ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS
Student:	Oldřich Malec
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je revize, reorganizace a nastavení řízení projektu vývoje a infrastruktury portálu DBS pro podporu výuky databázových systémů. Realizační týmy jsou sestaveny ze studentů SP1, SP2 a studentů pracujících na BP.

1. Nastudujte a krátce popište metody řízení vývoje SW systému, kde je třeba koordinovat práci více týmů a jednotlivců.
2. Popište stávající proces řízení projektu, proveďte analýzu s cílem odhalit slabá místa, navrhněte reorganizaci procesu a realizujte ji. Pracujte v roli projektového manažera.
3. Navrhněte, realizujte a zdokumentujte infrastrukturu, která se pro vývoj i samotné nasazení použije. Soustřeďte se zejména na efektivitu vývoje a znovupoužitelnost kódu.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 28. listopadu 2016

České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra softwarového inženýrství

Bakalářská práce

Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS

Oldřich Malec

Vedoucí práce: Ing. Jiří Hunka

7. května 2017

Poděkování

Na následujících řádcích bych chtěl poděkovat všem kolegům, kamarádům, mentorům i spolupracujícím studentům, se kterými jsem měl tu čest společně tvořit výukovou aplikaci, díky které mohla vznikout tato práce. Výčet bude dlouhý jelikož jsem u projektu již třetím rokem, a zejména studenti v realizačních týmech se střídají často. Nejprve musím samozřejmě poděkovat Ing. Jiřímu Hunkovi, který jakožto duševní autor aplikace, hlavní tester a vizionář nových požadavků a především jakožto vedoucí této práce, na toto místo patří. Dále musím zmínit dva excelentní studenty - Filipa Glazara a Pavla Kováře, kteří mi pomáhali s vedením týmů, podáváním zpětné vazby na kód studentů ale i řešením kritických chyb v produkční verzi, a to i přesto, že měli dost práce se svými vlastními bakalářskými pracemi. Zcela jistě patří díky i dalším výborným studentům, kteří pro DBS projekt vytvořili své bakalářské práce: Milanu Vlasákovi, Petru Pejšovi, Jiřímu Slavotínkovi, Martinu Kubišovi a Tomáši Fedorovi.
Při přesunu do historie musím každopádně uvést Jana Sýkoru, který mi především v začátcích, kdy jsem sám byl *obyčejným studentem v předmětu BI-SP1*, radil, jak udělat mnoho věcí a naučil mne základ Gitu a Nette. Mezi další studenty, kterým bych chtěl poděkovat, lze každopádně zařadit všechny ty, kteří snáší pracovat pode mnou jakožto jejich projektovým manažerem - tedy studenty BI-SP1 a BI-SP2 napříč semestry od LS 2016 až po LS 2017. Jmenovitě se jedná o: Juraje Polačka, Milana Vancla, Vojtěcha Bakaje, Jakuba Nováka, Jakuba Štercla, Ladislava Zemka, Daniela Ondřeje, Lukáše Krčála, Ondřeje Lakomého, Richarda Strnada, Jana Luxemburka, Matěje Hlaváčka, Marka Erbena, Annu Dřevíkovskou, Davida Ryzce, Filipa Machalu, Jana Potočiara, Jakuba Dvořáka,

Juraje Kožíka, Lukáše Banka a Marka Papinčáka.
Velké díky patří také Ing. Ivanu Halaškovi, za nahlašování obřího množství nedostatků a požadavků na funkcionalitu aplikace prostřednictvím veřejného bug reportu. Poděkovat bych také chtěl Ing. Michalu Valentovi a všem ostatním vyučujícím předmětu BI-DBS za snášení práce v portálu, který přes veškerou naši snahu stále obsahuje velké množství nedostatků.
Na závěr patří velké poděkování i Kristýně Miltnerové a Markétě Malcové za korekturu jazykové, gramatické a stylistické stránky této práce a mé rodině, za neutuchající podporu při mých studiích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. května 2017 .....

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

MALEC, Oldřich. *Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017. Dostupný také z WWW: ⟨`https://gitlab.fit.cvut.cz/malecold/bc-thesis`⟩.

Abstrakt

Práce se zaměřuje na efektivní řízení softwarového projektu, jehož cílem je vyvinout podpůrnou webovou aplikaci pro výuku předmětu BI-DBS na FIT ČVUT. Práce popisuje problémy s vedením vícečlenných týmů, zabývá se problematikou nasazování a udržitelnosti běhu výsledného softwaru a jeho průběžného vývoje. Cílem práce bylo navrhnout a začít používat postupy a nástroje, díky kterým se efektivně skloubí běžící systém a jeho průběžný vývoj za minimálního výskytu chyb. Jako metodika vývoje byla zvolena kombinace Extrémního programování a Feature-driven development, průběh práce je sledován v Redmine. Zdrojový kód aplikace je verzován a testován na Gitlabu. Poslední kapitola také obsahuje praktické ukázky konfigurace některých služeb potřebných pro běh samotné aplikace. Z práce lze vycházet při řízení jiných projektů obdobných rozměrů či pokračování v řízení stejného projektu.

Klíčová slova projektové řízení, metodiky vývoje, databázové systémy, řízení lidí, Redmine, Gitlab, SSL, Shibboleth, monitoring

Abstract

This thesis focuses on effective software project management, which aims to develop a supporting application for teaching BI-DBS at FIT CTU. Thesis describes issues with leading teams consisting of multiple people, deals with deployment, maintenance of deployed software and the development itself. The goal is to propose and start using procedures and tools, which will effectively combine a running system with its active development, while suppressing most of the bugs. The chosen software development methodology combines Extreme programming and Feature-driven development, the course of work is tracked in Redmine. Source code of the app is versioned and tested using Gitlab. The last chapter also includes samples of configuration for some services required to run the application. This work can be used as a base point for other projects with similar sizes or to continue in the process of managing the same project.

Keywords project management, development methodology, database systems, people management, Redmine, Gitlab, SSL, Shibboleth, monitoring

Obsah

Seznam ukázek kódu

Seznam tabulek

Seznam obrázků

Úvod

Téma *projektového řízení* je potřeba řešit u každého alespoň středně velkého projektu. Portál pro podporu výuky předmětu BI-DBS se již do takových rozměrů také dostal, zvlášť když je zároveň potřeba udržovat běh nasazené aplikace, opravovat chyby, které objevují koncoví uživatelé, a řídit další vývoj nových funkcí. Kromě řízení projektu obsahuje také práce téma *infrastruktury* vývoje softwaru, která zajišťuje efektivnější vývoj, komunikaci a plánování. Má působnost při řízení tohoto projektu a jeho infrastruktury by měla přispět k vyšší kvalitě používání zmíněné aplikace. Z výstupu by tedy měli těžit všichni studenti a vyučující předmětu BI-DBS.

Má motivace k řešení tohoto tématu vznikla ve chvíli, kdy jsem se v předmětu BI-SP2 stal vedoucím jednoho z týmů, které aplikaci vyvíjely pod vedením Jiřího Hunky. Viděl jsem, že mé programátorské schopnosti nedosahují takových úrovní, jako některých ostatních studentů, kteří například naprogramovali celou kostru nového systému, zato jsem však měl cit pro procesy v projektovém řízení, svědomitost při záznamech stráveného času nebo zájem o nastavování postupů vývoje a konfigurace nových podpůrných služeb, potřebných pro běh i vývoj aplikace.

V práci nejprve seznámím čtenáře s předmětem BI-DBS, pro který je portál vyvíjen a přiblížím podobu řízení projektu, jak probíhalo před započetím mé práce. Následně se zabývám analýzou metodik vývoje softwaru, ze kterých tvořím závěry pro jejich využití při vývoji DBS portálu. Třetí kapitola obsahuje jak teorii, tak praxi při řízení lidských zdrojů, což je jedním z mých hlavních zaměření při

práci na projektu. Čtvrtá a poslední kapitola popisuje veškeré podpůrné nástroje, ať už se jedná o konfigurace serveru nebo služby používané realizačním týmem, k jejichž nastavení nebo definici používání jsem jistou měrou přispěl. Práce by se dala zařadit k ostatním bakalářským a diplomovým pracem, které vznikly pro účely DBS projektu. Jejich autoři, chronologicky seřazeni podle termínu dokončení práce, jsou: Jan Sýkora, Martin Kubiš, Filip Glazar, Petr Pejša, Jiří Slavotínek, Tomáš Fedor, Pavel Kovář a Milan Vlasák.

Kapitola 1

Co je to DBS portál

Tato práce se zabývá především vývojem a správou portálu pro předmět BI-DBS na FIT ČVUT. Ještě než se tedy ponoříme do jednotlivých součástí vývoje tohoto portálu, chtěl bych čtenáře seznámit s předmětem samotným.

1.1 Historie výuky BI-DBS

Předmět databázové systémy je vyučován na katedře softwarového inženýrství a v současném studijním programu se s ním setkají studenti bakalářského programu Informatika ve svém druhém semestru. Výuka si klade za cíl přiblížit studentům problematiku ukládání dat, a to především pomocí relačních databází. Důraz je kladen na jazyky RA a SQL. Forma výuky předmětu pochází již z dob výuky na FEL ČVUT a používala se již déle než 10 let.

Semestrální práce Každý student si v rámci své semestrální práce vytvoří vlastní databázi, kterou naplní testovacími daty, a následně vymýšlí dotazy. Typická semestrální práce obsahuje ve své finální podobě alespoň 25 dotazů a pokrývá všechny standardní klauzule DML, jednak z jazyku *relační algebry* a jednak z SQL. DML v SQL vypadá následovně:

- `SELECT ... FROM ... WHERE ... ;`
- `INSERT INTO ... VALUES ... ;`
- `UPDATE ... SET ... WHERE ... ;`

- `DELETE FROM ... WHERE ... ;`

Semestrální práce se vypracovávala ve formátu XML, který byl následně pro přehlednější reprezentaci převeden do HTML pomocí XSLT. Zejména studenti si často na nutnost vyplňování XML šablony stěžovali.

Testy v semestru a zkouška Kromě semestrálních prací popsaných výše jsou také součástí hodnocení studentů průběžné testy v semestru a závěrečná zkouška. Testy v semestru se týkají především praktické části - tedy používání RA a SQL či modelování schémat databáze - které by student měl mít osvojené ze své semestrální práce. Závěrečná zkouška poté kromě praktických částí může obsahovat i teoretické otázky které byly probírány na přednášce.

1.2 Vznik DBS portálu

Obě výše popsané součásti výuky jsou poměrně náročné na korekturu vyučujícím. Typicky nebylo možné zajistit, aby vyučující zkontroloval každý jednotlivý dotaz, který student ve své semestrální práci vytvořil. Oprava testů, které se psaly na papír, trvala zbytečně dlouho a prakticky nebylo možné ji automatizovat. Z těchto důvodů přišel v roce 2013 Jiří Hunka - jeden z vyučujících předmětu BI-DBS - s nápadem, že se realizuje portál zaměřený na podporu výuky Databázových systémů, který bude umožňovat jak efektivnější korekturu studentských semestrálních prací, tak i rychlejší opravu testů a zkoušek.

Řešitelský tým Vývoj portálu takových rozměrů však nebylo možné financovat běžně dostupnými prostředky, kterými fakulta disponuje. Z toho důvodu bylo rozhodnuto, že bude portál vyvíjen s rámci předmětů BI-SP1 a BI-SP2. Jedná se o předměty vyučované v oboru Softwarové inženýrství, které si studenti typicky zapisují ve svém 4., respektive 5. semestru studia. Cílem předmětů je vytvořit 3-5 členné týmy, které budou pracovat na softwarovém projektu, počínaje návrhem, analýzou požadavků atp., a konče hotovým softwarem. Bylo tak vypsáno zadání na realizaci DBS portálu, do kterého se přihlásilo 13 studentů. Tento způsob získávání pracovní síly pro další vývoj portálu je využíván dodnes. Jelikož BI-SP1 a navazující BI-SP2 má celkové trvání jeden akademický

rok, jsou každý rok nabíráni noví studenti. Toto přináší jak obtíže s řízením projektu, tak příležitosti pro jeho změny. Řízení DBS projektu se věnuje kapitola 3.

Nasazení portálu Portál byl poprvé nasazen do výuky v LS 2016, kdy byl využíván pouze na cvičeních, která vedl Jiří Hunka, a ve zkouškovém období byl otestován také na jednom termínu závěrečné zkoušky. V ZS 2016 portál používali všichni studenti, kteří měli předmět BI-DBS zapsaný, protože jich v tomto semestru bylo pouze 36. Jednalo se tak o ideální testovací vzorek pro příští semestr. Následující semestr - LS 2017 - byl již portál využíván všemi cvičícími a počet studentů se pohyboval kolem 550. Detailům ohledně nasazování portálu se věnuje kapitola 4.

1.3 Historie řízení vývoje portálu

Já sám jsem se s portálem seznámil poprvé v letním semestru 2015 v rámci předmětu BI-SP1. V té době byla aplikace nasazena na testovací doméně, ale nebyla ještě používána „veřejností".
Další vývoj probíhal pomocí dvou řešitelských týmů po 4 členech, které navázaly na předchozí týmy. Projektové řízení bylo v té době - především z kapacitních důvodů - na velmi slabé úrovni: v týmech byli stanoveni vedoucí, kteří měli za práci týmu zodpovídat, ve skutečnosti však spíše pracoval každý sám za sebe. Hotová práce se poté prezentovala jednou týdně na schůzce s Jiřím Hunkou. Jeden z týmů měl navíc k dispozici ještě Jana Sýkoru, který zastával roli vedoucího týmu. Ten v té době pracoval na své bakalářské práci na téma *Podpora automatizované kontroly semestrální práce z předmětu Databázové systémy* a týmům zadával úkoly pro realizaci jeho návrhů UI a funkcionality. Následný výstup poté i osobně kontroloval.
Po dokončení předmětů BI-SP1 a BI-SP2 jsem u projektu zůstal a začal jsem se naplno věnovat jeho řízení, správě infrastruktury a opravě kritických chyb. V současnosti vedu již druhý běh studentů BI-SP1 a mezitím vznikly i čtyři další bakalářské a jedna diplomová práce související s portálem.

Kapitola 2

Metodiky vývoje softwaru

V této kapitole čtenáře seznámím s různými metodikami vývoje softwaru a zaměřím se také na možnosti jejich využívání při vývoji DBS portálu.

Proč používat metodiku vývoje softwaru Ještě než začneme s představením jednotlivých metodik, měli bychom si říci, proč bychom vlastně měli nějakou metodiku vývoje používat.
Při práci na *malém* softwaru, který píše pouze jedna osoba, je opravdu mnohdy zbytečené starat se o vývojové metodiky. Jakmile však chceme spojit práci více lidí a přitom požadujeme, aby výsledný produkt byl hotový do určitého termínu, splňoval všechny předem stanovené požadavky a byl jednoduše rozšiřovatelný i po jeho nasazení, začínáme mít potřebu zavádět určitá pravidla či předpisy, které nám pomohou všechny předpoklady naplnit. Právě v tuto chvíli vstupují do hry *metodiky vývoje softwaru*, které právě tento „balík" pravidel předepisují, a vedení projektu poté stačí zvolit nejvhodnější metodiku či jejich kombinaci pro daný projekt.

2.1 Tradiční metodiky

2.1.1 Vodopádový model

Jak zmiňuje Kadlec [12], nejedná se přímo o metodiku, ale pouze o životní cyklus. Je charakteristický tím, že všechny fáze vývoje jsou prováděny postupně

za sebou a není možné se vracet. Případné změny je možné zpracovat až v rámci údržby, která vývoj vždy vrátí do jedné z předchozích etap a zajistí průběh i všech následujících etap.
Seznam vývojových fází:

- Definice problému,
- Specifikace požadavků,
- Návrh,
- Implementace,
- Integrace a testování,
- Údržba.

Použití v DBS projektu Tento model je obecně pro dnešní vývoj softwaru nevhodný, jelikož software je typicky potřeba dodat nejprve se základní funkcionalitou a poté iterativně přidávat další funkce: přesně tak je vyvíjen i DBS portál.

2.1.2 Spirálový model

Vzniku Spirálového modelu dala za vznik především kritika Vodopádového modelu. Hlavní novinkou zde byl *iterativní přístup* a opakovaná *analýza rizik*. V tomto modelu se stále opakují fáze *Stanovení cílů*, *Analýza rizik*, *Vývoj a testování* a *Plánování*. Při každém průchodu těmito fázemi se provádí odlišná činnost než v předchozí iteraci (kromě analýzy rizik, ta je prováděna vždy stejně), například fáze vývoje v první iteraci pracuje pouze s koncepty, v příští iteraci specifikuje požadavky, v další navrhuje architekturu a teprve poté implementuje a testuje. Stále se však jedná převážně o jednosměrnou cestu, která nejen na počátku ale i v průběhu obsahuje velké množství analýz a návrhů.

Použití v DBS projektu Tato metodika sice zavádí iterace, ale ty jsou pouze čtyři základní, předem definované. Jak bylo zmíněno výše u vopádového modelu (2.1.1), DBS projekt potřebuje být vyvíjen pomocí skutečně iterativní metodiky, která bude v jednotlivých iteracích přidávat funkcionality.

Tradičních metodik existuje samozřejmě větší množství, pro potřeby této práce jich však nebudu více uvádět a přesuneme se k Agilním metodikám.

2.2 Agilní metodiky

Agilní metodiky řízení vznikly především z důvodu neustále se měnících požadavků na software a potřebnosti rychlejší reakce na požadované změny. Tradiční metodiky nebyly schopné reagovat na změny požadavků dostatečně rychle. Především když se začneme bavit o vývoji softwaru, kde se neustále mění dostupné technologie a požadavky na finální verzi, jsou tradiční metodiky zcela nevhodným modelem vývoje.

Rozdíl mezi tradiční a agilní metodikou lze popsat také jejich přístupem k *funkcionalitě*, *času* a *zdrojům* (Kadlec [12]). Zatímco u tradičních metodik je funkcionalita stanovena na začátku a musí jí být dosaženo pomocí *nějakého* času a *nějakého* množství zdrojů, u agilních metodik je to přesně naopak. Typicky je stanoven termín a dostupné zdroje, ze kterých vyplyne *nějaká* funcionalita.

Připomeňme si také *Manifesto for Agile Software Development* [1], který vznikl v roce 2001:

> „Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním. Při této práci jsme dospěli k těmto hodnotám:
>
> - *Jednotlivci a interakce* před procesy a nástroji.
> - *Fungující software* před vyčerpávající dokumentací.
> - *Spolupráce se zákazníkem* před vyjednáváním o smlouvě.
> - *Reagování na změny* před dodržováním plánu.
>
> Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.
>
> | | | |
> |---|---|---|
> | Kent Beck | Alistair Cockburn | James Grenning |
> | Mike Beedle | Ward Cunningham | Jim Highsmith |
> | Arie van Bennekum | Martin Fowler | Andrew Hunt |

Ron Jeffries	Robert C. Martin	Jeff Sutherland
Jon Kern	Steve Mellor	Dave Thomas
Brian Marick	Ken Schwaber	

Je tedy zřejmé, že Agilní metodiky více pracují s vývojáři samotnými, a umožňují i potlačení některých postupů pro dobro celku a dovolují ”zanedbat” i náležitosti, jako například dokumentaci. Následně se na některé agilní metodiky zaměříme více.

2.2.1 Extrémní programování

Na začátek je vhodné říci, že je rozdíl mezi *extrémním vývojem softwaru* a *extrémním projektovým řízením*. Zatímco projektové řízení se dá dělit na tradiční, agilní a extrémní (Wysocki, [29]), u vývoje softwaru se naopak dá říci, že XP položilo základ pro agilní metodiky a je tak jednou z nich.

XP rozvíjí standardní postupy, avšak posouvá je až do extrémů:

- Vždy budeme pracovat na co nejjednodušší verzi, která splňuje požadavky.
- Pokud se osvědčuje *revize kódu*, budeme neustále revidovat kód …
- Pokud se odvědčil *návrh*, budeme neustále navrhovat a vylepšovat specifikaci …
- Pokud se osvědčilo *testování*, budeme neustále testovat …
- Na rozdíl od tradičních metodik, které najednou dodají velký balík nových funkcionalit, bude XP dodávat i ty nejmenší funkční kousky. Toto vede ke *krátkým iteracím*.

