Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Platforma pre monitorovanie chýb a iných udalostí v aplikácii

Dokumentácia k riadeniu projektu

Vedúci tímu: Ing. Jakub Ševcech
Členovia tímu: Bc. Barbora Brocková, Bc. Matej Čaja, Bc. Adam Lieskovský, Bc. Martin Číž, Bc. Peter Kyseľ, Bc. Daniel Uderman, Bc. Michal Viskup
Akademický rok: 2014/2015

1 Úvod

V tomto dokumente sa nachádza dokumentácia riadenia projektu. Rozpísali sme pridelené manažérske pozície členov tímu v súvislosti s projektovým riadením a definovali ich úlohy.

Každý manažér si spísal metodiku, pomocou ktorej sa vytvoril prístup pre zjednotenie správania členov tímu ku určitému problému, jeho sledovanie. V závere opisu metodiku zhodnotil jej plnenie ostatnými členmi tímu.

Počas vývoja projektu sme sa snažili postupovať agilne a prideľovali sme úlohy členom tímu v súlade s dvojtýždňovými šprintami. V dokumente sa nachádza sumarizácia práce členov počas absolvovaných troch šprintov v semestri.

V závere sa nachádza retrospektíva práce na tímovom projekte a celkový progres, ktorý sa nám podarilo dosiahnuť.

1.1 Podiel na vytváraní dokumentu

Názov kapitoly	Autor
Úvod	Bc. Barbora Brocková
Role členov tímu	Všetci členovia tímu
Opis manažérskych činností	Všetci členovia tímu
Sumarizácia práce členov tímu	Všetci členovia tímu
Manažment kvality	Bc. Matej Čaja, Bc. Barbora Brocková, Bc. Michal Viskup
Manažment rizík	Bc. Martin Číž
Manažment plánovania	Bc. Daniel Uderman
Manažment monitorovania	Bc. Peter Kyseľ
Manažment podpory vývoja a integrácie	Bc. Adam Lieskovský
Manažment komunikácie	Bc. Michal Viskup
Manažment tvorby dokumentácie	Bc. Barbora Brocková, Bc. Michal Viskup
Záverečná retrospektíva za zimný semester	Bc. Matej Čaja
Preberacie protokoly	Všetci členovia tímu

Tab. 1: Podiel na tvorbe dokumentu

2 Role členov projektového tímu

Na prvom oficiálnom stretnutí sa zadelili manažérske pozície členom tímu. Každý manažér mal za úlohu vypracovať metodiku v súlade so svojou manažérskou pozíciou a kontrolovať iných členov tímu, či ju pri práci dodržiavajú.

Manažérska činnosť	**Pridelná osoba**
Manažér dokumentácie	Bc. Barbora Brocková
Manažér podpory vývoja a integrácie	Bc. Adam Lieskovský
Manažér kvality	Bc. Matej Čaja
Manažér rizík	Bc. Martin Číž
Manažér monitorovania	Bc. Peter Kyseľ
Manažér plánovania	Bc. Daniel Uderman
Manažér komunikácie Vedúci tímu	Bc. Michal Viskup

Tab. 2: Zoznam manažérskych úloh v tíme

3 Opis manažérskych činností

3.1 Manažment kvality

Manažér kvality je zodpovedaný najmä za stanovenie a dohliadanie na dodržiavanie metodík tvorby testov, spracovania chýb a tvorby, dokumentovania a refaktoringu kódu. Zároveň má na starosti aj meranie kvality kódu pomocou rôznych metrík s využitím podporných nástrojov.

3.2 Manažment rizík

Manažér rizík má na starosti riadenie a sledovanie rizík. V priebehu vývoja projektu určuje aké riziká môžu projekt alebo tím ovplyvniť. Sleduje charakteristiky rizík. Riziká analyzuje, hodnotí a plánuje ako na ne reagovať. Dohliada na dodržiavanie postupov ostatnými členmi tímu. Spolu s manažérom kvality dohliada na to, aby bol projekt otestovaný. Manažér rizík na základe vyhodnotených rizík komunikuje s manažérmi, ktorých sa týkajú.

3.3 Manažment plánovania

Manažér plánovania zodpovedá za Korektné a transparetné plánovanie úloh ktoré sú nutné pre dokončenie inkrementu projektu v každom zo šprintov. Informuje jednotlivých členov tímu o im pridelených úlohách a zodpovedá za rovnomerne distribuovanie úloh. Úzko spolupracuje s manažérmy rizík a monitorovania, s ktorými priebežne konzultuje stav vypracovania jednotlivých úloh a zodpovedá za ich včasné vyriešenie a uzatvorenie.

3.4 Manažment monitorovania

Manažér monitorovania má na starosti monitorovanie projektu, korektné uzatváranie a dokončovanie úloh, podľa časového harmonogramu a po inicializačnom stretnutí na začiatku šprintu, pridať, prípadne skontrolovať úlohy, pre nasledujúci šprint.
Monitorovanie projektu sa vykonáva cez project managment tool. V našom projekte využívame agilnú verziu Redmine. Okrem iného monitorovanie prebieha aj cez komunikačné kanály alebo priamou komunikáciou, s riešiteľom priradenej úlohy.

3.5 Manažment podpory vývoja a integrácie

Definovanie konfigurácii, údržba a aktuálnosť podporných prostriedkov spadajú medzi úlohy manažéra podpory vývoja.
Primárnou úlohou manažéra je udržiavanie prostredia pre vývoj a testovanie produktu. Medzi jeho kompetencie resp. zodpovednosti patrí aj nasadzovanie nových verzií aplikácii na

produkčný server, a samotný bezproblémový beh aplikácie. Manažér sleduje prácu v nástroji *Git* [1] a v sieti *Github*[2], je zodpovedný za dodržiavanie stanovených pravidiel pri verziovaní zdrojového kódu. Udržiava kompatibilitu medzi jednotlivými knižnicami využívaných v zdrojovom kóde a ich aktuálnosť.

3.6 Manažment komunikácie

Manažér komunikácie je zodpovedný za stanovenie komunikačných kanálov pre tím, prípravu týchto kanálov z hľadiska konfigurácie a stanovenie metodiky využívania týchto kanálov.
Zadefinované metodiky určujú, ktorý z kanálov použiť v ktorej situácií. Zavádzajú tiež štandardy ohľadom využívania niektorých vlastností zvolených kanálov a previazania kanálov medzi sebou.

3.7 Manažment tvorby dokumentácie

V rámci projektu je potrebné vytvoriť niekoľko rôznych dokumentácií. Úlohou manažéra dokumentácie je vytvorenie šablóny dokumentov, definovanie pravidiel a metodík pri ich vypĺňaní. Dokumenty budú vypĺňané viacerými členmi tímu, z toho dôvodu musí dohliadať na správne aplikovanie definovaných metodík a pravidiel, sledovať ich včasné zapracovávanie a kontrolovať obsah a kvalitu dokumentov. V závere projektu je jeho úlohou pripraviť dokumenty na odovzdanie a vytlačiť dokumentácie, ktoré sú vopred určené zadaním projektu.

Okrem vytvárania dokumentov sa v projekte dokumentuje aj zdrojový kód. Úlohou manažéra dokumentácie je určiť nástroj pre generovanie dokumentácie zo zdrojového kódu a definovať konvencie jeho dokumentovania.

[1] http://git-scm.com/
[2] https://github.com/

4 Sumarizácia práce členov tímu za zimný semester

V kapitole je zobrazené rozdelenie úloh členom tímu. Na ich ohodnotenie sa aplikoval prístup ohodnotenia zložitosti úloh pomocou čísel fibonacciho postupnosti. Aby bolo možné preukázať reálnu prácnosť členov tímu, ku každej úlohe sa priradil aj čas, ktorý musel člen tímu stráviť pri jej riešení.

4.1 Šprint č. 1

Zodpovedná úloha	Popis úloh	Ohodnotenie úloh
Bc. Barbora Brocková	Prijímanie správ cez API	5
Bc. Adam Lieskovský	Kostra aplikácie	11
Bc. Matej Čaja	Registrácia a prihlásenie	10
Bc. Martin Číž	Štúdium NewRelicAgent	21
Bc. Peter Kyseľ	Štúdium NewRelicInsights	5
Bc. Daniel Uderman	AirBrakeLib	8
Bc. Michal Viskup	Angular	4

Tab. 3: Podiel práce na prvom šprinte

Každý z členov dostal úlohu na implementáciu a analýzu alternatívneho existujúceho riešenia zadania projektu. Na začiatku 2. šprintu boli výsledky analýz odprezentované.

