



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Řízení projektu a infrastruktury portálu pro podporu výuky p edm tu 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 SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

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

Oldřich Malec

Vedoucí práce: Ing. Jiří Hunka

27. dubna 2017

Poděkování

TODO poděkování

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 27. dubna 2017

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2017 Oldřich Malec. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

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⟩](https://gitlab.fit.cvut.cz/malecold/bc-thesis).

Abstrakt

TODO

Klíčová slova TODO

Abstract

TODO

Keywords TODO

Obsah

Úvod	3
1 Co je to DBS portál	5
1.1 Historie výuky BI-DBS	5
1.2 Vznik DBS portálu	6
1.3 Historie řízení vývoje portálu	7
2 Metodiky vývoje softwaru	9
2.1 Tradiční metody	9
2.2 Agilní metody	11
2.3 Zvolené řešení pro DBS projekt	17
3 Řízení lidských zdrojů	21
3.1 Vytvoření plánu lidských zdrojů	23
3.2 Zajištění projektového týmu	24
3.3 Rozvoj projektového týmu	26
3.4 Řízení projektového týmu	31
4 Infrastruktura DBS projektu	35
4.1 Issue tracker	35
4.2 Verzování a testování	42
4.3 Komunikace projektového týmu	46
4.4 Správa aplikace	48
4.5 Návrhy budoucích vylepšení infrastruktury	57

Závěr	59
Zdroje	61
A Seznam použitých zkratk	65
B Slovník pojmů	67
C Diagramy	69
D Vstupní kvíz	71
E Hodnocení týmů	75
F Obsah přiloženého média	77

Seznam tabulek

Seznam obrázků

3.1	Maslowova pyramida potřeb	22
3.2	Organization Breakdown Structure používaná v DBS projektu	24
4.1	Stav před a po nasazení úpravy kódu Redmine pro zobrazování odkazu na Gitlab	39
4.2	Používání branchí v gitu při vývoji DBS portálu	44
C.1	Activity diagram průběhu práce v Redmine	70
D.1	První úloha vstupního kvízu DBS projektu. Autor: Oldřich Malec . . .	72
D.2	Druhá úloha vstupního kvízu DBS projektu. Autor: Pavel Kovář . . .	73
D.3	Třetí úloha vstupního kvízu DBS projektu. Autor: Milan Vlasák . . .	74
E.1	Tabulka hodnocení týmů TODO aktualizovat	76

Úvod

TODO úvod TODO číslování až od Úvodu?

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 portálem 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 vymyslí dotazy. Typická semestrální práce obsahuje ve své finální podobě alespoň 25 dotazů do a pokrývá všechny standardní klauzule DML, jednak z jazyku *relační algebry* a také 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 výuku jsou poměrně náročné na korekturu vyučujícím. Typicky nebylo možné zajistit, aby vyučující zkontroloval každý jednotlivý dotaz, který student vytvořil ve své semestrální práci. Oprava testů, které se psaly na papír poté trvala zbytečně dlouho a prakticky nebylo možné ji automatizovat. Z tohoto důvodu 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 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 také 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 používán dodnes. Jelikož SP1 a navazující 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 - již byl 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 - spíše než interní vedoucí - zastával roli vedoucího týmu. Honza 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 a výstup poté i kontroloval.

Po dokončení předmětů BI-SP1 a následně BI-SP2 jsem u projektu zůstal a začal jsem se naplno věnovat jeho projektovému 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.

Metodiky vývoje softwaru

V této kapitole čtenáře seznámím s jednotlivými metodikami vývoje softwaru a zaměříme se také na možnosti jejich použití při vývoji DBS portálu, který byl popisován v kapitole 1.

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 zbytečné starat se o *nějakou* vývojovou metodiku. Jakmile však chceme spojit práci více lidí, požadujeme aby výsledný produkt byl hotový do určitého termínu, splňoval všechny předem stanovené požadavky a byl udržovatelný i po jeho nasazení, začínáme mít potřebu zavádět určitá pravidla či předpisy, které nám pomohou všechny požadavky 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 metody

2.1.1 Vodopádový model

Jak zmiňuje Kadlec [11], 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 možná zpracovat až v rámci údržby, která vývoj vždy vrátí do jedné z předchozích etap a následně musí proběhnout i etapy následující.

- Definice problému
- Specifikace požadavků
- Návrh
- Implementace
- Integrace, 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 zmíněné výše u vodopádového modelu (2.1.1), DBS projekt potřebuje být vyvíjen pomocí skutečně iterativní metody, která bude v jednotlivých iteracích přidávat funkcionality.

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

2.2 Agilní metody

Agilní metody řízení vznikly především z důvodu neustále se měnících požadavků na software a rychlejší reakci na požadované změny. Tradiční metody 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 metody zcela nevhodným modelem vývoje.

