Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Migrace uzavřeného softwarového projektu na open-source

Bc. Filip Vondrášek

Vedoucí práce: Ing. Marcel Hlopko

23. června 2015

Poděkování

Děkuji v první řadě mému vedoucímu, Ing. Marcelu Hlopkovi, za cenné připomínky a vstřícný přístup, velké poděkování dále patří kolegovi z práce, Ing. Augustinu Šulcovi, za mentorování a navádění správným směrem – množství času, které se mnou strávil, už mu patrně nikdo nevrátí. V neposlední řadě bych pak rád poděkoval mé přítelkyni, Xeniye Valentové, rodičům a přátelům, konkrétně Václavu Jirovskému a Elišce Šestákové, za morální podporu, bez níž by mi již před několika lety absolutně přeskočilo, a to nejen kvůli náročnému studiu. Díky vám všem!

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) 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 23. června 2015 .....

Odkaz na tuto práci

Vondrášek, Filip. *Migrace uzavřeného softwarového projektu na open-source.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této práce je vysvětlit softwarovým vývojářům principy open-source softwaru, jeho vývoje a přechodu na open-source metodiku z uzavřeného vývoje. Práce popisuje postup, jehož pomocí lze převést stávající či vytvořit nový open-source projekt, od výběru licence a nástrojů až po vedení komunity. Dále je zanalyzována, navržena a implementována aplikace Deployment Framework sloužící pro správu a automatické nasazování softwarových projektů k zákazníkům. Na ní je dále předevedena migrace na open-source. Tato práce je ve výsledku kompletní příručkou k open-source vývoji.

Klíčová slova open-source, software, vývoj, komunita, společnost

Abstract

The goal of this work is to explain the principles of open-source software, its development and migration from closed to open-source development to developers. The work describes the process, due to which it is possible to migrate an existing project or to create a new one, beginning with choosing a license and tools and ending with community management. An application called Deployment Framework used for software projects management and automatic deployment was analysed, designed and implemented. Such migration is

then presented on it. This work is a complete guide to the open-source development.

Keywords open-source, software, development, community, company

Obsah

Úvod **1**
Motivace 1
Cíl 1
Struktura 2

I Teoretická část **3**

1 Open-source **5**
1.1 Definice 5
1.2 Výhody open-source 6
1.3 Nevýhody open-source 7
1.4 Financování open-source projektů 8

2 Open-source licence **9**
2.1 Creative Commons 9
2.2 MIT 11
2.3 BSD 11
2.4 GNU 12
2.5 Apache 14
2.6 Multi-licensing 14

3 Verzovací systémy **17**
3.1 Kategorie verzovacích systémů 17
3.2 CVS (Concurrent Versions System) 18
3.3 SVN (Apache Subversion) 18
3.4 Git 19
3.5 Mercurial 19

4 Rekvizity **21**

4.1 Hosting projektu . . . 21
4.2 Issue tracker . . . 23
4.3 Continuous integration a build server . . . 23
4.4 Changelog . . . 24
4.5 Mailing list . . . 24
4.6 Readme . . . 25
4.7 Dokumentace . . . 25
4.8 Webová stránka (prezentace) . . . 26
4.9 Ohlášení (announcement) . . . 26

5 Užitečné nástroje a SW doporučení 27
5.1 Git-flow . . . 27
5.2 Gitter . . . 28
5.3 HipChat . . . 28
5.4 Semantic Versioning (SemVer) . . . 30
5.5 Trello . . . 30

6 Vedení vývojového procesu 33
6.1 První fáze vývoje . . . 33
6.2 Druhá fáze vývoje . . . 33

7 Zpracování zdrojového kódu 39
7.1 Kvalita kódu . . . 39
7.2 Odstranění proprietárních modulů . . . 41

II Praktická část – Deployment Framework 43

8 Návrh a analýza 45
8.1 Účel . . . 45
8.2 Přehled uživatelů . . . 46
8.3 Funkční požadavky . . . 46
8.4 Nefunkční požadavky . . . 49
8.5 Architektura . . . 49

9 Implementace 51
9.1 Zvolené technologie . . . 52

10 Migrace na open-source 57
10.1 Licence . . . 57
10.2 Hosting projektu . . . 57
10.3 Build server . . . 58
10.4 Zakládání nezbytných dokumentů . . . 59
10.5 Webová stránka (prezentace) . . . 60
10.6 DEF Wiki . . . 62

10.7 Metodika vývoje 62

Závěr **63**

Literatura **65**

A Seznam použitých zkratek **69**

B Obsah přiloženého CD **71**

Seznam obrázků

2.1 Oblíbenost jednotlivých open-source licencí. 10
2.2 Rozložení uživatelské základy jednotlivých open-source licencí. . . 10

5.1 Označování záznamů z issue trackeru na Gitteru. 29
5.2 Trello účet hry The Dead Linger. 31

7.1 Detekce nedosažitelného kódu v IntelliJ IDEA 40

8.1 Architektura aplikace DEF. 50

9.1 Hlavní stránka – Dashboard. 51
9.2 Znečištění globálního prostoru NuGetem. 53

10.1 Využití Gitu oproti SVN. 58
10.2 Readme soubor pro DEF. 59
10.3 Javascriptový plugin bxSlider použitý ve webové prezentaci 61

Úvod

Motivace

Mezi open-source patří projekty jako linuxový kernel, kompilátory GCC a G++ či 3D modelovací software Blender, open-source však začínají pomalu využívat, či dokonce přecházet na vývoj, i společnosti, které se doposud open-source projektům bránily (kupříkladu Microsoft a jejich .NET).