4.2 Šprint č. 2

Zodpovedná úloha	Popis úloh	Ohodnotenie úloh
Bc. Barbora Brocková	Dokumentácia a generovanie konfig. súboru	7
Bc. Adam Lieskovský	RabbitMQ, Zachyt chýb, testovanie	10
Bc. Matej Čaja	Metodika testovania, Tomcat	5
Bc. Martin Číž	Vizualizácia chýb do grafu	11
Bc. Peter Kyseľ	Emailové notifikácie	5

Bc. Daniel Uderman	Komentáre ku udalostiam	3
Bc. Michal Viskup	Modul prijímania správ, modul spracovania správ	5

Tab. 4: Podiel práce na druhom šprinte

4.3 Šprint č. 3

Zodpovedná úloha	Popis úloh	Ohodnotenie úloh
Bc. Barbora Brocková	Dokumentácia API, Frontend zmeny	3
Bc. Adam Lieskovský	API load test, remote access db/msq	6
Bc. Matej Čaja	Monit M/T, Testy Angular	10
Bc. Martin Číž	Destroy	2
Bc. Peter Kyseľ	Testovanie JAVA	5
Bc. Daniel Uderman	Vytvorenie virtuálneho testovacieho prostredia	5
Bc. Michal Viskup	Modul prijímania správ, modul spracovania správ - rozšírenia	12

Tab. 5: Podiel práce na treťom šprinte

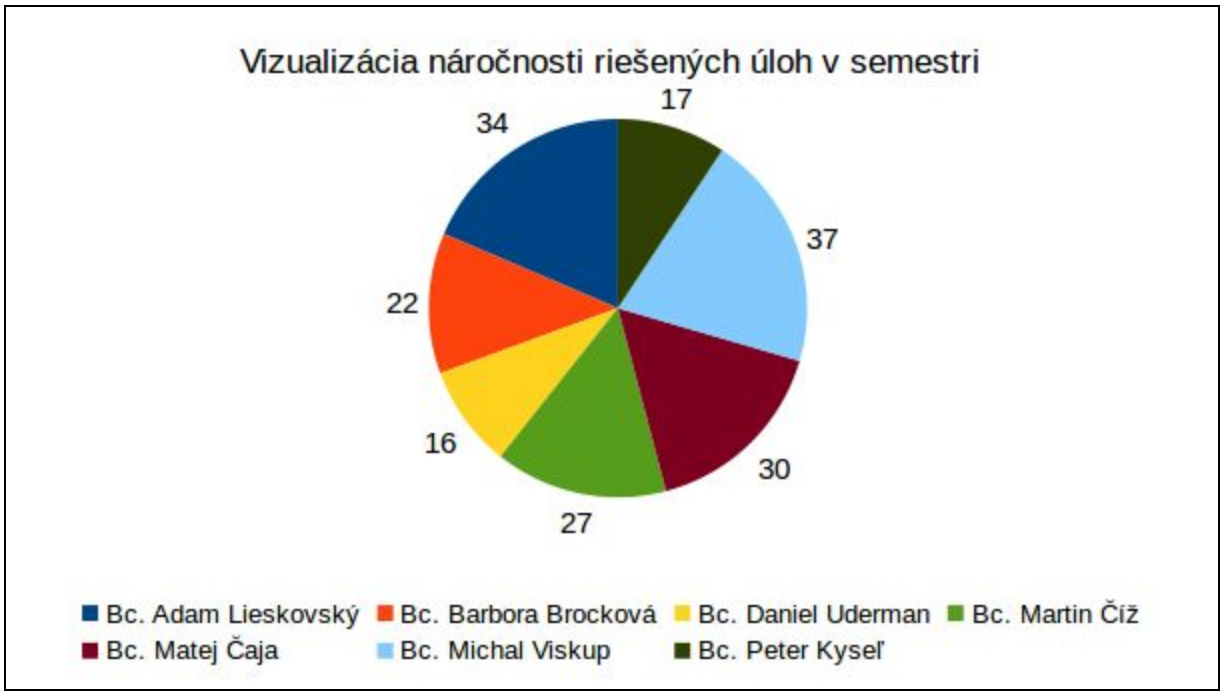
4.4 Šprint č. 4

Zodpovedná úloha	Popis úloh	Ohodnotenie úloh
Bc. Barbora Brocková	Spracovanie deploy msg	5
Bc. Adam Lieskovský	Datove kocky/Java - Add/Create	8
Bc. Matej Čaja	Hbase, Dopytovanie dátovej kocky z Rails aplikácie	10
Bc. Martin Číž	Webové rozhranie pre parametre datovej kocky - Backend	3
Bc. Peter Kyseľ	Filtrovanie udalostí podľa statusu	2
Bc. Daniel Uderman	Webové rozhranie	5

	preparametre datovej kocky - Frontend	
Bc. Michal Viskup	Datove kocky/Java - Add/Create, Performance testy	15

Tab. 6: Podiel práce na štrvtom šprinte

4.5 Vizuálna retrospektíva náročnosti riešených úloh za zimný semester



Graf 1: Vizualizácia náročnosti riešených úloh členmi tímu v semestri

V grafe sa nachádza vizualizácia súm náročností (fibonacciho ohodnotení) úloh, ktorá je vypočítaná cez jednotlivé šprinty v semestri na každého člena tímu. Ako je vidno, medzi ohodnoteniami úloh členov tímu existuje istý nepomer. Je to spôsobené tým, že nie každá pridelená úloha v úvodných šprintoch semestra bola ohodnotená. Dodatočne sa úlohy neohodnotili. V posledných dvoch šprintoch sme už ohodnotili všetky plánované úlohy, ale aj keď sme sa ich snažili rovnomerne prideliť členom tímu, nie vždy to bolo možné. Jednotliví členovia tímu si pri riešení úloh navzájom pomáhali a graf nezobrazuje reálny čas, ktorý venovali úlohám. Tento údaj je možné nájsť v kapitole *Záverečná retrospektíva za zimný semester*.

5 Záverečná retrospektíva za zimný semester

Osoba	Celkový čas
Adam Lieskovský	94.70
Peter Kyseľ	35.50
Daniel Uderman	24.20
Matej Čaja	37.50
Martín Číž	33.00
Barbora Brocková	43.00
Michal Viskup	80.00

Tab. 7: Celkový prehľad času jednotlivých členov tímu

V tabuľke sa nachádza čas členov tímu, ktorý si vykázali v nástroji na evidenciu úloh - Redmine. Rovnako, ako v predchádzajúcej kapitole, je možné si všimnúť, že medzi strávenými časmi členmi tímu je istý nepomer. Je to spôsobené tým, že nie každý člen si v Redmine zaznamenával reálny svoj čas. Mohli nastať nasledujúce situácie:

- člen pracoval na úlohe iného člena tímu a nevykázal sa,
- člen tímu si vykazoval aktivity, ktoré si iní členovia nevykázali,
- niektoré úlohy si vyžadovali väčšiu analýzu, ako iné,
- člen tímu prejavil počas semestra väčší záujem pracovať na projekte. Vyhľadával si nové úlohy a vypomáhal iným členom tímu pri riešení ich úloh.

Podrobnejší prehľad pre jednotlivé úlohy, ktoré vyriešili členovia tímu a na nich strávený čas, je k dispozícii v prílohe dokumentu s názvom retrospektiva_za_zimny_semester.pdf.

6 Sumarizácia práce členov tímu za letný semester

V kapitole je zobrazené rozdelenie úloh členom tímu počas šprintov v letnom semestri. Neaplikovávali sme priraďovanie hodnôt fibonacciho postupnosti pre jednotlivé úlohy a nie každý člen tímu zaznamenával čas riešenia každej riešenej úlohy.

6.1 Šprint č. 1

Zodpovedná úloha	Popis úloh

Bc. Barbora Brocková	SQL parser a preklad SQL na ElasticSearch dopyty
Bc. Adam Lieskovský	Rozbehať ElasticSearch - SQL plugin
Bc. Matej Čaja	Rozbehať Redis a ElasticSearch
Bc. Martin Číž	Template vizualizácie a Rails view
Bc. Peter Kyseľ	CRUD dopyt a dashboard
Bc. Daniel Uderman	Model dopytu v Angulare
Bc. Michal Viskup	Spojenie Redis, ElasticSearch a Shovel (prerobenie Message processora)

Tab. 7: Podiel práce na prvom šprinte

6.2 Šprint č. 2

Zodpovedná úloha	Popis úloh
Bc. Barbora Brocková	Vytvoriť issue na GitHub, Dotazník spokojnosti
Bc. Adam Lieskovský	Nastaviť routing v Railsoch a pridanie do Angularu, refresh dopytu
Bc. Matej Čaja	Získanie dát z ElasticSearchu, kontrola logov
Bc. Martin Číž	Testy na dashboard a boardcelly
Bc. Peter Kyseľ	Emailove notifikácie z ElasticSearchu
Bc. Daniel Uderman	Získanie detailu boadcellu v Angulari, Ukladanie sql dopytov
Bc. Michal Viskup	Oddelenie indexov pre správy a spojenie Rails s Message processorom

Tab. 8: Podiel práce na druhom šprinte

6.3 Šprint č. 3

Zodpovedná úloha	Popis úloh
Bc. Barbora Brocková	Vytvorenie issue na Github bude z Rails
Bc. Adam Lieskovský	SQL dopyty a ElasticSearch
Bc. Matej Čaja	Rollback správ z Elasticu do Postgresql

Bc. Martin Číž	Dizajn stránky a robehať staging
Bc. Peter Kyseľ	Emailové notifikácie a dashboardy
Bc. Daniel Uderman	Zrušiť filter udalostí pri vybraných situáciách
Bc. Michal Viskup	Profylaktika Java modulov