Rozdíl mezi tradiční a agilní metodikou lze popsat také jejich přístupem k *Funkcionalitě, Času a Zdrojům* (Kadlec [11]). Zatímco u tradičních metod je funkcionalita stanovena na začátku a musí být dosažena pomocí *nějakého* času a *nějakého* množství zdrojů, u agilních metod je to přesně naopak. Typicky je stanoven termín a dostupné zdroje a funkcionalita *nějaká* bude.

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	Ward Cunningham	Andrew Hunt
Mike Beedle	Martin Fowler	Ron Jeffries
Arie van Bennekum	James Grenning	Jon Kern
Alistair Cockburn	Jim Highsmith	Brian Marick

Robert C. Martin	Jeff Sutherland
Steve Mellor	Dave Thomas
Ken Schwaber	

© 2001, výše zmínění autoři Toto prohlášení může být volně kopírováno v jakékoli formě, ale pouze v plném rozsahu včetně této poznámky.

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, [24]), 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 osvědčil *návrh*, budeme neustále navrhovat a vylepšovat specifikaci ...
- Pokud se osvědčilo *testování*, budeme neustále testovat ...
- Narozdíl od tradičních metodik, které najednou dodají velký balík nových funkcionalit, bude XP dodávat i ty nejmenší funkční kousky. Toto vede na *krátké iterace*.

Tyto postupy přivedené do extrémů jsou reprezentovány v *12 základních postupech XP*, se kterými se seznámíme níže.

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 a i zákazník. Typicky jedna karta 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, tak 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. Žádné přidání vlastností nechce do návrhu zahrnovat, 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 píše 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.

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*: Součástí 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 zdrojových 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 ve stoje 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 mezi třemi až šesti členy.

Použití v DBS projektu Pro potřeby DBS projektu je tedy Scrum také nevhodný, především jelikož 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 měl vždy být fungující software, který je možné představit zákazníkovi.

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

- *měřitelnost*: 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 více 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 software 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 dané třídě nejvíce rozumí určitý programátor. Ti 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 funkcionálně 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 poté pouze volí nové vlastnosti k implementaci a integruje hotové vlastnosti do výsledné aplikace.

Použití v DBS projektu Výše popsáný 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 samotným zdrojovým kódem aplikace. Při přidávání funkcionality do systému je tedy nejprve napsán test, který bude funkcionality testovat. Požadovaná funkcionality 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, 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 funkční.

Na předchozích stránkách jsme si představili *některé* vývojové metodiky. Především těch agilních však existuje nepřeberné množství, a jak říká například Kadlec [11] nebo Rasmusson [16], každý projekt je jedinečný a určitě není nejlepší možností striktně dodržovat jednu z vývojových metodik. Klíčem je spíše skloubení dostupných metodik k vytvoření vlastní, pro daný projekt nejefektivnějšího přístupu k vývoji. Vedoucí týmu či projektový manažer by neměl *slepe* 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 vyvíjen pomocí metody 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 (více 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. Důvodu 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 pole jako *datum*, *stav*, *priorita* či *průběžné poznámky plnění*.

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

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 *pushi* do *Gitlabu* se automaticky spustí Continuous integration.

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 metody vývoje pro DBS projekt. Především rozdělení na jednotlivé *featurey* a role *hlavního programátora*:

Featurey Ve FDD je vývoj hnán kupředu především seznamem vlastností, které má finální podoba mít. Toho využívá i DBS projekt, u kterého takový seznam existuje a z jedné strany se stále rozšiřuje novými požadavky, na straně druhé je zpracováván programátory. Jednotlivé vlastnosti jsou měřitelné, v případě jejich nesrozumitelnosti jsou programátorovi dopřesněny 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.

Řízení lidských zdrojů

Řízení lidských zdrojů je jednou z neobtížnějších 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 ??.

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 poskytneme určitý vstup a podle jeho množství můžeme 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 poté 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 právě jídlo či pití. V DBS projektu



Obrázek 3.1: Maslowova pyramida potřeb

jsou namísto finančně studenti hodnoceni známkou z předmětu, který je pro jejich studijní obor povinný. Jelikož však *vystudování vysoké školy* obecně je spíše potřebou *úcty, uznání* či *seberealizace*, jedná se o vyšší patra pyramidy. Maslow 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 zajištěné tyto „nižší potřeby“, dostávají se do popředí právě ty, které může DBS projekt nabídnout. Toto otevírá 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 (viz kapitola ??), 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, navíc 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 [18] 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