Ve společnosti Baud spol. s r.o., jež je externím zadavatelem práce, byl od května 2012 vyvíjen (spoluvyvíjen autorem této práce) produkt Deployment Framework (dále jen DEF). Projekt byl původně zamýšlen jako interní nástroj, na němž by společnost vyzkoušela nové technologie pro pozdější použití na jiných projektech. Vyšlo však najevo, že aplikace je užitečná, a začaly se na ni nabalovat nové požadavky na funkčnost. Tímto ad-hoc vývojem bez návrhu se nicméně zdrojový kód stal těžce udržovatelným, bylo proto rozhodnuto DEF napsat od začátku s řádnou přípravou.

Současně s tímto bylo na základě porady dohodnuto, že aplikace bude převedena na open-source. Ukázalo se ale, že prakticky neexistuje žádný kompaktní zdroj, který by popisoval celý postup takové migrace – existují jen návody pro různé konkrétní problémy, nikoliv však návod říkající, na co se od začátku do konce postupně zaměřovat. Vznikl tedy nápad na tuto diplomovou práci.

Cíl

Cílem práce je usnadnit přechod na open-source metodiku vývoje softwarovým společnostem, jež se jinak zabývají tvorbou komerčního softwaru (tj. produktů, jež jsou dále prodávány koncovým zákazníkům). Dále si klade za cíl pomoci také vývojářům, kteří by rádi nahlédli do světa open-source a případně začali open-source sami vyvíjet. Dílo nicméně očekává od čtenářů již určité zkušenosti se softwarovým vývojem.

Práce se zabývá všemi podstatnými detaily od případů, kdy je vhodné na open-source přejít, a právních záležitostí, přes popis nejdůležitějších nástrojů a vhodných postupů při jednání s komunitou, až po ukázku, jak taková migrace probíhá v praxi.

Struktura

Práce je rozdělena na dvě části – teoretickou a praktickou. Teoretická, hlavní, část se zabývá způsoby, jak převodu na open-source dosáhnout. Tato část je dále dělena na logické celky, které popisují obecně open-source (definice, výhody, nevýhody...), open-source licence, verzovací systémy, rekvizity, užitečné nástroje, metody vedení vývojového procesu a metody zpracování zdrojového kódu.

Praktické kapitoly pak popisují návrh, analýzu a implementaci DEFu a konkrétní postup, jakým bylo dosaženo převodu na open-source. Ve výsledku jsou tedy v práci shrnuty veškeré podstatné záležitosti, které je při migraci na open-source potřeba znát.

Část I

Teoretická část

Kapitola 1

Open-source

1.1 Definice

Definice open-source dle Open Source Initiative [1]:

1. Licence nesmí nikoho omezovat v prodeji nebo rozdávání softwaru jako součásti celku obsahujícího programy z různých zdrojů. Licence nesmí vyžadovat licenční poplatek, ani jiný poplatek za takový prodej.
2. Program musí obsahovat zdrojový kód a musí umožňovat distribuci jak zdrojového kódu, tak zkompilované podoby. Pokud není nějaká forma produktu distribuována společně se zdrojovým kódem, musí být uveřejněna možnost tento zdrojový kód získat za cenu nikoliv vyšší než rozumný reprodukční poplatek, ideálně zdarma prostřednictvím stažení z Internetu. Zdrojový kód musí být preferovanou cestou, jíž by programátor program mohl měnit. Vědomě zmatený (z anglického obfuscated) kód není povolen. Meziprodukty jako výstup preprocesoru nebo překladače nejsou povoleny.
3. Licence musí dovolovat modifikace a deriváty, stejně jako jejich distribuci pod stejnými podmínkami a licencí jako původní software.
4. Licence může omezovat distribuci zdrojového kódu v upravené podobě pouze tehdy, pokud licence umožňuje distribuci patch souborů se zdrojovým kódem za účelem modifikace programu při jeho kompilaci. Licence musí výslovně povolit distribuci softwaru postaveného z upravených zdrojových kódů. Licence může požadovat, aby odvozené práce nesly jiný název či číslo verze.
5. Licence nesmí znevýhodňovat žádnou osobu či skupinu osob.
6. Licence nesmí nikoho omezovat při používání softwaru v konkrétních oborech. Kupříkladu nesmí omezovat použití programu v businessu nebo genetickém výzkumu.

7. Práva spojená s programem platí pro všechny, kterým je program dodán, bez nutnosti akceptovat další podmínky.

8. Práva spojená s programem nesmí záviset na tom, zda je program součástí nějaké softwarové distribuce. Pokud je program z dané distribuce vyjmut a použit či dále distribuován dle podmínek jeho licence, všechny strany, kterým je program distribuován, mají mít stejná práva jako ta, která poskytovala původní softwarová distribuce.

9. Licence nesmí omezovat jiný software, který je distribuován společně s licencovaným softwarem. Licence například nesmí trvat na tom, aby ostatní programy distribuované tímtéž médiem byly také open-source.

10. Žádné ustanovení licence nesmí být založeno na jakékoliv individuální technologii či stylu rozhraní.

Z uvedené definice přímo plyne, že zásadními rozdíly mezi open-source a proprietárním softwarem je především ona otevřenost, která znamená jak technickou, tak i právní dostupnost kódu. Uživatelé otevřených programů tedy mohou volně kód využívat (tudíž i upravovat). Open-source projekt se dále řídí dle různých licencí, které jsou popsány v kapitole 2, každá licence ovšem musí dodržovat popsanou definici.

