ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

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

Pokyny pro vypracování

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

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

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 28. listopadu 2016

České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra softwarového inženýrství

Bakalářská práce

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

Oldřich Malec

Vedoucí práce: Ing. Jiří Hunka

14. 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 14. dubna 2017 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

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

Abstrakt

TODO

Klíčová slova TODO

Abstract

TODO

Keywords TODO

Obsah

Seznam tabulek

Seznam obrázků

Úvod

TODO úvod

Kapitola 1

Co je to DBS portál

Tato práce se zabývá především vývojem a správou portálu pro předmět BI-DBS na FIT ČVUT. Ještě než se tedy ponoříme do jednotlivých součástí vývoje tohoto portálu, chtěl bych čtenáře seznámit s 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 vymýšlí dotazy. Typická semestrální práce obsahuje ve své finální podobě alespoň 25 dotazů do a pokrývá všechny standardní klauzule DML, jednak z jazyku *relační algerby* 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 zkontrolovatl 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.

Kapitola 2

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 zbytečené 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 [5], 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 možní 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 vopádového modelu (2.1.1), DBS projekt potřebuje být vyvíjen pomocí skutečně iterativní metody, která bude v jednotlivých iteracích přidávat funkcionality.

TODO more std methods?

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 [5]). 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 | James Grenning | Robert C. Martin |
> | Mike Beedle | Jim Highsmith | Steve Mellor |
> | Arie van Bennekum | Andrew Hunt | Ken Schwaber |
> | Alistair Cockburn | Ron Jeffries | Jeff Sutherland |
> | Ward Cunningham | Jon Kern | Dave Thomas |
> | Martin Fowler | Brian Marick | |

Je tedy zřejmé, že Agilní metodiky více pracují s vývojáři samotnými, a umožňuji 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, [12]), u vývoje softwaru se naopak dá říci, že XP položilo základ pro agilní metodiky a je tak jednou z nich.

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

- Vždy budeme pracovat na co nejjednodušší verzi, která splňuje požadavky
- Pokud se osvědčuje *revize kódu*, budeme neustále revidovat kód ...
- Pokud se odvědčil *návrh*, budeme neustále navrhovat a vylepšovat specifikaci ...
- Pokud se osvědčilo *testování*, budeme neustále testovat ...
- 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 přidáné vlastnosti 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 píší i sami zákazníci, kteří tvoří testy funkcionality.

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

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 otestván vždy, jakmile je to možné, typicky tedy ihned po dokončení nové funkcionality.

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

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

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

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

2.2.2 Scrum

Scrum je charakterizován především pojmy *Sprint* a *Scrum meeting*. První z nich má trvání většinou jeden měsíc a opakuje se několikrát. Jeho cílem je přinést do výsledného produktu nové funkcionality. Naopak Scrum meeting je *každodenní* schůzka, na které se shrnou nově dokončené úkoly a stanoví se, na čem se bude dále pracovat. Schůzku vede tzv. *Scrum master*. Typicky probíhá 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 Lean development

TODO ?

2.2.4 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:

- *měritelnost*: vlastnost je jasně uchopitelná komponenta systému, kterou lze porovnat s požadavkem zákazníka
- *srozumitelnost*: programátor musí rozumět tomu, jak má vlastnost fungovat a co je jejím cílem
- *realizovatelnost*: programátor musí vědět, zda je schopen vlastnost realizovat v odpovídajícím časovém úseky (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 softwaru přidávat.

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

Návrh Vlastnosti se vždy týkají různých tříd, za které má zpravidla zodpovědnout určitá osoba, či 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 funkcionalitě nové vlastnosti. Společně s programovým kódem jsou tvořeny i testy, a to především *unit testy*.
Jakmile je vlastnost dokončena, je vložena do sdíleného prostoru, u FDD nazývaného *class repository*
Hlavní programátor poté dokončené vlastnosti integruje do předchozí verze hlavní aplikace.

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

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

TODO popsat týmy podle vlastností? (Kadlec 194)

2.2.5 Test driven development

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

Použití v DBS projektu Jelikož TDD se svým konceptem *test first, program later* vzdaluje od klasických vývojových metodik, není vhodné pro vývoj DBS portálu, kde se každý rok mění složení programátorů. DBS projekt samozřejmě využívá testy, viz kapitola TODO nameref, 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.