Tab. 9: Podiel práce na treťom šprinte

6.4 Šprint č. 4

Zodpovedná úloha	Popis úloh
Bc. Barbora Brocková	Vytvorenie dotazníka (IIT SRC), Zmena dokumentácie GitHub API, Článok na robime.it, poster, vizitky,...
Bc. Adam Lieskovský	Vytvorenie príkladov na dopyty do EleasticSearch, úprava AppMonitor gemu, Deploy/Config serveru
Bc. Matej Čaja	Deploy/Config serveru, logovací skript
Bc. Martin Číž	Sidebar a prerobiť project na sidebar
Bc. Peter Kyseľ	Posielanie týždenných notifikácií a testy
Bc. Daniel Uderman	Usporiadanie zoznamu eventov
Bc. Michal Viskup	Článok na IIT SRC, úprava Message processoru pre ElasticSearch a Postgresql, deploy/Config serveru

Tab. 10: Podiel práce na štrvtom šprinte

7 Záverečná retrospektíva za letný semester

V letnom semestri sme neohodnocovali zložitosť úloh pomocou čísel fibonacciho postupnosti a nezaznamenávali sme čas riešenia úloh pre všetky úlohy. Z toho dôvodu nie je možné vytvoriť objektívne zhodnotenie vyriešených úloh členmi nášho tímu exportom sumárov z nástroja Redmine. Zo zimného semestra si už jednotliví členovia tímu preniesli svoju špecializáciu a rolu v tíme a pokračovali v nej aj v letnom semestri. Každý člen tímu riešil úlohy, ktoré boli náplňou jeho špecializácie.

8 Opis aplikovaných metodík

V tejto kapitole sa nachádza ***stručný opis metodík***, ktoré sme si navrhli v úvodných stretnutiach. Ich pôvodné znenie sa nachádza v dokumente *Dokumentácia k metodikám*, ktorý nie je prílohou tohto dokumentu, a na ktorého tvorbe sa podieľali všetci členovia tímu.

Každý manažér si vytvoril metodiku pre zjednotenie správania sa členov tímu k jednotlivým problémom, sledovanie ich činnosti a následné overenie jeho navrhnutého prístupu.

9 Manažment kvality

Manažér kvality je zodpovedaný najmä za stanovenie a dohliadanie na dodržiavanie metodík tvorby testov, spracovania chýb a tvorby, dokumentovania a refaktoringu kódu. Zároveň ma na starosti aj meranie kvality kódu pomocou rôznych metrík s využitím podporných nástrojov.

9.1 Konvencie pre testovanie a písanie testov v jazyku ruby

9.1.1 Všeobecné zásady návrhu a tvorby testov

Pokiaľ ide o funkcionalitu, ktorá pokrýva značnú časť projektu, alebo sa jedná o kritickú časť projektu, ktorá si vyžaduje dôkladne pretesovanie, na vytváraní návrhu testov sa môže podieľať aj celý tím s cieľom najmä odhalenia všetkých možných rizík, ktoré môžu vzniknúť, preto aj identifikácia takejto funkcionality spadá pod metodiku *identifikácie rizík*.
Testy však vytvára programátor prevažne sám a dohliada na to, aby pokryl celú naimplementovanú funkcionalitu. V prípade potreby môže programátor použiť podporné nástroje na meranie pokrytia aplikácie testami (*Code Climate*), pomocou ktorých môže nájsť ešte nepokryté časti kódu. Manažér kvality počas šprintu tieto nástroje využíva na monitorovanie stavu softvéru a prípadne urguje k písaniu testov.

9.1.2 Vytváranie testovacích súborov

Preferovaným spôsobom vytvárania súborov je formou automatického generovania, no je možné vytvoriť si súbory aj ručne tak, ako je popísané nižšie.

Ručné vytváranie testovacích súborov:

- Na pomenovanie súborov používame *snake_case*
- Súbory pomenúvamé tak, ako sa nazýva testovaný súbor, pridaním prípony *_spec* na koniec názvu súboru, napríklad:

  Testovaný súbor: *projects_controller.rb*

Testovací súbor: *projects_controller_spec.rb*

- Vytvorený súbor umiestňujeme do jednej zložky s názvom */spec/* pod príslušný priečinok, do ktorého spadá testovaný súbor. Ak teda testujeme *projects_controller,* testovací súbor sa bude nachádzať v:

  ~/project_root/spec/controllers/projects_controller_spec.rb

Automatické generovanie testovacích súborov:

- Vygenerovanie súboru pomocou príkazu *rails generate rspec:pattern názov_súboru.* Napríklad pre controller *project_controller* bude vyzerať príkaz následovne:

  rails generate rspec:controller project

  kde:
  rails generate rspec: - vyvolanie generátora
  controller - určuje, o aký typ súboru sa jedná a kde bude jeho konečné umiestnenie
  project - názov súboru, bez prípony controller, tá bude doplnená automaticky
  Príkaz zároveň vytvorí aj prázdnu šablónu v súbore *factories.rb,* v ktorom sú definované testovacie dáta *FactoryGirl.* Pre ďalšie príklady s použitím iného *pattern*-u je dostupná dokumentácia ku generovaniu súborov v Rspec.

9.1.3 Implementácia testov

Testy naimplementujeme buď podľa návrhu, ktorý vznikne na stretnutí alebo podľa uváženia programátora, na základe funkcionality, tak ako bolo popísané v časti *Všeobecné zásady písania a tvorby testov.*

Pri písaní testov dodržujeme následujúce zásady:

- Testy dostatočne jasne vetvíme pomocou *describe/context/it.* Pričom najvyššiu vetvu pomenúvame pomocou *describe,* poddružené vetvy pomocou *context* a samotné testy pomocou *it.*
- Názvy testov píšeme tak, aby boli tzv. self-explanatory. Ak sa napríklad jedná o vytvorenie nového používateľa projektu - *it "creates new project user"*
- Testy pomenúvame tak, aby pri súvislom čítani dávali zmysluplný text, vyhýbame sa teda pomenovaniam typu:
  - *context 'valid'*
  - *it 'create'*

  Namiesto toho vytvárame pomenovania typu:
  - context 'with valid user data'
  - it 'creates new project user'

  Výnimkou je len najvrchnejšia vetva *describe,* ktorá môže niesť aj jednoduchý názov

(napríklad názov metódy v *controller*-y). Vyhýbame sa pritom slovám typu *mali by (should)*:
namiesto *it 'should create new project user'* napíšeme *it 'creates new project user'*

- Testy by nemali byť príliš komplexné a mali by vždy vykonávať iba jednu úlohu, aby sa zabránilo neprehľadnosti kódu.
- Testovacie dáta a objekty vytvárame tak, aby sme zabránili opakovaniu sa.

Priebeh implementácie testov vychádza z TDD praktiky *red-green-refactor*, teda najprv napíšeme test, ktorý zlyhá (*red*), potom naimplementujeme kúsok kódu, ktorý spôsobi, že napísaný test prejde (*green*) a nakoniec urobíme refaktoring kódu (*refactor*).

9.1.4 Spúštanie a overenie testov

Na spúštanie a overenie testov je možné buď spustiť všetky testy naraz, alebo ich pretestovať jednotlivo. Pri hromadnom spúštaní je možné využiť RubyMine IDE, ktoré využíva celý tím a je preferovaným prostredím, spustením *Run -> Run -> spec:appmonitor.* Prostredie umožňuje aj jednoduché debugovanie, ktoré sa spúšta obdobne *Run -> Debug -> spec:appmonitor.* V prípade nevyužitia IDE je možné spustiť príkazy cez terminál:

1. prechodom do koreňového adresára projektu a
2. spustením príkazu:
    a. *rspec spec/**

    alebo jednotlivo po súboroch:
    b. *rspec spec/cestaKSúboru/nazov_súboru.rb.*

Príkladom pre *project_controller_spec.rb* bude teda príkaz:
rspec spec/controllers/project_controller_spec.rb.

Naimplementované testy musia po spustení všetky prejsť a musia byť 'zelené'. Akýkoľvek iný výsledok testov je považovaný za nedostatočný a testy je potrebné upraviť tak, aby boli dosiahnuté požadované výsledky (podľa praktiky *red-green-refactor*).

9.1.5 Zhodnotenie aplikovania metodiky v praxi

V úvodných fázach projektu (1. a 2. šprint) sa dohodli technológie, ktoré sa na testovanie použijú a vytvorili sa postupne testy na pokrytie časti kódu dovtedy naprogramovanej funkcionality rails aplikácie. Konkrétne sa jednalo o modely a controllery, pričom niektoré z testov boli v ďalších fázach jednotlivých šprintov ešte dodatočne upravené, aby bolo zaručené ich správne fungovanie. Týmto sa podarilo zvýšiť mieru pokrytia aplikácie testami, čo bolo overené aj podporným nástrojom Code Climate. Pri písaní testov boli dodržiavané aj pravidlá, ktoré stanovuje táto metodika.Túto časť hodnotíme kladne. V letnom semestri sme dodržiavali aj naďalej zadefinované pravidlá.