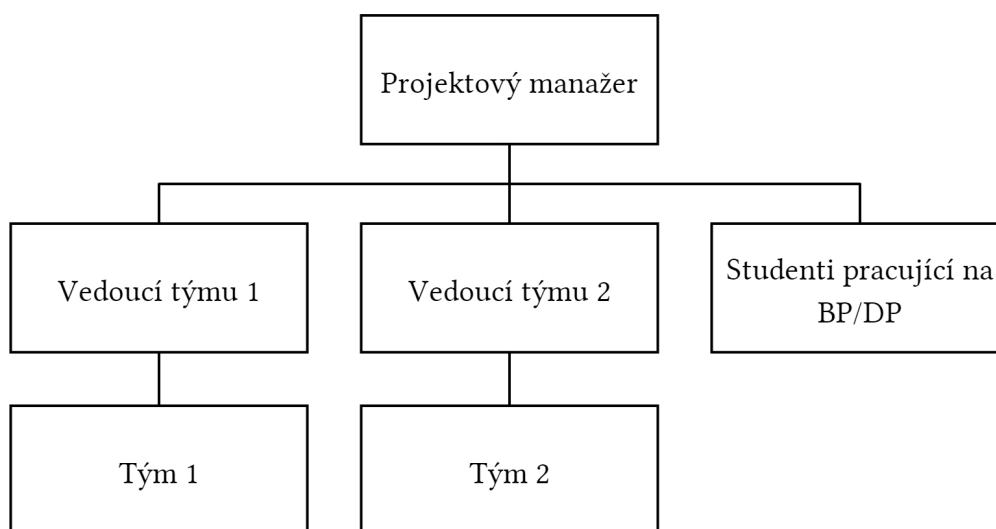
V následujících sekcích si jednotlivé procesy rozebereme podrobněji a zaměříme se i na jejich roli v DBS projektu.

3.1 Vytvoření plánu lidských zdrojů

Cílem tohoto procesu je vytvořit plán, jaké *typy* lidí potřebujem 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 dále se dělí například na jednotlivé části - programátoři, testéři, 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) a měla by být vytvořena na začátku projektu a podle ní 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, jak by to být mělo. Dělení na týmy fungovalo již od začátku, vedoucí týmu však často nezvládal dodržovat svou práci (více v úvodní kapitole 1.3). 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, kterou



Obrázek 3.2: Organization Breakdown Structure používaná v DBS projektu

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.

3.2 Zajištění projektového týmu

Tento proces se primárně stará o *příjem nových pracovníků, řízení jejich vytížení a vyrovnaní*. 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, kde je sledována práce přiřazená zaměstnanci a v případě, že je po něm požadování 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 na větší pohodlí zaměstnance a také snadnější 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, a také 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, a to *lov pracovníků*. Zároveň jsme však po nových studentech chtěli, aby měli buďto zkušenost s použitými technologiemi, nebo měli předpoklad k rychlému naučení těchto - pro někoho nových - technologií. Z toho důvodu jsme připravili *vstupní kvíz* (viz níže podsektce 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 web-server 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 nábořem, který probíhá neustále, je nábor studentů, kteří budou na projektu pracovat v rámci své bakalářské či diplomové práce. 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. Často také svou práci spíše než se mnou konzultují s Jiřím Hunkou - který vede nejen realizační týmy SP1/SP2 ale také právě bakalářské či diplomové práce.

Kromě nabírání nových lidí patří do tohoto procesu také řízení jejich vytížení a vyvažování přidělené práce. Jak vyplývá z obrázku Organization Breakdown Structure používaná v DBS projektu, 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 kompetenci 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* 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 na ukončení celého projektu nebo přeskupení členů týmů - pokud to povaha projektu dovoluje. Poté nastává opět první fáze *forming* s novým členěním týmu. Druhou doplňující fází je *rozpuštění týmu*, které 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 na různých kurzech, ať už s lektorem nebo jako e-learning. Tyto kurzy by měly být vhodně plánovány s ohledem na nadcházející práci, která bude zaměstnancům zadávána. V opačném případě se bude míjet účinkem a ponese 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 jak finanční, tak například ve formě delší dovolené atp. Pro manažera je důležité správně rozlišovat práci navíc a kladně ji hodnotit, aby byli pracovníci dostatečně motivováni, ale nikoliv aby tyto činnosti prováděli 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 semestru kdy vzniká tato práce jsou rozdíly mezi týmy velmi dobře sledovatelné, a to díky jejich rozdílnému 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* narozdíl 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 tak bylo nejvíce práce ve druhém týmu vykazováno právě jejich vedoucím, který sice opravdu zkušený byl a jeho výstup byl kvalitní, nezvládl ale dostat svůj tým do optimálního 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, tak 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 zezáčátku byl v rozpacích a jeho výstupy byly slabší, než týmu druhého, postupem času se díky průchodu jednotlivými fázemi dostává 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.