2.2.6 Kanban

TODO?

TODO more agile methods?

Na předchozích stránkách jsme si představili *některé* vývojové metodiky. Především těch agilních však existuje nepřeberné množství, a jak říká například Kadlec [5] nebo Rasmusson [8], každý projekt je jedinečný a určitě není nejlepší možností striktně dodržovat jednu z vývojivých metodik. Klíčem je spíše skloubení dostupných metodik k vytvoření vlastní, pro daný projekt nejefektivnějšího přístupu k vývoji. Vedoucí týmu či projektový manažer by neměl *slepě* následovat recept předložený například Extrémném programováním a zamítat jakékoliv jeho úpravy. Z toho důvodu i pro DBS projekt vznikla vlastní

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

2.3 Zvolené řešení pro DBS projekt

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

Na následujících řádcích budu pro potřeby této kapitoly popisovat i nástin *infrastruktury*, která se pro DBS projekt 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 *jakékoliv* 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 *push*i do *Gitlab*u 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 *featury* a role *hlavního programátora*:

Featury Ve FDD je vývoj hnán kupředu především seznamem vlastností, které má finální podoba mít. Toho využívá i DBS projekt, u kterého takový seznam existuje 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 členeny 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 věnuje TODO ref na příslušné kapitoly.

Kapitola 3

Řízení lidských zdrojů

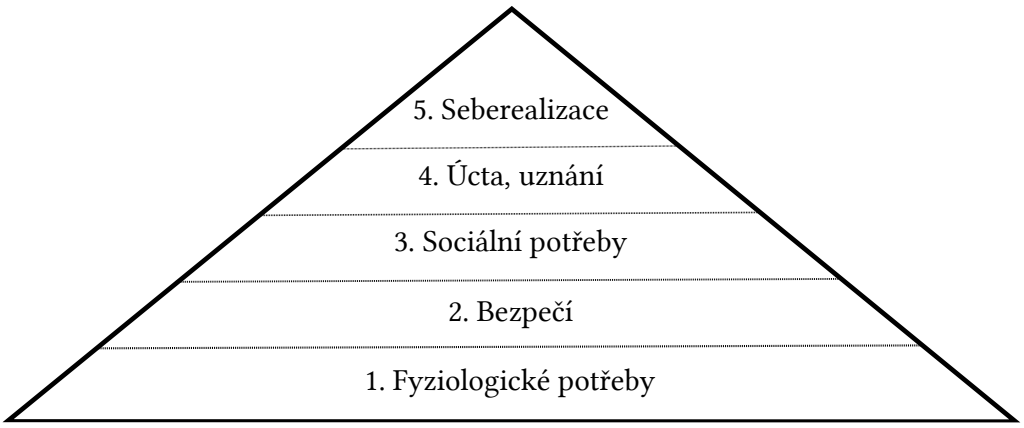
Řízení lidských zdrojů je jednou z neobtížnějších součástí projektového řízení, především proto, že se prolíná s dalšími odvětvími jako například *psychologií* či *sociologií*. Jelikož lidé jsou hnací silou projektu, mohou různé metody či samotná kvalita (ba dokonce absence!) řízení lidských zdrojů odrážet 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 lidkých 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.

TODO Schwalbe: 356+?