Negatívne hodnotíme nepísanie testov spolu s vytváranou funkcionalitou, testy boli dodatočne alebo neboli vôbec implementované v neskorších fázach projektu. V tomto smere vidíme zlepšenie v budúcnosti dodržiavaním vytvárania testov spolu s vytváranou funkcionalitou.

9.2 Základné pravidlá písania v jazyku Ruby

Pre prehľadné a jednotné vytváranie zdrojového kódu sa zaviedli pravidlá. Z nich sa vybrali najdôležitejšie a bližšie sa popísali v metodike, ktorá sa nachádza v dokumente metodík.

9.2.1 Pomenovanie

Pre pomenovávanie tried, metód, modulov, premenných je zaužívaných niekoľko konvencií. Medzi najdôležitejšie z nich patria: názvy sa píšu v anglickom jazyku, názvy premenných, symbolov a metód sú vždy malým písmenom a jednotlivé slová sú oddelené "_", tzv. snake_case, triedy a moduly sa pomenovávajú tzv. CamelCase, názvy ako HTTP, URL, XML sa píšu vždy veľkými písmenami.

9.2.2 Komentovanie

Komentáre sa píšu v anglickom jazyku. Nepíšu sa v rámci riadku, kde sa nachádza komentovaný prvok, ale riadok nad komentovaným prvkom. Píšeme tak, aby nebolo potrebné komentovanie. Účel metódy a triedy popisujeme konvenciami dokumentovania zdrojového kódu, ktoré sú bližšie opísané v kapitole *Dokumentovanie zdrojového kódu v jazyku Ruby*.

9.2.3 Výnimky (Exceptions)

Pre signalizáciu výnimiek použíme fail metódu. Raise používame, len ak zachytávame výnimku a zasielame ju o úroveň vyššie. Namiesto inštancie chybovej triedy vytvárame dva argumenty. Napr. "*fail SomeException, 'message*'".

9.2.4 Dátové štruktúry

Pre písanie dátových štruktúr sa definovalo niekoľko konvencií, ktoré sú rozdelené podľa štruktúry Hash table a Array (a Set). Medzi najdôležitejšie patrí využívame literálov, ktoré ponúka jazyk Ruby. Dlhšie formy zápisov sú zastarané, uprednostňujeme kratšie zápisy. Používame *Hash#fetch*, ak chceme získať hodnotu kľúča v hash tabuľke, alebo vykonať logické porovnanie. Používame *Hash#values_at*, ak chceme priradiť hodnoty z hash tabuľky viacerým premenným. V prípade, že potrebujeme uchovať pole unikátnych prvkov, použijeme štruktúru *Set*, nie *Array*.

9.2.5 Textové reťazce

Pri písaní textových reťazcov sme definovali niekoľko pravidiel. Pri spájaní reťazcov a ich formátovaní používame metódu format. Ak chceme v zdrojovom kóde napísať reťazec, píšemeho s jednoduchými úvodzovkami, nie zložitými. Ak je nejaký text v reťazci uzavretý v zátvorkách, kvôli pehľadnosti v okolí textu vkladáme medzeru. A iné.

9.2.6 Triedy a moduly

Snažíme sa mať každú triedu vo vlastnom súbore, ktorý je pomenovaný rovnako ako trieda. Hierarchia tried spĺňa princípy Liskovej substitúcie. Metódy sú od *public*, *protected*, *private* oddelené vždy jedným riadkom, kvôli prehľadnosti. V triedach používané príkazy zdrojového kódu splňajú pravidlá, ktoré sú definované v dokumente metodík.

9.2.7 Zhodnotenie aplikovania metodiky v praxi

Metodiku sme definovali na rozpätí druhého a tretieho šprintu. Jej obsah je kombináciou zaužívaných konvencií pri vývoji členmi tímu s najlepšími praktikami pri vývoji, ktoré sme získali z externých zdrojov. Ruby je špecifický jazyk, ktorý dovoľuje využitie širokého množstva postupov pre implementáciu jednotlivých úloh a prvkov. Preto nie je možné jednoznačne označiť jednotlivé implementácie za nevhodné alebo nekonvenčné.

Metodiku, ktorú sme zadefinovali, sme dodržiavali úspešne v celej jej miere, a to vrámci webovej aplikácie, ale aj klientského modulu. Je to z toho dôvodu, že sme našli prienik v prístupe vývoja u členov tímu.

9.3 Konvencie založené na platforme Rails

Existuje niekoľko používateľmi vyhľadávaných editorov zdrojového kódu. Odporúča sa používanie IDE RubyMine vo verzií 6.3, ktoré svojimi funkciami uľahčuje prácu nielen pokoročilým vývojárom, ale aj začiatočníkom.

9.3.1 Pomenovanie

Pri Ruby on Rails platia rovnaké konvencie ako v jazyku Ruby pre pomenovávanie premenných, tried a modulov. Avšak tieto konvencie sú rozšírené o nasledujúce pravidlá pomenovania databázovej tabuľky, pomenovania cudzích a primárnych kľúčov, pomenovanie modelov a controllerov.

9.3.2 Konfigurácia

Konfiguračné nastavenia, ktoré sú aplikovateľné pre všetky prostredia aplikácie, vkladáme do súboru *config/application.rb.* Ak chceme nastaviť špecifické vlastnosti pre konkrétne prostredie, urobíme tak v súbrore *config/environments/nazov_prostredia.rb*. Pre inicializáciu modulov a premenných používame vytváranie súborov v priečinku *config/initializers* tak, aby každý gem mal vlastný súbor.

9.3.3 Routing

Preferujeme používanie generátorov REST-ful routes využitím *resources, member* a *collection*. Ak to nie je nevyhutné, vyhýbame sa použitiu príkazu *match*. Pri definovaní podmodulov aplikácie preferujeme využitie príkazu *namespace*.

9.3.4 Controller

V našej aplikácii aplikujeme všebecne známy štýl “fat model, skinny controller”, v ktorom je výpočtová logika, v čo najväčšej miere, presunutá na úroveň modelu. Controller sa stará iba o spracovanie prichádzajucích požiadaviek a parametrov podsúvania odpovede, či už vo formáte json alebo html. Všetci členovia tímu používajú pri písani kódu IDE RubyMine VLOZIT ODKAZ.

9.3.5 Model (Active Record)

Ak to nie je nutné, neprepisujeme prednastavenú konfiguráciu atribútov modelu (názov databázovej tabuľky, primárny kľúč). Validácie, callback a prístupne atribúty zgrupejeme na začiatku súboru.

9.3.6 View

Pri vytváraní views preferujeme vytváranie tzv. čiastočných (angl. partial) views, ktoré zabezpečujú ich znovupoužiteľnosť. Cieľom využívania komplexného formátovania je premiestniť výpočtovú logiku z views do pomocných súborov (angl. helpers) alebo priamo do modelov.

9.3.7 Migrácia

Ak potrebujeme vytvoriť migrácie nad tabuľkami v databáze, preferujeme metódu *change* namiesto využitia starších *up* a *down*. Súbor *schema.rb* integrujeme do VCS. Ak potrebujeme

definovať prednastavené hodnoty alebo rôzne constrainty, robíme tak už na databázovej úrovni v migráciach.

9.3.8 Assets

V našej aplikácii sa snažíme využívať asset pipeline v maximálne možnej miere. Pri umiestňovaní assetov sme stanovili pravidlá, kedy namiesto manuálneho pridávania jednotlivých súborov (napr. *bootstrap*), použijeme gemifikovanú verziu assetu. Vytvorili sme zložky, do ktorých umiestňujeme assety tretích strán, štýlovacie súbory, javascripty alebo obrázky.

9.3.9 Bundler

Gemy využívane len na určitých prostrediach vkladáme do príslušnej podskupiny gemov v súbore *Gemfile,* ktorými sú napríklad *test* a *development*.

Zhodnotenie aplikovania metodiky v praxi

Pri práci na projekte, sme už od prvých týždňov dodržiavali konvencie frameworku Ruby on Rails. Veľmi pozitívne hodnotíme aj fakt, že členovia tímu, ktorí nemali priame predchádzajúce skúsenosti v tvorbou webových aplikácii postavených na skriptovacích jazykoch a MVC modely sa dokázali úspešne adaptovať hneď od začiatku. Tento fakt potvrdzuje vhodný výber použitých technologií.

Pri funkcionalite odosielania emailových notifikácii jeden člen tímu použil namiesto triedy ActiveMailer iba Model, načo bol pri prehliadke kódu upozornený a napravil svoju chybu.

9.4 Základné pravidlá písania v jazyku Java

Metodika vychádza z metodiky spoločnosti Oracle, *Code Conventions for the Java Programming Language.*[3] Uplatňuje voči uvedenej metodike výnimky v oblasti formátovania zdrojového kódu. Rozširuje ju o metodické pokyny pre pomenúvanie tried, metód a premenných.

9.4.1 Maximálna dĺžka riadku

Maximálna dĺžka riadku je 130 znakov.