Tyto postupy přivedené do extrémů jsou reprezentovány v *12 základních postupech XP*, se kterými se seznámíme na následující straně.
Ještě než se dostaneme k samotným postupům, potřebujeme se seznámit s pojmem *karta zadání.*

Karty zadání a karty úkolů Karty zadání vznikají během plánování návrhu softwaru. Tohoto plánování se účastní všichni vývojáři, vedení i zákazník. Jedna karta obvykle popisuje jednu funkcionalitu či jeden požadavek. Kartě náleží vlastnosti jako *datum vytvoření*, *stav*, *priorita*, *odhad vyřešení* a také samozřejmě vlastní zadání úkolu a dodatečné poznámky. Když se poté na jedné kartě pracuje, je aktualizována průběžnými poznámkami, které popisují které části již byly splněny, co je potřeba udělat atp.
Kromě karet zadání existují v XP také *karty úkolů*. Tyto vznikají nikoliv na začátku celého procesu vývoje, ale na začátku jednotlivé iterace.

12 základních postupů XP

1. *Plánovací hra*
   Probíhá jednak na začátku vývoje v rámci návrhu celé aplikace, jednak na začátku jednotlivých iterací. Jejím výstupem jsou *Karty zadání* a *Karty úkolů*, popisované v předchozím odstavci.

2. *Malé verze*
   Verze jsou dodávány neustále. I nejmenší změna funkcionality, která přináší užitek pro zákazníka, je vydána jako nová verze.

3. *Metafora*
   Celý systém je popisován jednoduchým příběhem, jak má ve výsledku fungovat. Tento příběh je sdílen jak vývojáři a managementem, tak zákazníkem a případně dalšími zainteresovanými osobami. Tato metafora u XP nahrazuje tradiční popis architektury.

4. *Jednoduchý návrh*
   XP navrhuje vždy pouze to, co je přímo požadováno. Nechce do návrhu zahrnovat žádné přidané vlastnosti, snaží se udělat minimum pro to, aby byly splněny požadavky.

5. *Testování*
   Během vývoje jsou neustále psány i testy pro nově vznikající komponenty. Nikdy nesmí existovat jednotka, která nemá vlastní unit test. Testy dokonce píší i sami zákazníci, kteří tvoří testy funkcionality.

6. *Refaktorizace*
   Refaktorizace je změna zdrojového kódu, která ovšem nemění jeho chování. Je prováděna z důvodu zvýšení čitelnosti kódu pro ostatní programátory či jeho zefektivnění. V XP by měl programátor vždy buď pouze implementovat novou funkčnost, nebo refaktorizovat stávající kód, nikdy ne obě činnosti současně.

7. *Párové programování*
   Zdrojový kód je vždy psán dvěma programátory u jednoho počítače. Zatímco jeden píše kód, druhý sleduje postup a přemýšlí nad dalšími souvislostmi: Je tato funkce napsána správně? Zapadá do celkového konceptu? Změní tato úprava chování jiného modulu? Je potřeba napsat nový test? …

8. *Společné vlastnictví*
   Zatímco některé jiné metodiky přiřazují jednotlivé třídy určitým programátorům, které za ně poté mají zodpovědnost, v XP je to právě naopak. Všichni mají zodpovědnost za celý kód a kdokoliv tak může měnit libovolnou součást systému.

9. *Nepřetržitá integrace*
   Systém je sestaven a otestován vždy, jakmile je to možné, typicky tedy ihned po dokončení nové funkcionality.

10. *Čtyřicetihodinový pracovní týden*
    XP klade důraz i na spokojenost lidských zdrojů. Snaží se tedy o nepřetěžování programátorů, jelikož při jejich únavě se razantně snižuje kvalita jejich výstupu.

11. *Zákazník na pracovišti*
    Součastí týmu by měl být i sám zákazník, neboli expert na cílovou doménu. Jeho účelem je odpovídat na dotazy, tvořit testy či určovat priority.

12. *Standardy pro psaní zdrojového kódu*
    Jelikož XP nedefinuje žádné standardní dokumenty, jsou veškeré informace o projektu obsaženy ve zdojových kódech samotných. Proto je potřeba, aby byly rozumně strukturované, čitelné a pochopitelné.

Použití v DBS projektu Některé z těchto postupů jsou jistým způsobem používány i při vývoji DBS portálu, více v sekci 2.3.

2.2.2 Scrum

Scrum je charakterizován především pojmy *Sprint* a *Scrum meeting*. První z nich má trvání většinou jeden měsíc a opakuje se několikrát. Jeho cílem je přinést do výsledného produktu nové funkcionality. Naopak Scrum meeting je *každodenní* schůzka, na které se shrnou nově dokončené úkoly a stanoví se, na čem se bude dále pracovat. Schůzku vede tzv. *Scrum master*. Typicky probíhá vestoje a netrvá déle než 30 minut. Dalo by se říci, že tyto každodenní schůzky jsou stěžejní aktivitou metodiky Scrum. Od těchto schůzek se také odvíjí velikost týmu: Scrum preferuje malé týmy o třech až šesti členech.

Použití v DBS projektu Pro potřeby DBS projektu je Scrum také nevhodný, především proto, že předpokládá každodenní schůzky.

2.2.3 Feature driven development

Jak název napovídá, FDD se soustředí především na *featury*, neboli jednotlivé vlastnosti, funkce či rysy výsledného softwaru. Na počátku je vytvořen základní model softwaru, který je dále členěn na jednotlivé vlastnosti. Vývoj probíhá zpravidla v dvoutýdenních iteracích, jejichž výsledkem by vždy měl být fungující software, který je možné představit zákazníkovi.

Klíčový pojem *feature (vlastnost)* lze popsat jako samostatnou část systému, která splňuje následující parametry:

- *Měritelnost*: vlastnost je jasně uchopitelná komponenta systému, kterou lze porovnat s požadavkem zákazníka.
- *Srozumitelnost*: programátor musí rozumět tomu, jak má vlastnost fungovat a co je jejím cílem.
- *Realizovatelnost*: programátor musí vědět, zda je schopen vlastnost realizovat v odpovídajícím časovém úseku (typicky iterace 2 týdny). V opačném případě je většinou potřeba vlastnost rozdělit na několik menších a na těch pracovat samostatně.

V tuto chvíli by se mohlo zdát, že FDD vůbec nedisponuje *návrhem a modelováním*. Ve skutečnosti je tomu právě naopak, společně s FDD je často zmiňován ještě *model-driven approach*. Jelikož vlastnosti jsou klíčovou stavební jednotkou FDD, je potřeba je kvalitně navrhnout a namodelovat.

Posloupnost kroků při vývoji pomocí FDD zachycuje následující seznam:

1. vytvoření celkového (globálního) modelu,
2. vypracování podrobného seznamu vlastností,
3. plánování podle vlastností,
4. návrh podle vlastností,
5. implementace.

Dvě poslední se poté stále opakují, dokud existují další vlastnosti, které je potřeba pro softwaru přidávat.

Vzhledem k vývoji DBS portálu zde stojí za zmínku především fáze návrhu a implementace.

Návrh Vlastnosti se vždy týkají různých tříd, za které má zpravidla zodpovědnout určitá osoba, či určitý programátor, který dané třídě nejlépe rozumí. Vývojáři jsou tedy rozděleni do týmů dle potřeby a sestavení týmů se tak pro každou iteraci obměňuje.
Tento nově sestavený tým má poté za úkol připravit návrh implementace vlastnosti, na jehož základě je poté naprogramována.

Implementace Týmy sestavené v předchozí fázi pracují na vlastní funkcionalitě nové vlastnosti. Společně s programovým kódem jsou tvořeny i testy, a to především *unit testy*.
Jakmile je vlastnost dokončena, je vložena do sdíleného prostoru, u FDD nazývaného *class repository*.
Hlavní programátor poté dokončené vlastnosti integruje do předchozí verze hlavní aplikace.

Typicky na projektu pracuje několik týmů a hlavní programátor pouze volí nové vlastnosti k implementaci a integruje hotové vlastnosti do výsledné aplikace.

Použití v DBS projektu Výše popsaný průběh práce hrubě odpovídá i organizaci vývoje DBS projektu, které je popsáno v sekci 2.3.

2.2.4 Test driven development

Součástí každého vývojového procesu je bezesporu i testování výstupu. TDD této skutečnosti využívá a posouvá tvorbu testů ještě před samotný zdrojový kód aplikace. Při přidávání funkcionality do systému je tedy nejprve napsán test, který bude funkcionalitu testovat. Požadovaná funkcionalita je poté naprogramována přesně tak, aby procházela testy. Testy samozřejmě stále přibývají a tak je zapotřebí, aby nová funkcionalita prošla nejen novým testem napsaným přímo pro ni, ale také všemi ostatními. Tento přístup vyžaduje jisté odhodlání programátora, jelikož vždy na první pohled vypadá jednodušeji pouze *dopsat tu jednu řádku*. Při korektním postupu pomocí TDD je však i pro nejmenší změnu kódu vždy potřeba napsat test.

Použití v DBS projektu Jelikož TDD se svým konceptem *test first, program later* vzdaluje od klasických vývojových metodik, není vhodné pro vývoj DBS portálu, kde se každý rok mění složení programátorů. DBS projekt samozřejmě využívá testy, viz kapitola 4.2.2.1, testuje se však vždy funkcionalita, která již existuje. Testy tedy především hlídají, zda změny v kódu nepoškodily jinou funkcionalitu, a testy pro nové funkce jsou dopsány až poté, co je nová funkce dokončena.

Na předchozích stránkách jsme si představili *některé* vývojové metodiky. Především těch agilních však existuje nepřeberné množství, a jak říká například Kadlec [12] nebo Rasmusson [18], každý projekt je jedinečný a určitě není nejlepší možností striktně dodržovat jednu z vývojivých metodik. Klíčem je spíše skloubení dostupných metodik k vytvoření vlastního, pro daný projekt nejefektivnějšího přístupu k vývoji. Vedoucí týmu či projektový manažer by neměl *slepě* následovat recept předložený například Extrémním programováním a zamítat jakékoliv jeho úpravy. Z toho důvodu i pro DBS projekt vznikla vlastní

metodika, které se bude věnovat jak následující sekce, tak celá následující kapitola.

2.3 Zvolené řešení pro DBS projekt

Jak bylo zmíněno v sekcích o *Extrémní programování* a *Feature driven development*, DBS projekt je vyvíjem pomocí metodiky na pomezí XP a FDD.

Na následujících řádcích budu pro potřeby této kapitoly popisovat i nástin *infrastruktury*, která se v DBS projektu využívá. Tomuto tématu je však věnována také vlastní kapitola 4.

2.3.1 Inspirace v XP

Jedním z požadavků vývoje pomocí *Extrémního programování* je předpoklad, že budou dodržovány všechny jeho náležitosti. Jelikož jsme však na akademické půdě a vývojáři DBS projektu jsou především studenti předmětů BI-SP1, případně BI-SP2 (jak již bylo zmíněno v kapitole Co je to DBS portál), není možné abychom využívali například *párové programování* či *čtyřicetihodinový pracovní týden*, které XP „předepisuje“. Už minimálně z tohoto důvodu nemůže být XP použito ve svém plném obsahu. Důvodů je více, nyní se však pojďme zaměřit naopak na to, co vývoj DBS portálu z Extrémního programování *využívá*.

Revize kódu XP mluví o neustálé revizi kódu. Jelikož studenti, kteří portál programují, jsou často nezkušení, je opravdu důležité jejich kód často revidovat, ideálně ihned po jejich *commitu* do *Gitu*. Revize provádí vedoucí týmu a také vedení projektu, zejména Pavel Kovář.

Krátké iterace, malé verze Jelikož DBS portál je *webová aplikace*, je velmi jednoduché vydat novou verzi. Toho je také využíváno a nové veřejné verze jsou publikovány téměř každý týden. Kromě toho jsou publikovány i testovací verze, které je možné používat paralelně s produkční verzí, a zde jsou změny nasazovány dle potřeby - někdy i vícekrát denně.

Karty zadání XP popisuje *Karty úkolů* (2.2.1), které víceméně odpovídají úkolům, které jsou v DBS projektu zadávány v *Redmine*. Odpovídají zejména jejich pole *datum*, *stav*, *priorita* či *průběžné poznámky plnění*.

Obecně se ale dá říct, že takovouto funkci *issue trackeru* musí určitý systém plnit prakticky v *jakékoliv* vývojové metodice, proto mluvit o *inspiraci z XP* u tohoto bodu nemá takový význam.

Testování a nepřetržitá integrace V XP je „předepsáno“ neustále dopisovat testy pro nově vznikající komponenty. To probíhá i při vývoji DBS portálu, což je podpořeno i *neustálou integrací*. Jakmile je dopsána nová funkčnost, tak při jejím *push*i do *Gitlab*u se spustí automatické testy.

Zákazník na pracovišti Tato součást není při vývoji DBS portálu dodržována zcela, již z důvodu, že prakticky neexistuje sdílené pracoviště. Jednotliví členové pracují typicky odděleně, kdekoliv. Zákazník ovšem *je* součástí týmu. V případě DBS projektu se jedná o Jiřího Hunku, vedoucího této práce, který jakožto vyučující předmětu BI-DBS je zároveň jedním z hlavních uživatelů výsledné aplikace. Určuje tedy priority, testuje nové verze a hlásí chyby či požadavky.

Vývoj DBS portálu tedy přebírá z Extrémního programování zhruba *polovinu* jeho základních postupů (2.2.1). Dále se pojďme zaměřit na to, co je přebráno z Feature driven development.

2.3.2 Inspirace v FDD

I z Feature driven development jsou přebrány některé metodiky vývoje pro DBS projekt. Především rozdělení na jednotlivé *featury* a role *hlavního programátora*.

Featury Ve FDD je vývoj hnán kupředu především seznamem vlastností, které má finální podoba mít. Toho využívá i DBS projekt, u kterého takový seznam existuje. Ten se z jedné strany stále rozšiřuje novými požadavky, zatímco na

straně druhé je zpracováván programátory. Jednotlivé vlastnosti jsou měřitelné, v případě nesrozumitelnosti jsou programátorovi upřesněny a jsou členěny tak, aby byly realizovatelné do následující iterace (1-2 týdny).

Hlavní programátor Ve FDD dominuje role *hlavního programátora*, který určuje priority jednotlivých vlastností připravených k implementaci a integruje jejich hotové řešení do výsledné aplikace. Tuto roli zastávám v DBS projektu já, jakožto *projektový manažer*. Mým úkolem je mimo jiné zpracovávat požadavky a nahlášené chyby do jednotlivých zadání. Této činnosti se blíže věnují mnohé z následujících sekcí, jak z kapitoly Řízení lidských zdrojů, tak Infrastruktura DBS projektu.

Kapitola 3

Řízení lidských zdrojů

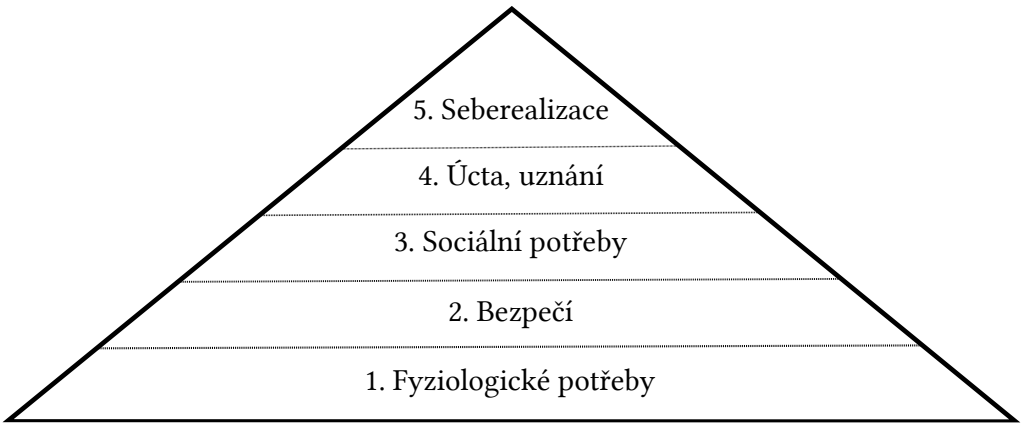
Řízení lidských zdrojů je jednou z nejobtížnějších součástí projektového řízení, především proto, že se prolíná s dalšími odvětvími jako například *psychologií* či *sociologií*. Jelikož lidé jsou hnací silou projektu, mohou různé metody či samotná kvalita (ba dokonce absence!) řízení lidských zdrojů odrážet výsledný úspěch či neúspěch projektu. I v DBS projektu tedy vyvstala potřeba tuto pozici zaujmout a chopil jsem se jí já, jak již bylo popsáno v kapitole 1.3.

3.0.3 Role psychologie v řízení lidí

Ještě než se ponoříme do samotných procesů popisujících řízení lidských zdrojů, věnujme se chvíli právě oněm dalším odvětvím, které se s řízením lidí prolínají. Jak již bylo řečeno, jedná se především o *psychologii*.

Lidé nejsou stroje, kterým bychom poskytli určitý vstup a podle jeho množství mohli očekávat výstup. Je potřeba se zamýšlet nad dalšími okolnostmi, proč pro nás daný člověk pracuje, zda je spokojený, kdy odvádí nejlepší práci atp. Za veškerou lidskou činností stojí *nějaká* motivace, cílem projektového manažera je proto najít, co motivuje právě jemu svěřené pracovníky.

Maslowova pyramida potřeb Ve standardních projektech, kde jsou vývojáři odměňováni finančně, se dá plat považovat za splnění základních fyziologických potřeb, protože je za něj možné nakoupit jídlo či pití. V DBS projektu jsou namísto finančně studenti hodnoceni známkou z předmětu, který je pro jejich



Obrázek 3.1: Maslowova pyramida potřeb

studijní obor povinný. Jelikož však *vystudování vysoké školy* obecně je spíše potřebou *úcty, uznání* či *seberealizace*, jedná se o vyšší patra pyramidy. Maslow [16] tvrdí, že není možné uspokojovat vyšší patra pyramidy, pokud nejsou zajištěny ty nižší. Z toho důvodu může být obtížné motivovat k práci ty studenty, kteří nejprve potřebují vyřešit své zabezpečení (1. a 2. úroveň pyramidy), takoví studenti budou logicky dávat přednost před prací na projektu opravdové práci, za kterou dostanou peníze. Jakmile však má student zajištěny tyto „nižší potřeby", dostávají se do popředí právě ty, které může DBS projekt nabídnout. Tím se otevírají skvělé možnosti pro rozvoj jak samotných studentů, tak výsledného portálu. Jako příklad uvedu Pavla Kováře, který v tuto chvíli pracuje na bakalářské práci na téma *Automatizované testování webového portálu dbs.fit.cvut.cz*. K projektu se dostal stejnou cestou jako já, ale o rok později. Po dokončení předmětu BI-SP2 se rozhodl u projektu zůstat a pokračovat v jeho vývoji. Jelikož již během vývoje v rámci SP týmů sám od sebe přicházel s inovacemi a kvalitními návrhy na zlepšení, jedná se dle mého názoru o jasný příklad naplňování nejvyšší potřeby, a to *seberealizace*.

Právě seberealizace je jedna z potřeb, na kterou se snažím při řízení DBS projektu klást důraz. Studenti jsou hodnoceni jednak za práci, která je jim zadána, zároveň ale dostávají příležitosti rozvíjet vlastní nápady a přínosy projektu a jsou za tyto přínosy také kladně hodnoceni. Více o hodnocení je popisováno v sekci 3.3.3.3.

3.0.4 Procesy řízení lidských zdrojů

Jak popisuje Schwalbe [20] ve své knize *Řízení projektů v IT: kompletní průvodce*, při řízení lidských zdrojů se rozeznávají následující procesy:

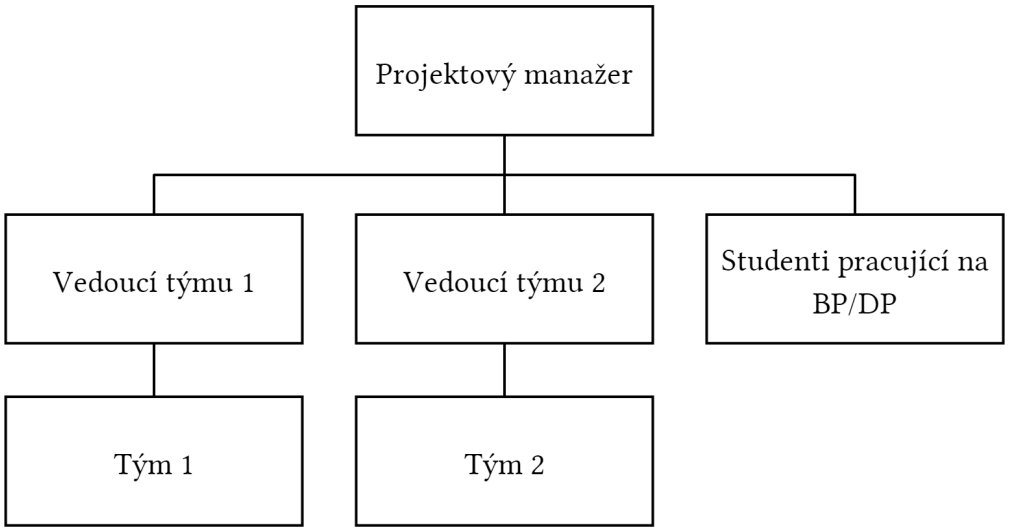
- Vytvoření plánu lidských zdrojů.
- Zajištění projektového týmu.
- Rozvoj projektového týmu.
- Řízení projektového týmu.