1.2 Výhody open-source

Důvodů, proč by mohla vývojářská skupina přejít z vývoje proprietárního softwaru na open-source, je povícero.

Typicky to bývají následující:

- Vytvoření dobrého jména: Vytvoření užitečného nástroje a uvolnění jeho zdrojových kódů (nebo již vytváření tohoto nástroje za přispění komunity) rozhodně negativní dojem nevytvoří. Když si uživatelé spojí jméno společnosti s dobrým produktem a přátelským přístupem, může ji to pomoci.

- Vlastnosti a opravy, které je potřeba implementovat, stojí vývojáře čas, a jejich zaměstnavatele tím pádem i peníze. Pokud se povede vytvořit zaběhlejší open-source skupinu, která na projektu pracuje ze zájmu, společnosti to ušetří čas i peníze. Fakt, že vývojáři pracují na projektu ze zájmu, navíc výrazně přispívá ke kvalitě kódu, a to také proto, že nově napsané části prochází přísnou kontrolou dalších nadšených vývojářů. Podaří-li se tedy úspěšně open-source projekt rozjet, společnost má v podstatě pracovní sílu zdarma.

- Open-source může být bezpečnější. Na toto téma existuje mnoho dohadů, nicméně pravidlem palce zůstává, že čím více uživatelů do kódu vidí, tím větší je pravděpodobnost, že si někdo všimne kritické chyby a pokud ji rovnou neopraví, alespoň ji nahlásí. Tento bod má ovšem i druhou stranu, jak je popsáno v sekci 1.3.

- Občas se stane, že společnost potřebuje najmout nové zaměstnance. Musí tedy podat nabídku práce (či čekat, že si firmy někdo sám všimne), domluvit pohovory, všechny zájemce prověřit... Se zaběhlou open-source komunitou má společnost tu výhodu, že se může pokusit najmout vývojáře, kteří již na jejích projektech pracují. Vývojář už sám ví, co od práce může očekávat, a firma naopak ví, co může očekávat od daného vývojáře. Nejlepší pohovor je ten, který vůbec nemusíte vést.

1.3 Nevýhody open-source

Open-source neskýtá jen výhody, ale přináší i stinné stránky.

- V open-source sféře probíhá jinak vydělávání peněz (viz sekce 1.4), což může spoustu manažerů zpočátku vystrašit, a tím ztížit ještě samotný přechod na open-source vývoj.

- Už ze samotné povahy open-source, tedy otevřenosti kódu, společnosti nemohou používat své interní know-how. Pokud tedy mají některou část aplikace výborně napsanou, ale používají v ní interní postupy či moduly, budou ji muset přepsat (nebo odstranit dané proprietární moduly, viz sekce 7.2).

- Kontroverzní stránkou je bezpečnost – existují případy, kdy byla nalezena chyba v open-source projektu, která byla díky možnosti jednoduchého prostudování zneužita dříve, než byla opravena. Nejznámějšími případy, které se oba shodou náhod objevily v roce 2014 (a jednalo se o výrazně kritické chyby přímo ohrožující bezpečnost běžných uživatelů), jsou bezesporu chyby Heartbleed [2] v knihovně OpenSSL a Shellshock [3] v *nixovém Bashi. Konkrétně chyba Heartbleed existovala v OpenSSL dokonce přes dva roky (vznikla 14. března 2012, objevena byla v dubnu 2014), ne vždy tedy platí, že víc lidí víc vidí – tito lidé totiž nejprve musí vědět, co vlastně hledají, obvykle je nenapadne studovat kód řádek po řádku a hledat, zdali v něm nejsou chyby (zvlášť, když kód celou dobu funguje). Stále je ovšem pravděpodobnost, že si chyby někdo všimne, větší než u proprietárního softwaru.

1.4 Financování open-source projektů

Financování open-source projektů neprobíhá standardním prodáváním licencí k použití. Možností, jak projekt zpeněžit, nicméně stále existuje několik:

- **Prémiová funkcionalita**: Často bývá mylně zaměňována za demoverze. Tato monetizace typicky funguje tím způsobem, že produkt poskytuje veškerou proklamovanou funkcionalitu, ale lze si zaplatit za velmi specifické vlastnosti (takové vlastnosti, které využije například někdo v průmyslu, ne tolik však běžný koncový uživatel). Mezi takové vlastnosti může například patřit zvýšená stabilita pro enterprise uživatele.
- **Placená podpora**: Open-source software je obvykle dodáván „as-is“, tj. bez jakékoliv jiné podpory nežli komunitní. Jedním ze způsobů, jakým software zmonetizovat, je poskytnutí placené podpory (často v různých variantách, kde nejdražší verzí bývá 24/7 telefonická podpora, případně možnost na míru řešených úprav softwaru).
- **Multi-licensing**: Open-source software lze vydat pod více licencemi, například jednou otevřenou a jednou komerční, více o multi-licensingu je je popsáno v sekci 2.6.
- **Školení**: Další možností, jak open-source software zmonetizovat, je nabízení školení. Společnosti, které budou chtít open-source aplikaci používat v praxi, obvykle nechtějí, aby se uživatelé naučili s aplikací pracovat svépomocí, a často si tak objednávají školení.