[3] http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

9.4.2 Otváracie a ukončovacie zátvorky tried, metód a blokov kódu

Otváracia zátvorka triedy, metódy a bloku sa píše na rovnaký riadok ako definícia triedy, metódy alebo bloku. Ukončovacia zátvorka sa uvádza na nový riadok za posledným príkazom triedy, metódy alebo bloku.

9.4.3 Pomenúvanie balíkov, tried, metód a členských premenných

Balíky, triedy, metódy a členské premenné sa pomenúvajú v anglickom jazyku. Názvy balíkov sa píšu s malým počiatočným písmenom. Názvy tried sa píšu s veľkým počiatočným písmenom. Názvy metód sa píšu s malým počiatočným písmenom. Názvy premenných sa píšu s malým počiatočným písmenom. Ak názov balíka, triedy, metódy alebo členskej premennej pozostáva z viacerých slov, každé ďalšie slovo sa píše s veľkým počiatočným písmenom.

9.4.4 Zhodnotenie aplikovania metodiky v praxi

Metodiku pre písanie Java kódu v plnom jej znení sme pri implementácií Java modulov aplikácie dodržiavali.

10 Manažment rizík

V tíme sme identifikovali možné riziká. Určili sme aké riziká môžu projekt alebo tím ovplyvniť. Pre projekt sme riziká určovali aj všeobecne pre celý projekt, ale aj pre samostatné úlohy, z ktorých projekt pozostával. Manažér rizík tieto riziká spísal a ohodnotil ich pravdepodobnosť a rozsah škôd.

10.1 Opis metodiky

Na identifikáciu a analýzu rizík sme sa riadili nasledovnými metodikami.

10.1.1 Identifikácia rizík

Počas prideľovania úloh je nutné identifikovať možné riziká, ktoré môžu nastať počas ich plnení. **Riziká musia byť identifikované v procese tvorby, hodnotenia a prideľovania úlohy na začiatku šprintu.** Je nežiadúce identifikovať riziká úlohy až na konci šprintu.

Keď počas plnenia úlohy vypláva na povrch nové neočakávané riziko, člen tímu ktorému bola úloha pridelená **je povinný informovať** manažéra rizík, prípadne ostatných manažérov, ktorých

sa môže riziko priamo dotýkať. Rovnako reaguje aj člen tímu, ktorý je pridelený na review úlohy a ktorý identifikoval nové riziko.

10.1.2 Analýza rizík

Po procese identifikácie rizík by sme mali získať zoznam možných rizík. Ten preriedime na zoznam rizík, ktoré vyžadujú nejakú akciu a určíme im prioritu. K rizikám hľadáme možné reakcie a tie ďalej analyzujeme.

Prioritu určujeme na základe pravdepodobnosti výskytu a rozsahu škôd podľa obrázku 1 určovanie priority rizika.

Obr. 1: Určovanie priority rizika[4]

10.1.3 Zhodnotenie aplikovania metodiky v praxi

V začiatočných fázach projektu bolo náročné určiť všetky riziká, pretože sme ešte presne nepoznali ktoré technológie budeme používať. Zároveň sme sa ešte ako tím dobre nepoznali. Z týchto dôvodov boli niektoré riziká identifikované až vtedy, keď reálne hrozili.

Na riziká sme však vždy včas reagovali a každé riziko sa nám podarilo vyriešiť alebo mu celkom predísť.

[4] Prednáška *Manažment v SI a IS #3 Plánovanie a riziká, Unicorn Systems*

Zoznam udalostí v tíme, kedy nastali rizikové situácie:

- Člen tímu nestihol dokončiť svoju úlohu počas šprintu. Riešením bol presun úlohy na ďalší šprint.
- Člen tímu počas tvorby úlohy vytvoril pesimistický odhad rizík, ktoré môžu nastať na serveri. Riešením bolo vytvorenie novej úlohy do ďalšieho šprintu na vytvorenie riešenia.
- Viacerí členovia tímu raz alebo viac krát meškali na stretnutie. Vymysleli sa rôzne motivačné úlohy pre meškajúcich v závistosti od toho ako dlho meškali.
- Člen tímu zverejnil funkcionalitu, ktorá vyradila predošlú funkcionalitu. Táto zmena sa však dostala len do vetvy vývoja, kde sa následne opravila, takže nedošlo k škode v produkcii.

V letnom semestri sme neidentifikovali žiadne nové riziká.

11 Manažment plánovania

Manažér plánovania prezentoval metodiku plánovania úloh v nástrojoch Trello a Redmine. Definoval procesy, ktoré je protrebné vykonávať pri otváraní, priebežnej práci, uzatváraní a kontrolovaní jednotlivých úloh. Manažér plánovania túto metodiku spísal a skontroloval jej dodržiavanie.

11.1 Opis metodík

11.1.1 Evidovanie vlastností systému v nástroji Trello

Táto metodika definuje proces navrhovania, schvaľovania a archivácie jednotlivých produktových vlastností v nástroji Trelo. Pomocou tohto nástroja udržiavame produktový backlog. Jednotlivé vlastnosti sú zobrazené vo forme virtuálnych nalepovacích štítkov a ich zoznamov umiestneních na interaktívnej tabuli. Tieto štítky farebne označujeme podľa stavu ich svaľovania. Rovnako archivujeme tie, ktoré sú aktuálne vyriešené a implementované.

11.1.2 Evidovania práce na úlohách v nástroji Redmine

Predmetom tejto metodiky je definovanie procesov spojených s evidenciou práce na úlohách evidovaných v nástroji Redmine. Definuje spôsoby a jednotlivé používané stavy v ktorých sa môžu nachádzať jednotlivé úlohy. Zároveň definuje spôsob evidencie času strávenom riešením jednotlivých úloh a periodicitu aktualizácie stavu riešenia jednotlivých úloh. Nezabúda ani na spôsob evidencie výstup z konkrétnych úloh.

11.1.3 Zhodnotenie aplikácie metodík v praxi

Vypracované metodiky členovia tímu dodržiavajú až na ojedinelé situácie ktoré manažér plánovania včasne reviduje. Jednotlivé výstupy úloh sú vhodne uvedené a relevantné k zadaniu úlohy. Úlohy sú vypracovávané prevažne s časovým predstihom.
Situácie kedy nebol plán v riadnom čase dokončený boli riešené presunutím úloh do nasledujúceho šprintu.

V Letnom semestri členovia tímu pravidelne pracovali na zverených úlohách, Úlohy boli po každudom stretnutí evidované v nástroji redmine avšak progres práce na nich bol prevažne reportovaný priamo na oficiálnych tímových stretnutiach a mimo nich. Boli určené osoby zodpovedajúce za každú z logických častí vyvýjaného riešena ako napríklad vizuálizácia dát, spracovávanie prúdu dát, a nasadzovanie používaných technológií a udržiavanie servového prostredia. Takýmto modelom sme vedeli včasne riešiť drobné časové sklzy voči plánu práce a úlohy dokončiť v očakávanom čase.

12 Manažment monitorovania

Manažér monitorovania navrhol metodiku pridávania úloh do project managment toolu Redmine. Následne skontroloval či sú úlohy priradené osobám, ktoré danú úlohu riešili. Na začiatku šprintu pridával nové úlohy do Redmine a na konci šprintu prípadné neuzatvorené úlohy konzultoval s riešiteľmi.
V rámci monitorovania projektu, sme hneď v prvom šprinte začali používať Redmine. Problémom bolo však, že sme nesprávne estimovali úlohy ako aj nepriaďovali user points. Preto, následne výsledný burndown chart, na konci šprintu neuvádzal skutočný stav šprintu. Druhý šprint sme ohodnotili všetky pridelené úlohy, tým pádom sa tento problém eliminoval. Taktiež sme začali používať planning poker, pre určenie náročnosti úloh vychádzajúc, z čísel Fibbonaciho postupnosti. Úlohy pridávané do nasledujúceho šprintu sú vyberané z product backlogu, pre ktorý sme zvolili Trello.
Koncom druhého šprintu a začiatkom tretieho sme narazili na problém nedostatku času na dokončenie pridelených úloh. Riešením bolo prenesenie nedokončených úloh do nasledujúceho šprintu pričom sa nemenili ich user story points. Avšak navyše boli členom tímu priradené ďalšie úlohy na daný šprint.

12.1 Opis metodiky

V tejto kapitole sa nachádza opis aplikovaných metodík, ktoré sa dodržiavali členmi tímu.

12.1.1 Proces vytvárania a prideľovania úloh na začiatku šprintu

12.1.2 Vstupný stav:

- Existuje backlog s používateľskými príbehmi a ich prioritami.
- Tento backlog sa nachádza v nástroji Redmine v Backlogs
- Prebieha stretnutie členov tímu s vlastníkom produktu.
- Začína nový šprint.

12.1.3 Výstupný stav:

- Sú zadané úlohy, pre nasledujúci šprint spísané v zápisnici zo stretnutia.
- Úlohy majú svoje zodpovedné osoby a sú odhadnuté veľkosti/čas trvania daných úloh.