3.0.4 Procesy řízení lidských zdrojů

Jak popisuje Schwalbe [10] 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

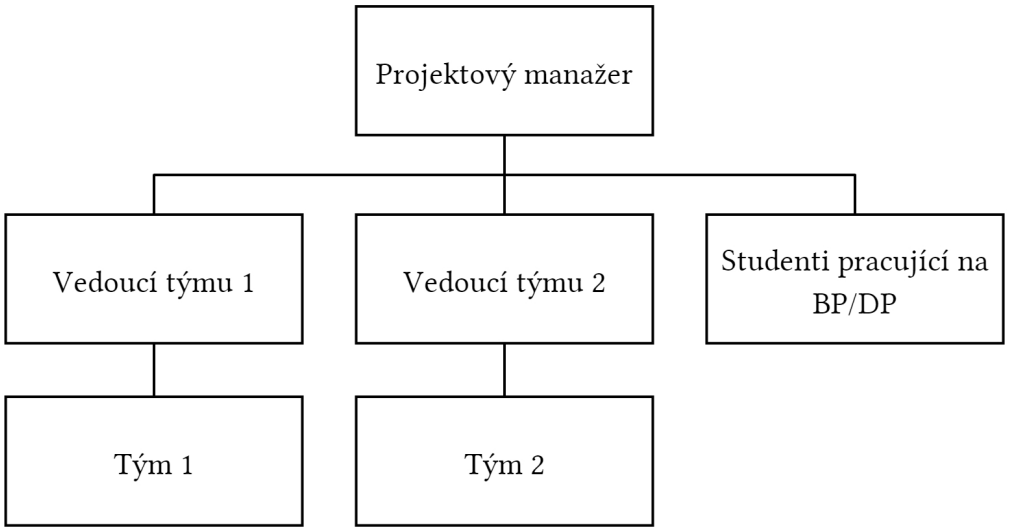
V následujících sekcích si jednotlivé procesy rozebereme pordobněji a zaměříme se i na jejich roli v DBS projektu.

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, testeři, grafici, z nichž například programátoři mohou být dále rozděleni do týmů starajících se o jednotlivé komponenty systému, kde každý tým má ještě svého vedoucího. Takovému plánu se říká *OBS* (Organization Breakdown Structure) a měla by být vytvořena na začátku projektu a podle ni následně zvoleni vedoucí daných oblastí.

Použití v DBS projektu V DBS projektu popisuje organizační členění obrázek 3.2.

Je však nutné přiznat, že tato struktura v případě našeho projektu vznikla spíše *na základě dostupných lidí*, než naopak, 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 activitz diagramu v příloze C.1.

3.2 Zajištění projektového týmu

Tento proces se primárně stará o *příjem nových pracovníků*, *řízení jejich vytížení* a *vyrovnání*. První ze zmíněných je především v IT hodně diskutované téma, kdy najít člověka se správnou kvalifikací je často nadlidský úkol. Firmy se tak přetahují kdo nabídne zajímavější pracovní prostředí, lepší benefity či *evergreen*: vyšší plat.
Když už je *ten správný* zaměstnanec přijat, je potřeba řídit jeho *vytížení* a případně jeho vznik *vyrovnávat*: Jedná se o metodiky, 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 shánění 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 podsekce Vstupní kvíz), který měl původně sloužit jako nutný požadavek pro vstup do týmů, z důvodu nízkého zájmu však byla poté jeho role přehodnocena. Kvíz byl studentům udělen 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 aplikace 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 náborem, který probíhá neustále, je nábor studentů, kteří budou na projektu pracovat v rámci své bakalářské či diplomové práci. Takoví studenti zpravidla přichází sami s požadavkem o vymyšlení tématu BP/DP či zůstávají u projektu po skončení jejich působení v předmětu BI-SP2. Tito studenti vyžadují zpravidla méně „péče“ projektového manažera, protože již vědí, co chtějí realizovat a často jsou v daném tématu i zběhlejší než já. 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 kompentenci týmových vedoucích, což dále rozvíjí jejich schopnosti vést tým. Pro správu přidělené práce využíváme *Redmine*, jehož použití se více věnuji v kapitole 4.1.0.2.

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 člěněním 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 odměňování 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 rozdílenému složení:

- *Team 1* vznikl spojením 5 studentů, kteří se do té doby vůbec neznali. U tohoto týmu byla patrná především první fáze *formingu*, kde byli jednotliví členové rozpačití, nevěděli, co si mohou dovolit a vedoucí byl ve své pozici nejistý. V průběhu následujících týdnů se však stav výrazně zlepšil a tým je nyní schopen provádět i komplexnější úlohy a rozvrhnout si práci. Předpokladem je, že do následujícího semestru, kdy budou studenti na projektu pracovat v rámci BI-SP2, dosáhne tým fáze *performingu* a jeho výstup bude na vysoké úrovni.
- *Team 2* 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 zezačátku 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 technologe - Nette Frameworku, ale také například zaměřená na Git či různá „how-to“, která jsou užitečná při vývoji DBS portálu.
Tato školení jsou většinou pořádána studenty, kteří na projektu pracovali v minulosti v rámci předmětů BI-SP1/2 a u projektu zůstali v rámci svých BP či DP.
Jelikož letošní rok bylo školení pořádáno hned ve druhém týdnu semestru, studenti byli ještě rozpačití a ostýchali se 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 cokoliv - a také „rozbít“ *groupthink* a *konformitu s většinou*, jak je definoval Procházka [7] ve své lekci *Skupinová dynamika, Týmová spolupráce* v kurzu *Manažerské psychologie*, které jsem se zúčastnil v zimním semestru 2015.

