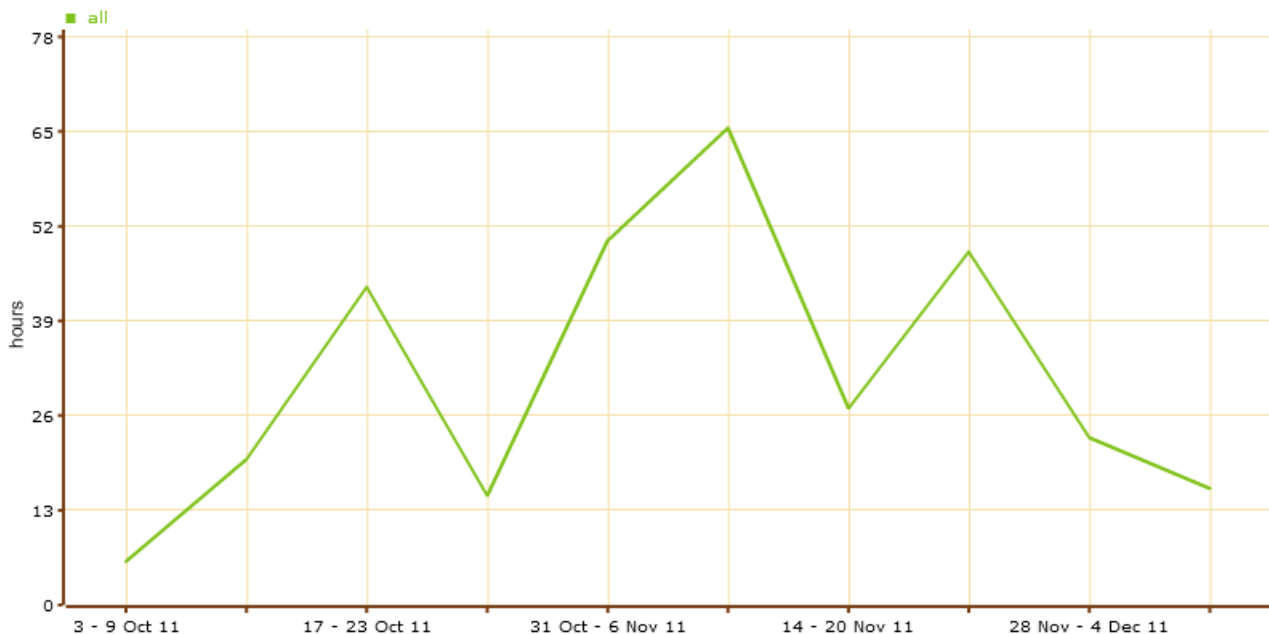
Potom môže prejsť na „Logged hours ratio“, kde sa mu zobrazí počet zaznamenaných hodín všetkých členov tímu.

Po kliknutí na „Logged hours timeline“ sa mu zobrazí podobný graf ako pri „Burndown“, lenže tento graf je zameraný na počet zaznamenaných hodín. Manažer monitorovania môže pridať filter „Users“, kde sa mu zobrazí graf so všetkými členmi tímu, každý inou farbou.

Môže si ďalej pozrieť strávené hodiny nad jednotlivými úlohami pridaním filtru „Issues“.

Kliknutím na „Issues ratio“ si môže pozrieť stav úloh v koláčovom grafe, ktoré sú rozdelené do piatich častí:

- a) New b) Assigned c) Closed d) Rejected e) In progress



Obr. 15 Prehľad intenzity logovania stráveného času na projekte

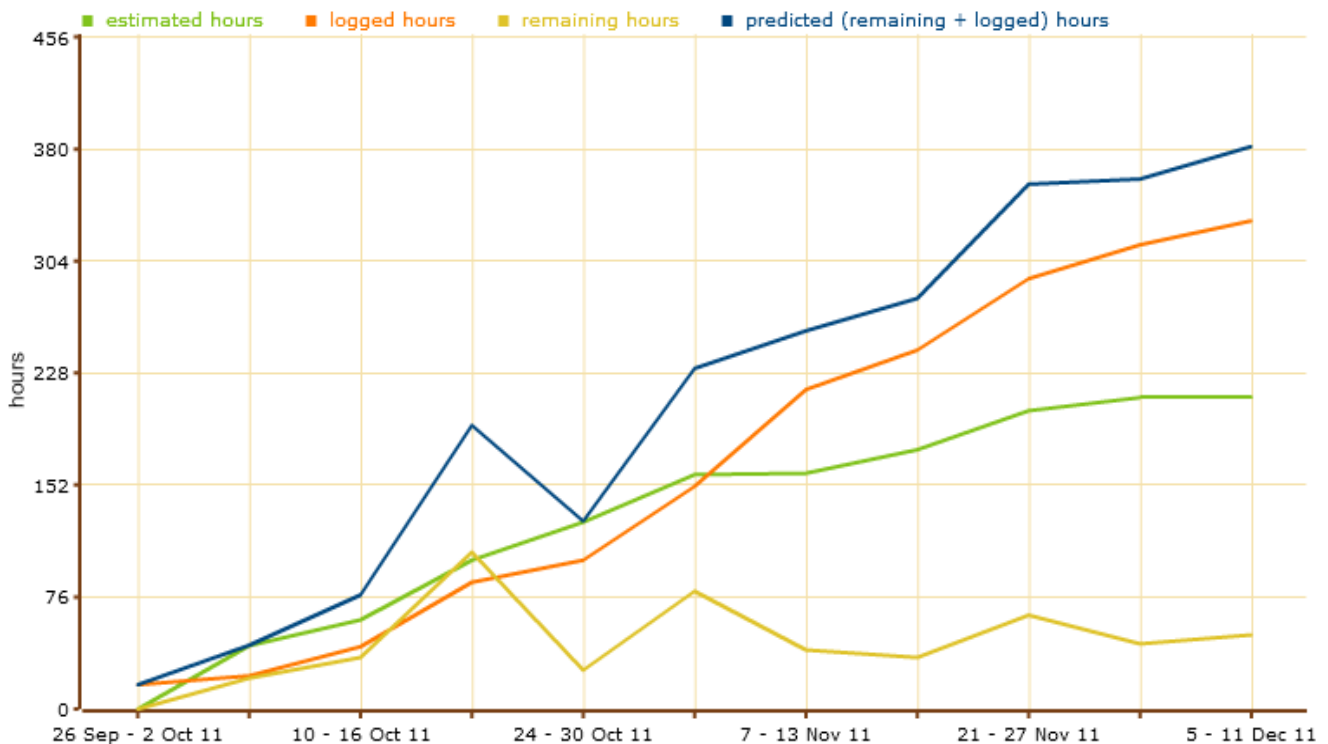
## 7.8.2 Monitorovanie úloh v Redmine

Manažer monitorovania sa prihlási do nástroja Redmine.