V následujících sekcích si jednotlivé procesy rozebereme podrobněji a zaměříme se i na jejich roli v DBS projektu.

3.1 Vytvoření plánu lidských zdrojů

Cílem tohoto procesu je vytvořit plán, jaké *typy* lidí potřebujeme do projektu zapojit. Může se jednat například o *profesionálního C++ programátora* či *webdesignéra s citem pro UI a UX* atp. Typicky je vytvořena stromová struktura, kterou zastřešuje projektový manažer a která se dále dělí například na jednotlivé části - programátoři, testeři, grafici, z nichž například programátoři mohou být dále rozděleni do týmů starajících se o jednotlivé komponenty systému, kde každý tým má ještě svého vedoucího. Takovému plánu se říká *OBS* (Organization Breakdown Structure). Tato struktura by měla být vytvořena na začátku projektu a podle ni následně zvoleni vedoucí daných oblastí.

Použití v DBS projektu V DBS projektu popisuje organizační členění obrázek 3.2. Je však nutné přiznat, že tato struktura v případě našeho projektu vznikla spíše *na základě dostupných lidí*, než naopak. Dělení na týmy fungovalo již od začátku, vedoucí týmu však často nezvládal plnit práci, která po něm byla požadována. Teprve po dalším zastřešení jejich zodpovědnosti *projektovým manažerem* se stala pozice vedoucího týmu důležitější, a to především na úrovni komunikace manažera s týmem. Veškeré úkoly, které manažer do týmu zadává, prochází přes vedoucího týmu, který práci dále rozděluje, a po jejím dokončení opět kontroluje. Tento proces je znázorněn i na activity diagramu v příloze C.1.



Obrázek 3.2: Organization Breakdown Structure používaná v DBS projektu

3.2 Zajištění projektového týmu

Tento proces se primárně stará o *příjem nových pracovníků*, *řízení jejich vytížení* a *vyrovnání*. První ze zmíněných je především v IT hodně diskutované téma, kdy najít člověka se správnou kvalifikací je často nadlidský úkol. Firmy se tak přetahují, kdo nabídne zajímavější pracovní prostředí, lepší benefity či *evergreen*: vyšší plat.
Když už je *ten správný* zaměstnanec přijat, je potřeba řídit jeho *vytížení* a případně jeho vznik *vyrovnávat*. Jedná se o metodiky, při kterých je sledováno množství práce zadané zaměstnanci, a v případě, že je po něm požadováno více, než je schopen zvládnout, je buďto přehodnocen harmonogram a úkoly s nižší prioritou jsou odloženy na později, nebo je k úkolu přiřazen další pracovník.
Plánování přiřazení práce je důležité především z důvodu předcházení výkyvů v množství přidělené práce, což vede k většímu pohodlí zaměstnance a také snadnějšímu řízení.

3.2.1 Použití v DBS projektu

Jelikož u DBS projektu se pohybujeme na akademické půdě a nikoliv ve světě byznysu, je zde přístup k náboru nových pracovníků trochu odlišný. Noví pracovníci - studenti - jsou sháněni jednak nárazově jednou ročně - při startu

nového letního semestru - jednak kontinuálně - jakožto zájemci o bakalářskou či diplomovou práci.
První ze zmíněného náboru vyžaduje vypsání tématu na stránku předmětu BI-SP1, kde se snažíme studenty nalákat právě k našemu projektu. Především v tomto roce (2017) se zvýšila zajímavost ostatních nabízených projektů (různé mobilní aplikace atp.), a tak se i zde začal projevovat podobný efekt jako u reálných firem: *lov pracovníků*. Zároveň jsme však po nových studentech chtěli, aby měli buď zkušenost s použitými technologiemi, nebo měli předpoklad k rychlému naučení těchto - pro někoho nových - technologií. Z toho důvodu jsme připravili *vstupní kvíz* (viz níže podsekce Vstupní kvíz), který měl původně sloužit jako nutný požadavek pro vstup do týmů, z důvodu nízkého zájmu však byla poté jeho role přehodnocena. Kvíz byl studentům zadán až po jejich vstupu do projektu a na základě jejich výsledků byli rozděleni do podobně schopných týmů a také byli určeni vedoucí týmů - ti, kteří vykazovali nejvyšší aktivitu a zároveň nejlépe zpracovali kvíz.

Vstupní kvíz

Jedná se o malou aplikaci v Nette, která obsahuje 3 jednoduché úkoly s možností pokročilého rozšíření řešení. Ukázky kvízu jsou dostupné v příloze D.
Studenti si museli na svém počítači nainstalovat a nastavit webserver a stáhnout zdrojové kódy ze školního Gitlabu. Databáze se vytvořila sama při prvním spuštění - pro potřeby kvízu stačil databázový stroj SQLite. Následně byl student přihlášen do *kostry aplikace*, která obsahovala 3 úkoly:

- Zobrazování dat z databáze v tabulce a práci se zobrazováním data a času v čitelné formě.
- Jednoduché CMS: Přidávání, úprava a mazání příspěvků.
- Práce s komponentou Grido: Přidání sloupce tabulky, nastavení řazení, přidání akce.

Všechny tyto úkoly měly navíc *volitelé* rozšíření, které se většinou zaměřovalo na korektní návrh architektury MVP.

Druhým náborem, který probíhá neustále, je nábor studentů, kteří budou na projektu pracovat v rámci své bakalářské či diplomové práci. Takoví studenti zpravidla přichází sami s požadavkem o vymyšlení tématu BP/DP či zůstávají u projektu po skončení jejich působení v předmětu BI-SP2. Tito studenti vyžadují zpravidla méně „péče“ projektového manažera, protože již vědí, co chtějí realizovat, a často jsou v daném tématu i zběhlejší než já. Běžně také svou práci spíše než se mnou konzultují s Jiřím Hunkou - který vede nejen realizační týmy SP1/SP2, ale také právě bakalářské a diplomové práce.

Kromě nabírání nových lidí patří do tohoto procesu také řízení jejich vytížení a vyvažování přidělené práce. Jak vyplývá z obrázku 3.2, já jakožto projektový manažer se nestarám přímo o zátěže jednotlivých členů týmu, ale práci přiděluji pouze týmům jako celkům. Rozdělení práce v týmu je plně v kompentenci týmových vedoucích, což dále rozvíjí jejich schopnosti vést tým. Pro správu přidělené práce využíváme *Redmine*, jehož použití se více věnuji v kapitole 4.1.1.

3.3 Rozvoj projektového týmu

V předchozí sekci jsme zajistili vznik týmu, tím však zdaleka práce nekončí. Hned po vytvoření týmu se spouští jednotlivé fáze vývoje týmu, jak je definoval *Tuckman* [28] v roce 1965:

1. *Forming (formování)*
   Při samotném vzniku nebo při příchodu nového člena dochází k představování, určování rolí v týmu atp. Většinou není během této fáze vyprodukována žádná práce, ale jedná se o nedílnou součást vývoje týmu, která nemůže být opomenuta.

2. *Storming (krize)*
   Jakmile jsou stanoveny role v týmu a začíná samotná práce, začne docházet k nedorozuměním, ujasňování rolí, odpovědností a kladených požadavků na jednotlivé členy. Toto zákonitě přináší jisté rozepře, které musí tým řešit.

3. *Norming (stabilizace)*
   Pokud jsou rozepře z předchozí fáze vývoje týmu úspěšně vyřešeny, dostává se tým do fáze stabilizace, ve které dochází k vyjasnění a vylepšení komunikace týmu a tím i jeho výkonnosti.

4. *Performing (optimální výkon)*
   Po průchodu všemi předchozími fázemi se tým dostává na vrchol své výkonnosti, je schopen zvládnout mnohem složitější úkoly a má stabilizované vnitřní vztahy.

Kromě těchto čtyř základních fází je vhodné zmínit ještě dvě doplňující, a to *rozpad týmu*, který může nastat po neúspěšném *stormingu* a který buďto vede k ukončení celého projektu nebo přeskupení členů týmu - pokud to povaha projektu dovoluje. Poté nastává opět první fáze *forming* s novým členěním týmu. Druhou doplňující fází je *rozpuštění týmu*, která nastává po skončení projektu.

3.3.1 Školení

Další součástí rozvoje týmu je školení pracovníků v nových dovednostech. Firmy často nabízejí svým zaměstnancům možnost účastnit se různých kurzů, ať už s lektorem, nebo ve formě e-learningu. Tyto kurzy by měly být vhodně plánovány s ohledem na nadcházející práci, která bude zaměstnancům zadávána. V opačném případě se budou školení míjet účinkem a ponesou tak pouze náklady.

3.3.2 Systém odměňování a oceňování

Důležitou součástí je také odměňování lidí. Typicky jsou speciálně odměňováni ti pracovníci, kteří řeší úkoly nad rámec zadání nebo ve svém volném čase pomáhají kolegům s jejich prací. Odměny mohou být finanční nebo mít například podobu delší dovolené. Pro manažera je důležité správně rozlišovat práci vykonanou navíc a kladně ji hodnotit, tak aby byli pracovníci dostatečně motivováni. Nesmí však docházet k jevu, kdy pracovníci dělají práci navíc *pouze* pro získání zmíněných bonusů.

3.3.3 Použití v DBS projektu

Proces rozvoje projektového týmu má několik částí, i jejich použití v DBS projektu si tedy pro lepší přehlednost rozčleníme.

3.3.3.1 Rozvoj projektového týmu

Čtyři fáze vývoje týmu popisované výše probíhají i v týmech, které programují DBS portál. Zvláště v letním semestru 2017, kdy vznikala tato práce, byly rozdíly mezi týmy velmi dobře sledovatelné, a to díky jejich rozdílenému složení:

- *Team 1* vznikl spojením 5 studentů, kteří se do té doby vůbec neznali. U tohoto týmu byla patrná především první fáze *formingu*, kde byli jednotliví členové rozpačití, nevěděli, co si mohou dovolit, a vedoucí byl ve své pozici nejistý. V průběhu následujících týdnů se však stav výrazně zlepšil a tým je nyní schopen provádět i komplexnější úlohy a rozvrhnout si práci. Předpokladem je, že do následujícího semestru, kdy budou studenti na projektu pracovat v rámci BI-SP2, dosáhne tým fáze *performingu* a jeho výstup bude na vysoké úrovni.

- *Team 2* na rozdíl od prvního týmu vznikl ze 4 studentů, kteří se již před začátkem projektu znali. Měli jasně zvoleného vedoucího, který byl zároveň nejzkušenějším z nich. U tohoto týmu tak vůbec neproběhly fáze *forming*, *storming* ani *norming* a nebylo tak možné se dostat k nejefektivnější poslední fázi: *performing*. V době psaní této práce bylo nejvíce práce ve druhém týmu vykazováno právě vedoucím, který sice opravdu zkušený byl a jeho výstup byl kvalitní, nezvládl ale dovést svůj tým k optimálnímu výkonu. To znamenalo, že ostatní členové často záviseli právě na svém vedoucím, který „zařídil vše“. Já, jakožto manažer projektu, se snažím do tohoto rozložení zasáhnout, aby se i team 2 dostal do své nejefektivnější fáze, a doufám, že do příštího semestru se mi podaří tohoto cíle dosáhnout.

Když tedy porovnáme tyto dva týmy, dostáváme se k jednoduchému závěru: *Team 1*, přestože ze začátku byl v rozpacích a jeho výstupy byly slabší než týmu druhého se postupem času díky průchodu jednotlivými fázemi dostal do lepší pozice. Naopak *Team 2* začal díky svému zkušenému vedoucímu vykazovat hned od začátku kvalitní práci, dá se však předpokládat, že pokud nedojde

k zásadní změně v rozložení týmu, nebude se již tato efektivita zvyšovat, naopak se dá očekávat i její pokles.

V době dokončování této práce se navíc vedoucí Teamu 2 dozvěděl, že kvůli neúspěšné zkoušce odložené do LS nemůže dále pokračovat ve studiu. Podle posledních informací by měl však pokračovat s vedením svého týmu minimálně do konce tohoto semestru, na semestr příští si však budou muset zbývající členové zvolit mezi sebou nového vedoucího.

3.3.3.2 Školení

Na začátku letního semestru, kdy se k projektu dostávají noví studenti, jsou kromě úvodu do projektu pořádána i dodatečná školení, především týkající se použité technologe - Nette Frameworku, ale také například zaměřená na Git či různá „how-to“, která jsou užitečná při vývoji DBS portálu.
Tato školení jsou většinou pořádána studenty, kteří na projektu pracovali v minulosti v rámci předmětů BI-SP1/2 a u projektu zůstali v rámci svých BP či DP.
Jelikož letošní rok bylo školení pořádáno hned ve druhém týdnu semestru, studenti byli ještě rozpačití a ostýchali se pokládat dotazy ohledně věcí, které by je zajímaly. Z toho důvodu jsem se školení také zúčastnil, přestože v roli lektorů již byli Pavel Kovář a Milan Vlasák, kteří také pracovali na BP týkajících se DBS portálu. Já jsem tedy měl příležitost zúčastnit se jako posluchač, čímž jsem se snažil přiblížit „druhé straně bariéry“ - novým studentům - a schválně jsem kladl i *základní dotazy*, které jsem považoval za vhodné právě pro nové týmy. Mým cílem tedy bylo jednak navodit přátelskou atmosféru - ve které je možné zeptat se na cokoliv - jednak „rozbít“ *groupthink* a *konformitu s většinou*, jak je definoval Procházka [17] ve své lekci *Skupinová dynamika, Týmová spolupráce* v kurzu *Manažerské psychologie*, které jsem se zúčastnil v zimním semestru 2015.

Groupthink, konformita s většinou První zmíněný, v překladu *skupinové myšlení*, je jev, který nastává při neřízené debatě několika, i velmi inteligentních, lidí. Při groupthinku může taková skupina dojít i ke zcela špatným rozhodnutím, neboť nikdo nechce vystupovat jako „černá ovce“ a rozbít skupinový souhlas,

všichni tak *v tichosti* odsouhlasí první návrh, který padl. Často se stává problémem, že osoba ve vůdčí pozici vyjádří svůj názor a poté se zeptá na názor podřízených. Ti ve valné většině budou pouze souhlasit, jelikož je jim nepříjemné vystoupit proti názoru jejich nadřízeného. Kdyby však byl nejprve dán prostor pro vyjádření podřízených a poté prezentována představa vedení, existuje mnohem větší šance, že budou probrány všechny aspekty řešeného problému. Pro prevenci groupthinku je vhodné do každé debaty zařadit alespoň jednoho *rozvraceče*, jehož úkolem bude zpochybňovat veškeré názory a rozvířit tak mnohem produktivnější diskuzi.
Konformita s většinou je podobný jev jako groupthink a nastává ve chvíli, kdy v určité skupině lidí většina zastává nějaký názor. Člověk, který si myslí něco jiného, se pak velmi často *přizpůsobí* většině, z důvodu „aby nebyl divný“, aby si ostatní „něco nemysleli“ atp. Průzkumy [17] ukázaly, že stačí 3-4 další osoby, které mohou tvrdit i naprostý nesmysl, a nezávislá další osoba se i přes vnitřní nevoli velmi často *přizpůsobí většině*. Při kladení sady otázek, kde 4 figuranti schválně odpovídali špatně, se 32% osob přizpůsobilo jejich *špatné* odpovědi pokaždé, 74% se přizpůsobilo alespoň u jedné otázky. Stačí však jedna další osoba, která bude odpovídat *jinak než většina*, a konformita rázem klesá na pouhých 6% - přesně to bylo i mým cílem u školení nových studentů: aby se nebáli klást otázky, přestože většina studentů jen *kývala*.

3.3.3.3 Systém odměňování a oceňování

Odměnou, kterou student získává za práci na projektu, je především známka do předmětu BI-SP1/(BI-SP2) a také zápočet do předmětu BI-SI1. Tato známka se odvíjí od výsledků, kterých student při vývoji projektu dosahoval.
Hodnocení probíhá na dvou úrovních:

1. Hodnocení týmu jako celku, udělované projektovým manažerem.
2. Hodnocení jednotlivce v rámci týmu, udělované vedoucím týmu.

První ze zmíněného uděluje manažer projektu a skládá se z následujících položek:

- *Vypracování přiděleného úkolu*
  Každý úkol, který je do týmů přiřazen, má nastavené body, které za něj lze získat. Výši bodů určuje manažer projektu (= zadavatel úkolu) a v případě

odhalení vyšší obtížnosti v průběhu řešení úkolu mohou být body ještě dodatečně upraveny, nikdy však sníženy. Proces vypracování úkolu popisuje Activity diagram průběhu práce v Redmine dostupný v příloze C.1. Výsledné hodnocení úkolů je definováno v tabulce 3.1.

Míra splnění úkolu	Bodový zisk
Úkol splněn nad rámec zadání	125% bodů
Úkol plně splněn, funguje	100% bodů
Úkol splněn, má malé nedostatky (ale lze nasadit)	75% bodů
Úkol splněn pouze z části, lze nasadit	50% bodů
Úkol řešen, je vidět náznak (správného) řešení, ale výsledek nefunguje (nelze nasadit)	25% bodů
Úkol není na Gitu, není nastaven jako vyřešený v Redmine nebo řešení absolutně nelze použít	0 bodů

Tabulka 3.1: Bodování týmů dle míry splnění úkolu

- *Nahlášení nového úkolu*
  Studenti jsou vedeni k vlastní iniciativě. Když přijdou s novým úkolem, který vede ke zlepšení výsledné aplikace nebo opravuje závažnou, dosud nenahlášenou chybu, je jejich tým kladně ohodnocen.

- *Práce vedoucího týmu*
  Jelikož vedoucí týmu má za úkol nejen sám pracovat na úkolech, ale i rozdělovat práci, kontrolovat výstupy svého týmu a řídit komunikaci, je každý týden hodnocen i za tuto práci. Manažer tedy sleduje aktivitu vedoucího především na Slacku a Redmine a jednou týdně vyhodnotí kvalitu jeho práce. Získané body se také počítají jako zisk celého týmu.

Výše popsané možnosti získávání bodů přináší body vždy celému týmu a jednotlivým členům jsou rovnoměrně rozdistribuovány. Vedoucí týmů však mají ještě možnost hodnotit svůj tým interně, a to jak uznají za vhodné. Jedinou podmínkou je, že součet bodů udělených v týmu se musí vždy rovnat nule, pokud má tedy některý člen týmu získat plusové body, musí být body odebrány jinému členovi. Manažer projektu se snaží do tohoto interního hodnocení týmů nezasahovat, ale kontroluje například, zda není některý člen cíleně utlačován.

Tabulka hodnocení týmů v letním semestru 2017 je dostupná v příloze E.

3.4 Řízení projektového týmu

Zbývá poslední proces řízení lidských zdrojů, a to *řízení projektového týmu*. To velmi často probíhá pomocí tzv. *soft skills* neboli měkkých dovedností, jelikož je zde potřeba sledovat rozhovory a mezilidské vztahy, hodnotit výkonnost projektu, řešit konflitky atp.
Mezi hlavní problémy, ke kterým může docházet, lze zařadit:

- nedostatek důvěry,
- strach z konfliktů,
- nedostatek zapojení,
- vyhýbání se zodpovědnosti,
- lhostejnost k výsledkům.

Cílem manažera by mělo být snažit se tyto problémy řešit, či ještě lépe, zařídit, aby k nim vůbec nedocházelo.