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.

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

Tabulka 3.1: Bodování týmů dle míry splnění úkolu

- *Nahlášení nového úkolu:* Studenti jsou vedeni k vlastní iniciativě: Když přijdou s novým úkolem, který vede ke zlepšení výsledné aplikace nebo opravuje závažnou, dosud nenahlášenou chybu, je 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 rozdistribuovány. Vedoucí týmů však mají ještě možnost hodnotit svůj tým interně, a to jak uznají za vhodné. Jedinou podmínkou je, že součet bodů udělených v týmu se musí vždy rovnat nule, pokud 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 konflitky atp.
Mezi hlavní problémy, ke kterým může docházet, lze zařadit:

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

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

3.4.1 Použití v DBS projektu

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

- Studenti jsou vedeni k vyjadřování vlastních názorů, snažíme se být otevřeni jakýmkoliv návrhům včetně kritiky vedení. Ze zkušeností vyplývá, že 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 *rozčilovalo* 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.

Kapitola 4

Infrastruktura DBS projektu

DBS projekt je webová aplikace, která je dostupná na adrese `https://dbs.fit.cvut.cz/`. Aplikace běží na Nette Frameworku (PHP) a je servírována pomocí apache. TODO

4.1 Infrastruktura projektu před realizací práce

4.1.0.1 Git

Git [11] je distribuovaný systém pro správu verzí, navržený jak pro malé tak pro velmi velké projekty, kladoucí důraz na rychlost a efektivitu. TODO více popsat git V DBS projektu byl využíván TODO

4.1.0.2 Redmine

4.2 Analýza a návrh zlepšení infrastruktury

Git Jako první nedostatek byla identifikována neexistence oddělení vývojové verze od produkční. TODO

4.3 Realizované řešení

Git V Gitu je nově dodržována přísnější struktura a například k produkční verzi má přístup pouze projektový manažer a ověření vývojáři, kteří již prošli předměty SP1 a SP2.

TODO

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. HRONČOK, Miroslav. *Diplomová práce* [online]. 2016 (cit. 2017-03-05). Dostupné z: `https://github.com/hroncok/diplomka`.
4. HRONČOK, Miroslav. *FIT ČVUT thesis tips* [online]. 2016 (cit. 2017-03-05). Dostupné z: `https://github.com/hroncok/fit-thesis-tips`.
5. KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.
6. LANG, Jean-Philippe. *Overview - Redmine* [online]. 2006 (cit. 2017-03-05). Dostupné z: `http://www.redmine.org`.
7. PROCHÁZKA, Marek. *Skupinová dynamika, Týmová spolupráce* [online]. 2015 (cit. 2017-04-13). Dostupné z: `https://ekonom.feld.cvut.cz/cs/student/predmety/manazerska-psychologie`.
8. RASMUSSON, Jonathan. *The agile Samurai: how agile masters deliver great software*. Raleigh: Pragmatic Bookshelf, 2010. ISBN 978-1-934356-58-6.
9. ŘEHÁČEK, Petr. *Projektové řízení podle PMI*. Praha: Ekopress, 2013. ISBN 978-80-86929-90-3.
10. SCHWALBE, Kathy. *Řízení projektů v IT: kompletní průvodce*. Brno: Computer Press, 2011. ISBN 978-80-251-2882-4.