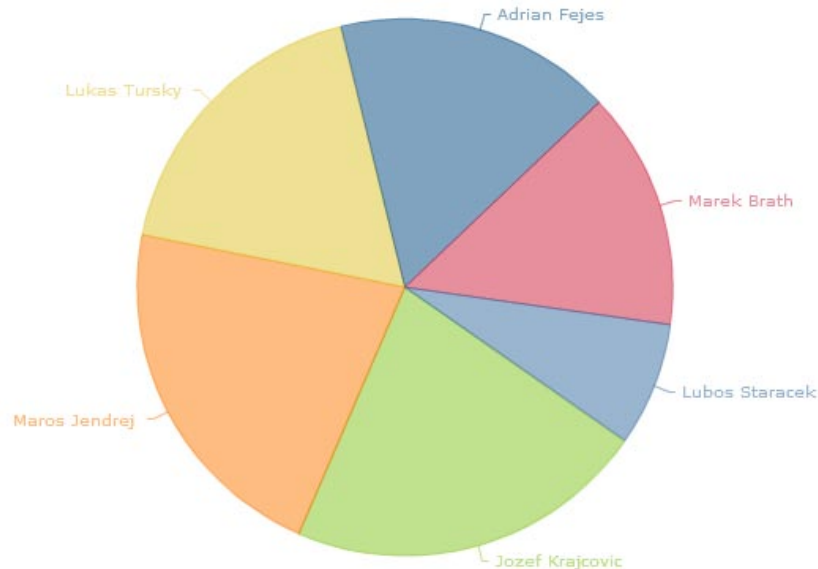
Prejde na sekciu „Charts“, kde si vyberie status „All“, aby sa mu zobrazili všetky úlohy členov tímu. V pravidelných intervaloch (napr. raz do týždňa) skontroluje úlohy, ktoré mali byť ukončené v danom intervale. Kliknutím na názov úlohy sa mu zobrazia podrobnosti o úlohe. Kliknutím na počet hodín pri „Spent time“ si môže pozrieť počet zaznamenaných hodín, dátum a ak nim komentár.

Stlačením čísla úlohy sa vráti späť na podrobnosti o úlohe, kde môže pridať komentár stlačením „Update“. Manažér monitorovania pridá komentár v prípade, keď:

- Úloha nie je uzavretá včas.
- Člen tímu nemá zaznamenané žiadne hodiny alebo sa čas výrazne odchyľuje od odhadovaného času.
- Popis alebo parametre úlohy sú nepresné alebo nedostatočné.



Obr. 16 Priebežný burndown graf



Obr. 17 Sledovanie činnosti členov tímu (nemusí odrážať celkovú činnosť na projekte)

## 7.9 Manažment tvorby dokumentácie

V tejto časti sú definované pravidlá riadenia písania dokumentácii, ktoré musí každý člen tímu dodržiavať v rámci firemnej kultúry.

### 7.9.1 Roly a zodpovednosti

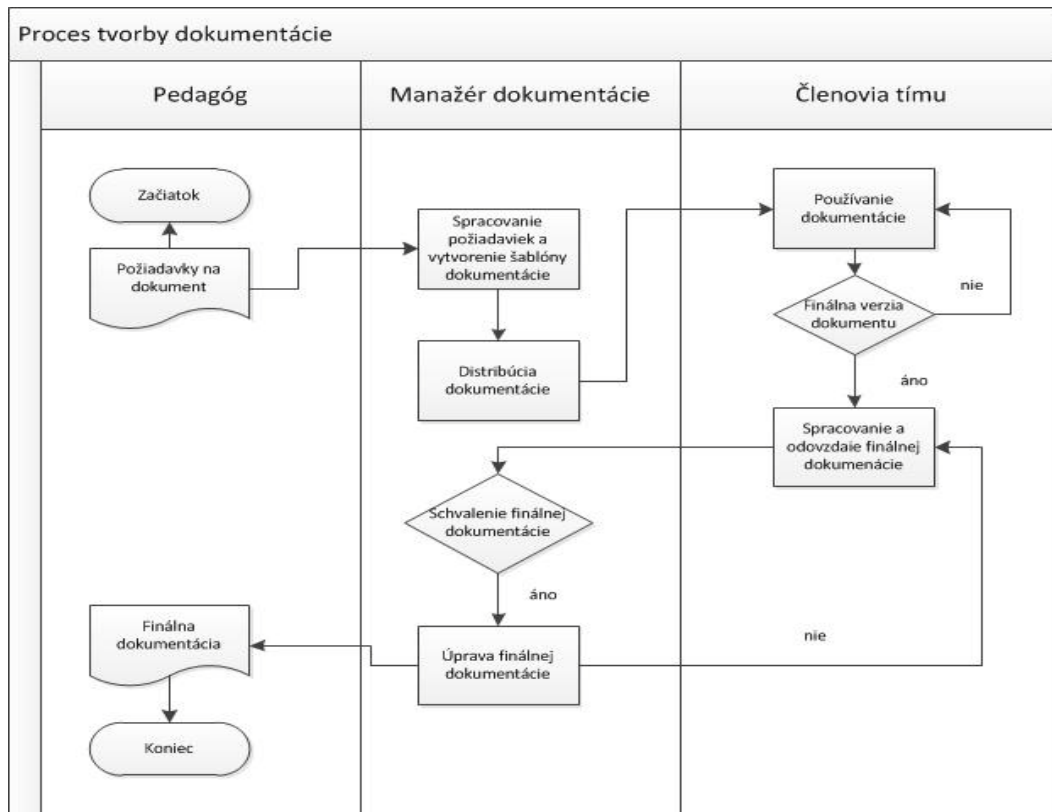
Tab. 21 Roly a zodpovednosti manažmentu dokumentácie