Kapitola 2

Open-source licence

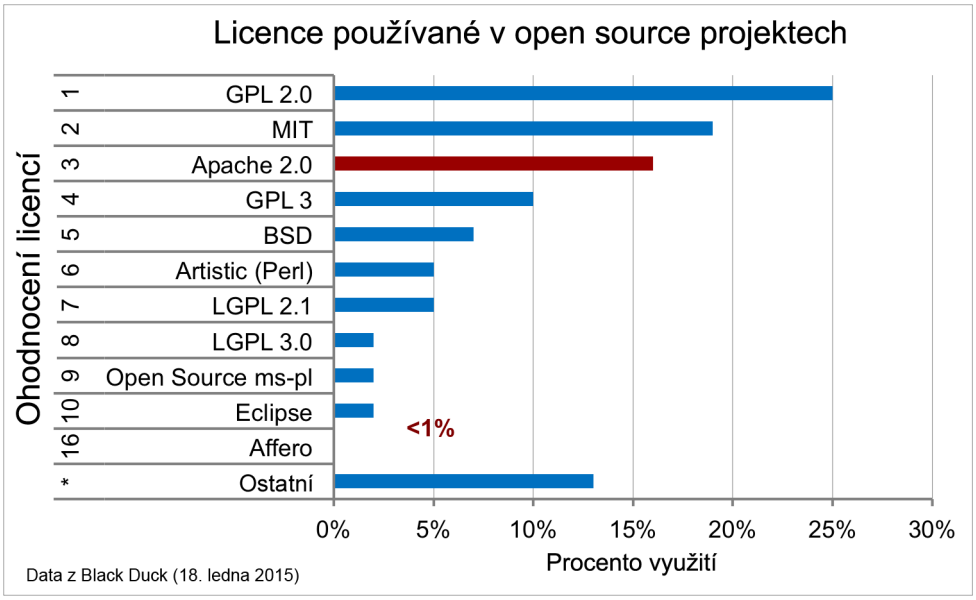
Na samém začátku vývoje open-source produktu je nutné si říci, jaká konkrétní licence bude potřeba. Není jich totiž málo a přestože všechny dodržují desetibodovou definici (část 1.1), liší se v zásadních věcech.

Výběr správné licence je pravděpodobně nejdůležitějším krokem při vývoji open-source projektu. Neexistuje dělení na špatné a dobré, ale na pro konkrétní účely vhodné a méně vhodné, až nevhodné. Je velice důležité si uvědomit veškeré právní aspekty a dopady jednotlivých licencí, jako příklad lze zvolit GNU GPL (sekce 2.4.1), která je specifická tím, že nejen odvozená díla, ale i díla používající v libovolné své části jiný projekt licencovaný pod GNU GPL, musí pak také používat stejnou licenci. Čili se jedná pak o rozhodnutí, zda svůj projekt jednou licencí „pojistit," ovšem za cenu menší popularity projektu (licence GNU GPL, i přes nesporné výhody, často kvůli této podmínce pro vývojáře bývá dostačujícím důvodem poohlédnout se po podobném projektu s jinou licencí – zřejmě málokdo by chtěl kvůli i té nejdrobnější knihovničce licencovat celý svůj počin pod jednou konkrétní licencí).

Následující popisy licencí jsou stručným výtahem. U každé z nich je proto uveden odkaz na oficiální znění. Aktuální oblíbenost, respektive rozložení uživatelské základny jednotlivých licencí lze vidět na obrázku 2.1, respektive 2.2.

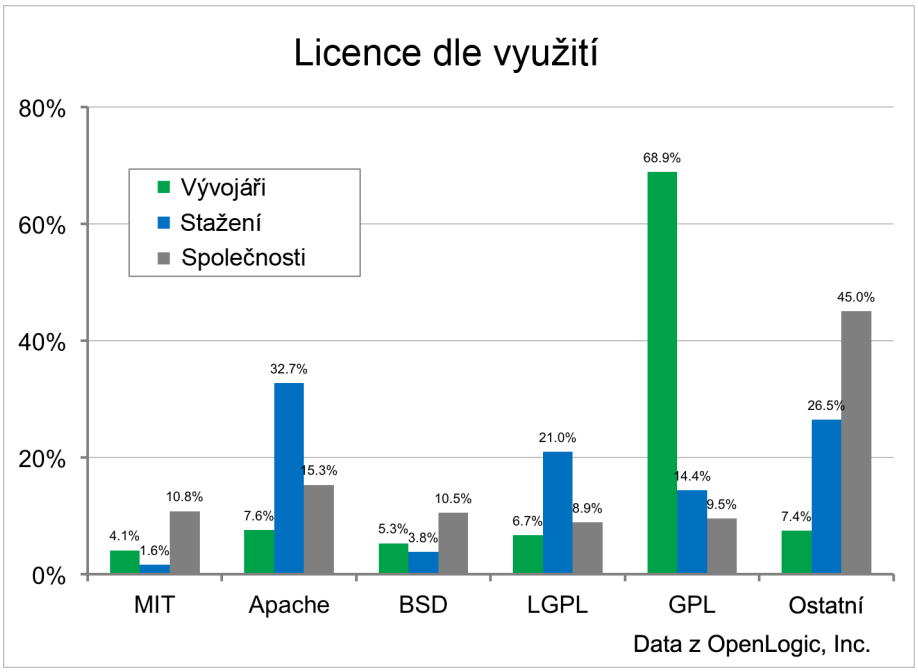
2.1 Creative Commons

Je vhodné alespoň se zmínit o americké neziskové organizaci Creative Commons [5] (dále jen CC), jejíž cílem je postarat se o svobodné šíření znalostí a tvůrčí činnosti. Existuje několik druhů CC licencí, dokonce i webová stránka, na níž si uživatel zvolí klíčové vlastnosti a ona sama pomůže s výběrem. [6] Nicméně i přes fakt, že spousta vlastností CC by byla vhodná pro tvorbu open-source softwaru, samotní autoři CC silně nedoporučují a naopak upozorňují na existenci jiných, pro software vhodnějších, licencí. Důvod je prostý – žádná z CC licencí neobsahuje výslovné podmínky šíření zdrojového kódu. Z tohoto důvodu se dále v práci nebudeme CC zabývat.