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

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

Dodatek A

Seznam použitých zkratek

AJAX	Asynchronous JavaScript and XML
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
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
KOS	Komponenta studium
LS	letní semestr
MVP	Model-View-Presenter
OBS	Organization Breakdown Structure
PHP	PHP: Hypertext Preprocessor
PMI	Project Management Institute
RA	Relational algebra / Relační algebra
SP	Softwarový projekt

SQL	Structured query language
TDD	Test driven development
UI	User interface
UX	User experience
XML	Extensible Markup Language
XP	Extrémní programování
XSLT	Extensible Stylesheet Language Transformations
ZS	zimní semestr
ČVUT	České vysoké učení technické

Dodatek B

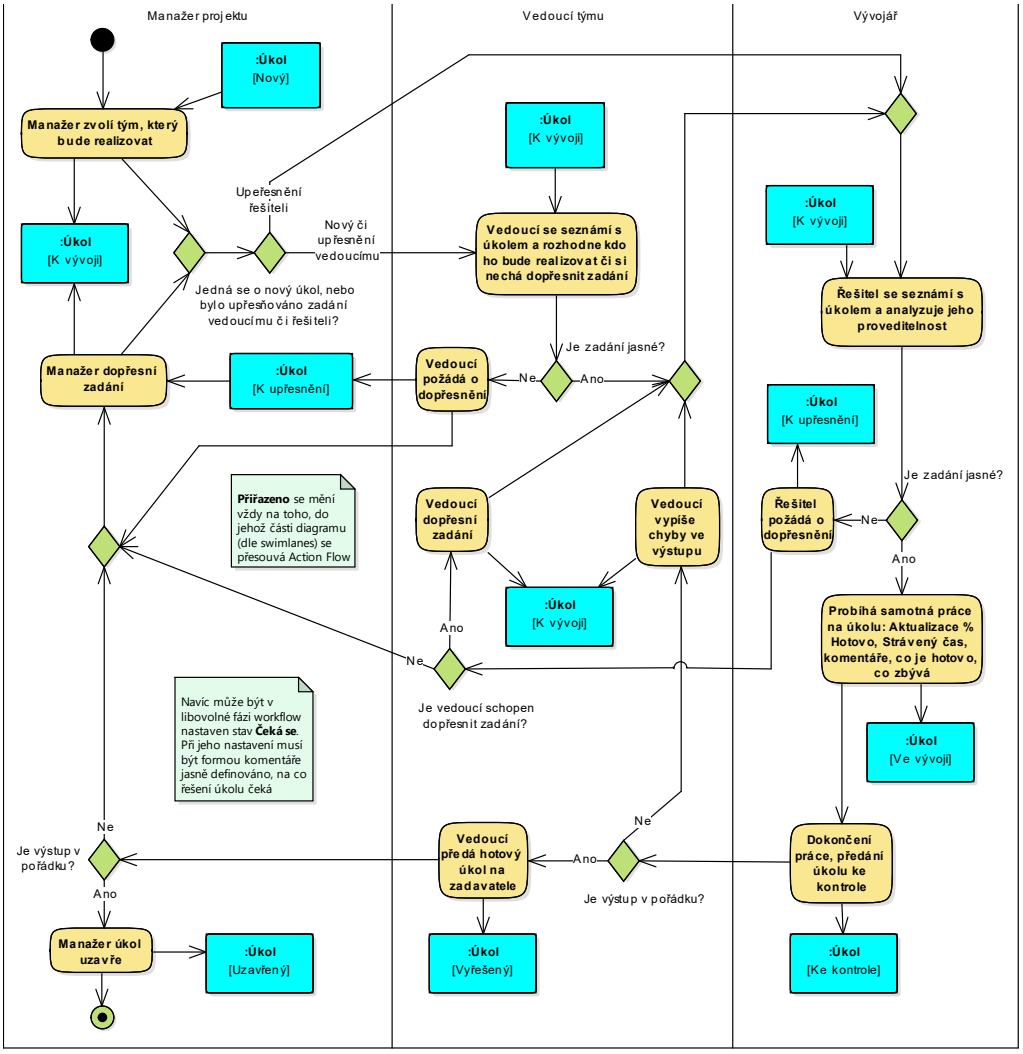
Slovník pojmů

Continuous integration	Systém automatického sestavováno 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. Nejznáměnší 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 přijmutí nových změn zdrojového kódu. V tomto textu používáno především v souvislosti s *Gitem*
Feature	Vlastnost, funkce, rys.
Framework	Softwarová struktura, která slouží jako podpora pro pohodlnější programování. Může obsahovat podpůrné funkce, knihovny či nástroje pro efektivnější, bezpečnější a pohodlnější vývoj softwaru