3.4.1 Použití v DBS projektu

Při řízení DBS projektu se snažím předcházet výše zmíněným problémům pomocí následujících technik:

- Studenti jsou vedeni k vyjadřování vlastních názorů, snažíme se být otevřeni jakýmkoliv návrhům včetně kritiky vedení. Ze zkušeností vyplývá, že je toto pro studenty často jednodušší vyjádřit při osobní konzultaci, nikoliv při schůzce, kdy poslouchají i všichni ostatní členové. Po skončení úvodní „společné“ části schůzky tedy začínám obcházet jednotlivé studenty a ptám se na jejich problémy, názory atp.
- Hodnocení výsledků je transparentní, snažím se o jeho smysluplnost a srozumitelnost. Každý vidí, kolik za svůj úkol získal bodů, a také, kolik bodů získávají ostatní členové. To by mělo napomáhat snížit lhostejnost k *bodovým* výsledkům, stále však přetrvává problém lhostejnosti k *softwarovým*

výsledkům. Stává se, že někdy je úkol vyřešen pouze „tak, aby splňoval zadání“ a výsledné funkčnosti poté schází jednoduchost při používání cílovým uživatelem. Tomuto problému se snažím předcházet zvýšením bodového hodnocení za velmi kvalitní řešení úkolu, hlavním problémem však je, že programátoři zároveň nejsou uživatelé systému. Pokud by tomu tak bylo, kvalita UI a UX by šla výrazně nahoru, protože by samotné autory aplikace *rozčilovalo* její používání. Tento cíl však není efektivně realizovatelný, protože předmět, na který systém cílí, mají již tito studenti absolvovaný a pro roli učitele zase nemají dostatečnou kompetenci. Často tak chyby v použitelnosti nahlašují až koncoví uživatelé, a to především Jiří Hunka.

Softwarová podpora řízení lidských zdrojů Pro řízení projektu používáme mnohé podpůrné programy a aplikace. Této problematice se více věnuje samostatná kapitola 4.

Kapitola 4

Infrastruktura DBS projektu

Z předchozích kapitol již víme, jak efektivně vyvíjet software a jak řídit lidské zdroje. V poslední kapitole své práce bych se chtěl věnovat infrastruktuře - tedy softwaru, procesům a nástrojům, které při vývoji DBS portálu využíváme.
V této kapitole často zmiňuji práci *manažera projektu*, čímž mám na mysli svou vlastní práci.

4.1 Issue tracker

Základním stavebním kamenem jakéhokoliv projektu, je *issue tracker*, neboli „systém sledování úkolů“. Do něj jsou zadávány úkoly, které je potřeba realizovat. Tyto úkoly se poté přiřazují lidem, kteří je budou řešit, a všechny změny úkolu jsou vždy jednoduše dohledatelné a měřitelné. Do těchto systémů je také často možné zaznamenávat čas strávený řešením daného úkolu, nastavovat termíny, hierarchie úkolů a řídit přístup různých rolí k určitým funkcionalitám.
Při vývoji DBS portálu využíváme následující issue trackery:

- *Redmine:* interní issue tracker využívaný vývojovým týmem a manažerem. Přístup k němu mají pouze role definované v předchozí kapitole v organizačním členění vývoje projektu (Obrázek 3.2). Slouží pro vývoj, rozdělení práce i záznam času.
- *Gitlab:* veřejný issue tracker dostupný všem studentům i vyučujícím na FIT ČVUT, který slouží jako *bug report* neboli „hlášení chyb“. Chyby zde

nahlašují typicky koncoví uživatelé portálu a z vývojového týmu na ně reaguje pouze manažer projektu, který poté potřebné úkoly zadává na interní issue tracker: Redmine.

4.1.1 Redmine

„Redmine je flexibilní webová aplikace pro správu řízení projektu. Díky použití frameworku Ruby on Rails je multiplatformní a multidatabázová“[15].

Redmine je při vývoji DBS portálu využíván především pro:

- Shromažďování úkolů k řešení.
- Sledování práce na přiřazeném úkolu.
- Záznam času stráveného řešením úkolu či u projektu.
- Shromažďování návodů a dokumentací - Wiki.

Jednotlivé body si následně popíšeme blíže, předtím bych však ještě chtěl zmínit možnost rozdělení úkolů do jednotlivých *projektů*.

4.1.1.1 Projekty v Redmine

Redmine umožňuje na jedné instalaci provozovat několik oddělených (a nebo hiararchicky uspořádaných) projektů. V rukou hlavního administrátora (kterého u DBS projektu zastávám také já) je možnost umožnit určitým uživatelům či rolím přístup k různým projektům a také udělit různé pravomoce v těchto projektech.
Pro vývoj DBS portálu využíváme následující projekty:

- `newDBS`: projekt shromažďující veškeré úkoly, které bude potřeba řešit, nemají však zatím přiřazeného řešitele.
- `SP-DBS-2017`: projekt zastřešující projekty jednotlivých týmů studentů, kteří mají zapsané předměty BI-SP1/BI-SP2. Tento projekt a jeho dílčí subprojekty se obměňují každým rokem, podle aktuálního počtu přihlášených studentů. V letním semestru 2017 se tento projekt dále dělí na:
  - `Team 1`

- `Team 2`

Proč dělit práci na projekty? Rozdělení na projekty je v běžné firmě jasné. V jednu chvíli se může pracovat na projektech několika klientů, které se sebou vůbec nemusí souviset. Proč ale dělit práci na projekty i u DBS portálu, který je jako celek pouze jeden projekt? Důvodem je především možnost využívání hromadných statistik, ale také rozdělení na realizační týmy.
Rozdělení má význam ve chvíli, kdy chce vedoucí týmu nebo manažer projektu zobrazit například celkový strávený čas jednoho z týmů, nebo zobrazení přidělené práce danému týmu. Díky odlišným projektům je snadné rozdělit úkoly, na kterých pracuje Team 1, Team 2 a nebo které nejsou zatím přiděleny k realizaci. Také vedoucí týmů mají snazší práci s přehledem práce svého vlastního týmu, mohou si zapnout emailové notifikace pouze pro změny v projektu, který patří jejich týmu atp.

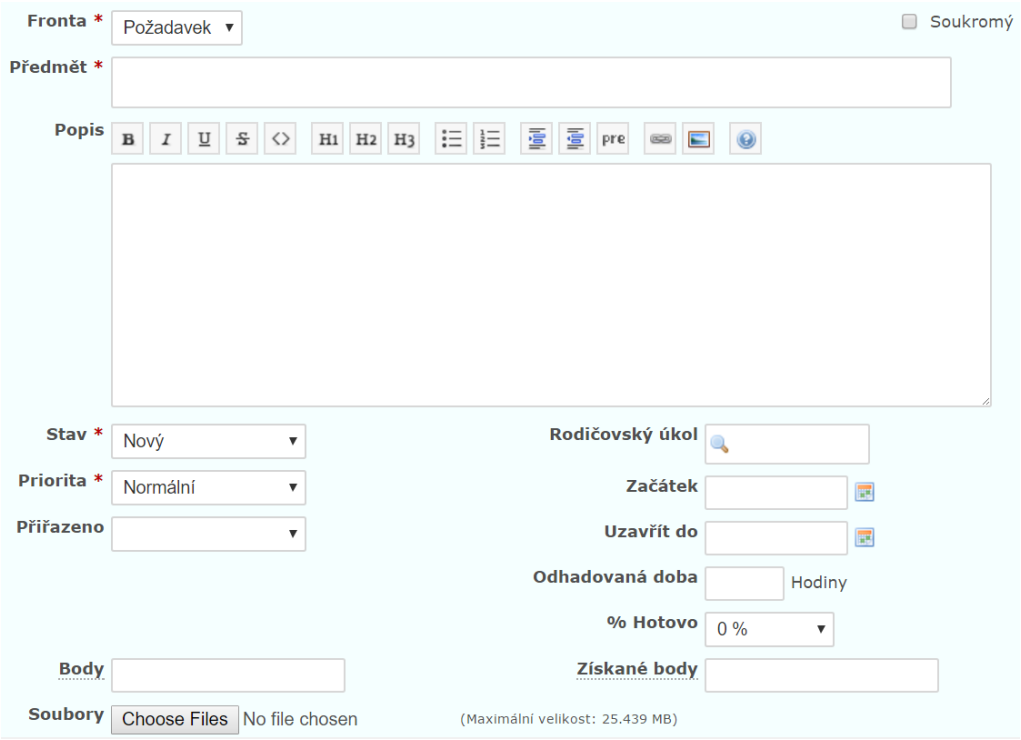
4.1.1.2 Shromažďování úkolů k řešení

Během práce na projektu často vyvstávají nové požadavky či se objevují nové chyby. Tyto chyby a požadavky je potřeba *ihned* zaznamenávat do issue trackeru. K tomu slouží projekt `newDBS`, který slouží jako „shromaždiště úkolů“, které jsou *volné* k řešení.
Odtud úkol typicky putuje dále zásahem manažera, který rozhodne o nutnosti jeho realizace a přiřadí úkol do jednoho z týmů - tedy projektu `Team 1` nebo `Team 2`. Některé úkoly jsou řešeny i přímo zde, ale to pouze výjimečně - na těch pracuji buďto přímo já nebo některý ze studentů pracujících na BP či DP.
Úkol, který je nově vytvořen v `newDBS` projektu má typicky vyplněna pouze následující pole:

- *Název:* stručný název úkolu. Na začátku názvu mohou být dodatečné popisné *tagy*, například `[databaze]` nebo `[menu]`, které jasně definují, které komponenty systému se úkol týká.

- *Popis:* detailní zadání úkolu, obsahující kroky k replikaci chyby, popis požadavku či nástin, jak úkol řešit.

- *Stav:* výčet stavů, v tuto chvíli nastaven na *Nový*. Všechny změny stavů popisuje Activity diagram průběhu práce v Redmine dostupný v příloze C.1.
- *Priorita:* priorita úkolu, u většiny úkolů nastavena na *Normální*. Dalšími možnými stavy je *Nízká*, *Vysoká*, *Urgentní* a *Okamžitá*.



Obrázek 4.1: Prostředí zadávání nového úkolu do Redmine

4.1.1.3 Sledování práce na přiřazeném úkolu

Jakmile je úkol přeřazen do jednoho z týmů, vyplní manažer i další dostupná pole:

- *Přiřazeno:* pole, které určuje člověka, po kterém je k úkolu požadován další vstup. Proces změny tohoto pole také popisuje Activity diagram průběhu práce v Redmine dostupný v příloze C.1.
- *Uzavřít do:* úkolu se nastaví termín, do kterého musí být vyřešen. V případě nevyřešení může být tým, kterému byl úkol přiřazen, bodově penalizován.

- *Body:* manažer odhadne obtížnost řešení úkolu a přiřadí mu určité bodové ohodnocení. Více o bodování bylo probíráno v sekci 3.3.3.3: Systém odměňování a oceňování.

V tuto chvíli přechází úkol do kompetence daného týmu, který na úkolu začne pracovat a postupně vykazuje svůj postup. K tomu slouží jednak možnost přidávat k úkolu komentáře, jednak možnost měnit pole *% Hotovo* a také zaznamenávat strávený čas (viz sekce 4.1.1.4).

Průběh práce je také možné sledovat na *gitu*, respektive *Gitlabu*, kterým se více věnují samostatné sekce 4.2.1, respektive 4.2.2.

Návaznost commitů v gitu na Redmine úkoly Redmine umožňuje k projektu připojit i repozitář - v našem případě git. Při takovém provázání je vhodné, aby jednotlivé commity v gitu byly určitým způsobem provázány s úkolem, který řeší. Toho lze docílit přidáním ID úkolu do *commit message*. Jedná se o jedno z pravidel pro psaní commitů, kterým se blíže věnuje sekce 4.2.1.2. Takto provázané commity se poté zobrazují pod úkolem vedle poznámek uživatelů a je možné si je jednoduše otevřít a zobrazit změny v kódu.

Automatický report chyb v testech Jelikož nové *commity* jsou po nahrání do sdíleného repozitáře testovány automatickými testy, je možné ihned zjistit případné nedostatky řešení. Pokud se při automatickém testu objeví chyba, je přes Redmine API přidán komentář k úkolu, ke kterému se commit váže, s informací, že test selhal, a s odkazem na Gitlab.

Provázání commitů na Gitlab Kromě automatického reportu chyb, který zrealizoval Pavel Kovář v rámci své BP, jsem také já upravil zdrojové kódy naší instalace Redmine pro snazší integraci Redmine a Gitlab. Jak již bylo zmíněno výše, Redmine sám zobrazuje u úkolu příslušné commity, samotné zobrazení commitu ale nedosahuje takových kvalit jako dedikované řešení: Gitlab. Z toho důvodu jsem k zobrazení odkazu na commit přidal i odkaz na Gitlab, jak znázorňuje obrázek 4.2.

Dokončení úkolu Když je úkol dokončen, je dle workflow (příloha C.1) přiřazen zpět na manažera, který zhodnotí kvalitu výstupu a vyplní pole

Obrázek 4.2: Stav před a po nasazení úpravy kódu Redmine pro zobrazování odkazu na Gitlab

Získané body. Množství získaných bodů se odvíjí od předem stanovené bodové dotace za daný úkol a kvality jeho řešení definované v tabulce 3.1.

4.1.1.4 Záznam času

Řešitel úkolu si k němu také zaznamenává strávený čas, spolu s poznámkami, na čem v zaznamenané době pracoval. Tento čas se poté zobrazuje u úkolu a lze tak sledovat, zda byl odhad doby řešení správný, nebo je potřeba zvýšit časovou či bodovou dotaci. Také je možné v přehledu projektu či projektů zobrazovat, kolik hodin kteří uživatelé odpracovali a podle toho například rozdělovat hodnocení jejich práce.
Je však nutno zmínit, že záznam času není vždy ideálním měřidlem výkonu uživatele. Stejně obtížný úkol může zkušený programátor zrealizovat během 20 minut, kdežto nezkušený nováček může nad úkolem strávit i několik hodin a výstup i přesto zůstává sporný. Pro hodnocení by tak kromě vykázaného času měl být brán ohled i na kvalitu výstupu, případně další ukazatele.

4.1.1.5 Wiki

Redmine poskytuje také možnost tvorby *knowledge base* či *Wiki*, kterou využíváme především u newDBS[1] projektu. Informace zde shromažďované slouží jak pro podporu nových studentů, tak jako dokumentace aplikace. Jsou zde návody jak nasadit vývojové prostředí, spouštět testy, požadované formáty kódu i commit message a další. Struktura Wiki v současnosti vypadá následovně:

[1]Dle rozdělení projektů v Redmine v sekci 4.1.1.1

- Instalujeme projekt - *Sekce popisující nasazení vývojové verze projektu na počítač programátora*
  - Vagrant
    * Instalace a používání
    * Integrace do PHPStorm
    * Debug Selenium testů
    * Řešení problémů
  - Manuální instalace
- První spuštění a konfigurace nástrojů - *Popis prvotního nastavení aplikace a vývojových nástrojů*
  - První spuštění aplikace
  - Kontrola code style
  - Integrace s Tracy v PHPStorm
  - Integrace Codeception s PHPStorm
- Code practices - *Seznam pravidel pro vývoj projektu. Obsahuje informace jako například požadavky na formátování a strukturu kódu, pojmenovávání tabulek databáze, strukturu commit message v gitu a další*
- Testování - *Informace o tvorbě a spouštění automatických testů aplikace*
  - Základní informace
  - Tvorba testů
  - Správa CI
- Issue tracker, hodnocení - *Informace o používání redmine*
  - Redmine workflow - *Activity diagram průběhu práce v Redmine, který je v této práci dostupný v příloze C.1*
  - Veřejný issue tracker na GitLabu

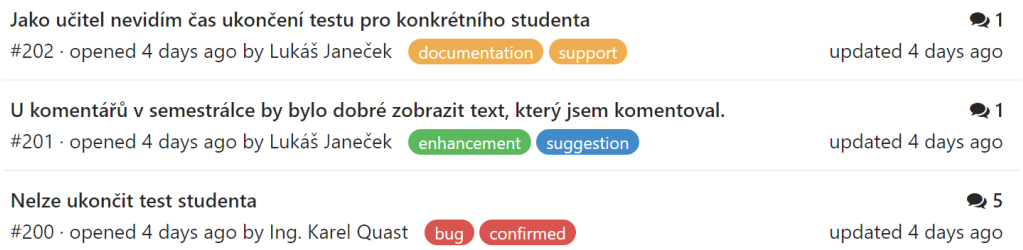
    - Bodové hodnocení - *Popis bodového hodnocení, v této práci popsaného v sekci 3.3.3.3*
- Technologie - *Dokumentace používaných technologií*
    - Nástroje a task runner
    - Databázové migrace
    - Dokumentace různých komponent systému
- Repozitář - *Informace o repozitáři, návod jak používat git*
    - Gitlab
    - Doporučený git workflow
- Server - *Informace o serveru, na kterém výsledná aplikace běží*
    - Datadog monitoring - *Informace o Datadogu, v této práci blíže popsáno v sekci 4.4.6.1*
    - Offsite zálohy - *Informace o zálohách nahrávaných na externí úložiště*

Je tedy jasné, že v případě jakýchkoliv nejasností je Wiki výchozím místem, kde hledat informace o projektu a všem, co s ním souvisí. Podoba Wiki se v průběhu času mění, jelikož nejen aplikace samotná, ale i způsob jejího vývoje prochází inovacemi.
Je důležité na Wiki zaznamenávat i dokumentaci různých procesů, které většina členů projektu využívá pouze uživatelsky a nikoliv administrativně, a to především z důvodu, aby byl *kdokoliv v projektu nahraditelný*. Kvalitní dokumentace umožní při výpadku člena týmu pokračování ve vývoji i v případech, kdy se daný člen staral například o nasazování aplikace na server nebo spravoval jinou důležitou součást.

4.1.2 Gitlab

Gitlab je druhým issue trackerem využívaným při vývoji DBS portálu. Používá se pro *spojení s koncovými uživateli*, kteří zde nahlašují chyby. Jelikož issue tracker na Gitlabu je jednodušší než ten na Redmine a všichni studenti i vyučující na FIT ČVUT jsou na něm automaticky registrováni, je vhodný pro využití

právě veřejného bug reportu.
Já jakožto manažer projektu jsem emailem upozorňován na všechny nově nahlášené bugy či žádosti o podporu s používáním aplikace a snažím se na úkoly v co nejkratší době reagovat. Podpora uživatelů je typicky vyřízena do několika hodin, opravy chyb či požadavky nových funkcí jsou však interně předány na Redmine a poté zadány k realizaci týmům studentů, z toho důvodu může jejich vyhotovení trvat dny až týdny.
Uživatelé aplikace se k tomuto issue trackeru dostanou přes proklik přímo z aplikace, je však dostupný i přes link `https://gitlab.fit.cvut.cz/malecold/newDBS-bug/issues`.



Obrázek 4.3: Nahlášené úkoly veřejného bug reportu v Gitlabu

4.2 Verzování a testování

Prakticky jakýkoliv projekt, na kterém pracuje více lidí nebo trvá více než několik hodin je potřeba *verzovat*. Jedním z nejpopulárnějších nástrojů pro verzování softwaru a jiných textově-založených projektů (jako je například tato práce psaná v XeLaTeX) je *Git*.
Git sám o sobě nabízí pouze command line ovládání a prostředí, existují však různé nástavby nad gitem, které usnadňují či zpřehledňují jeho použití. Nejpopulárnějším je bezesporu Github, pro účely DBS projektu je však využíván Gitlab, který je na fakultě dostupný všem studentům i vyučujícím. S verzováním softwaru úzce souvisí jeho testování. Automatické testy, neboli CI, je spouštěno na Gitlabu při každé změně zdrojových kódů.