Obrázek 2.1: Oblíbenost jednotlivých open-source licencí.

Zdroj: `http://osswatch.jiscinvolve.org/wp/2015/02/05/open-source-software-licensing-trends/` [4]



Obrázek 2.2: Rozložení uživatelské základy jednotlivých open-source licencí.

Zdroj: `http://osswatch.jiscinvolve.org/wp/2015/02/05/open-source-software-licensing-trends/` [4]

2.2 MIT

MIT licence uděluje všem bezplatné právo na manipulaci s programem a zdrojovým kódem bez jakýchkoliv omezení, tedy včetně kopírování, modifikace a dokonce i prodeje. Je pouze nutné spolu se softwarem distribuovat i znění licence a zřeknutí se zodpovědnosti. [7]

Příkladem projektu, který je vydáván pod licencí MIT, je například oblíbená JavaScriptová knihovna jQuery, jejíž tvůrci na svém webu výslovně uvádějí, že je možné jQuery použít v jakémkoliv jiném projektu, včetně komerčních, pouze za podmínky zachování licenční hlavičky. [8]

2.3 BSD

BSD spolu s MIT patří mezi nejliberálnější licence – umožňuje svobodné šíření pouze pod podmínkou uvedení licence, autora a zřeknutí se odpovědnosti. Primárně se dělí na dva druhy: Starší BSD 3-Clause (BSD New) a novější BSD 2-Clause.

2.3.1 BSD 3-Clause

Přestože se BSD 3-Clause také nazývá „BSD New," nejedná se o nejnovější verzi; pouze je novější než původní BSD, které vyžadovalo, aby ve veškerých reklamách na vyvinutý produkt bylo uvedeno, že program používá kód Univerzity v Berkeley (pokud ovšem takový kód použit byl).

Tři podmínky, které BSD 3-Clause požaduje, jsou [9]:

1. Šířený zdrojový kód musí obsahovat informace o copyrightu, tento seznam podmínek a zřeknutí se odpovědnosti.

2. Šířená binární verze programu musí obsahovat informace o copyrightu, tento seznam podmínek a zřeknutí se odpovědnosti v dokumentaci programu a/nebo dalších materiálech distribuovaných s programem.

3. Odvozený produkt nesmí být prezentován jménem původních tvůrců, držitelů práv či přispěvatelů, pokud k tomu tito nedali písemný souhlas.

2.3.2 BSD 2-Clause

Licence BSD 2-Clause je novější verzí třísložkového BSD, pouze s vypuštěným třetím bodem. [10]

2.4 GNU

2.4.1 GNU GPL

V době psaní této práce se používá verze 3 (označována jako GNU GPL v3 nebo jednodušeji GPL v3) a spolu s Apache licencí patří mezi nejkomplikovanější open-source licence. GPL v3 řeší následující záležitosti [11]:

2.4.1.1 Zveřejňování doslovných kopií

Je povoleno zveřejňovat doslovné kopie zdrojového kódu na jakémkoliv médiu za předpokladu, že budou splněny všechny následující body:

1. Na každé kopii bude jasně a vhodně uveden copyright.
2. Veškeré zmínky odkazující na tuto licenci budou ponechány beze změn.
3. Veškeré zmínky o absenci záruky budou ponechány beze změn.
4. Všichni, kteří kopii obdrží, obdrží také kopii této licence.

Za každou kopii je povoleno účtovat poplatek a je povoleno za poplatek poskytovat podporu či záruku.

2.4.1.2 Zveřejňování upravených verzí

Pro zveřejňování upravených verzí platí stejná pravidla jako v sekci 2.4.1.1, krom toho navíc platí následující body:

1. Dílo musí nést viditelné poznámky s datem a autorem úprav.
2. Dílo musí nést viditelné poznámky říkající, že je šířeno pod touto licencí a dalšími podmínkami, které v této práci zmíněny nebudou (neboť nejsou nosnými pilíři licence).
3. Je nutné licencovat dílo jako celek touto licencí. Tato licence tedy bude platit na celé dílo a všechny jeho části bez ohledu na to, jakým způsobem jsou zabaleny.
4. Pokud dílo obsahuje interaktivní uživatelská rozhraní, každé z nich musí obsahovat příslušné právní poznámky (copyrighty). Pokud však interaktivní rozhraní původního díla tyto poznámky neobsahují, není nutné tento bod dodržet.