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

Dodatek C

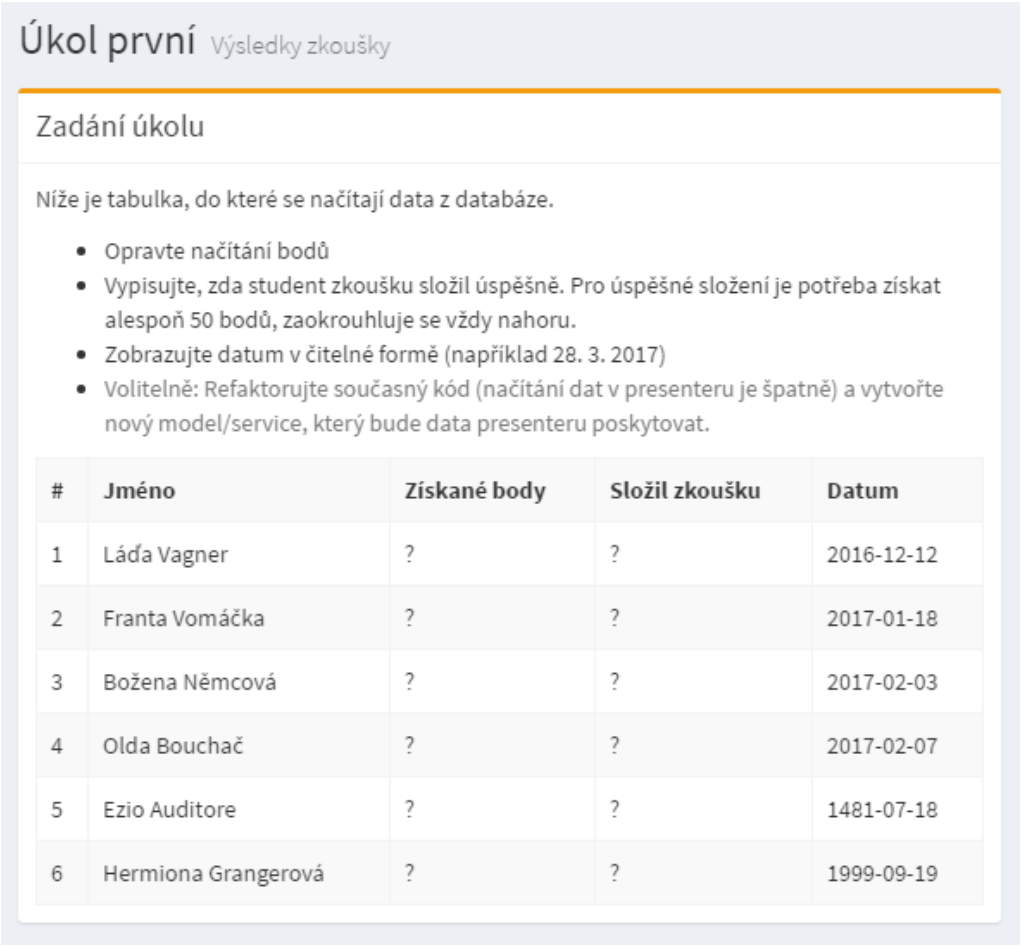