4.2.1 Git

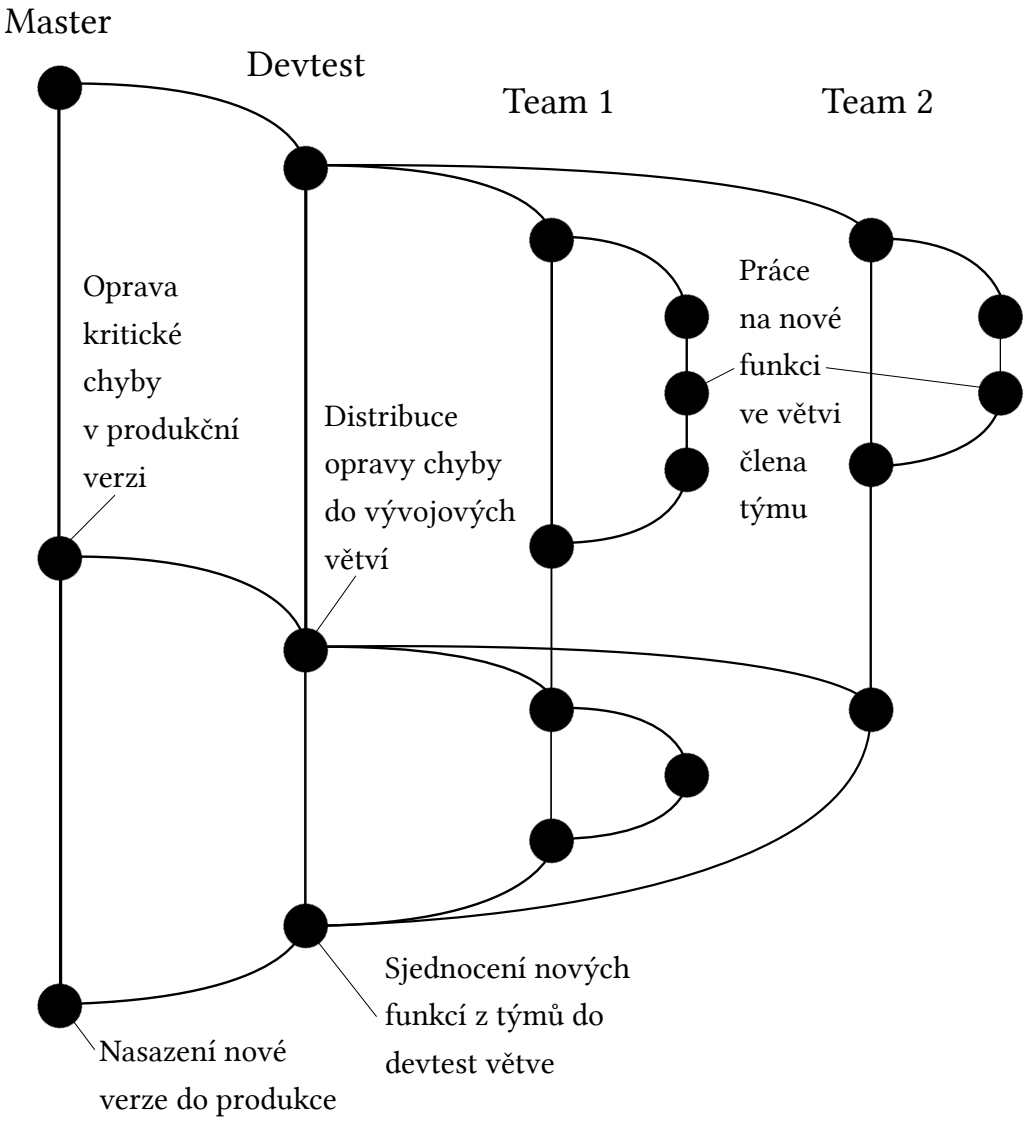
> „Git je distribuovaný systém správy verzí, který je zdarma a open source. Je navržený tak, aby zvládal vše od malých po velmi velké projekty s rychlostí a efektivitou“[22].

4.2.1.1 Branche v gitu

Jednou z hlavních výhod vývoje softwaru pomocí gitu je možnost využívání branchí, neboli různých vývojových větví. Toto umožňuje v jednu chvíli udržovat informace o nasazené - produkční - verzi aplikace a současně mít v dalších větvích rozpracované nové funkce. Když se poté v produkční verzi objeví kritická chyba, může vývojář přepnout z branche s rozpracovanou novinkou na stabilní nasazenou branch, zde provést opravu a poté opět pokračovat ve vývoji nové funkce. Tento proces je ilustrován na obrázku 4.4, který dokumentuje používání branchí v DBS projektu.

- *`master` branch* obsahuje vždy verzi aplikace, která je nasazena na doméně `dbs.fit.cvut.cz`
- *`devtest` branch* slouží pro testování nových funkcí. Shromažďují se do ní novinky z týmových branchí a bývá pravidelně nasazována na doménu `dbs2.fit.cvut.cz`
- *Týmové větve* pojmenované `team1` a `team2` jsou již v režii týmových vedoucích. Je však doporučeno další rozdělení na branche jednotlivých členů týmu, jelikož každý typicky pracuje na jiné funkci.
- *Branche členů týmů* pojmenované `team1-username` či `team2-username`, kde `username` je uživatelské jméno studenta, které mu bylo přiděleno fakultou (5 písmen příjmení + 3 písmena jména, duplikáty nahrazovány odzadu čísly)

Díky tomuto rozdělení je mnohem snažší identifikovat, kdo na čem pracuje, a také je zaručena lepší kontrola výstupů. S tímto nastavením branchí souvisí také Activity diagram průběhu práce v Redmine dostupný v příloze C.1. Jakmile je úkol předáván vedoucím týmu zpět na manažera projektu, musí být výsledek již v týmové větvi `team1` nebo `team2`. Manažer projektu tedy výstup



Obrázek 4.4: Používání branchí v gitu při vývoji DBS portálu

zkontroluje v jedné z těchto větví a v případě, že byl úkol splněn a může být nasazen, provádí *merge* do větve `devtest`. Zde provedené novinky jsou poté v určitých intervalech nasazovány na adresu `dbs2.fit.cvut.cz`, kde je ještě jednou zkontrolována jejich funkčnost. Vydání nové verze (tedy merge branche `devtest` do `master`) probíhá typicky pouze jednou týdně, po několikadenním testování nových funkcí na testovací doméně.

4.2.1.2 Formát commit message

Zanesení implementace nových funkcí do sdíleného repozitáře probíhá pomocí *commitů*, ke kterým se píší krátké zprávy, které popisují, co daný commit přináší nového, co mění atp.
V DBS projektu jsme nastavili i požadovanou strukturu této *commit message*. Pokud commit řeší nějaký úkol z Redmine, musí obsahovat na svém začátku ID úkolu formou tagu: například `[#2065] New version of DBSDM deployed`. Jak je vidět, i komentář samotný by měl být napsán anglicky a měl by znovu shrnout, co bylo úkolem. Kromě tagu s ID úkolu mohou být volitelně přidány i další tagy, které jednoslovně shrnují, co commit řeší.
Přidání ID úkolu z Redmine do commit message v gitu umožňuje jejich *propojení*, které již bylo popisováno v sekci 4.1.1.3.

4.2.1.3 Tagy v gitu

Tagy [24] jsou *značky*, které lze přiřazovat určitým commitům. To umožňuje jednak zachytit jejich významnost, jednak rychlý návrat k některé z takto označených verzí.
Tagování commitů probíhá u DBS projektu automaticky, během nasazování nové verze aplikace. Tagy přidává skript, o kterém pojednává následující sekce.

4.2.1.4 Hooky v gitu

Hooky [23], česky háky nebo háčky, jsou vlastně skripty, které se automaticky spouští před nebo po provedení některé akce v gitu. Jako příklad lze uvést `pre-commit` hook, který je spuštěn vždy ve chvíli, kdy uživatel (nebo jiný skript) použije `commit`. Teprve po skončení `pre-commit` hooku se pokračuje ve vykonání samotného *commitu*.
V DBS projektu je na celoprojektové úrovni využíván `post-merge` hook, který je součástí repozitáře s nasazenou aplikací umístěného na serveru. Tento se spouští *vždy*, když je proveden *merge* (tedy při nasazení jakékoliv nové změny na server, protože v `pull` je interně zahrnut i `merge`). Jeho cílem je jednak změnit verzi aplikace, která se zobrazuje cílovému uživateli a používá se i pro verzování proměnných zdrojů - načítané Javascript a CSS soubory, jednak aktualizuje changelog, jak bude popsáno v sekci 4.4.1, která se tomutu tématu věnuje blíže.

Nakonec je ještě přidán *tag*, který byl popsán v sekci předchozí. Zkrácená ukázka `post-merge` hooku je znázorněna v ukázce kódu 4.1.

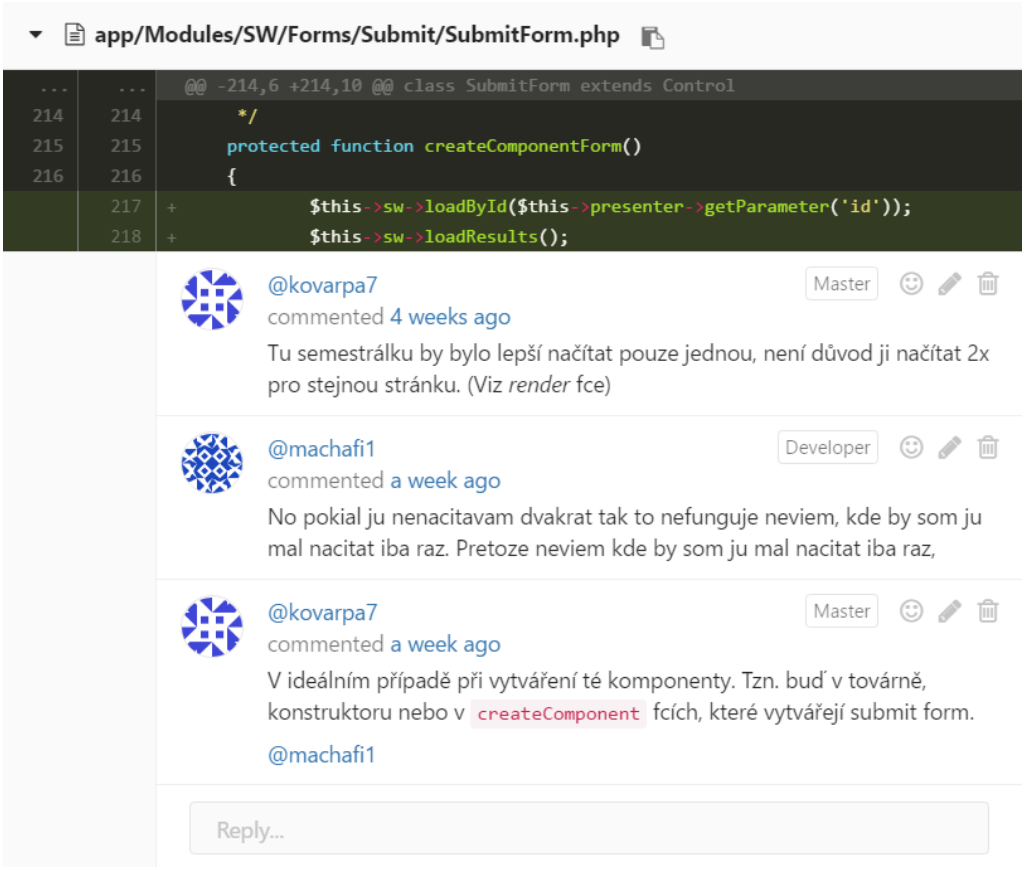
```
awk '
    BEGIN {
        FS = ".";
    }
    {
        if (/version:/) {
            ORS = "";
            for( i = 1; i < NF-1; ++i ){
                print $i
                print "."
            }
            print $(NF-1)+1
            ORS = "\n"
            print ".0"
        } else {
            print
        }
    }
' app/Config/config.neon > app/Config/config.temp
    && mv app/Config/config.temp app/Config/config.neon

git add app/Config/config.neon

echo "Changing 'Unreleased' to new version in CHANGELOG.md"

version=$(grep -m1 "version" app/Config/config.neon | tr -d " " |
    cut -d: -f2 | cut -d. -f1,2)
date=$(date +%Y-%m-%d)

if [ $(head -n1 CHANGELOG.md | grep "Unreleased" | wc -l) == 1 ]; then
    change="## $version - $date"
    sed "1s/.*/$change/" CHANGELOG.md > CHANGELOG.temp
        && mv CHANGELOG.temp CHANGELOG.md

    git add CHANGELOG.md
fi

echo "Committing new version number and changelog"
git commit -am"[deploy][ci skip] Increased minor version number,
    updated changelog"
git push

echo "Creating new tag"
git tag -a v${version} -m"Version ${version}, released ${date}"
git push --tags
```

Ukázka kódu 4.1: Post-merge hook pro automatizaci verzování aplikace

4.2.2 Gitlab

> „Gitlab je webové rozhraní nad Gitem, které navíc obsahuje wiki, CI (Continuous integration) a CD (Continuous deployment). Nabízí jednotnou správu Git repozitářů, ať už soukromých nebo veřejných“[3].

Využívání Gitlabu přináší především přehlednější správu celého projektu - interaktivní webové rozhraní je mnohem přívětivější než terminálové prostředí čistého Gitu. Kromě zobrazování nových commitů je často využívána možnost přidávat komentáře k určitému řádku souboru/commitu, což se hodí především při revizi kódu od studentů. Tuto funkci ilustruje obrázek 4.5



Obrázek 4.5: Komentáře ke commitu v Gitlabu

4.2.2.1 Automatické testování

O automatické testy se v projektu stará především Pavel Kovář v rámci své bakalářské práce [13]. Ty se spouští autonomně, při každému commitu, a umožňují tak ihned odhalit chybu ve funkci pokryté testem. Oznámení o chybě je jednak programaticky zapsáno na Redmine, jak již bylo popsáno v sekci 4.1.1.3, jednak se nahlásí na Slack, kterému se věnuje následující sekce 4.3.1. Testy nejsou spouštěny přímo na Gitlabu, ale zpracovává je server, na kterém běží i výsledná aplikace. Jejich zatěžování systému je monitorováno v Datadogu (4.4.6.1) a je dbáno na to, aby neomezovaly rychlost používání nasazené aplikace.

4.3 Komunikace projektového týmu

I pro komunikace týmu je vhodné zvolit ty správné nástroje. Email je rozhodně neumírajícím komunikačním kanálem například s externím klientem. Uvnitř týmu je však často potřeba používat rychlejší a flexibilnější komunikační nástroj.

4.3.1 Slack

Slack je komunikační nástroj 21. století [21]. Jedná se o webovou, desktopovou i mobilní aplikaci, která umožňuje vytvořit i několik týmů.
Tým je ucelené sdružení lidí, kteří pracují na stejném projektu. V rámci jednoho týmu může komunikovat jakýkoliv člen s kterýmkoliv dalším přes soukromé zprávy, navíc ale mohou být vytvořeny kanály, ve kterých probíhá komunikace se zaměřením na určité téma. Tyto kanály mohou být jednak veřejné - kdokoliv z týmu vidí obsah a může se do kanálu připojit, nebo soukromé - autor kanálu musí pozvat ty členy, kteří mají mít přístup. U DBS projektu využíváme následující kanály:

- *#ci_build:* kanál slouží pro automatický report neúspěšných testů z Gitlabu, jak již bylo popsáno v sekci 4.2.2. Typicky se poté kolem oznámení o špatném buildu vytvoří další diskuze manažera a autora testu, ohledně toho, proč se testování nezdařilo.

- *#dbsdm:* komunikace s autorem DBSDM komponenty - nástroje pro tvorbu ER diagramů [6].

- *#general:* základní výchozí kanál každého Slack týmu - slouží pro vše, co je potřeba napsat všem a nezapadá do ostatních kanálů.

- *#grido_render:* kanál zabývající se možností nasazení lepšího frontendu pro tabulky použité v aplikaci.

- *#meetings:* slouží pro domlouvání termínů schůzek, omluvy při absenci atp.

- *#quiz:* kanál sloužil zejména na začátku semestru pro řešení problémů se vstupním kvízem, o kterém jsem již psal v předchozí kapitole (3.2.1).

- *#random:* druhý z výchozích kanálů, spolu s `#general`. Slouží pro neprojektové povídání, spam či odreagování.

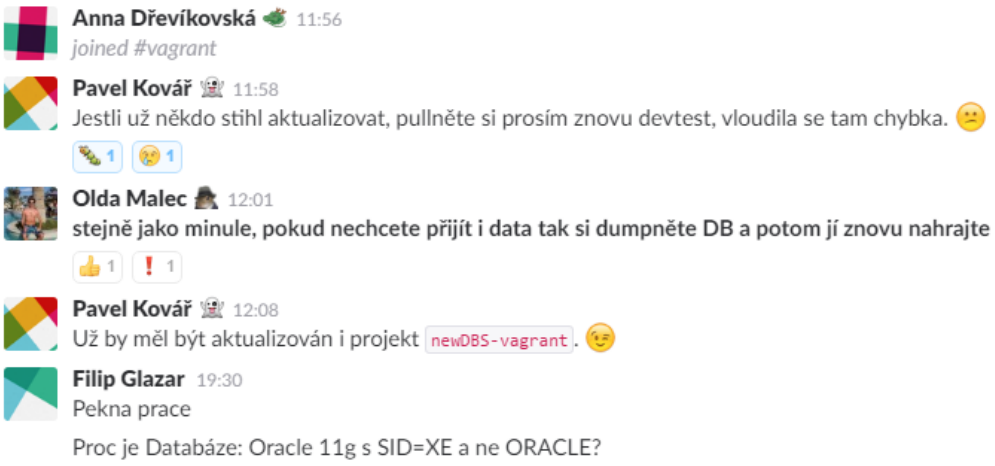
- *#team_leaders:* kanál určený pro komunikaci manažera s vedoucími týmů.

- *#tests:* v tomto kanále se řeší problémy s automatickými testy.

- *#vagrant:* řešení problémů s Vagrantem - vývojovým prostředím programátorů na jejich lokálních strojích. Také se využívá pro oznamování vydání nových verzí Vagrantu a sběr požadavků pro zakomponování do nové verze.

- *#vedeni:* uzavřený kanál vedení projektu, členem jsem já, Jiří Hunka a studenti pracující na BP týkajících se projektu, kteří mají přímý zásah i do týmů, tento semestr tedy Pavel Kovář a Milan Vlasák. Z minulého semestru také v kanálu zůstal Filip Glazar, který je autorem kostry systému pro semestrální práce [7], ale v současnosti se již projektu nevěnuje v takové míře.

Kromě těchto kanálů jistě existují ještě další uzavřené, které si například vytvořily jednotlivé týmy a já k nim nemám přístup.



Obrázek 4.6: Ukázka komunikace ve Slacku, kanál #vagrant

4.3.2 Redmine

Kromě Slacku je samozřejmě důležitým komunikačním kanálem i Redmine, na kterém se řeší úkoly. Především formou poznámek k úkolu dochází často k přímé komunikaci mezi manažerem, vedoucím týmu a řešitelem. Velikou výhodou je, že si uživatel může nastavit emailové notifikace na změny v Redmine, a to přesně podle svých požadavků. Například manažer projektu bude chtít být informován o všech změnách ve všech projektech, vedoucí týmu poté pouze o všech změnách ve svém týmu a nakonec řadový člen bude odebírat pouze změny, které se týkají přímo jeho práce.

4.3.3 Schůzka

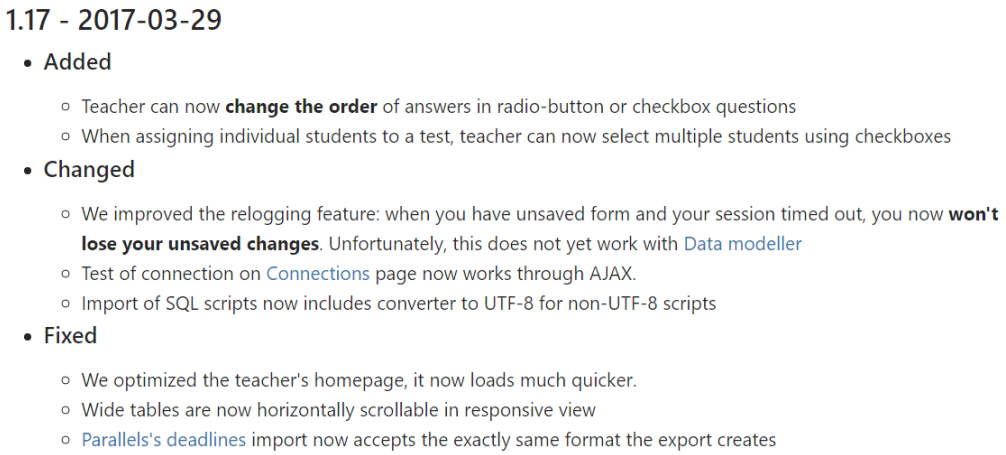
Nejméně častým, ale zato nejintenzivnějším komunikačním kanálem je bezesporu *schůzka*, která probíhá jedenkrát týdně a trvá většinou kolem jedné hodiny, často však po schůzce zůstanou někteří studenti déle a probírají s vedením řešení svých úkolů, či neočekávané chyby. Schůzka byla již částečně popisována v předchozí kapitole v sekci 3.4.1 a tak ji v kapitole o *Infrastruktuře* nebudu dále rozvádět.

4.4 Správa aplikace

V poslední sekci správy infrastruktury se zaměřím na služby a procesy, které se týkají běhu samotné aplikace. Budu popisovat například Changelog, SSL, jednotné přihlašování, používání různých podpůrných skriptů nebo monitoring výkonu. Jedná se především o *praktické* ukázky konfigurací, nastavených procesů, ale místy i analýzu výsledků, například monitorovacích služeb.