2.4.1.3 Zveřejňování nezdrojových verzí

Pro zveřejňování nezdrojových verzí (ve formě strojového kódu) platí stejná pravidla jako v sekci 2.4.1.2, krom toho navíc musí platit některý z následujících bodů:

1. Strojový kód musí být zveřejněn na trvalém fyzickém médiu běžně používaného pro výměnu softwaru spolu s úplným kódem.[1]

2. Strojový kód musí být zveřejněn na trvalém fyzickém médiu spolu s psanou nabídkou (platnou na alespoň tři roky a tak dlouho, dokud jsou nabízeny náhradní části nebo zákaznická podpora) na dodání úplného kódu za cenu nepřevyšující náklady na výrobu a dodání, nebo na přístup ke kopii úplného kódu ze síťového serveru bez poplatku.

3. Jednotlivé kopie díla musí být zveřejněny s kopií psané nabídky na poskytnutí úplného kódu. Tato alternativa bývá ovšem povolena jen občas a pro nekomerční účely a pouze pokud distributor dílo původně obdržel s takovouto nabídkou.

4. Strojový kód díla musí být nabízen přístupem z určených míst (zdarma, či za poplatek). Úplný kód musí být nabízen stejnou cestou bez dalších poplatků. Pokud je strojový kód nabízen ke stažení ze síťového serveru, úplný kód může být umístěn na jiném serveru s ekvivalentními podmínkami ke stažení, pokud jsou jasně dané instrukce, jak se k úplnému kódu dostat. Distributor strojového kódu je povinen zajistit dostupnost úplného kódu nezávisle na hostiteli.

5. Strojový kód musí být nabízen zdarma na peer-to-peer sítích, pokud jsou uživatelé obeznámeni s tím, kde strojový a úplný kód najdou.

2.4.2 GNU-LGPL

Licence GNU-LGPL (GNU Lesser General Public License, dříve GNU Library General License) je odlehčenou verzí GNU GPL. Nejvýraznějším rozdílem mezi LGPL a GPL je možnost projekty vydané pod licencí GNU-LGPL užívat i v proprietárním softwaru a ve svobodném softwaru pod jinou licencí nežli (L)GPL. Program, který v sobě používá jiný produkt licencovaný pod GNU-LGPL, je možné dále vydávat pod jinou licencí, nejedná-li se o odvozené dílo. [12]

[1] Úplný kód znamená veškerý zdrojový kód potřebný k vygenerování, nainstalování a (v případě spustitelného díla) spuštění programu, stejně jako k úpravě díla.

2.5 Apache

Licence Apache z dílen společnosti Apache Software Foundation, je v lecčems podobná GPL v3, narozdíl od ní ovšem nevyžaduje po uživatelích copyleft.[2] Požadavky Apache licence na distribuci jsou následující [13]:

1. Uživatelé díla nebo odvozených děl musí obdržet kopii této licence.

2. Jakékoliv změněné soubory v sobě musí nést zřejmou zmínku o tom, že byly modifikovány (spolu se jménem autora úpravy).

3. Ve zdrojovém kódu jakéhokoliv odvozeného díla, které uživatel distribuuje, je nutné zachovat veškeré copyrighty, patenty, obchodní značky a poznámky z původního díla, kromě těch částí, jež nejsou použity v žádné části odvozeného díla.

4. Pokud původní dílo obsahuje textový soubor `NOTICE`, potom všechna odvozená díla musí obsahovat čitelnou kopii copyright sekce tohoto souboru, vyjma těch copyrightů, které nesouvisí s žádnou části odvozené práce, na alespoň jednom z těchto umístění:

   - Uvnitř `NOTICE` souboru distribuovaného s odvozeným dílem.
   - Uvnitř zdrojového kódu nebo dokumentace, je-li spolu s odvozeným dílem poskytnuta.
   - Na obrazovce generované odvozeným dílem, pokud a kdekoliv se copyrighty třetích stran objevují.

   Obsah `NOTICE` souboru slouží čistě pro informativní účely a nemění nijak licenci. Do odvozených děl je možné doplňovat vlastní copyrighty za předpokladu, že se tyto nesnaží nijak upravovat licenci.

Licencí schválených společností OSI je ještě více [14], licence výše popsané nicméně patří mezi nejpopulárnější a v naprosté většině případů si z nich vývojář zvolí, nemá proto smysl v této práci popisovat další, méně časté licence.

2.6 Multi-licensing

Zvláštní možností je licencovat dílo pod více licencemi, typicky pod komerční a open-source. Dělává se to s ohledem na komerční důvody společností – koncoví uživatelé (a hobby programátoři) si dílo zdarma stáhnou a pokud jej dále upraví (nebo začlení do svého programu), jednoduše jej znovu uveřejní. To ovšem často nepřichází v úvahu u společností, které by sice také rády některá open-source díla ve svých programech využívaly, ovšem v žádném případě

[2] Nutnost vydat své dílo pod stejnou licencí.

si nepřejí dále kvůli jedné převzaté části zveřejňovat zdrojové kódy celého svého díla. V takové situaci pak přichází na řadu multi-licensing, díky němuž si vývojáři mohou program koupit, a tím pádem se zbavit nutnosti jednat nadále s kódem podle open-source pravidel.

Počet licencí, pod nimiž lze dílo licencovat, je v podstatě neomezený, zvykem však bývá dvojí licence (dual-licensing).

Kapitola 3

Verzovací systémy

Ještě předtím, než budeme hovořit o projektových hostinzích pro projekty v kapitole 4.1, proberme si stručný základ sdílení zdrojových kódů a dalších souborů – verzovací systémy. Jejich výběr totiž následnou volbu projektového hostingu přímo ovlivňuje.