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 technologie - 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 ptát na otázky, které by pro ně byly zajímavé. Z toho důvodu jsem se školení také zúčastnil, přestože v roli lektora již byl Pavel Kovář a Milan Vlasák, kteří oba v současnosti pracují 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 ukázat přátelskou atmosféru - možnost zeptat se na cokoli - a také „rozbít“ *groupthink* a *konformitu s většinou*, jak je definoval Procházka [15] 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 k 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, je 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, jednak z důvodu „aby nebyl divný“, aby si ostatní „něco nemysleli“ atp. Průzkumy [15] 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 de odvíjí od výsledků, kterých student v našem 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. Proces vypracování úkolu popisuje Activity diagram průběhu práce v Redmine dostupný v příloze C.1. Výsledné hodnocení úkolů je definováno v tabulce 3.1.
- *Nahlášení nového úkolu*: Studenti jsou vedeni k vlastní iniciativě: Když přijdou s novým úkolem, který vede ke zlepšení výsledné aplikace

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

nebo opravuje závažnou, dosud nenahlášenou chybu, je jeho tým kladně ohodnocen.

- *Práce vedoucího týmu:* Jelikož vedoucí týmu má za úkol nejen sám pracovat na úkolech, ale i rozdělovat práci, kontrolovat výstupy jeho 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 rozdělovány. 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 tedy některý člen týmu má 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 konflikty 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: zařadit, 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 otevření jakýmkoliv návrhům včetně kritiky vedení. Ze zkušeností vyplývá, že toto je 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 vidí, kolik bodů získávají ostatní členové. Toto 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 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 *rozčillovalo* její používání.

Tento cíl však není efektivně realizovatelný, protože předmět, na který systém cílí, mají již tito studenti hotový 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.

Infrastruktura DBS projektu

Z předchozích kapitol již víme, jak efektivně vyvíjet software a jak řídit lidské zdroje. V poslední kapitole mé 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 mou 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 úkoly 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 využíván 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. - *představení Redmine na oficiálním webu [14]*.

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*:

Projekty v Redmine Redmine umožňuje na jedné instalaci provozovat několik oddělených (a nebo hierarchicky uspořádaných) projektů. V rukou hlavního administrátora (kterého u DBS projektu zastávám také já) poté je možnost umožnit určitým uživatelům či rolím přístup k různým projektům a také 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 zapsáni 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, team2 a nebo nejsou zatím přiděleny k realizaci. Také vedoucí týmů mají snazší práci s přehledem práce jejich vlastního týmu, mohou si zapnout emailové notifikace pouze pro změny v projektu, který patří jejich týmu atp.

4.1.1.1 Shromažďování úkolů k řešení

Během práci na projektu často vyvstávají nové požadavky či se objevují nové chyby. Tyto 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 a 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 *Nizká*, *Vysoká*, *Urgentní* a *Okamžitá*

4.1.1.2 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 4.1.1.3).

Průběh práce je také možné sledovat na *gitu*, respektive *Gitlabu*, kterým se více věnují samostatných sekcích 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 jednu z guidelines pro psaní commitů, kterým se blíže věnuje sekce 4.2.1. 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 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 4.1.



Obrázek 4.1: Stav před a po nasazení úpravy kódu Redmine pro zobrazování odkazu na Gitlab

Dokončení úkolu Když je úkol dokončen, je dle workflow (příloha C.1) přiřazen zpět na manažera, který zhodnotí kvalitu výstupu a vyplní pole *Získané body*. Množství získaných bodů se odvíjí od předem stanovené bodové dotace za daný úkol a kvality jeho řešení definované v tabulce 3.1.

4.1.1.3 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 měřídlem výkonu

uživatelé. 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.

4.1.1.4 Wiki

Redmine poskytuje také možnost tvorby *knowledge base* či *Wiki*, kterou využíváme především u newDBS projektu. Informace zde shromažďované slouží jak pro podporu nových studentů, tak jako dokumentace aplikace, jsou zde návody jak spouštět testy atp. Struktura Wiki v současnosti vypadá následovně:

- 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*

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, jak nejen aplikace samotná ale i její vývoj prochází inovacemi.

Je důležité na Wiki zaznamenávat i dokumentaci různých procesů, které většina uživatelů využívá pouze uživatelsky a nikoliv administrativně, a to především z důvodu, aby byl *kdokoliv v projektu nahraditelný*. Kvalitní dokumentaci 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 na email 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>.

4.2 Verzování a testování

Prakticky jakýkoliv projekt, na kterém pracuje více lidí nebo trvá více než několik hodin je potřeba *verzovat*. Jedním z nejpoužívanějších nástrojů pro verzování softwaru a jiných textově-založených projektů (jako je třeba tato práce psaná v \LaTeX) 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.