Postup:

1. Vlastník produktu vyberie z backlogu používateľské príbehy, pre ďalší šprint. Nasleduje diskusia s tímom. Prípadné návrhy a pripomienky tímu.
2. Používateľské príbehy sa rozdelia na menšie úlohy, na vypracovanie.
3. Jednotlivé úlohy sa ohodnotia zložitosťou a to tak, že každý člen tímu odhadne zložitosť úlohy číslom z Fibonaciho postupnosti (1,2,3,5,8,13...). Maximálny a minimálny odhad vysvetlí prečo si myslí, že je jeho odhad adekvátny. Postup sa opakuje pokiaľ sa celý tím nezhodne na jednom čísle. Následne sa určí náročnosť úlohy.
4. Členovia tímu si vyberajú úlohy v iteráciách pokiaľ sa nevyberú všetky úlohy.
5. Porovná sa, či sa rovnajú súčty zložitostí úloh každého člena tímu. Ak to nie je možné dosiahnuť, tak aspoň s minimálnymi odchýlkami medzi jednotlivými členmi.
6. Po skončení stretnutia na začiatku šprintu manažér monitorovania vytvorí nový šprint v nástroji Redmine a následne priradí dohodnuté úlohy členom tímu.

12.1.4 Proces vytvárania a prideľovania úloh počas šprintu

12.1.5 Vstupný stav:

- Člen tímu si myslí, že je potrebné vytvoriť novú úlohu.

12.1.6 Výstupný stav:

- Úloha má svoju zodpovednú osobu a zložitosť.

12.1.7 Postup:

1. Člen tímu navrhne tímu cez vopred určený komunikačný kanál pridanie novej úlohy počas šprintu.
2. Na základe diskusie s ostatnými členmi sa rozhodne, či sa vytvorí nová úloha. Ak sa nová úloha zamietne proces končí, inak pokračuje v bode 3.
3. Určí sa zložitosť úlohy na základe dohody medzi členmi tímu.

4. Vyberá sa zodpovedná osoba. Pri výbere sa zohľadňujú odpovede na otázky:
    - Kto má záujem úlohu riešiť?
    - Kto mal na začiatku šprintu úlohy s najnižším číslom zložitostí (ich súčet)?
5. Člen tímu, ktorý inicioval vytvorenie novej úlohy sa stáva zadávateľom. Nová úloha je neodkladne zadaná zadávateľom do nástroja Redmine.

12.1.8 Zhodnotenie aplikovania metodiky v praxi

V priebehu prvého šprintu sme nedodržali metodiku a neohodnotili sme pri všetkých úlohách zložitosť. Tento problém sme eliminovali pridaním zložitosti v druhom šprinte. Následne nám vznikali úlohy počas šprintu, ktoré sme pridávali a spracovávali bez použitia metodiky na pridávanie úloh počas šprintu. Problém bol vyriešený tak, že počas prebiehajúceho šprintu sa už nové úlohy nevytvárajú, bez konzultácie s ostatnými členmi tímu a ak má niekto návrh na pridanie úloh, aplikuje sa vytvorená metodika. Počas ohodnocovania úloh Fibbonaciho postupnosťou počas druhého šprintu sme neohodnocovali korektne. Správne ohodnocovanie má vzniknúť vzájomnou dohodou na konkrétnom čísle Fibbonaciho postupnosti. My sme vybrali strednú hodnotu navrhnutej zložitosti. Zmena bude zapracovaná v nasledujúcom šprinte.

13 Manažment podpory vývoja a integrácie

Manažér vývoja navrhol metodiku pre udržiavanie prostredia pre vývoj a testovanie produktu. Kontroloval dodržiavanie stanovených pravidiel pri verziovaní a odovzdávaní nových častí zdrojového kódu. Naša metodika vychádzala z prístupu V.Driessena - A successfull Git branching model[5], ktorú sme jemne modifikovali a z práce v nástrojoch *Git*[6] a v sieti *Github*[7].

13.1 Opis metodiky

V tejto kapitole sa nachádza opis aplikovaných metodík, ktoré sa dodržiavali členmi tímu.

13.1.1 Postup vetvenia zdrojového kódu

Pre spôsoby vetvenia zdrojového kódu existuje viacero spôsobov a ustálených metodík. Nami definovaná metodika je modifkáciou “A successfull Git branching model”.

Pre pomenovanie vetiev platia vždy nasledovné pravidlá:

[5] http://nvie.com/posts/a-successful-git-branching-model/
[6] http://git-scm.com/
[7] https://github.com/

- Názvy vetiev, podobne ako aj názvy odovzdaní píšeme výlučne v anglickom jazyku s malými písmenami.
- Na oddelovanie anglických slov slúži pomlčka

V rámci zdrojového kódu definujeme nasledovné vetvy:

- **master - produkčná vetva (angl. production)** je vetva, ktorá je nasadená na serveri v ostrej prevádzke zákazníkmi.
- **development - vývojová vetva (angl. development),** do ktorej sa spájaju vetvy nových funkcionalít a aj kritických opráv.
- **feature - vetvy s novou funkcionalitou (angl. feature),** pričom v jednej vetve sa implementuje jedna nova funkcionalita. Tieto vetvy sú spájané do vývojovej vetvy.
- **hotfix - Vetvy s opravou kritických chýb (angl. hotfix).** Ich zmeny sa spájajú do produkčnej aj vývojovej vetvy.

13.1.2 Princípy odovzdávania nových funkcionalít

Jednotlivé odovzdania (commits) by mali obsahovať vždy ucelenú zmenu len jednej funkcionality. Môžu zasahovať do rôznych častí zdrojového kódu, ale ako celok spolu musia súvisieť. Odovzdania prinášajú možnosť stručne identifikovať úpravy v kóde pomocou prislúchajúcej správy (angl. Commit message). Pri odovzdávaní sme zadefinovali nasledovné pravidlá:

- Odovzdania sa píšu výlučne v anglickom jazyku.
- Pri odovzdávaní neponechávame zakomentované predchádzajúce verzie zdrojového kódu. Na možný návrat slúži práve nástroj Git.
- Ak prvý znak správy neoznačuje názov súboru alebo objektu, ktorý si to vyžaduje, píšeme ho vždy veľkým písmenom, bodku na konci správy nepíšeme.
- Ak chceme vytvoriť správu pri odovzdaní s obsahom väčším ako 50 znakov, za prvým riadkom dĺžky 50 znakov necháme prázdny riadok a zvyšok správy píšeme pod ním.
- Ak vykonávame zmenu previazateľnú s PMS, do správy pridáme príslušný identifikátor úlohy v nasledovnom tvare #číslo_úlohy.
- Pri uvádzaní odkazov alebo skratiek používame korektný tvar napr. GitHub, JSON. Ak špecifikujeme triedy, metódy alebo súbory zo zdrojového kódu, používame ich existujúci, korektný tvar.

13.1.3 Odovzdávanie kritických opráv zdrojového kódu

Pre spájanie vetiev kritických opráv (hotfixov) platia nasledujúce pravidlá:

Vetví sa od	**Spája sa do**	**Názov**

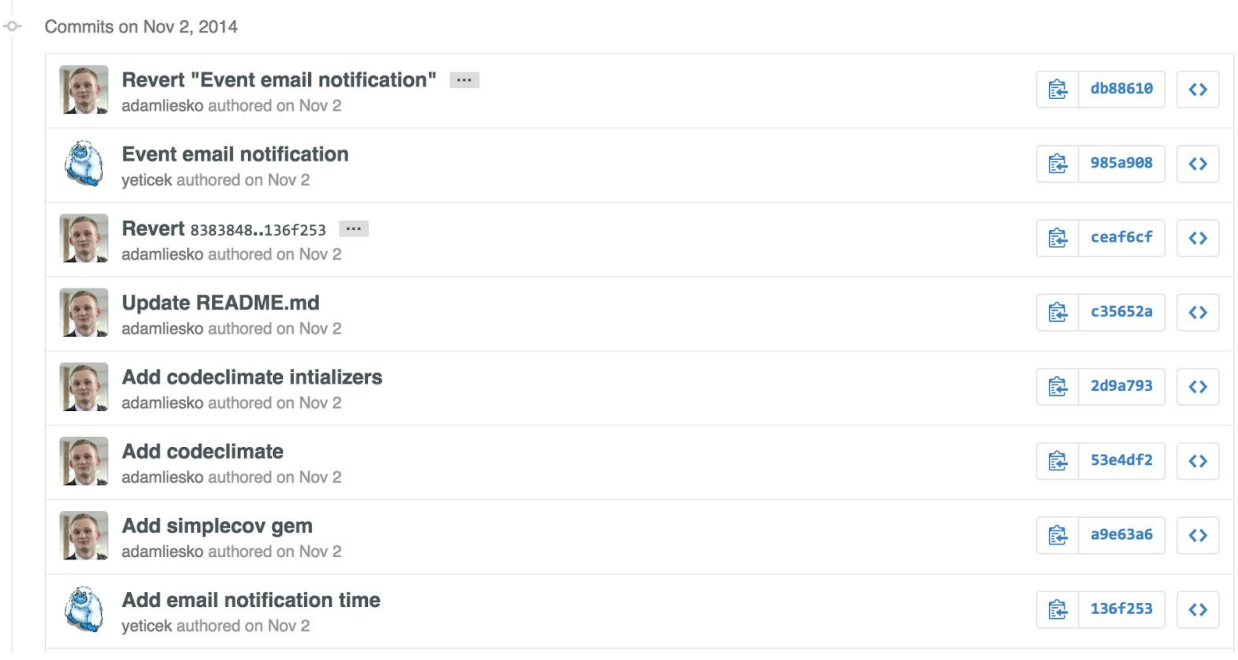
master	development, master	Hotfix-* Jej názov obsahuje číslo verzie na produkcii a číslo problem v PMS.