4.4.1 Changelog

Changelog popisuje změny aplikace v jejích jednotlivých verzích. Je užitečný jak pro koncové uživatele, kteří se pomocí changelogu mohou dozvědět, co je v aplikaci nového, tak pro testery, kteří vědí, které změny testovat.
Jak psát changelog popisuje například Olivier Lacan [14]. Changelog je pro lidi, rozhodně by tedy jeho obsah neměl být automaticky generován například z výpisu commitů z gitu. Tento výpis již existuje a vývojáři i testeři k němu mají v rámci repozitáře přístup, kopírovat jej do changelogu by tedy navíc byla duplikace informací. Správný changelog by měl jednoduše vypíchnout nejdůležitější novinky, změny a opravy v nové verzi.



1.17 - 2017-03-29

- Added
  - Teacher can now **change the order** of answers in radio-button or checkbox questions
  - When assigning individual students to a test, teacher can now select multiple students using checkboxes
- Changed
  - We improved the relogging feature: when you have unsaved form and your session timed out, you now **won't lose your unsaved changes**. Unfortunately, this does not yet work with Data modeller
  - Test of connection on Connections page now works through AJAX.
  - Import of SQL scripts now includes converter to UTF-8 for non-UTF-8 scripts
- Fixed
  - We optimized the teacher's homepage, it now loads much quicker.
  - Wide tables are now horizontally scrollable in responsive view
  - Parallels's deadlines import now accepts the exactly same format the export creates

Obrázek 4.7: Ukázka changelogu DBS portálu

Pro DBS aplikaci píši v současné době changelog já, a to ve chvíli, kdy se slévají změny z jednotlivých týmů do branche devtest, jak bylo popisováno ve

workflow gitu v sekci 4.2.1.1. Nejnovější sekce změn je nadepsána *Unreleased* a v této verzi je aplikace nasazena na `dbs2.fit.cvut.cz`. Učitelé, kteří se rozhodnout na této doméně otestovat novou verzi, tak mají k dospozici seznam bodů, které popisující co se v systému měnilo, a mohou se na ně zaměřit. Při následném nasazení na hlavní doménu `dbs.fit.cvut.cz` je součástí `post-merge` hooku (více v sekci 4.2.1.4), automatická změna nadpisu *Unreleased* na aktuální verzi a datum.

4.4.2 SSL

> „SSL (Secure Sockers Layer) je zabezpečovací technologie, používaná pro navazování zašifrovaných spojení mezi serverem a klientem - typicky webovým serverem a prohlížečem, případně mezi mailovým serverem a mailovým klientem.
> Toto umožňuje bezpečně posílat citlivé údaje - čísla kreditních karet, hesla k účtům atp.“[5].

Pro DBS portál je toto důležité, jelikož umožňuje přihlašování uživatelů - učitelů i studentů. Jelikož je pro přihlašování navíc využívána technologie Shibboleth, o které ještě pojednává následující sekce 4.4.3, a pro jejíž běh je SSL vyžadováno, muselo být pro funkčnost portálu nasazeno.
Aplikace standardně běží na adrese `dbs.fit.cvut.cz`, v letním semestru 2016 však zároveň probíhalo přepsání celé aplikace z důvodu nového lepšího návrhu a efektivnější znovupoužitelnosti kódu. Z toho důvodu byla zřízena nová doména `dbs2.fit.cvut.cz`, na které měla souběžně se starou verzí běžet i nová verze aplikace. I tuto doménu však bylo potřeba zabezpečit pomocí SSL, o což jsem se postaral vykonáním následujících kroků:

1. *Vygenerování certifikátu:* certifikáty generuje cetrifikační autorita - pro naši fakultu CESNET. Komunikace s CESNETem navíc zajišťuje Ing. Martin Bílý, napsal jsem mu tedy email a za pár dní mi byl zaslán certifikát pro novou doménu.

2. *Nasazení certifikátu na server:* přes standardní konzolové příkazy je vygenerovaný certifikát a klíč potřeba nahrát do složky `/etc/ssl/certs/`, respektive `/etc/ssl/private/`.

3. *Konfigurace webserveru:* v `/etc/apache2/sites-available` (jedná se o Debian) je potřeba nakonfigurovat jednak přesměrování standardních ne-https požadavků na https, jednak nastavit samotné https. Konfigurace `apache` webserveru pro využívání SSL je zachycena v ukázce kódu 4.2.

```
SSLEngine on

SSLCipherSuite ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA
    256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-
    RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:
    ECDHE-RSA-AES128-SHA:DHE-RS$
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On

SSLCertificateFile /etc/ssl/certs/dbs2.fit.cvut.cz.pem
SSLCertificateChainFile /etc/ssl/certs/chain_TERENA_SSL_CA_3.pem
SSLCertificateKeyFile /etc/ssl/private/dbs2.fit.cvut.cz.key

# ostatní standardní konfigurace, netýkající se SSL
```

Ukázka kódu 4.2: Konfigurace Apache pro použití SSL

4.4.3 Shibboleth

Jak již bylo zmíněno v předchozí sekci, přihlašování do aplikace zajišťuje *Shibboleth:*

> „Shibboleth je jedním ze světově nejpoužívanějších řešení pro propojení uživatelů s aplikacemi, ať už uvnitř jedné nebo napříč různými organizacemi. Každá softwarová komponenta Shibbolethu je zdarma a open source.
> Shibboleth poskytuje SSO (Single Sign-On) a dovoluje stránkám provádět autorizovaná rozhodnutí o přístupu k chráněnému obsahu pro jednotlivé uživatele“[26].

Tento systém je využíván právě i na ČVUT, o jeho používání pro přihlašování do DBS portálu tedy nebylo pochyb. Systém umožňuje koncovému uživateli přihlásit se na *jakoukoliv* stránku, kde autorizace přístupu probíhá pomocí Shibbolethu a následně má přístup i k ostatním aplikacím, které rovněž využívají

Shibboleth, bez nutnosti dalšího přihlašování.

```
<ApplicationOverride id="dbs2.fit.cvut.cz" entityID="https://dbs2.fit.
    cvut.cz/shibboleth">
    <CredentialResolver type="File">
        <Key>
            <Path>/etc/ssl/private/dbs2.fit.cvut.cz.key</Path>
        </Key>
        <Certificate>
            <Path>/etc/ssl/certs/dbs2.fit.cvut.cz.pem</Path>
        </Certificate>
    </CredentialResolver>
</ApplicationOverride>
```

Ukázka kódu 4.3: Konfigurace Shibboleth pro novou doménu s vlastním SSL certifikátem

Zprovoznění systému však není triviální, stejně jako v předchozím případě s SSL, i Shibboleth jsem pro doménu `dbs2.fit.cvut.cz` nasazoval já, a její hlavní část sestávala v úpravě konfiguračních souborů na serveru ve složce `/etc/shibboleth/`, kam bylo do souboru `shibboleth2.xml` potřeba přidat nový `ApplicationOverride` (ukázka kódu 4.3), který bude odkazovat na SSL certifikáty probírané v předchozí sekci. Override je potřeba z důvodu, že na serveru již běží jiná aplikace, která Shibboleth využívá - `dbs.fit.cvut.cz` a přihlášení do jednotlivých aplikací má být na sobě nezávislé.
Dále bylo potřeba opět upravit konfiguraci webserveru (`apache`), což znázorňuje ukázka kódu 4.4. Na adrese, kam směřuje přepisovací pravidlo v ukázce kódu 4.4, tedy `https://dbs2.fit.cvut.cz/Shibboleth.sso/Metadata` je zároveň během konfigurace Shibbolethu možné sledovat, zda je nastaven správně. Přístup na tuto adresu stáhne soubor Metadata, který obsahuje informace o této doméně, její jméno, adresu, certifikát, vlastníka atp. Toto jsou informace, na základě kterých je uživatelům umožněno k aplikaci přistupovat. Informace o vlastníkovi domény se standardně definují formou šablony v podobném umístění jako první ukázka konfigurace - pro zprovoznění domény `dbs2.fit.cvut.cz` však nebylo zapotřebí konfigurovat novou šablonu popisujících informací, jelikož již jedna existovala pro hlavní doménu.

```
RewriteEngine on
RewriteRule ^/shibboleth
    https://dbs2.fit.cvut.cz/Shibboleth.sso/Metadata [R,L]

# ostatní konfigurace, netýkající se Shibbolethu
# je však potřeba konfigurovat SSL
```

Ukázka kódu 4.4: Konfigurace Apache pro použití Shibboleth autentizace

Poznámka k označení *doména* V předchozím textu jsem často používal označení *doména* pro adresy `dbs.fit.cvut.cz` a `dbs2.fit.cvut.cz`. Ve skutečnosti se však nejedná o rozdílné domény, pouze rozdílná `CNAME` neboli Canonical name [27]. To znamená, že se jedná pouze o alias, který odkazuje na stejný server a zde probíhá rozdělení na základě name-based virtual hostů, kteří jsou nakonfigurováni ve webserveru, v našem případě `apache`.

4.4.4 Skripty pro nasazení aplikace

Nasazování aplikace je jedním z úkonů, které opakují stále stejné postupy, není však cílem je plně automatizovat. Důvodem je kontrola nad aktuální nasazenou verzí, především ta na produkčním serveru *musí* být vždy stabilní. Nasazování provádím já a pro zjednodušení práce jsem si vytvořil skripty, například pro nasazení `master` branche na `dbs.fit.cvut.cz` je využíván skript z ukázky kódu 4.5. Tento skript se spouští z rootu lokálně nasazené aplikace a využívá Git, ve kterém nejprve do `master` branche provede merge z `devtest`, jak bylo popisováno v sekci 4.2.1.1. Následně se připojí na server přes `ssh` a zde provede `pull`, čímž aktualizuje zdrojové soubory ze sdíleného repozitáře. Nakonec jsou ještě promazány cache, které se mohly z důvodu aktualizace zdrojových souborů stát neaktuálními.
Podobné skripty mám připraveny i pro samotné nasazení `master` branche, bez přidávání změn z `devtest`, což je využíváno při aplikaci hotfixů přímo do produkční větve. Dalším skriptem je nasazování testovací verze na `dbs2.fit.cvut.cz`, který se velmi podobný výše zmíněnému, pouze jsou prohozeny větve `master` a `devtest`, jelikož je cílem do druhé zmíněné dostat i případné hotfixy, které byly provedeny přímo v té produkční.

```
#!/bin/bash

set -o errexit -o nounset -o pipefail

cd ~/Documents/DBS/newDBS-vagrant/newDBS/
git checkout master
git pull
git fetch origin devtest:devtest
git merge devtest
git push
echo "Connecting to the server"
ssh -t root@dbs.fit.cvut.cz '
        echo "Switching to project dir"
        cd /var/www/newDBS
        echo "Switching to master branch"
        git checkout master
        echo "Pulling git"
        git pull
        echo "Clearing cache"
        rm -rf temp/cache
        exit
    '
# return to original directory
git pull
cd -
```

Ukázka kódu 4.5: Skript pro nasazení nové verze aplikace pomocí gitu

4.4.5 Zálohování

DBS portál v současnosti využívá zhruba 550 studentů a řady vyučujících. Studenti mají na serveru uložena data svých rozpracovaných i odevzdaných semestrálních prací, učitelé například testové otázky nebo hodnocení studentů. Je proto velmi žádoucí vše bezpečně zálohovat.

Zálohování probíhá jednak na serveru - na stejný filesystém. Tento typ zálohy se hodí ve chvíli, kdy se v současných datech objevil problém a je potřeba navrátit starší verzi. Pro případ problému s úložištěm samotným je však potřeba provédět i tzv. *off-site* zálohy, neboli zálohy na *jiný počítač*.

Oba typy záloh provádí bash script znázorněný v ukázkách kódu 4.6 a 4.7, který na serveru spouští *cron*. Zálohy jsou spouštěny jedenkrát denně - kolem 4 hodiny ranní. Zálohuje se databáze a soubory na filesystému. Zálohy jsou uchovávány vždy týden, starší jsou mazány. Navíc jedenkrát měsíčně je

uchována i persistentní záloha. Na konci skriptu probíhá i nahrání nových souborů na externí úložiště - IDrive [11].

```
#!/bin/bash
PROJECTPATH="/var/www/newDBS"

# ===== SWFILES =====
for DIR in swFiles swFilesSubmitted images
do
    mkdir -p ${PROJECTPATH}/backup/${DIR}

    # create backup
    cd ${PROJECTPATH}/www
    tar zcf ${PROJECTPATH}/backup/${DIR}/$(date +%Y%m%d).tar.gz ${DIR}

    # delete backups older than a week
    cd ${PROJECTPATH}/backup/${DIR}/
    rm $(find . -maxdepth 1 -type f -regex '.*\.tar\.gz' | sort |
        head -n-7) 2>/dev/null

    # once a month, create a persistent backup
    if [ $(date +%d) == 01 ] ; then
        cd ${PROJECTPATH}/backup/${DIR}/
        mkdir -p persistent
        cp $(find . -maxdepth 1 -type f -regex '.*\.tar\.gz' | sort |
            tail -1) persistent/
    fi
done

# ===== DATABASE =====
mkdir -p ${PROJECTPATH}/backup/database

# create dump
cd ${PROJECTPATH}/backup/database/
pg_dump -h localhost -d <dbname> -U <dbuser> > $(date +%Y%m%d).sql
gzip -f $(date +%Y%m%d).sql

# delete dumps older than a week
rm $(find . -maxdepth 1 -type f -regex '.*\.sql\.gz' | sort |
    head -n-7)

# once a month, create a persistent dump
if [ $(date +%d) == 01 ] ; then
    mkdir -p persistent
    cp $(find . -maxdepth 1 -type f -regex '.*\.sql\.gz' | sort |
        tail -1) persistent/
fi
```

Ukázka kódu 4.6: Skript pro automatické zálohování - část A: vytvoření lokálních záloh souborů a databáze

```
# ===== OFFSITE BACKUP =====
# delete older backups, we only have 5 gigs of space there
for DIR in swFiles swFilesSubmitted images
do
    cd ${PROJECTPATH}/backup/${DIR}/
    find . -maxdepth 1 -type f -regex '.*\.tar\.gz' | sort |
        head -n-7 > /etc/idrive/to_delete.txt
    /etc/idrive/idevsutil --password-file=/etc/idrive/pwfile
        --delete-items --files-from=/etc/idrive/to_delete.txt
        dbsfitcvutcz@gmail.com@evs1247.idrive.com::
        home/var/www/newDBS/backup/${DIR}
done
cd ${PROJECTPATH}/backup/database/
find . -maxdepth 1 -type f -regex '.*\.sql\.gz' | sort |
    head -n-7 > /etc/idrive/to_delete.txt
/etc/idrive/idevsutil --delete-items
    --password-file=/etc/idrive/pwfile
    --files-from=/etc/idrive/to_delete.txt dbsfitcvutcz@gmail.com
    @evs1247.idrive.com::home/var/www/newDBS/backup/database

# upload new backups
/etc/idrive/idevsutil --password-file=/etc/idrive/pwfile
    --files-from=/etc/idrive/filelist.txt /
    dbsfitcvutcz@gmail.com@evs1247.idrive.com::home/
```

Ukázka kódu 4.7: Skript pro automatické zálohování - část B: odeslání záloh na externí úložiště

Kromě zálohování databáze a dat semestrálních prací a obrázků použitých v testech jsou samozřejmě zálohovány i zdrojové kódy aplikace. Toto je však řešeno implicitně používáním gitu, který je *distribuovaný*, a v případě problému s hlavním repozitářem ho je možné obnovit z jakéhokoliv klienta.

4.4.6 Monitoring serveru a aplikace

Každý server, který poskytuje obsah koncovým uživatelům, musí být nějakým způsobem monitorovaný - sledována jeho dostupnost, rychlost atp. Kromě serveru samotného je navíc vhodné sledovat i výkon aplikace, což může odhalit problémy s nastavením nebo neefektivitu kódu, které mohou vést až na dlouhé načítání stránek koncových uživatelů.
O monitorovacích službách, které se používají v DBS projektu, pojednávají následující sekce.

4.4.6.1 Datadog

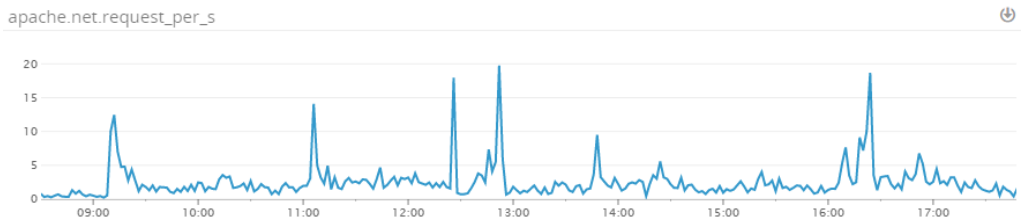
Datadog [4] je monitorovací služba, která umožňuje získávat data z mnoha různých služeb běžících na serveru, jako jsou například webserver, databáze, verzovací systém či obecné zatížení procesoru, paměti a diskových operací. Pro tyto služby jsou nabízeny připravené integrace, které nabízejí jednoduché nastavení jejich monitoringu. Dále je možné také do výsledných grafů přidat aktivitu z Gitu či Redmine, což ulehčuje následné hledání chyby - například navýšení využívání operační paměti lze poté jednoduše spojit i commitem v gitu, který začal způsobovat memory leaky. Právě toto propojení *všech* služeb je hlavní výhodou Datadogu, spolu s možností nastavení vlastních *monitorů* - lze sledovat buďto dostupnost jednotlivých služeb, jejich výkon, nebo dokonce kombinovat více monitorů dohromady, díky čemuž je možné sledovat například zatížení procesorů v závislosti na počtu připojených klientů.

Do DBS projektu jsem nasadil Datadog až v letním semestru 2017, jelikož se standardně jedná o placenou službu, avšak nyní jsem objevil možnost využívání zdarma pro studijní účely - což DBS projekt rozhodně je.
Monitorované služby jsou:

- *obecné statistiky:* zatížení CPU, využití paměti,
- *síťové statistiky:* velikost přenesených dat (download vs upload),
- *diskové statistiky:* prodleva přístupu k disku, množství zápisu a čtení, využití CPU při diskových operacích,
- *monitoring webserveru (apache):* množství přístupů, velikost servírovaných dat, poměr pracujících a čekajících threadů,
- *monitoring databáze (PostgreSQL):* počet načtených řádků, počet připojení, počet úprav databáze, využití disku, zatížení systému, využití paměti, velikost přenesených dat po síti,
- *monitoring Docker kontejnerů:* sledování výkonu automatických testů spouštěných v Dockeru (4.2.2.1). U kontejnerů je sledováno velké množství informací jako například využití CPU, paměti, swapu, cache, sítě, disku atp.

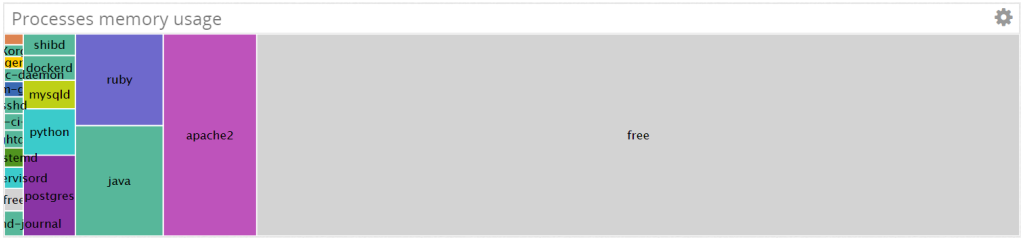
V tuto chvíli jsou navíc do Datadogu integrovány i novinky z Redmine, commity z Gitu však zatím integrované nejsou. Je to z důvodu, že Datadog nabízí

buďto integraci Gitu, jehož hlavní repozitář je právě na monitorovaném serveru, nebo externě na GitHubu. My však máme repozitář na školním GitLabu, který v současnosti jako integrace podporovaný není. Snažil jsem se integraci provést úpravou skriptů, které integrují repozitář hostovaný na monitorovaném serveru, bohužel však neúspěšně.
Níže přikládám ukázku grafu 4.8, který znázorňuje počet příchozích požadavků na webserver. Graf je ze dne, kdy byly v aplikaci psány testy - je vidět zvýšená aktivita v době spouštění testů: 9:15, 11:00, 12:45 a 16:15, což odpovídá začátku výukových hodin.