Diagramy



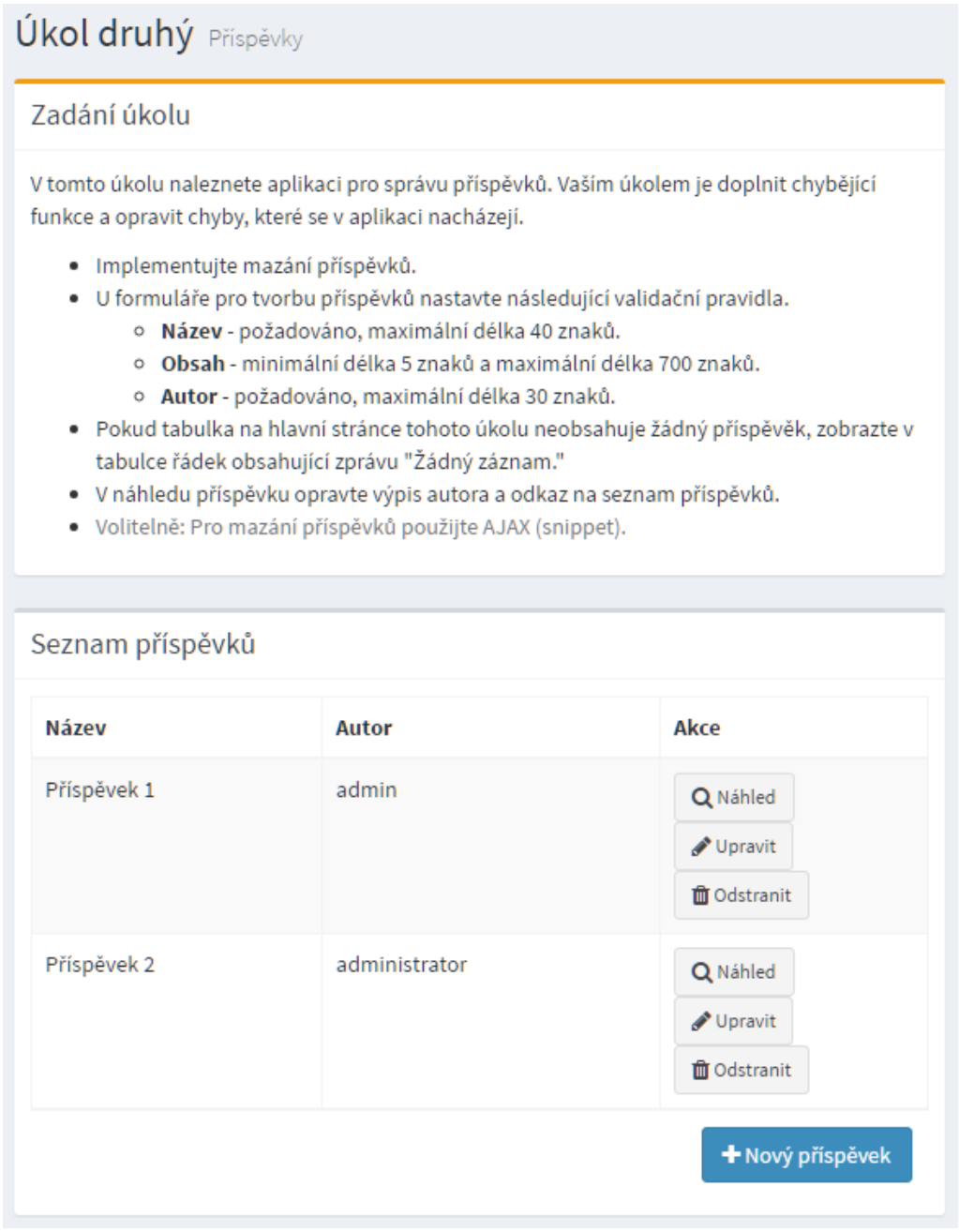
Obrázek C.1: Activity diagram průběhu práce v Redmine