13.1.4 Zhodnotenie aplikovania metodiky v praxi

Zhodnotenie aplikovania metodiky rozdelíme vzhľadom na realizované šprinty do viacerých časti. Dôvodom je, že pre mnoho členov tímu, bola práca s vybraným webovými technológiami niečim novým. Navyše, aj keď väčšina členov nášho tímu používala v minulosti nástroj na verziovanie, nemusel to byť práve nami využívany Git, prípadne s Gitom nepracovali až do takej úrovne, akú sme si stanovili (vetvenie kódu, pull requesty).

Prvý šprint

V tejto úvodnej fázi projektu úspešne hodnotíme dodržiavanie pravidiel pri písaní odovzdaní jednotlivých častí kódu (angl. commits). Veľmi pozitívne môžeme zhodnotiť aj dodržiavanie vetvenia kódu, pravidiel pri pomenúvaní, kedy až na jedného človeka, členovia tímu pracovali s vývojou vetvou, a pre jednotlivé funkcionality výtvarali samostatné vetvy, ktoré neskôr spájali do vývojovej vetvy. Obrázok 2 viditeľne ukazuje dvojnásovbný pokus o nahratie nedokončnej a nedostatočne otestovanej funkcionality na produkčńú vetvu, kde sme následne anulovali túto akciu.



Obr. 2: Revertovanie commitu na produkčnej vetve

Negatívne hodnotíme nevyužívanie funkcionality “pull requestu”. Dôvodom je ale neskoršie vyjasnenie si medzi členmi tímu, ako a kedy túto funkcionalitu využívať. V tejto praktike vidíme priestor na zlepšenie práce v nasledovných šprintoch.

Druhý šprint

Priestor pre nasadzovanie aplikácie využil len manažér podpory vývoja, čo sa dá vysvetliť tým, že v tejto fázy bol prototyp aplikácie len na experimentálnej úrovni, a zároveň sa členovia tímu ešte len zoznamovali s využívanými technológiami, pričom uprednostnili tie najnutnejšie na fungovanie v tíme. Proces nasadzovania aplikácie mohol vykonávať a aj môže naďalej iba jeden človek, čiže v tomto smere nemusíme ako tím nutne meniť svoj proces práce.

Tretí šprint

Od predošlého šprintu sa už nevyskytli situácie, kedy členovia tímu odovzdávali kód priamo do produkčnej vetvy. Stále ale pretrvávajú rezervy vo forme zabúdania vytvárania tkzv. pull reuqestov, pri odovdzávaní väčších častí kódu.

Štvrtý šprint

V období štvrtého šprintu bola frekvencia rozširovania zdrojového kódu nižšia, implementácia väčšieho funkčného celku bola vrámci vetiev spojená cez pull request, čím členovia tímu dodržali stanovené konvencie v metodikách.

Priestor na zlepšenie a záver

Hlavným a pretvrávajúcim problém je absencia vytvárania tzv. pull requests na odovzdanie ucelených funkcionalít do vývojovej vetvy. GitHub poskytuje veľmi jednoduché a prehľadné rozhranie, ktoré nám ako tímu ponúka rýchly prehľad na odovzdania, príjemný spôsob kolaboráie (komentáre priamo pri častiach kódu). Preto si myslíme, že je škoda nevyužívať danú funkcionalitu, ktorá pomáha zvýši't kvalitu nami odvedenej práce. Celkovo sme ako tím vytvorili len 8 pull requestov, čo nekorešponduje s množstvom implementovanej funkcionality a počtu commitov.

Aplikácia bola na server nasadzovaná stále len manažérom podpory vývoja. Do tejto aktivity by sa mali zapojiť aj ostatní členovia tímu. Nejedná sa len o samotnú webovú aplikáciu, ale aj o ostatné moduly systému.

Aktívny záujem o integráciu jednotlivých prvkov na servery prejavujú konštantne len dve osoby, čo nie je negatívny fakt. Vytvára to predpoklad, že v tejto práci budú naďalej pracovať v tomto menšom počte.

Obvykle máme aktívnych päť a viac feature vetiev, ktoré ale jednotliví členovia tímu aktualizujú o pridanú funkcionalitu do hlavnej vývojovej vetvy až v skutočne posledných momentoch, čím si spôsobujú možné riziká.

14 Manažment komunikácie

Manažér komunikácie stanovil komunikačné kanály pre tím, pripravil tieto kanály z hľadiska konfigurácie a stanovil metodiky používania využitím ich vlastností a ich previazania medzi sebou.

Navrhol dve úrovne komunikácie:

1. komunikačný kanál pre každodennú komunikáciu,
2. potreba zdieľania dát.

14.1 Opis metodiky

V tejto kapitole sa nachádza opis aplikovaných metodík, ktoré sa dodržiavali členmi tímu.

14.1.1 Komunikačný kanál pre každodennú komunikáciu

Pre každodennú komunikáciu bol vybraný nástroj Hipchat, pretože je prístupný prostredníctvom webovej, desktopovej aj mobilnej aplikácie.

14.1.2 Princípy využívania chatových skupín

Pre potreby separátnej komunikácie podľa doménových špecifík sme vytvorili 3 chatové skupiny:

- Development (vývoj)
- Documentation (dokumentácia)
- Gappers (miestnosť pre iné potreby tímu)

Princípy využívania chatovej skupiny Development:

1. miestnosť je určená na diskusiu o implementácií projektu (ujasnenie technických detailov implementácie a otázky tímu ohľadom zdrojového kódu),
2. miestnosť je prepojená s repozitárom zdrojového kódu (GitHub), takže sa v nej zobraujú informácie o každom odovzdaní novej funkcionality do zdielaného repozitára.
3. miestnosť je prepojená s nástrojom Capistrano, ktorý do nej posiela správu pri každom nasadení aplikácie.
4. pomocou znaku @ sa označujú osoby, ktoré majú na danú otázku alebo problém odpovedať.

Princípy používania chatovej skupiny Documentation:

1. miestnosť je určená na diskusiu o formáte, obsahu a stave dokumentácie projektu.

2. po zásahu do dokumentácie do nej autor zmeny v dokumentácií vloží krátku správu o povahe zásahu.
3. pomocou znaku @ sa označujú osoby, ktoré majú na danú otázku alebo problém odpovedať, majú úlohu v správe definovanej dokumentácií.

Princípy používania chatovej skupiny Gappers:

1. miestnosť je využívaná pre komunikáciu o záležitostiach, ktoré nespadajú do prvých dvoch skupín.
2. v prípade, že ide o informácie irelevantné k projektu, je zakázané označovať členov tímu.

14.1.3 Prístup k zdieľaniu dát

Pre zdieľanie dokumentov a iných dát, na ktorých tvorbe spolupracujú členovia tímu sa stanovili nasledujúce služby:

- GoogleDrive - člen tímu pracuje na dokumente. Dokument sa nachádza v priebežnej verzií. Po sfinalizovaní dokumentu sa začlení do príslušnej zložky.
- Dropbox - ďalší prístup pre zdieľanie pracovných verzií dokumentov.

14.1.4 Zhodnotenie aplikovania metodiky v praxi

Nástroj Hipchat bol využitý v priebehu celého semestra v značnej miere. Pri potrebe kontaktovania iných členov tímu sme primárne využívali práve Hipchat a funkciu znaku @, ktorá osobe označenej týmto znakom posiela priamo email s textom správy a linkom na miestnosť kde bola označená. Komunikácia tak bola sústredená na jedno miesto. Nepotvrdila sa obava, že použitím Hipchatu stratíme možnosť spätne dohľadať komunikáciu (čo je bez problémov možne napr. v prípade emailu). Vzhľadom na to, že táto komunikácia bola výhradne vnútrotímová, takáto potreba nevznikla.

V letnom semestri intenzita komunikácie vnútri tímu poľavila. Rôzna úroveň aktivity členov tímu sa prejavila i na intenzite komunikácie, niektorí členovia tímu mimo tímových stretnutí vôbec nekomunikovali. Jediným komunikačným kanálom zostal Hipchat, avšak nastávali situácie kde na správy v ňom nik nereagoval aj niekoľko dní. Upatňovanie metodiky komunikácie v letnom semestri preto možno označiť ako žiadne až mizivé.

15 Manažment tvorby dokumentácie

Manažér dokumentácie vytvoril šablóny dokumentov, ktoré je potrebné vypracovať a následne odovzdať v kontrolných bodoch. V jednotlivých dokumentoch stanovil metodiku ich vypĺňania tak, aby sa zjednotila forma a štýl písania všetkých členov tímu.