Obrázek 4.8: Sledování počtu požadavků na webserver v Datadogu během psaní testů

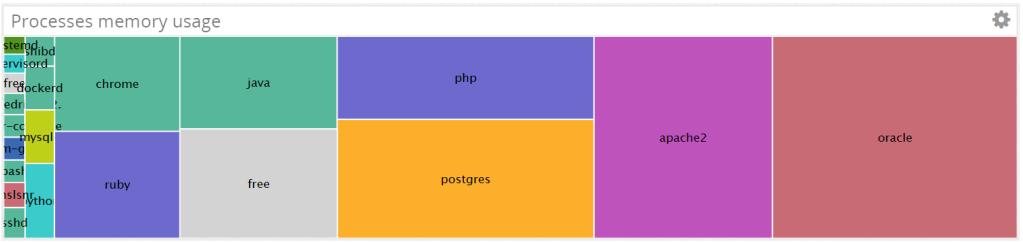
Následující graf 4.9 shrnuje průměrné využití paměti ve stejném časovém období. Je jasně vidět, že pro samotný běh DBS portálu není zatím potřeba paměť zvyšovat.



Obrázek 4.9: Sledování využití paměti v Datadogu během psaní testů

Poslední graf 4.10 znázorňuje také využití paměti, ovšem ve chvíli, kdy jsou na serveru spuštěny automatické testy z Gitlabu, o kterých jsem se již zmiňoval v sekci 4.2.2.1. Je vidět, že testy potřebují velké množství prostředků, proto

je důležité omezovat jejich dostupné zdroje, aby se neprojevily výkonnostní problémy pro koncového uživatele aplikace. Jedním z možných vylepšení do budoucna je zařízení dedikovaného *build serveru*, který by sloužil pouze pro CI a mohl být využíván na 100% pouze těmito testy.



Obrázek 4.10: Sledování využití paměti v Datadogu během automatizovaného testování nové verze aplikace (Gitlab CI)

Během nasazování Datadogu jsem také narazil na problém s konfigurací monitoringu Postgres databáze. Kontaktoval jsem proto podporu a společně jsme problém vyřešili, musel jsem však udělit datadog-agentu více přístupových práv i k ostatním databázím. Dle všeho se zdá, že se jedná o chybu v datadogu, který ignoruje nastavení zvolené databáze. Záznam komunikace je na přiloženém médiu v souboru `datadog_support.md`, pro vložení do této práce je moc dlouhý.

4.4.6.2 Google Analytics

Analytics [8] je webová služba, která nabízí přehledy o uživatelích webové stránky, zdroj jejich příchodu, navštívené stránky, používaný operační systém, prohlížeč, rozlišení obrazovky atp.
V komerční sféře jsou tato data využívána především pro zvýšení *konverzního poměru*, jelikož je možné například sledovat, jak se návštěvník eshopu dostane k finální koupi produktu, či rozhodnout, zda je vhodné investovat i do optimalizace webu pro mobilní zobrazení díky znalosti poměru počtu návštěv z mobilního zařízení.
Pro DBS portál má však nasazení Google Analytics také svůj význam, jelikož je měřena i rychlost načítání stránek nebo průchod uživatelů systémem. Prvním zmíněným můžeme detekovat pomalé stránky a případně optimalizovat rychlost jejich načítání. Pomocí průchodu uživalů systémem zase můžeme

zlepšovat rozdělení menu nebo zobrazování důležitých informací. Analýza současného *user flow* odhalila například následující skutečnost:

- V komponentě semestrálních prací je velmi často navštěvována stránka *termínů odevzdání*. Zde student získává pouze informaci, do kdy musí vyhotovit jednotlivé kontrolní body své práce. Na základě tohoto zjištění byl zadán úkol k realizaci zobrazování informací o termínech odevzdání přímo na domovské stránce studenta, kde bude přehledně vidět jak *nejbližší termín*, tak přehled všech termínů.

Kromě toho také získáváme obecné statistiky o uživatelích. Některé ze zajímavých informací získaných z Google Analytics v období 1. 2. 2017 až 26. 4. 2017:

- *61,16% připojení je z OS Linux* - toto je způsobeno jednak oblibou GNU/Linux mezi studenty FITu, ale především používáním systému ve školních učebnách.

- *57% připojení je přes ISP ČVUT* - přesně odpovídá předchozímu zjištění - většina studentů se systémem pracuje ze školy.

- *7% připojení je buďto z mobilu, nebo tabletu* - DBS portál není *vůbec* optimalizován pro mobilní zařízení. Přesto jsou u této statistiky zajímavá data:

  - *Míra okamžitého opuštění*, neboli Bounce Rate, je u mobilních zařízení 3x vyšší: 18% oproti 6% u desktopu. Stále to však znamená, že *82%* uživatelů se rozhodne i přesto portál z mobilního zařízení používat.

  - *Průměrná doba strávená na webu* je u mobilních zařízení 4 minuty, kdežto u desktopu 6 minut. Tento rozdíl je velmi malý, a poukazuje na to, že studenti opravdu portál používají i z mobilních zařízení i přes jeho absolutní neoptimalizovanost. Tato data tedy poukazují na to, že by portál do budoucna měl být vyvíjen i pro použití na menších rozlišeních.

	Mobile Device Info	Acquisition			Behavior		
		Sessions ↓	% New Sessions	New Users	Bounce Rate	Pages / Session	Avg. Session Duration
		3,143 % of Total: 7.07% (44,479)	17.91% Avg for View: 55.32% (-67.62%)	563 % of Total: 2.29% (24,608)	17.88% Avg for View: 7.51% (138.19%)	3.94 Avg for View: 9.34 (-57.80%)	00:03:53 Avg for View: 00:05:56 (-34.45%)
☐	1. Apple iPhone	**828** (26.34%)	17.15%	142 (25.22%)	14.49%	3.80	00:04:08
☐	2. Apple iPad	**205** (6.52%)	26.83%	55 (9.77%)	9.27%	4.09	00:03:42
☐	3. (not set)	**112** (3.56%)	29.46%	33 (5.86%)	18.75%	3.58	00:03:41
☐	4. Xiaomi Redmi Note 3	**108** (3.44%)	7.41%	8 (1.42%)	11.11%	5.59	00:10:27
☐	5. Huawei VNS-L21 P9 Lite	**70** (2.23%)	12.86%	9 (1.60%)	24.29%	3.89	00:04:13
☐	6. Google Nexus 5X	**68** (2.16%)	14.71%	10 (1.78%)	19.12%	4.29	00:02:53
☐	7. Huawei FRD-L09 Honor 8	**66** (2.10%)	3.03%	2 (0.36%)	15.15%	4.62	00:05:45
☐	8. Lenovo P70-A	**64** (2.04%)	6.25%	4 (0.71%)	28.12%	3.25	00:04:00
☐	9. Google Nexus 5	**56** (1.78%)	14.29%	8 (1.42%)	10.71%	3.98	00:06:21

Obrázek 4.11: Počty připojení z různých mobilních zařízení, změřeno pomocí Google Analytics

4.5 Návrhy budoucích vylepšení infrastruktury

Na závěr bych rád prezentoval některé další návrhy na zlepšení infrastuktury, které je možné v projektu dále rozvíjet. Mohou sloužit buď jako inspirace pro další studenty pracující na portálu, nebo pro mě samotného, pokud se rozhodnu na téma DBS portálu vytvořit i svou diplomovou práci.

4.5.1 Používání Datadogu

Datadog (4.4.6.1) - tedy monitorovací službu serveru - jsem zavedl během tohoto semestru. Tento nástroj však nabízí *obrovské* množství nastavitelných monitorů, upozornění atp. Cílem do budoucna by tedy mělo být další rozšiřování konfigurace Datadogu, z nichž některé návrhy dále rozvedu.

- *Propojení s Gitem*
  Na konci sekce o Datadogu jsem zmiňoval chybějící podporu integrace GitLabu do Datadogu. Při vylepšování monitoringu je jedním z cílů

zahrnout do sledovaných událostí i informaci o commitech, což by usnadnilo případné hledání chyb.

- *Nastavení monitorů upozorňujících na výkonnostní problémy*
  Datadog umožňuje konfigurovat vlastní monitory, které poté hlídají například počet příchozích spojení, využití CPU nebo jakoukoliv jinou statistiku, která je to Datadogu integrovaná. K těmto měřitelným jevům poté umožňuje nastavit limity, kterých by měly za normálního běhu dosahovat. Při překročení limitu je vyvoláno upozornění, buďto formou emailu, nebo zasláním zprávy na Slack. V současnosti jsou nastaveny pouze monitory *nedostatku operační paměti* a *nedostatku místa na disku.*

- *Statistiky přímo z aplikace*
  Datadog umožňuje i integraci přímo do běžící aplikace, formou úpravy PHP souborů. Je tak možné tvořit vlastní grafy, měřit počty zobrazení určitých stránek nebo rychlost určitých bloků kódu. Pro ještě jemnější sledování využívání aplikace je tedy možné přidat na určitá místa kódu odesílání dat do Datadogu.

4.5.2 Třetí testovací prostředí

V současnosti má aplikace jednu produkční verzi - `dbs.fit.cvut.cz` s nasazenou větví `master` a jedno „testovací“ prostředí nových verzí - `dbs2.fit.cvut.cz` s nasazenou větví `devtest`.
Tato dvě prostředí mají *sdílenou databázi*, což přináší jak výhody, tak nevýhody. Mezi výhody lze zařadit možnost používání testovací verze na ostrých datech - tester může s portálem pracovat na druhé doméně a testovat tak nové funkce. Když narazí na nepřekonatelnou chybu, tak ji nahlásí a pokračuje v práci na stabilní produkční verzi, jelikož data aplikací jsou stejná. Naopak nevýhodou je nemožnost testovat „agresivně“, právě z důvodu možného poničení produkčních dat. Kvůli sdílení databáze také nemáme v projektu implementováno automatické nasazování změn z Gitu na server - jedná se o choulostivou akci, kterou je lepší provádět manuálně (byť skriptem (4.4.4), ale manuálně spuštěným).
Cílem tohoto návrhu je tedy zajištění *třetího testovacího prostředí*, kam by se nové verze nasazovaly automaticky, a které by mělo vlastní, oddělenou databázi. Dále je před realizací vhodné zvážit:

- Které větve z Gitu budou na nové testovací prostředí nasazovány?
- Bude umožněno využívat toto prostředí i studentům z BI-SP1/BI-SP2 týmů?
- Bude se při nasazení vždy kopírovat produkční databáze, nebo bude databáze zcela nezávislá?
- Na jaké adrese bude testovací verze nasazena?
- Bude využíván Shibboleth (4.4.3), jako pro přihlášení na produkční verzi, nebo bude jednoduché přihlášení přes heslo?

Zodpovězení těchto otázek je již nad rámec mé bakalářské práce a nechávám je na případném řešiteli tohoto bodu infrastruktury.

4.5.3 Převod projektů na Gitlabu do společné skupiny

V současnosti je na školním Gitlabu využíváno hned několik repozitářů, ve kterých jsou jednotlivé součásti projektu:

- `newDBS`: hlavní aplikace naprogramovná v Nette,
- `newDBS-bug`: veřejné nahlašování chyb,
- `newDBS-env`: konfigurace Dockeru a Vagrantu - automatické testy a vývojářské prostředí,
- `newRAT`: používaný překladač relační algebry do SQL, fungující jako samostatná komponenta, s kterou portál komunikuje přes API,
- `DBSDM`: datový modeller používaný v projektu, pravidelně integrovaný do hlavního projektu, avšak vyvíjen samostatně,
- `newDBS-quiz`: nyní již nevyužívaný repozitář obsahující vstupní kvíz pro nové studenty v LS 2017.

Problém je, že tyto projekty jsou různě rozděleny mezi mne, Filipa Glazara a Pavla Kováře, což se může stát problémem po našem ukončení působnosti na škole. Cílem by tedy mělo být vytvořit novou skupinu projektů, do které budou všechny výše zmíněné projekty přesunuty. Proces to však není tak jednoduchý, jelikož změna „umístění“ repozitářů mění jejich adresu a je tedy

potřeba aktualizovat i všechny repozitáře, které mají vývojáři staženy lokálně. *Každý*, kdo s některým ze zmíněných projektů pracuje, si bude muset *všechny* své lokální repozitáře aktualizovat tak, aby odkazovaly na novou adresu. Kromě lokálních repozitářů bude také potřeba aktualizovat skripty, ve kterých se adresy repozitářů mohou také vyskytovat.

Závěr

Práce určitě splnila své předpoklady, projektové řízení celého projektu se posunulo na mnohem vyšší úroveň. Podařilo se mi nastavit postup práce v Redmine tak, aby byly jasně stanovené role vedoucích týmů a byl kladen důraz na jejich důležitost při kontrole výstupu, zanalyzoval jsem i psychologické jevy nastávající při práci s lidmi a pokusil jsem se jejich nežádoucí efekty co nejvíce eliminovat.

Studenti realizující portál v rámci předmětů BI-SP1 a BI-SP2 mají mnohem kvalitnější feedback a i nově zavedené nástroje zvyšují efektivitu práce. Informovanost napříč všemi službami využívaných v projektu se značně zlepšila díky provázání jednotlivých služeb, ať už přes nabízená API nebo pomocí vlastních úprav zdrojových kódů.

Co se infrastruktury týká, některá nasazená řešení byla *nutná* pro umožnění používání aplikace cílovým uživatelům, jiná zlepšují efektivitu vývoje. I přes množství nově nasazených služeb je u infrastruktury stále možnost ji dále vylepšovat, jednak novými nástroji, jednak zlepšením konfigurací těch současných. Tyto možnosti jsem nastínil na konci kapitoly, která se infrastrukturou zabývá.

Během práce jsem také řešil řadu akutních chyb, které se i přes veškerou naši snahu dostaly do produkční verze. Při prvotním nasazování aplikace do výuky jsem se osobně účastnil testů a zkoušek a poskytoval podporu jak učitelům, tak studentům. Při jednom testu nastala i situace, kdy jsem *během běžících testů* připravil opravu kritické chyby a nasadil ji na server, zatímco studenti pracovali na zadaných otázkách.

Při pohledu na čas, který jsem u projektu strávil, lze započítat nejprve období,

kdy jsem byl studentem BI-SP1 a BI-SP2. Za tyto dva semestry jsem strávený čas přesně zaznamenával, a jednalo se o 362 hodin. Následně jsem se přesunul do role projektového manažera, kde jsem již hodiny nezaznamenával, dle mých odhadů se však každý semestr jedná o zhruba 150 - 200 hodin. V této roli jsem již třetím semestrem, výsledný strávený čas se tedy blíží 1000 hodinám.

Zdroje

1. BECK, Kent et al. *Manifesto for Agile Software Development* [online]. 2001 (cit. 2017-03-15). Dostupné z: `http://agilemanifesto.org/`.
2. BLANCHARD, Kenneth. *Minutový manažer*. Praha: Pragma, 1993. ISBN 80-85213-29-X.
3. CARIAS, Karen. *Simple words for a GitLab Newbie* [online] (cit. 2017-04-26). Dostupné z: `https://about.gitlab.com/2015/05/18/simple-words-for-a-gitlab-newbie/`.
4. DATADOG. *Modern monitoring & analytics* [online] (cit. 2017-04-22). Dostupné z: `https://www.datadoghq.com`.
5. DIGICERT INC. *What Is SSL (Secure Sockets Layer)? | DigiCert.com* [online] (cit. 2017-04-21). Dostupné z: `https://www.digicert.com/ssl.htm`.
6. FEDOR, Tomáš. *ER Diagrams Web Component II.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.
7. GLAZAR, Filip. *Systém pro podporu BI-DBS - semestrální práce*. Bakalářská práce. Czech Technical University in Prague, Faculty of Information Technology, 2016.
8. GOOGLE. *Oficiální webové stránky Google Analytics – webová analýza a vytváření přehledů – Google Analytics* [online] (cit. 2017-04-22). Dostupné z: `https://www.google.com/analytics/`.
9. HRONČOK, Miroslav. *Diplomová práce* [online]. 2016 (cit. 2017-03-05). Dostupné z: `https://github.com/hroncok/diplomka`.

10. HRONČOK, Miroslav. *FIT ČVUT thesis tips* [online]. 2016 (cit. 2017-03-05). Dostupné z: https://github.com/hroncok/fit-thesis-tips.

11. IDRIVE INC. *Backup and restore Linux servers using Command Line Utility* [online] (cit. 2017-04-22). Dostupné z: https://www.idrive.com/cmd_steps.

12. KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.

13. KOVÁŘ, Pavel. *Automatizované testování webového portálu dbs.fit.cvut.cz*. Bakalářská práce. Czech Technical University in Prague, Faculty of Information Technology, 2017.

14. LACAN, Olivier. *Keep a Changelog* [online] (cit. 2017-04-20). Dostupné z: http://keepachangelog.com/en/0.3.0/.

15. LANG, Jean-Philippe. *Overview - Redmine* [online]. 2006 (cit. 2017-03-05). Dostupné z: http://www.redmine.org.

16. MASLOW, Abraham. Developmental sequence in small groups. *Psychological Review*. 1942, roč. 50, č. 4. ISSN 370-396.

17. PROCHÁZKA, Marek. *Skupinová dynamika, Týmová spolupráce* [online]. 2015 (cit. 2017-04-13). Dostupné z: https://ekonom.feld.cvut.cz/cs/student/predmety/manazerska-psychologie.

18. RASMUSSON, Jonathan. *The agile Samurai: how agile masters deliver great software*. Raleigh: Pragmatic Bookshelf, 2010. ISBN 978-1-934356-58-6.

19. ŘEHÁČEK, Petr. *Projektové řízení podle PMI*. Praha: Ekopress, 2013. ISBN 978-80-86929-90-3.

20. SCHWALBE, Kathy. *Řízení projektů v IT: kompletní průvodce*. Brno: Computer Press, 2011. ISBN 978-80-251-2882-4.

21. SLACK. *Slack: Where work happens* [online] (cit. 2017-04-16). Dostupné z: https://slack.com/is.

22. SOFTWARE FREEDOM CONSERVANCY. *Git* [online] (cit. 2017-03-05). Dostupné z: https://git-scm.com/.

23. SOFTWARE FREEDOM CONSERVANCY. *Git - Git Hooks* [online] (cit. 2017-04-30). Dostupné z: https://git-scm.com/book/gr/v2/Customizing-Git-Git-Hooks.

24. SOFTWARE FREEDOM CONSERVANCY. *Git - Tagging* [online] (cit. 2017-04-30). Dostupné z: `https://git-scm.com/book/en/v2/Git-Basics-Tagging`.

25. ŠTRUPL, Petr. *Shibboleth pro správce - ČVUT IT návody - VICwiki* [online] (cit. 2017-04-21). Dostupné z: `https://wiki.cvut.cz/confluence/pages/viewpage.action?pageId=27822483`.

26. THE SHIBBOLETH CONSORTIUM. *Shibboleth* [online] (cit. 2017-04-21). Dostupné z: `https://shibboleth.net/`.

27. TRLICA, Adam. *Jaké jsou typy DNS záznamů? - GIGASERVER.CZ / Znalostní báze* [online] (cit. 2017-04-21). Dostupné z: `https://kb.gigaserver.cz/jake-jsou-typy-dns-zaznamu`.

28. TUCKMAN, Bruce. Developmental sequence in small groups. *Psychological Bulletin*. 1965, roč. 63, č. 6. ISSN 0033-2909.

29. WYSOCKI, Robert K. *Effective project management: traditional, agile, extreme*. Indianapolis: Wiley, 2012. ISBN 978-1-118-01619-0.