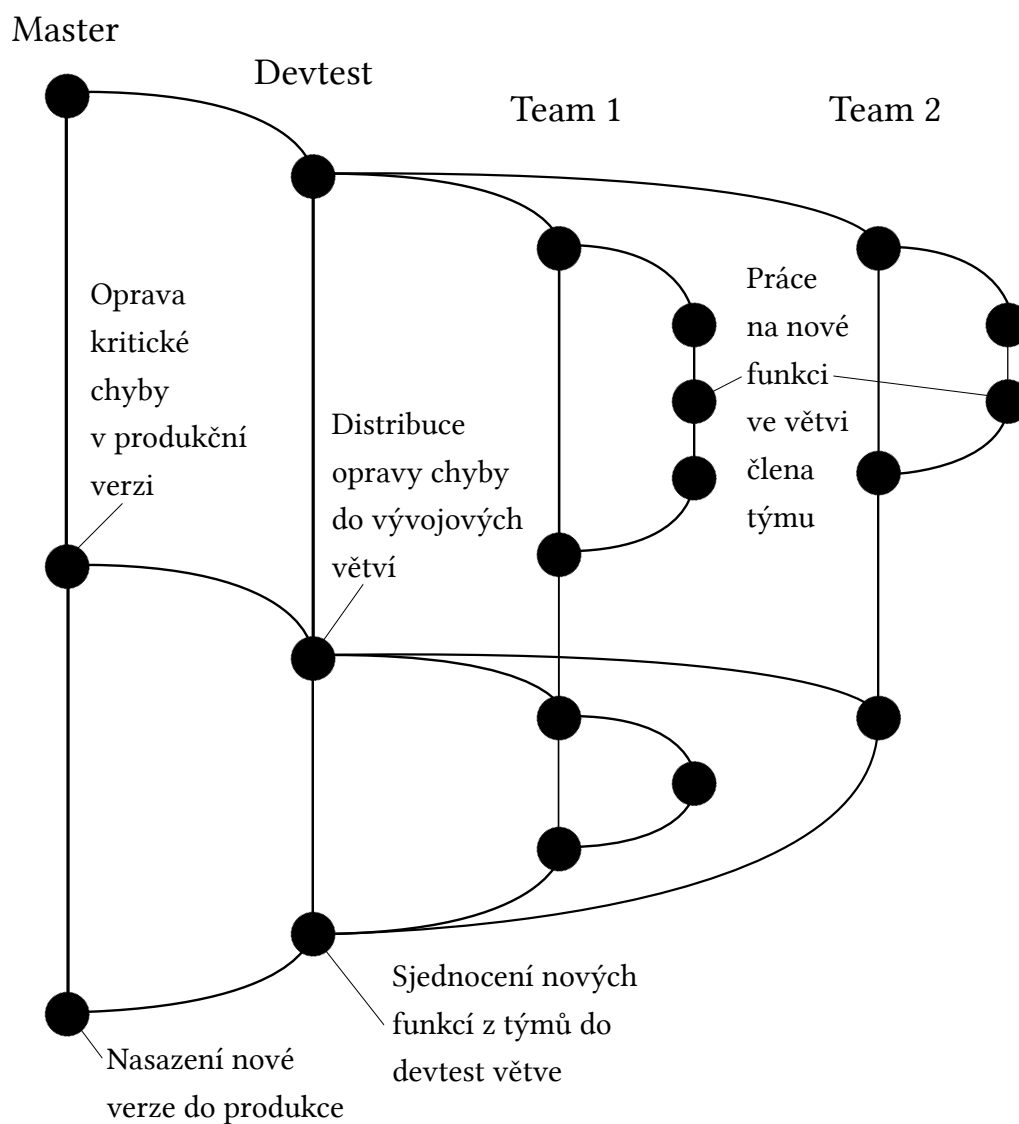
4.2.1 Git

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. - *představení Gitu na oficiálním webu [20]*.

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 branchy s rozpracovanou novinkou na stabilní nasazenou branch, zde provést opravu a poté opět pokračovat ve vývoji nové funkce. Tento proces je ilustrován na obrázku 4.2, který dokumentuje používání branchí v DBS projektu.

- *master branch* obsahuje vždy verzi aplikace, která je nasazena na doménu `db.fit.cvut.cz`
- *devtest branch* slouží pro testování nových funkcí. Shromažďují se do ní novinky z týmových branchí a bývá pravidelně nasazována na doménu `db2.fit.cvut.cz`
- *Týmové větve* pojmenované `team1` a `team2` jsou již v režii týmových vedoucích. Je však doporučeno další rozdělení na branchy 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 snazší 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 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 `db2.fit.cvut.cz`, kde je ještě jednou zkontrolována jejich funkčnost. Vydání nové verze (tedy *merge* branchy *devtest* do *master* probíhá typicky pouze jednou týdně, po několikadenním testování nových funkcí na testovací doméně.



Obrázek 4.2: Používání branchí v gitu při vývoji DBS portálu

Formát commit message Zanesení implementace nových funkcí do sdíleného repozitáře probíhá pomocí *commitů*, ke kterým se píše 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 *připojení*, které již bylo popisováno v sekci 4.1.1.2.

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.

Hlavní výhodou je však bezesporu integrované CI:

Automatické testování O automatické testy se v projektu stará především Pavel Kovář v rámci své bakalářské práce [12]. Ty se spouští autonomně, při každému commitu, a umožňují tak ihned odhalit chybu ve funkci pokryté testem. Oznámení o chybě je jednak programicky zapsáno na Redmine, jak již bylo popsáno v sekci 4.1.1.2, 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 [19]. 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 - kdokoli 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 možnostmi 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 či návrhy a požadavky nové 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 týkající 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 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.