3.1 Kategorie verzovacích systémů

3.1.1 Lokální verzovací systémy

Prvními verzovacími systémy (zde se píše počátek sedmdesátých let dvacátého století) byly takzvané lokální verzovací systémy. Název této kategorie je zjevný – veškerý zdrojový kód je uchováván na lokálním souborovém systému. Takové verzovací systémy jsou tedy vhodné jen pro jednotlivce, nebo týmy pracující na směny a tedy nevhodné pro open-source projekty. V této práci se jimi proto dále nebudeme zabývat.

3.1.2 Verzovací systémy s architekturou klient-server

Druhou nejstarší kategorií jsou verzovací systémy pracující na bázi klient-server. Práce s nimi funguje tím způsobem, že uživatel si (za pomocí konkrétního verzovacího systému) stáhne kompletní repozitář, ovšem bez jakékoliv historie a dalších souborů, kterými lze přepínat mezi verzemi atp. Nejznámějšími systémy pracující na bázi klient-server jsou například CVS (sekce 3.2) a SVN (sekce 3.3). Vývojáři pracující se systémy založených na architektuře klient-server pracují s jedním sdíleným repozitářem, se kterým se synchronizují. Aby ale vývojář zanesl změny do repozitáře, musí k němu být připojen, tedy být online. Tatáž nevýhoda platí pro návrat ke starším verzím projektu.

3.1.3 Distribuované verzovací systémy

Distribuované verzovací systémy jsou patrně nejsložitějšími systémy. Neexistuje žádný centrální repozitář – každý vývojář, který si kopii naklonuje, obdrží celý repozitář (včetně historie změn, takže může snadno přecházet mezi jednotlivými verzemi projektu), s nímž pak může dělat cokoliv chce. Veškeré změny se promítají lokálně a uživatelé si je dle libosti vyměnují mezi sebou.

Existence repozitářů, které se tváří jako server (například GitHub), by mohla budit dojem, že mezi systémy fungujícími jako klient-server a distribuovanými systémy neexistuje výrazný rozdíl. Je ale třeba uvědomit si, že tyto „centrální“ repozitáře jsou také jen dalšími klienty, kteří zkrátka přijímají změny na základě dohody. Začleněním změn třebas na GitHub se proto stále říká anglicky pull, což je stejná operace, jako když jeden klient stahuje data od jiného klienta.

Nejznámějšími distribuovanými verzovacími systémy jsou bezesporu Git a Mercurial. V dnešní době dokonce Git už předstihl i SVN v počtu projektů (obrázek 10.1).

3.2 CVS (Concurrent Versions System)

CVS je jedním z nejstarších verzovacích systémů – první verze, tehdy ještě ve formě shell skriptů, vznikla již v roce 1986, oficiální release napsaný v jazyce C pak v roce 1990. Poslední oficiální update vyšel v roce 2008 a je otázkou, nakolik je to z důvodu vyspělosti projektu, jak tvrdí například Jennifer Vesperman v knize Essential CVS [15], a nakolik z důvodu opuštěnosti, jak – možná poněkud nechtěně – dokládá oficiální bug tracker. [16]

CVS pracuje na architektuře klient-server a v mnoha ohledech je podobný systému SVN (více o jednotlivých rozdílech si probereme v sekci 3.3).

3.3 SVN (Apache Subversion)

Systém Apache Subversion (dříve pod názvem Subversion) vznikl v roce 2000 s cílem odstranit nedostatky staršího CVS a narozdíl od CVS dodnes vznikají nové verze (v době psaní této práce byla nejnovější verzí 1.9.0-beta1 vydaná 18. března 2015).

Jedním z největších rozdílů je atomicita commit operace (operace, která začleňuje změny do centrální repozitáře) – commit se provede buď celý, nebo vůbec, zatímco u CVS přerušení commitu vede k inkonzistenci celého repozitáře. Atomicita je u CVS totiž brána na bázi souborů, nikoliv celé revize. Občas se tedy může stát, že když vývojář začne commitovat několik souborů a ve zhruba stejnou dobu začne commitovat i někdo jiný, navzájem si začnou soubory přepisovat a vznikne konflikt, který je třeba ručně řešit, a dostat tím způsobem repozitář z nekonzistentního stavu.

SVN taktéž umožňuje nahrávat do repozitáře i binární soubory (CVS je vytvořeno pouze pro nahrávání textových souborů; toto omezení lze sice obejít, ale stojí to práci navíc). Existují další, například výkonnostní rozdíly, které ovšem práci tolik nenarušují, nebudeme se jimi proto dále zabývat.

3.4 Git

Git je projektem Linuse Torvaldse (autora Linuxu). Torvalds zastává poněkud radikální názory [17] na SVN, a snažil se proto Git udělat zcela jiným způsobem. Git není jen jedním nástrojem, ale obsahuje hned mnoho desítek programů, které dohromady tvoří jeden velký ekosystém. Z verzovacích systémů popsaných v této práci je Git zcela bezesporu tím nejsložitějším (což ostatně dokazuje i existence předmětů věnovaných pouze Gitu na několika fakultách ČVUT, nově pod vedením Ing. Petra Pulce ve zkušebním běhu i na FITu).

Nespornou výhodou je, že Git původně vytvářel člověk, který má ohromné zkušenosti s tvorbou velice rozsáhlého softwaru a vyzná se i ve výkonu souborových systémů, lze tedy očekávat, že věděl, co, jak a proč dělá. Díky této znalosti souborových systémů je Git rychlý a efektivní.