Dodatek A

Seznam použitých zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BI-DBS	Databázové systémy
BI-SI1	Softwarové inženýrství I
BI-SP1	Softwarový týmový projekt 1
BI-SP2	Softwarový týmový projekt 2
BP	Bakalářská práce
CI	Continuous integration
CMS	Content management system
CNAME	Canonical name
CPU	Central Processing Unit
ČVUT	České vysoké učení technické
DBS	Databázové systémy
DML	Data manipulation language
DP	Diplomová práce
FDD	Feature driven development
FEL	Fakulta elektrotechnická
FIT	Fakulta informačních technologií
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
ISP	Internet service provider
KOS	Komponenta studium

LS	letní semestr
MVP	Model-View-Presenter
OBS	Organization Breakdown Structure
OS	Operační systém
PHP	PHP: Hypertext Preprocessor
PMI	Project Management Institute
RA	Relational algebra / Relační algebra
SP	Softwarový projekt
SQL	Structured query language
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign-On
TDD	Test driven development
UI	User interface
UX	User experience
XML	Extensible Markup Language
XP	Extrémní programování
XSLT	Extensible Stylesheet Language Transformations
ZS	zimní semestr

Dodatek B

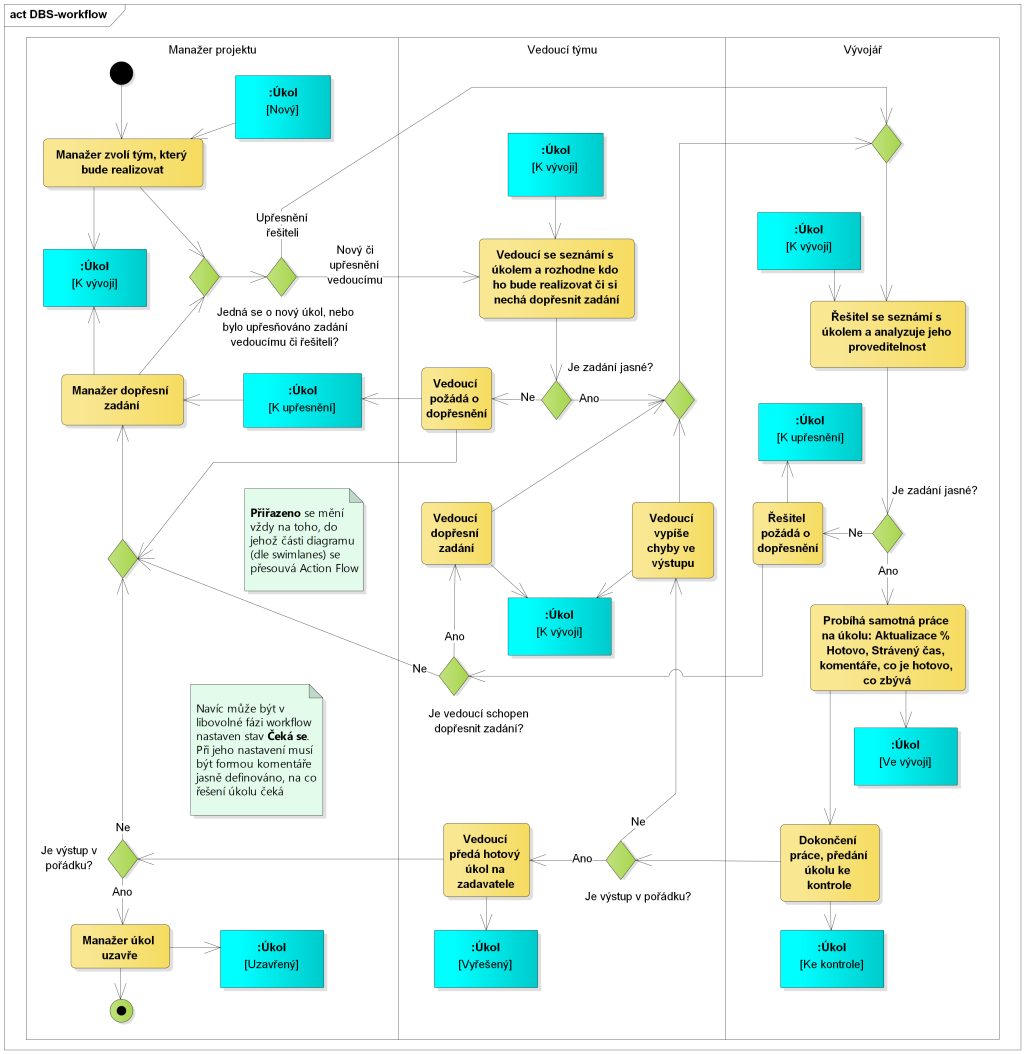
Slovník pojmů

Continuous integration	Systém automatického sestavování a testování změn zdrojového kódu.
commit message	Krátká zpráva připojená ke *commitu*, která shrnuje, co commit řeší.
commit	Aplikování či přijmutí nových změn zdrojového kódu. V tomto textu používáno především v souvislosti s *Gitem*.
konverzní poměr	Pravděpodobnost, že se z návštěvníka stane zákazník.
cron	Služba operačního systému, která umožňuje v určitý čas nebo v určitých intervalech spouštět nějaký příkaz nebo skript.
Data manipulation language	Soubor klíčových slov se stanovenou syntaxí, podobný programovacímu jazyku, určený pro manipulaci s daty uloženými v databázi. Nejznámější DML je součástí SQL.
Feature	Vlastnost, funkce, rys.
Framework	Softwarová struktura, která slouží jako podpora pro pohodlnější programování. Může obsahovat podpůrné funkce, knihovny či nástroje pro efektivnější, bezpečnější a pohodlnější vývoj softwaru.

Gitlab	Nástavba nad Gitem ulehčující používání repozitáře ve webovém rozhraní.
Git	Distribuovaný systém pro správu verzí vhodný především pro soubory s textovým obsahem.
Grido	PHP komponenta umožňující jednoduchý výpis dat do tabulky, jejich řazení, filtrování, stránkování a hromadné akce.
merge	Sloučení dvou vývojových větví v *Gitu*.
Model-View-Presenter	*Návrhový vzor* softwaru, který klade důraz na oddělení jednotlivých vrstev aplikace. Využívá jej například Nette Framework.
Nette	Webový framework pro PHP.
pull	Stažení nových změn větve ze sdíleného repozitáře do lokální složky.
push	Odeslání *commitu* do sdíleného repozitáře, například Gitlabu.
SQLite	Jednoduchý databázový stroj, který uchovává celou databázi pouze v jednom souboru.
Slack	Komunikační nástroj pro týmy.

Dodatek C

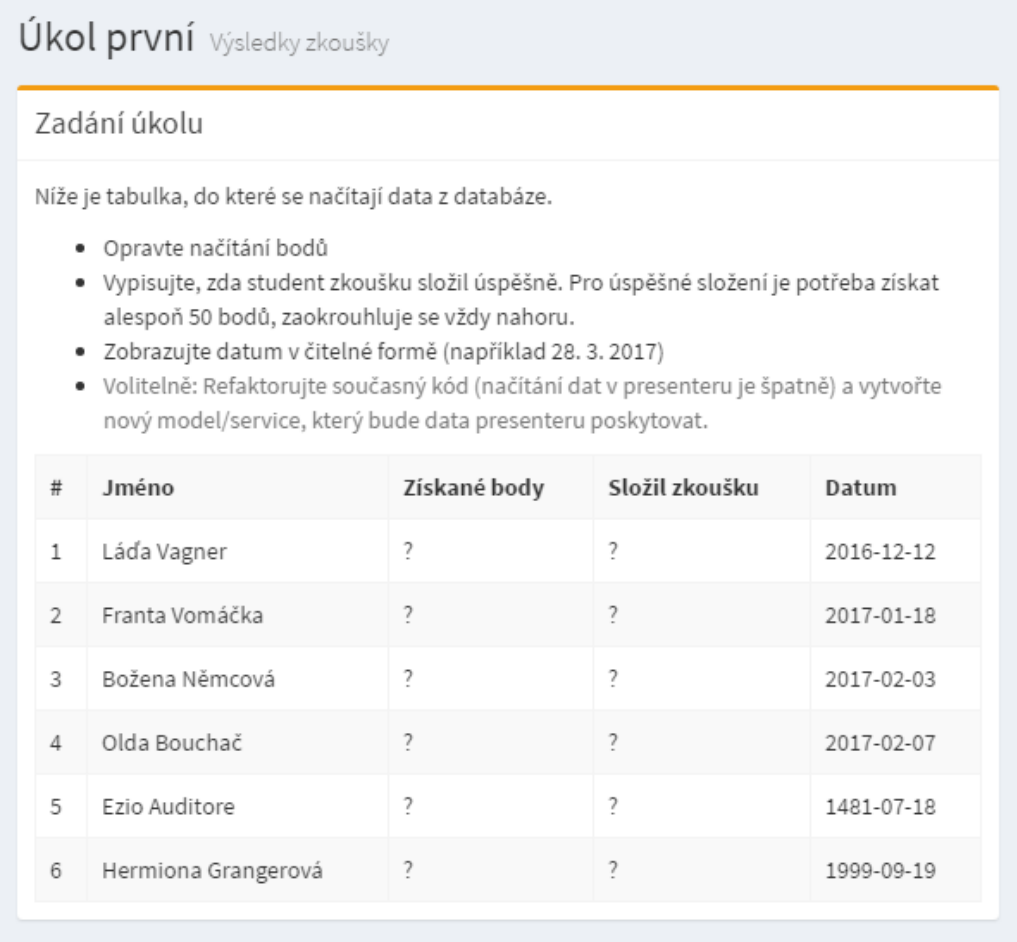
Diagramy



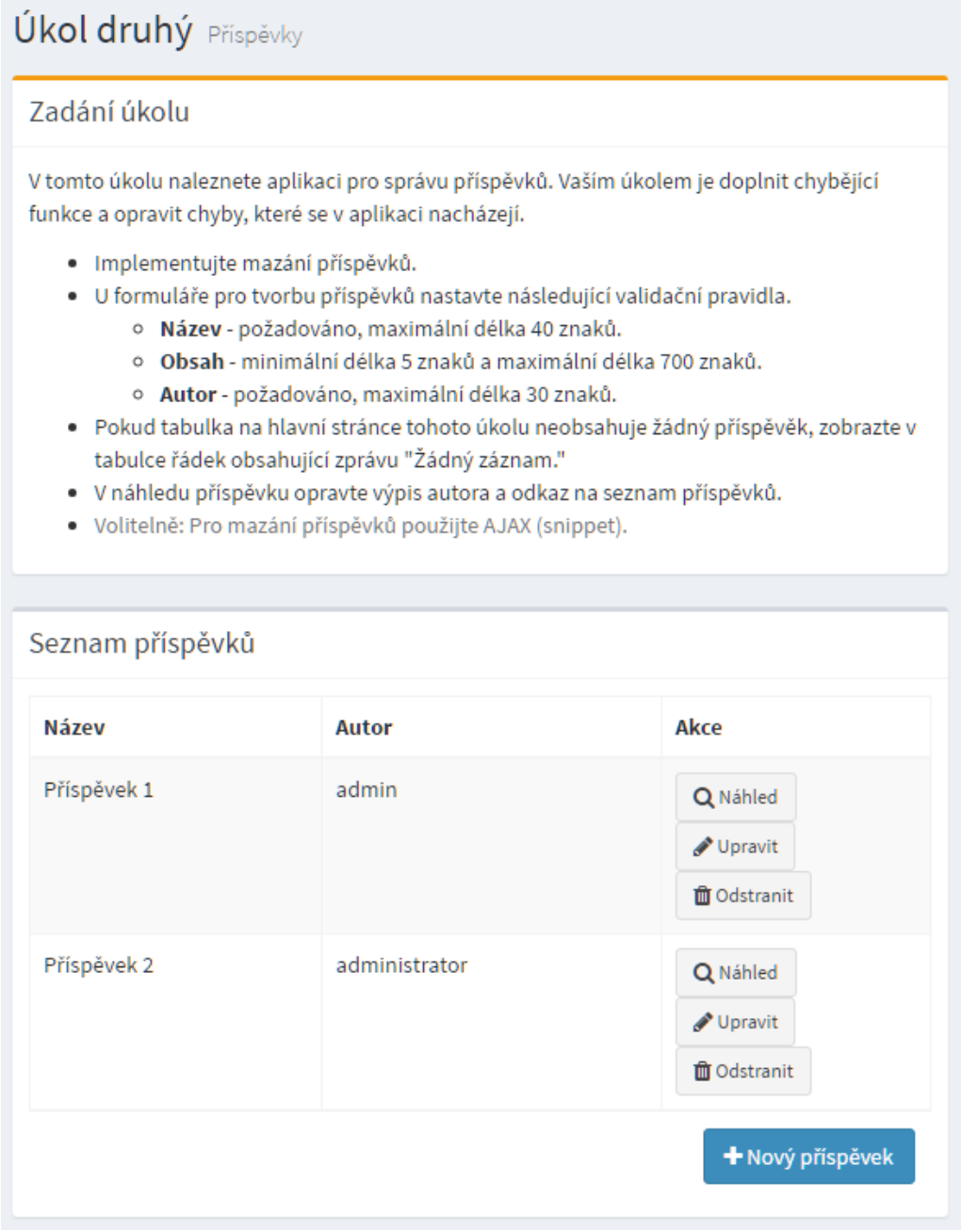
Obrázek C.1: Activity diagram průběhu práce v Redmine

Dodatek D

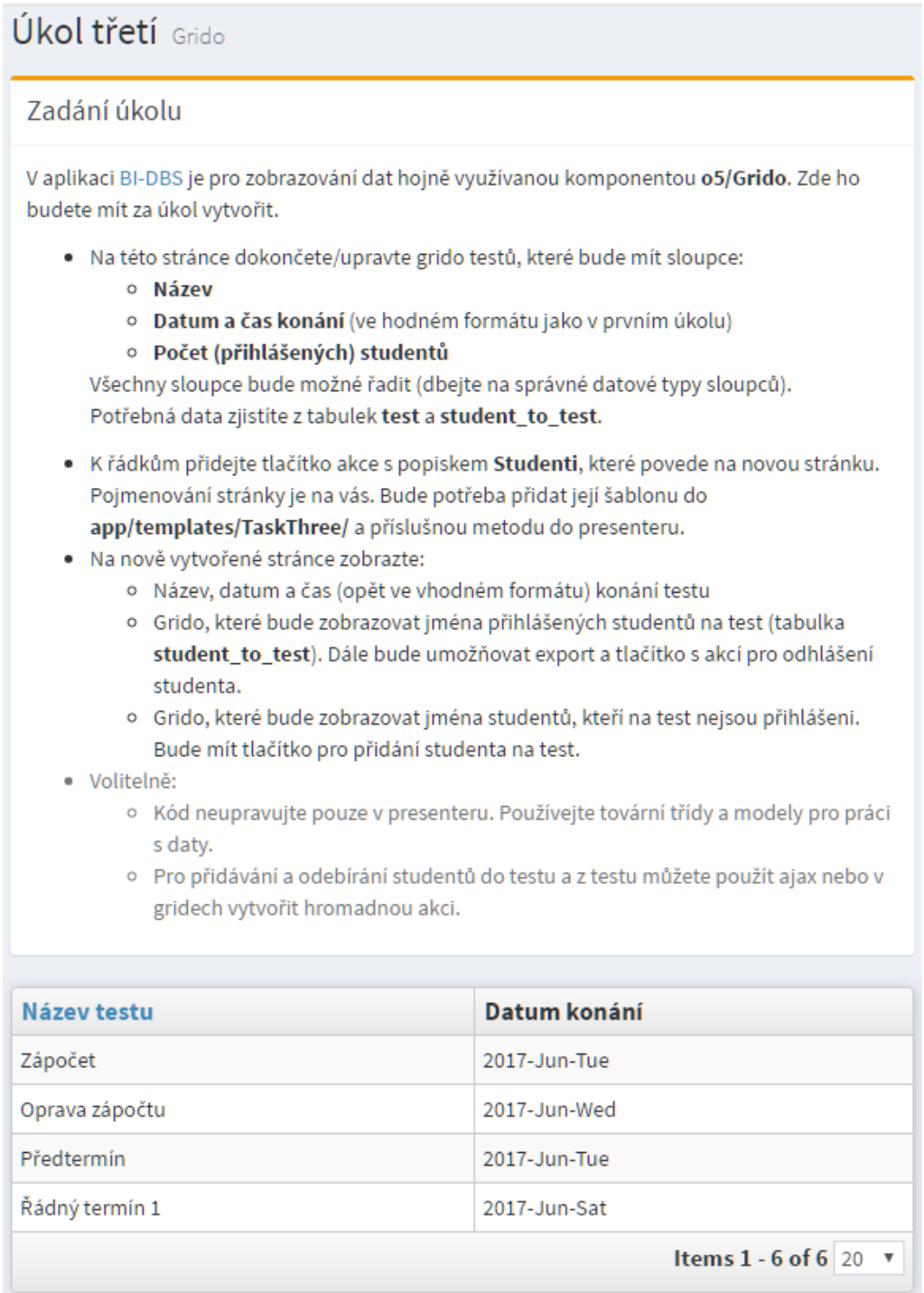
Vstupní kvíz



Obrázek D.1: První úloha vstupního kvízu DBS projektu. Autor: Oldřich Malec



Obrázek D.2: Druhá úloha vstupního kvízu DBS projektu. Autor: Pavel Kovář



Úkol třetí Grido

Zadání úkolu

V aplikaci BI-DBS je pro zobrazování dat hojně využívanou komponentou **o5/Grido**. Zde ho budete mít za úkol vytvořit.

- Na této stránce dokončete/upravte grido testů, které bude mít sloupce:
  - **Název**
  - **Datum a čas konání** (ve hodném formátu jako v prvním úkolu)
  - **Počet (přihlášených) studentů**

  Všechny sloupce bude možné řadit (dbejte na správné datové typy sloupců). Potřebná data zjistíte z tabulek **test** a **student_to_test**.
- K řádkům přidejte tlačítko akce s popiskem **Studenti**, které povede na novou stránku. Pojmenování stránky je na vás. Bude potřeba přidat její šablonu do **app/templates/TaskThree/** a příslušnou metodu do presenteru.
- Na nově vytvořené stránce zobrazte:
  - Název, datum a čas (opět ve vhodném formátu) konání testu
  - Grido, které bude zobrazovat jména přihlášených studentů na test (tabulka **student_to_test**). Dále bude umožňovat export a tlačítko s akcí pro odhlášení studenta.
  - Grido, které bude zobrazovat jména studentů, kteří na test nejsou přihlášeni. Bude mít tlačítko pro přidání studenta na test.
- Volitelně:
  - Kód neupravujte pouze v presenteru. Používejte tovární třídy a modely pro práci s daty.
  - Pro přidávání a odebírání studentů do testu a z testu můžete použít ajax nebo v gridech vytvořit hromadnou akci.

Název testu	Datum konání
Zápočet	2017-Jun-Tue
Oprava zápočtu	2017-Jun-Wed
Předtermín	2017-Jun-Tue
Řádný termín 1	2017-Jun-Sat

Items 1 - 6 of 6 20

Obrázek D.3: Třetí úloha vstupního kvízu DBS projektu. Autor: Milan Vlasák

Dodatek E

Hodnocení týmů

Hodnocení SP1 poslední aktualizace: 04.05.2017

	Maximální body		Team 1										Team 2									
	SP/SI	Redmine	úkoly	vedoucí	bonus		součet		Marek E.	Anna D.	David R.	Filip M.	Jan P.	úkoly	vedoucí	bonus		součet		Jakub D.	Juraj K.	Lukáš B.	Marek P.
1-2. týden	5	10	x	x	x		x		10	9,75	9	9,5	9,75	x	x	x		x		10,25	9,75	8	10
3-4. týden	9	18	49,25	8	0	→	57,25	→	0	0	0	0	0	57,3	8	0	→	65,3	→	8	-3	-4	-1
									12,7	12,7	12,7	12,7	12,7							24,3	13,3	12,3	15,3
5-6. týden	9	18	48	8	3	→	59	→	3	3	-2	-1	-3	68,75	8	3	→	79,75	→	4	2	-4	-2
									16,1	16,1	11,1	12,1	10,1							23,9	21,9	15,9	17,9
7-8. týden	9	18	29,5	6	1	→	36,5	→	2	0	0	0	-2	23	6	1	→	30	→	0	0	0	0
									10,1	8,1	8,1	8,1	6,1							7,5	7,5	7,5	7,5
9-10. týden	9	18	3	4	0	→	7	→						4,5	0	0	→	4,5	→	0	0	0	0
									1,6	1,6	1,6	1,6	1,6							1,1	1,1	1,1	1,1
11-12. týden	9	18	30,75			→	30,75	→						6			→	6	→				
									6,8	6,8	6,8	6,8	6,8							1,5	1,5	1,5	1,5
13+zkouškové	0	0				→	0	→									→	0	→				
	50	**100**																					
Souhrn Redmine bodů									57,3	55,05	49,3	50,8	47,05							68,55	55,05	46,3	53,3
Souhrn SI1/SP1 bodů									28,65	27,525	24,65	25,4	23,525							34,275	27,525	23,15	26,65

Obrázek E.1: Tabulka hodnocení týmů

Dodatek F

Obsah přiloženého média

- `README.md` ..... stručný popis obsahu média
- `src`
  - `diagram.EAP` ..... zdrojový soubor Activity diagramu práce v Redmine
  - `thesis` ..... zdrojové soubory práce ve formátu tex, pdf, png a svg
- `datadog_support.md` ..... záznam komunikace s podporou Datadogu
- `BP_Malec_Oldrich_2017.pdf` ..... text práce ve formátu PDF