Dodatek D

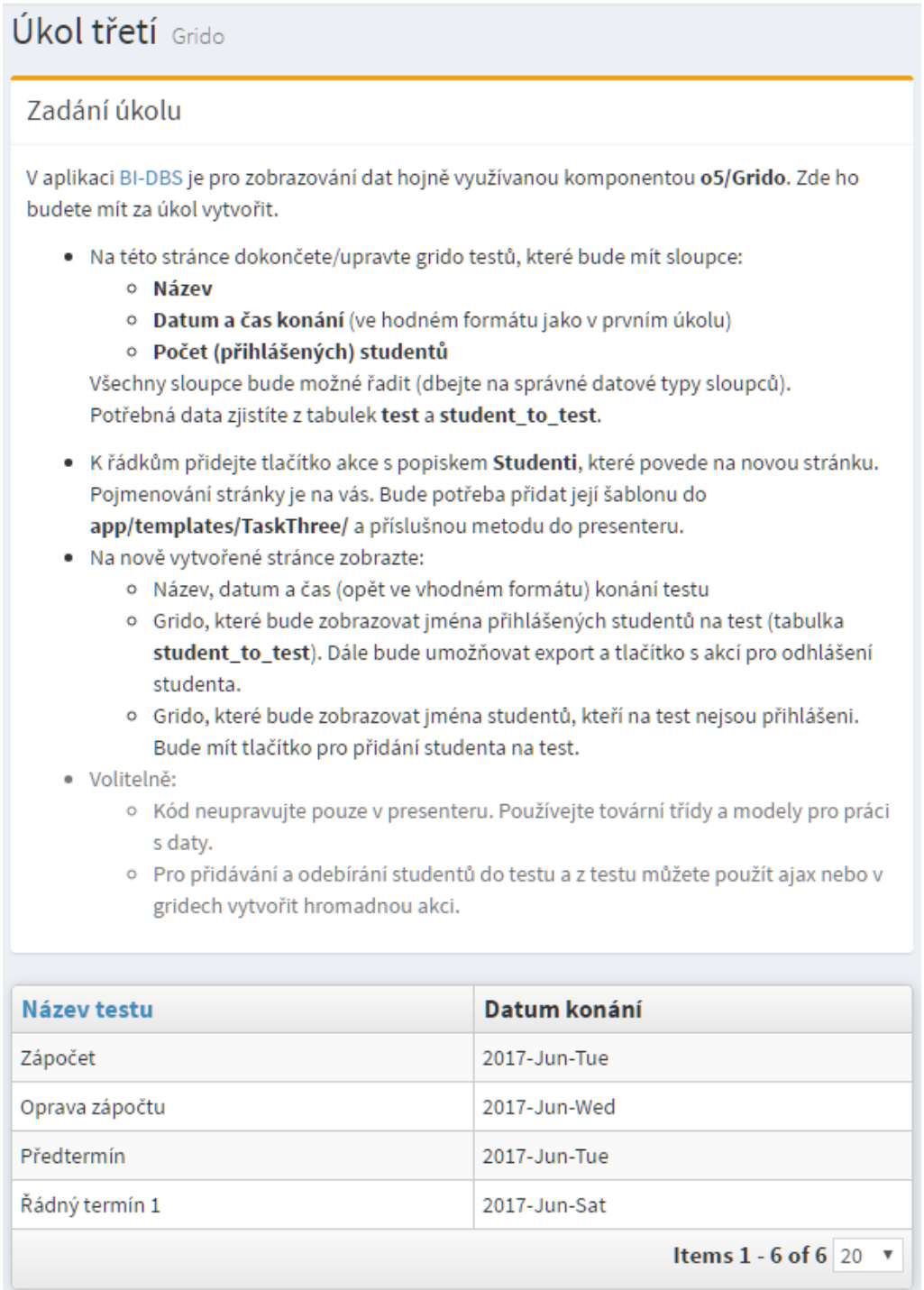
Vstupní kvíz



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



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



Úkol třetí Grido

Zadání úkolu

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

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

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

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

Items 1 - 6 of 6 20

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

Dodatek E

Hodnocení týmů

Hodnocení SP1 poslední aktualizace: 14.04.2017

	Maximální body	
	SP/SI	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	úkoly	vedoucí	bonus	Marek E.	Anna D.	David R.	Filip M.	Jan P.
x	x	x	x	10	9,75	9	9,5	9,75
57,25	49,25	8	0	0	0	0	0	0
				12,7	12,7	12,7	12,7	12,7
59	48	8	3	3	3	-2	-1	-3
				16,1	16,1	11,1	12,1	10,1
18,25	15,25	2	1					
				4,1	4,1	4,1	4,1	4,1
0								
0								
0								

Team 2							
součet	úkoly	vedoucí	bonus	Jakub D.	Juraj K.	Lukáš B.	Marek P.
x	x	x	x	10,25	9,75	8	10
65,3	57,3	8	0	8	-3	-4	-1
				24,3	13,3	12,3	15,3
79,75	68,75	8	3	4	2	-4	-2
				23,9	21,9	15,9	17,9
28	23	4	1				
				7	7	7	7
0							
0							
0							

Souhrn Redmine bodů	42,9	42,65	36,9	38,4	36,65	65,45	51,95	43,2	50,2
Souhrn SI1/SP1 bodů	21,45	21,325	18,45	19,2	18,325	32,725	25,975	21,6	25,1

Obrázek E.1: Tabulka hodnocení týmů