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 jeho požadavků. Například manažer projektu bude chtít být informován o všech změnách ve všech projektech, vedoucí týmu poté pouze o všech změnách v jeho týmu a nakonec řadový člen bude odebírat pouze změny, které se týkají přímo jeho práce.

4.3.3 Schůzka

Nejméně častým, ale zato neintenzivnějším 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 jejich úkolů či se řeší 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 hezky popisuje například Olivier Lacan [13]. 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 testéři k němu mají v rámci repozitáře přístup, kopírovat jej do changelogu by tedy byla navíc duplikace informací. Správný changelog by měl jednoduše vypíchnout nejdůležitější novinky, změny a opravy v nové verzi.

Pro DBS aplikaci píše v současné době changelog já, a to ve chvíli, kdy se slévají změny z jednotlivých týmů do branchu devtest, jak bylo popisováno ve workflow gitu v sekci 4.2.1. Nejnovější sekce změn je nadepsána *Unreleased* a v této verzi je aplikace nasazena na `db2.fit.cvut.cz`. Učitelé, kteří se rozhodnout na této doméně otestovat novou verzi tak mají k dospizici seznam změn, které se měnily, aby se na ně mohli zaměřit. Při následném nasazení na hlavní doménu `db2.fit.cvut.cz` je součástí skriptu, který nasazení provádí (více v sekci 4.4.4), automatická změna nadpisu *Unreleased* na aktuální verzi a datum.

TODO git tag?

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 umižňuje 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 `dbś.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 `dbś2.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 standarní 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. Základní konfigurace SSL může pro apache webserver vypadat následovně:

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
```

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 [22].

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.

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*, které 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 cíleně být na sobě nezávislé:

```
<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>

```

Dále bylo potřeba opět upravit konfiguraci webserveru (apache):

```

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

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

```

Na adrese, kam směřuje toto přepisovací pravidlo, 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 vlastníkově 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.

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 [23]. To znamená, že se jedná pouze o alias, který ukazuje na stejný server a zde probíhá rozdělení na základě name-based virtual hostů, které jsou nakonfigurované 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 branch na `dbs.fit.cvut.cz`:

```
#!/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 -
```

Tento skript se spouští z rootu lokálně nasazené aplikace a využívá Git, ve kterém nejprve do master branchy provede merge z devtest, jak bylo popisováno v sekci 4.2.1. 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 branchy, 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 podobná 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.

4.4.5 Zálohování

Zálohovat data je potřeba vždy a všude. Když nastane problém s úložištěm, první otázka bývá „A máte zálohy?“. Především když se jedná o studijní data 550 studentů a připravené testové otázky mnoha vyučujících, je žádoucí o tato data pečovat.

Zálohování probíhá jednak na serveru - na stejný filesystem. 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 provést i tzv. *off-site* zálohy, neboli zálohy na *jiný počítač*.

Oba typy záloh provádí bash script, který na serveru spouští *cron*. Zálohy jsou spouštěny 1x denně - kolem 4 hodiny ranní. Zálohuje se databáze a soubory na filesystemu - data semestrálních prací studentů. Zálohy jsou uchovávány vždy týden, starší jsou mazány. Navíc 1x měsíčně je uchována i persistentní záloha. Na konci skriptu probíhá i nahrání nových souborů na externí úložiště - IDrive [10].

Skript vypadá následovně (pro účely tohoto textu byl mírně zjednodušen a byly zcenzurovány přístupové údaje):

```
#!/bin/bash

PROJECTPATH="/var/www/newDBS"

# ===== SWFILES =====
# preparations
mkdir -p $PROJECTPATH/backup/swFiles

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

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

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

# ===== DATABASE =====
# preparations
mkdir -p $PROJECTPATH/backup/database
echo 'localhost::newdbs:<db_user>:<db_pass>' > ~/.pgpass
chmod 0600 ~/.pgpass

# create dump
cd $PROJECTPATH/backup/database/
pg_dump -h localhost -d newdbs -U <db_user> > $(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

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

Kromě zálohování databáze a dat semestrálních prací jsou samozřejmě zálohovány i zdrojové kódy aplikace. Toto je však řešeno implicitně používáním gitu, který je *distribovaný* a v případě problému s hlavním repozitářem je možné jej 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 neefektivnou kódu, které mohou vést až na dlouhé načítání stránek koncových uživatelů.

Pro DBS portál jsou v současnosti využívány následující služby:

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). 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.

4.4.6.2 Ansible, Nagios

TODO?