Rola	Zodpovednosť
<b>Manažér dokumentácie</b>	Vytvára šablóny pre dokumentáciu
	Zodpovedá za obsah, dodržiavanie formátovania a kontroluje chyby v dokumentácii
	Vytvára a verifikuje finálnu dokumentáciu ktorú treba odovzdať
<b>Zapisovateľ</b>	Pridáva obsah do dokumentácie
	Môže meniť obsah dokumentácie po dohode z manažérom

### 7.9.2 Základné pravidlá pri písaní dokumentácie

Na základe dodržania konzistencie pri písaní dokumentácie boli spísané všeobecné pravidlá pre písanie dokumentácie, ktorými by sa mali všetci členovia tímu riadiť. Vzhľadom na rozsah tohto dokumentu, sa pravidlá nachádzajú v Prílohe C – Pravidlá dokumentácie.

### 7.9.3 Postup tvorby dokumentácie



Obr. 18 Proces tvorby dokumentácie

- 1) Na začiatku sú definované požiadavky od nášho vedúceho tímu (pedagóga) alebo od prof. Bielikovej.
- 2) Na základe týchto požiadaviek vytvorí manažér dokumentácie šablónu dokumentácie, uloží ju do repozitára a oznámi členom tímu účel dokumentácie.
- 3) Každý člen tímu používa dokumentáciu a pridáva svoje časti, ktoré mu boli pridelené manažérom tímu.
- 4) Ak každý člen tímu doplnil svoju časť do dokumentácie, manažér dokumentácie verifikuje obsah, a v prípade výskytu chýb oznámi zodpovednému za časť obsahu, v ktorej sú chyby, aby ju prerobil. Ak je dokumentácia v poriadku, tak ju manažér dokumentácie označí ako finálnu a znemožní jej úpravu. Takto vytvorenú finálnu dokumentáciu archivuje do repozitára pod platným číslom verzie.
- 5) Manažér dokumentácie odovzdá finálnu dokumentáciu nášmu vedúcemu tímu.

### 7.9.4 Vytváranie zápisníc zo stretnutí

Zápisnice sa vytvára podľa šablóny ktorá je umiestnená v tímov repozitáre. Zápisnicu vytvára ten člen tímu, ktorý je nato určený podľa poradovníka. V zápise zo stretnutia by

malo byť zachytené všetko, o čom sa na stretnutí diskutovalo. A taktiež určenie úloh, ktoré vyplynuli zo stretnutia a tiež vyhodnotenie plnenia úloh z predchádzajúceho stretnutia.

#### **7.9.4.1 Pravidlá vytvárania zápisníc**

- 1) Zápisnicu vytvára člen tímu podľa šablóny
- 2) Zápisnicu treba vytvoriť do 8 hodín od ukončenia tímového stretnutia
- 3) Zápisnicu uložiť do repozitára a oznámiť členom tímu o jej vytvorení
- 4) Členovia tímu verifikujú úplnosť zápisu v prípade nejakej nezrovnalosti oznámi zapisovateľovi zápisnice aby ju modifikoval
- 5) Manažér plánovania pridá úlohy, ktoré treba vykonať zo zápisnice do nástroja Redmine.