V projekte sa definovalo niekoľko typov dokumentácií, ktoré sú určené pre:

- zaznamenávanie priebehu stretnutí (zápisnice zo stretnutí)
- dokumentovanie riadenie projektu (Dokumentácia k riadeniu projektu)
- dokumentovanie inžinierskeho diela (Dokumentácia inžinierskeho diela)
- opis modulov systému (Dokumentácia modulov systému)
- vytvorenie inštalačnej príručky (Inštalačná príručka)
- dokumentácia kompetencií členov tímu (Zoznam kompetencií tímu)

Okrem vytvárania dokumentov sa v projekte dokumentuje aj zdrojový kód. Úlohou manažéra dokumentácie je určiť nástroj pre generovanie dokumentácie zo zdrojového kódu a definovať konvencie jeho používania.

15.1 Opis metodiky

V tejto kapitole sa nachádza opis aplikovaných metodík, ktoré sa dodržiavali členmi tímu pri vytváraní dokumentov projektu a dokumentovaní zdrojového kódu.

15.2 Konvencie pri vytváraní dokumentov

15.2.1 Princípy vytvorenia dokumentu

Dokumenty sú vytvárané vo voľne dostupnom textovom editore Google Doc[1].

Pomenovanie dokumentu je nasledujúce: *tp_názov_dokumentu*. Každý dokument sa skladá z týchto častí:

- úvodná strana (prvá strana dokumentu; obsahuje tieto povinné údaje: názov a sídlo školy, názov projektu, typ dokumentu, mená členov tímu a akademický rok)
- obsah dokumentu (druhá strana dokumentu)
- Kapitoly a podkapitoly (podľa definovanej šablóny dokumentu)

15.2.2 Princípy formátovania textu

Pre jednotný formátovania textu sa pri písaní využívajú tieto pravidlá. Pre rozlišovanie textu sú využívané vstavané štýly v Google docs. (1. úroveň nadpisu je štýl Heading 1, 2. úroveň je štýl Heading 2, obyčajný text je štýl Normal text, atď.)
Nastavenie úvodnej strany je nasledujúce:

- Názov a sídlo školy.
- Názov projektu.
- Typ dokumentu.
- Mená členov tímu a akademický rok.

Vkladaný obsah je generovaný v Google docs. Definovali sa konvencie pre vkladanie prádznych riadkov kvôli prehľadnosti voľného textu. Rovnako aj formy citovania zdrojov a použitej literatúry. Pre každý typ dokumentu je vytvorená šablóna. Táto šablóna sa používa.

15.2.3 Princípy uloženia a zdieľania súborov

V službe GoogleDrive je vytvorená zložka Gappers, kde sa uchovávajú všetky dokumenty a ich šablóny, ktoré súvisia s projektom. Zápisnice sa ukladajú do podpriečinka Zápisnice. Pre každý semester sa nachádza zvlášť.

15.2.4 Zhodnotenie aplikovania metodiky v praxi

V prvom šprinte sa ako prvý dokument vytvorila priebežná verzia šablóny zápisnice. Následne sa manažérom dokumentovania vytvorila finálna šablóna, ktorá sa používala vo zvyšných šprintoch. Spolu s ňou bolo vytvorené aj úložisko zápisníc a šablón pre dokumenty odovzdávané v zimnom semestri. Zápisnice boli pravidelne vytváraným dokumentom, ktorý sa riadil podľa pravidiel metodiky a v súlade s ňou boli ukladané na úložisko zápisnic priamo po stretnutí. Dokument metodík sa začal napĺňať na prielome prvého a druhého šprintu, kedy vznikla metodika k manažmentu vývoja a podpory integrácie, manažmentu komunikácie a manažmentu dokumentovania. Tým, že metodika je postavená na východískových štýloch google docu, členovia tímu s jej napĺňaním nemali väčšie problémy. Koncom tretieho šprintu sa uzavreli dokumenty tak, aby ich bolo možné odovzdať v 1. kontrolnom bode. Rovnako sa stalo aj pri odovzdaní dokumentov v 2. kontrolnom bode.

V letnom semestri sa používali šablóny dokumentov, ktoré boli vystavené na začiatku zimného semestra. Počas každého stretnutia do 16.4. (pred prezentáciou IIT SRC) sa pravidelne vytvárali zápisnice zo stretnutí a ukladali na vopred určené miesto - úložisko zápisníc LS. Aktualizácia dokumentácie inžinierskeho diela a riadenia sa začala približne týždeň pred finálnym odovzdaním tímového projektu. Každý člen si zapracoval časť dokumentu, ktorá sa týkala jeho manažmentu a metodiky alebo časti zdrojového kódu.

15.3 Dokumentovanie zdrojového kódu pomocou Yardoc

Dokumentácia je generovaná pomocou Yardoc-u na základe štruktúrovaného opisu (jej vstupných a výstupných parametrov, účel metódy alebo triedy) nad každou definíciou metódy a triedy. Tento opis je zakomentovaný, a ďalej ho budeme označovať ako “popisný komentár”.

15.3.1 Pravidlá používania tagov a direktív

Pre písanie popisných komentárov nad metódami a triedami platí niekoľko pravidiel. Popisné komentáre sa píšu v anglickom jazyku. Pre prehľadnosť a čitateľnosť sa definovali pravidlá oddeľovania textu prázdnym riadkom alebo medzerou od znaku “#”.
Štandardný formát popisného komentáru:

```
# Method description
#
# @param name [Types] description
# @return [Types] description
def method_name(name) # method body end
```

Pre odkaz na popisný komentár inej metódy, používanie typov parametra sa ukázala jeho syntax. Preferujeme komentovanie hash parametrov a atribútov so zložitým (komplexným) typom. V dokumente metodík je zobrazený príklad na *overload* popisu vstupných atribútov.
Definovanie logickej viditeľnosti metód, tried a parametrov sa nepoužíva na skrytie nezdokumentovaných alebo nedôležitých metód, ale na označenie viditeľnosti tých objektov, ktorým tak nenastavujú pravidlá v Ruby. Možné výnimky a chyby taktiež popisujeme, aby sme sa v budúcnosti vystríhali možných nepredvídateľných situácií.

15.3.2 Špeciálne nastavenie Yardocu

Vo východiskovom nastavení Yardoc generovania dokumentácie je, že sa generujú len public metódy. My sme si v konfiguračnom súbore .yardopts upravili nastavenie tak, aby sa nám do dokumentácie generovali aj private metódy (--private).

15.3.3 Zhodnotenie aplikovania metodiky v praxi

Metodiku dokumentovania zdrojového kódu sme vytvorili až v úvode 3. šprintu, z toho dôvodu nebola metodika uvedená do praxe v rozsahu, v akom sa popisuje. Na konci 3. šprintu členovia tímu dokumentovali svoj zdrojový kód priebežne a to niekoľkými spôsobmi:

- Popis účelu triedy a metódy. Avšak nie každá metóda a trieda je zdokumentovaná.

- Popis internej výpočtovej logiky triedy v prípade, že rieši zložitejšiu funkcionalitu.
- Popis vstupných a výstupných atribútov sa nerealizoval.

V letnom semestri sme upustili od dokumentovania zdrojového kódu v Ruby, pretože ten sám je prehľadný a metódy sú zrozumiteľne pomenované.

15.4 Dokumentovanie zdrojového kódu v jazyku Java

Metodika stanovuje písanie Javadoc komentárov a štandardných komentárov. Stanovuje jednotný formát opisu tried, premenných a metód prostredníctvom komentárov Javadoc. Predpisuje anotácie, ktoré majú byť použité.

15.4.1 Javadoc komentáre

Javadoc komentáre sa píšu výhradne v anglickom jazyku. Odkazy na iné triedy v Javadoc komentároch sa píšu prostredníctvom anotácie @link.

15.4.2 Komentáre tried

Komentár triedy obsahuje meno autora a verziu. Tieto sa píšu prostredníctvom anotácií @author, @version. Trieda sa opisuje z globálneho hľadiska - akú entitu predstavuje/akú funkcionalitu zastrešuje. Členské premenné triedy, ktoré sú viditeľné mimo tela triedy sa komentujú Javadoc komentármi. Metódy viditeľné mimo tela triedy sa komentujú Javadoc komentármi.

15.4.3 Komentáre metód

Pri popise parametrov sa používa Javadoc anotácia @param. Pri opise návratovej hodnoty sa používa @return. Ak metóda deklaruje vyhodenie výnimky, príslušná situácia je opísaná prostredníctvom anotácie @throw.

15.4.4 Poznámky programátora

Poznámky programátora sa píšu výhradne v anglickom jazyku. Píšu sa prostredníctvom štandardných komentárov. Komentáre sa píšu vždy na samostatný riadok. Ak má komentár viac ako 1 riadok, každý riadok sa začína otváracími znakmi komentáru. Otvárací a zatvárací znak viacriadkového komentára sa nepoužíva.

15.4.5 Zhodnotenie aplikovania metodiky v praxi

Metodika dokumentovania Java kódu bola pri dokumentovaní Java modulov aplikácie uplatňovaná v celom svojom rozsahu v zimnom aj v letnom semestri.