4.4.6.3 Google Analytics

Analytics [7] je webová služba, která nabízí přehledy o uživateli 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 sledovat jak se například 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 uživatelů 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 navštívována stránka *termínů odevzdání*. Zde student získává pouze informaci, do kdy musí vyhotovit jednotlivé kontrolní body své práce. Díky tomuto zjištění byl zadán úkol k realizaci zobrazování informací o termínech odevzdání přímo na domovské stránce studenta, kde bude přehledně vidět jak „nejbližší termínu“, tak přehled všech termínů.

Kromě toho také získáváme obecné statistiky o uživateli. Některé za 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 ze školních učeben.
- 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 neoptimalizaci. Tato data tedy poukazují na to, že by portál do budoucna měl být vyvíjen i pro použití na menších rozlišeních.

4.4.7 Oprava kritických chyb

Kromě všech výše zmíněných součástí infrastruktury se také starám o opravu *kritických chyb*, které se v aplikaci objevují při jejím běhu. Velmi zkrácený výčet obsahuje například:

- TODO?

4.5 Návrhy budoucích vylepšení infrastruktury

Na závěr bych rád prezentoval některé další návrhy na zlepšení infrastruktury, které jsem nestihl realizovat do termínu odevzdání této práce. Mohou sloužit buďto jako inspirace pro další studenty pracující na portálu, nebo i pro mě, pokud se rozhodnu na téma DBS portálu vytvořit i mou diplomovou práci.

4.5.1 Integrace Gitu do Datadogu

Na konci sekce o Datadogu (4.4.6.1) jsem zmiňoval chybějící podporu integrace GitLabu do Datadogu. Cílem do budoucna by mělo být zahrnout do tohoto monitoringu i informaci o commitech, což by usnadnilo případné hledání chyb.

4.5.2 Třetí testovací prostředí

V současnosti má aplikace jednu produkční verzi - `db.fit.cvut.cz` s nasazenou větví `master` a jedno „testovací“ prostředí nových verzí - `db2.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. Právě z důvodu sdílené databáze také nemáme v projektu implementováno automatické nasazování změn z Gitu na server - jedná se o choulostivou akci, kterou je lepší provést 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 SP1/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.

Závěr

TODO závěr

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. 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/>.
8. HRONČOK, Miroslav. *Diplomová práce* [online]. 2016 (cit. 2017-03-05). Dostupné z: <https://github.com/hroncok/diplomka>.
9. HRONČOK, Miroslav. *FIT ČVUT thesis tips* [online]. 2016 (cit. 2017-03-05). Dostupné z: <https://github.com/hroncok/fit-thesis-tips>.

10. IDRIVE INC. *Backup and restore Linux servers using Command Line Utility* [online] (cit. 2017-04-22). Dostupné z: https://www.idrive.com/cmd_steps.
11. KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.
12. KOVÁŘ, Pavel. *Automatizované testování webového portálu dbs.fit.cvut.cz*. Bakalářská práce. Czech Technical University in Prague, Faculty of Information Technology, 2017.
13. LACAN, Olivier. *Keep a Changelog* [online] (cit. 2017-04-20). Dostupné z: <http://keepachangelog.com/en/0.3.0/>.
14. LANG, Jean-Philippe. *Overview - Redmine* [online]. 2006 (cit. 2017-03-05). Dostupné z: <http://www.redmine.org>.
15. PROCHÁZKA, Marek. *Skupinová dynamika, Týmová spolupráce* [online]. 2015 (cit. 2017-04-13). Dostupné z: <https://ekonom.feld.cvut.cz/cs/student/predmety/manazerska-psychologie>.
16. RASMUSSEN, Jonathan. *The agile Samurai: how agile masters deliver great software*. Raleigh: Pragmatic Bookshelf, 2010. ISBN 978-1-934356-58-6.
17. ŘEHÁČEK, Petr. *Projektové řízení podle PMI*. Praha: Ekopress, 2013. ISBN 978-80-86929-90-3.
18. SCHWALBE, Kathy. *Řízení projektů v IT: kompletní průvodce*. Brno: Computer Press, 2011. ISBN 978-80-251-2882-4.
19. SLACK. *Slack: Where work happens* [online] (cit. 2017-04-16). Dostupné z: <https://slack.com/is>.
20. SOFTWARE FREEDOM CONSERVANCY. *Git* [online] (cit. 2017-03-05). Dostupné z: <https://git-scm.com/>.
21. 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>.
22. THE SHIBBOLETH CONSORTIUM. *Shibboleth* [online] (cit. 2017-04-21). Dostupné z: <https://shibboleth.net/>.

23. 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>.
24. WYSOCKI, Robert K. *Effective project management: traditional, agile, extreme*. Indianapolis: Wiley, 2012. ISBN 978-1-118-01619-0.