#### **7.9.4.2 Pravidlá vedenia zápisníc**

- 1) Tvorca zápisnice ju prinesie na oficiálne stretnutie
- 2) Tvorca zápisnice oboznámi vedúceho tímu z stavom vykonania úloh zo zápisnice
- 3) Zapisovateľ, ktorý je určený podľa poradovníka pozorne počúva a zapisuje body o ktorých sa diskutuje na tímovom stretnutí.

### **7.10 Štýl programovania**

V tejto kapitole definujeme štýl písania kódu, ktorý budeme dodržiavať v rámci firemnej kultúry tímového projektu. Definované sú pravidlá, ktoré musí každý člen tímu dodržiavať pre sprehládnenie zdrojového kódu a tým zamedzeniu možných nedorozumení a konfliktov, ktoré môžu vzniknúť z nejednotného štýlu programovania.

#### **7.10.1 Vytváranie názvov**

Použitie správnych názvov je kľúčové k sprehládneniu kódu, názvy treba zvoliť zmysluplne aby vystihovali podstatu riešenia.

Názvy budú písane po anglicky

##### **Triedy**

1. používať notáciu PascalCase
2. názvy by mali byť podstatnými menami

##### **Metódy**

1. používať notáciu camelCase
2. názvy by mali byť slovesného tvaru



**Premene**

1. názvy sú písane malými písmenami
2. voliť zmysluplne názvy nie nič nehovoriace skratky ako (napr. „v“)
3. v prípade dlhších názvov používať pre oddelenie slov podtrhovník (napr. „nazov\_nazov“)
4. ak je možné tak premenu v tom istom riadku aj inicializovať

**Konštanty**

1. písane sú veľkými písmenami

**Ovládacie prvky (GUI)**

1. používať Maďarsku notáciu t.j. prefix, ktorý vystihuje o aký typ ovládacieho prvku ide (napr. „btnOK“ - btn pre tlačidlo a OK je názov tlačidla)

**7.10.2 Odsadenia**

Pre sprehládnenie štruktúry blokov v zdrojovom kóde treba dodržiavať nasledujúce pravidla:

1. Každý vnorený riadok musí byť odsadený tabulátorom o jednu pozíciu do ľavá

Správne:

```
nazovMetody()  
{  
    if()  
    {  
        nejakyprkaz;  
    }  
}
```

**7.10.3 Písanie zátvoriek**

1. Pri písaní zložených zátvoriek nepoužívať štýl K&R!

Správne:

```
if()  
{  
    nejakyprkaz;  
}
```

Nesprávne:

```
if() {  
    nejakyprkaz;  
}
```

2. Písanie okrúhlych zátvoriek za kľúčovým alebo nejakým príkazom bez použitia väčšieho počtu medzier.

Správne:

```
if (a == 2)
```

Nesprávne:

```
if    (a == 2)
```

#### 7.10.4 Písanie komentárov pre potreby nástroja doxygen

Vo všeobecnosti sa písanie komentárov pre nástroj doxygen skoro vôbec nelíši od bežného komentovania. Pokiaľ niekto používal komentáre pre Javadoc, alebo iný dokumentačný prístup, tak je to v podstate to isté.

Pre Doxygen platí, že pokiaľ sa "blok komentáru" nachádza či už pred metódou, triedou, alebo nejakou štruktúrou, tak tento komentár sa priradí a bude tykať tejto danej časti kódu.

##### 7.10.4.1 Všeobecné zásady pri písaní dokumentácie:

1. Komentáre písať čo možno stručne nevytvárať v žiadnom prípade literárne diela
2. Komentovať treba každú triedu, metódu (funkciu), zložitejší cyklus, prípadne aj premennú
3. Komentáre písať bez diakritiky a prvé slovo pri opise funkcie a metódy začínať s veľkým písmenom
4. Komentáre treba písať po anglicky pre budúce generácie

##### 7.10.4.2 Komentovanie súborov

1. Rozlišujeme dva druhy súborov:
  - *.h* súbor obsahujúci definície (hlavičky metód, premenné, atď). Je reprezentovaný ako trieda *Class*
  - *.cpp* súbor obsahujúci implementáciu. Pozostáva z viacerých metód (funkcií).
2. Každý takýto súbor by mal obsahovať v hlavičke komentár so základnými informáciami.
3. Hlavička obsahuje popis, ktorý nás informuje čo daný súbor obsahuje a k čomu je určený. Akú časť aplikačnej logiky zastrešuje.
4. Komentár pre doxygen musí byť uzavretý v blokovom komentári. Oproti klasickému komentáru začína */\*\** a končí klasickým *\*/*
5. Riadky medzi začiatkom a koncom komentára môžu ale nemusia začínať znakom *\**

Formát:

```
/**
 * @Title nazov suboru
 * -----
 * @Description
 * [ popis suboru, na čo slúži a čo sa v nom rieši ]
 *
 * @Category o aky typ suboru ide
 * @Author ak ide o nový subor, tak je vhodné mať meno autora suboru.
 * @Verzion
 */
```

#### 7.10.4.3 Komentovanie metód

1. Okomentovať každú metódu. Sprehl'adňuje nielen dokumentáciu ale aj zdrojový kód.
2. Stručný popis metódy, aké sú vstupy a čo vracia.

Formát:

```
Pred funkciou
/**
 * @Description Popis funkcie, v skratke čo má robiť
 *
 * @param meno_vstunej_premennej na čo slúži
 *
 * @see metodaXY() - odvolávka na nejakú inú metódu, bude ako link vygenerované, nie je nutné
 *
 * @return popis čo vracia daná metóda
 */
```

#### 7.10.4.4 Jednoriadkové komentáre

1. V prípade potreby je možné pre doxygen okomentovať aj jednoriadkový kód.
2. Pri definovaní typov môžeme použiť pre ich popis nasledujúci spôsob komentovania.
3. Použije sa komentárová značka `//!`

napr.

```
int var;      //!< Stručny popis daného riadku kodu, premennej, priradenia atd.
```

#### 7.10.4.5 Komentovanie vetvení

1. Zložitejšie vetvenia je vhodné vždy komentovať za podmienkou v tom istom riadku
2. Využívať 2x stlačenie tabulátora pre odsadenia, prípadne mať vhodné zarovnanú odsadenú časť s komentárom v rámci celého vetvenia.

Správne:

```
if(podmienka)           //! Komentar
{
}
else if(podemienka)      //! Komentar
{
}
else                     //! Komentar
{
}
```

#### 7.10.4.6 Používanie skratiek v komentároch

Už klasicky používame. Pre interné účely určené.

1. TODO: bude značiť niečo čo je potrebné v budúcnosti implementovať.

napr.

```
//TODO: doplnit funkcionalitu
```

2. BugID: bude signalizovať že na tomto mieste je známa chyba a ak je zaznačená v nástroji pre manažment zmien tak aj jej ID.

napr.

```
//Bug#12: chyba nespravneho vypisu hodnot
```

#### 7.10.5 Písanie metód

##### Odporúčania:

1. Snažiť sa programovať krátke a jasné metódy
  - ak je problém rozložený na viacero menších problémov a každý z nich je riešený na samostatnom mieste, je jednoduchšie pochopiť celý problém
2. Odstránenie duplicity kódu
  - zvyšuje kvalitu kódu a prispieva k lepšej udržiavateľnosti
3. Vyhybať sa tzv. „mŕtvemu kódu“
  - taký kód, ktorý je napr. v komentároch, už sa nepoužíva, len tam ostal ako história po predošlých verziách a zneprehľadňuje zdrojový kód