Seznam použitých zkratk

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

A. SEZNAM POUŽITÝCH ZKRATEK

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 Transformati- ons
ZS	zimní semestr

Slovník pojmů

Continuous integration	Systém automatického sestavování a testování změn zdrojového kódu
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
Grido	PHP komponenta umožňující jednoduchý výpis dat do tabulky, jejich řazení, filtrování, stránkování a hromadné akce
Nette	Webový framework pro PHP
Organization Breakdown Structure	Hierarchický model popisující rozdělení odpovědnosti za jednotlivé části projektu
SQLite	Jednoduchý databázový stroj, který uchovává celou databázi pouze v jednom souboru
commit	Aplikování či přijetí nových změn zdrojového kódu. V tomto textu používáno především v souvislosti s <i>Gitem</i>
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

B. SLOVNÍK POJMŮ

Gitlab	Nástavba nad Gitem ulehčující používání repozitáře ve webovém rozhraní
Git	Distribuovaný systém pro správu verzí vhodný především pro soubory s textovým obsahem
Model-View-Presenter	<i>Návrhový vzor</i> softwaru, který klade důraz na oddělení jednotlivých vrstev aplikace. Využívá jej například Nette Framework.
push	Odeslání <i>commitu</i> do sdíleného repozitáře, například Gitlabu
Slack	Komunikační nástroj pro týmy

Diagramy



Vstupní kvíz

Úkol první

Výsledky zkoušky

Zadání úkolu

Níže je tabulka, do které se načítají data z databáze.

- Opravte načítání bodů
- Vypisujte, zda student zkoušku složil úspěšně. Pro úspěšné složení je potřeba získat alespoň 50 bodů, zaokrouhluje se vždy nahoru.
- Zobrazujte datum v čitelné formě (například 28. 3. 2017)
- Volitelně: Refaktorujte současný kód (načítání dat v presenteru je špatně) a vytvořte nový model/service, který bude data presenteru poskytovat.

#	Jméno	Získané body	Složil zkoušku	Datum
1	Láďa Vagner	?	?	2016-12-12
2	Franta Vomáčka	?	?	2017-01-18
3	Božena Němcová	?	?	2017-02-03
4	Olda Bouchač	?	?	2017-02-07
5	Ezio Auditore	?	?	1481-07-18
6	Hermiona Grangerová	?	?	1999-09-19

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

Úkol druhý Příspěvky

Zadání úkolu

V tomto úkolu naleznete aplikaci pro správu příspěvků. Vaším úkolem je doplnit chybějící funkce a opravit chyby, které se v aplikaci nacházejí.

- Implementujte mazání příspěvků.
- U formuláře pro tvorbu příspěvků nastavte následující validační pravidla.
 - **Název** - požadováno, maximální délka 40 znaků.
 - **Obsah** - minimální délka 5 znaků a maximální délka 700 znaků.
 - **Autor** - požadováno, maximální délka 30 znaků.
- Pokud tabulka na hlavní stránce tohoto úkolu neobsahuje žádný příspěvek, zobrazte v tabulce řádek obsahující zprávu "Žádný záznam."
- V náhledu příspěvku opravte výpis autora a odkaz na seznam příspěvků.
- Volitelně: Pro mazání příspěvků použijte AJAX (snippet).

Seznam příspěvků

Název	Autor	Akce
Příspěvek 1	admin	<div><div>🔍 Náhled</div><div>✎ Upravit</div><div>🗑 Odstranit</div></div>
Příspěvek 2	administrator	<div><div>🔍 Náhled</div><div>✎ Upravit</div><div>🗑 Odstranit</div></div>

+ Nový příspěvek

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 vhodném 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 přihlášení. 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 odebrání 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

Hodnocení týmů

Hodnocení SP1

poslední aktualizace: 14.04.2017

	Maximální body	
	SP/Sl	Redmine
1-2. týden	5	10
3-4. týden	9	18
5-6. týden	9	18
7-8. týden	9	18
9-10. týden	9	18
11-12. týden	9	18
13+zkouškové	0	0
	50	100

Team 1									
součet	úkolý		vedoucí		bonus		Marek E.	Anna D.	David R.
	x	x	x	x	x	x			
57,25	49,25	8	0				10	9,75	9
59	48	8	3				12,7	12,7	12,7
18,25	15,25	2	1				3	3	-2
							16,1	16,1	11,1
							4,1	4,1	4,1
0									
0									
0									

Team 2									
součet	úkolý		vedoucí		bonus		Jakub D.	Juraj K.	Lukáš B.
	x	x	x	x	x	x			
65,3	57,3	8	0				10,25	9,75	8
79,75	68,75	8	3				24,3	13,3	12,3
28	23	4	1				4	2	-4
							23,9	21,9	15,9
							7	7	7
0									
0									
0									

Souhrn Redmine bodů	42,9	42,65	36,9	38,4	36,65	
Souhrn Sl/SP1 bodů	21,45	21,325	18,45	19,2	18,325	
						65,45
						51,95
						43,2
						50,2
						32,725
						25,975
						21,6
						25,1

Obrázek E.1: Tabulka hodnocení týmů TODO aktualizovat

Obsah přiloženého média

TODO