Git ale pravděpodobně nebude příliš vhodný pro menší týmy, či dokonce pro jednotlivce – některé jeho vlastnosti mohou působit značně zmatečně, nebo i zbytečně.

Git patří mezi distribuované verzovací systémy.

3.5 Mercurial

Mercurial je jakýmsi kompromisem mezi SVN a Gitem. Jedná se sice také o distribuovaný systém jako Git, zachovává ale většinu příkazů, které vývojáři znají z SVN – umí-li tedy uživatel pracovat s SVN, práce s Mercurialem jej nijak nepřekvapí. Narozdíl od Gitu je Mercurial monolitickým nástrojem sestávající z jednoho binárního souboru pojmenovaného `hg`. Schopnosti Mercurialu ve slučování změn, vytváření nových větví atp. jsou srovnatelné s Gitem, pouze nepatrně pomalejší. Mercurial je vytvořený v jazyce Python.

Kapitola 4

Rekvizity

Každý open-source projekt musí splňovat určité charakteristiky, pakliže to s ním jeho tvůrci myslí vážně. V této kapitole si postupně všechny potřebné rekvizity probereme.

Vývoj open-source softwaru se obvykle neřídí dle žádných standardů (vyjma licencí), následující postup tedy není třeba brát doslova a v uvedeném pořadí, jedná se spíše o vodítko, jakýsi seznam neoficiálních pravidel inspirovaný knihou Karla Fogela. [18]

4.1 Hosting projektu

Hosting projektu tvoří neopomenutelný základ; jedná se o web, na nějž vývojáři nahrávají to nejdůležitější – zdrojové kódy. Následuje abecedně seřazený seznam dnešních nejpopulárnějších hostingů:

4.1.1 Bitbucket

Služba Bitbucket byla spuštěna v roce 2008 s podporou Mercurialu. V roce 2011 byla následně implementována podpora Gitu. Bitbucket nabízí jak účet zdarma, tak i placenou variantu, která se od verze zdarma liší v počtu uživatelů, kteří se mohou účastnit vývoje projektu (pět vývojářů s možností zvětšit tento počet na osm přizváním tří nových uživatelů ke službě). V červnu 2013 Bitbucket dosáhl jednoho milionu uživatelů. [19]

4.1.2 GitHub

GitHub je v době psaní této práce (a pravděpodobně ještě po nějakou dobu dále bude) největším hostingem pro open-source projekty – v březnu 2015 čítal 8,9 milionů uživatelů pracujících na dohromady 20,8 milionech projektů. [20]. GitHub sice pro účty zdarma nenabízí soukromé projekty (narozdíl od zmíněného Bitbucketu), umožňuje však aktivaci studentského účtu, který skýtá

stejné výhody jako plně placený účet. Mezi spřátelené školy, jejichž studenti a zaměstnanci mají nárok na studentský účet, patří i FIT ČVUT. GitHub podporuje SVN a Git.

4.1.3 Google Developers (dříve Google Code)

Google Developers, služba dříve známa jako Google Code, je na tomto světě od března 2005 a podporuje práci s SVN, Gitem a Mercurialem. Tato služba dále nabízí některé prvky, které jiné služby neumí – jmenovitě například možnost zažádat o startup, zpěněžení obsahu, ale i nespočet tutoriálů k vývoji aplikací. Krom toho také jeden z projektů Google Developers – Google App Engine – slouží jako hosting webových aplikací. Tato nová verze už nicméně nepodporuje distribuci binárních souborů. Zajímavostí je, že přístup na stránku je zakázán návštěvníkům ze zemí, na něž Spojené státy uvalily sankce, kupříkladu Severní Korea či Kuba. Hosting tedy není vhodný pro vývojáře, jejichž cílová skupina leží v těchto zemích. Není přesně známo, kolik uživatelů a projektů Google Developers v současné době hostují; pravděpodobně už to však nikoho zajímat nebude, neboť tato hostovací služba bude bohužel 25. ledna 2016 končit svou činnost. [21]

4.1.4 Launchpad

Launchpad takřka jistě znají vývojáři pracující s unixovými systémy – tento hosting si zvolili tvůrci Linuxu pro jejich kernel. [22] Launchpad je mezi námi ještě déle než Google Developers – již od roku 2004. V červenci 2009 tvůrci dokonce i zveřejnili zdrojové kódy celého projektu. Služba podporuje verzování pouze pomocí nástroje GNU Bazaar – z ostatních nástrojů umí povětšinou data jen importovat. Launchpad nyní čítá přes 2,75 milionů uživatelů.

4.1.5 SourceForge

SourceForge je z této rešerše výrazně nejstarší hostovací službou – byla založena již v roce 1999. Podporuje také velké množství verzovacích systémů – CVS, SVN, Git, Mercurial a pro starší projekty ještě GNU Bazaar. Na SourceForge hostuje například MinGW, 7-zip a kodeky LAME a ffdshow. SourceForge do současnosti nasbíral 3,7 milionů uživatelů a hostuje přes 430 tisíc projektů. [23] Stejně jako Google Developers, ani SourceForge není přístupný pro státy na seznamu sankcí.

Hostovací služby se obvykle liší skutečně jen v drobnostech, nejčastěji v kvalitě jednotlivých dostupných nástrojů. V dnešní době nicméně vývojář neudělá chybu, zvolí-li si GitHub – už jen neuvěřitelné množství uživatelů je dobrým předpokladem ke kvalitní podpoře.

4.2 Issue tracker

Issue tracker je nástrojem, bez něhož nelze provádět žádný organizovaný vývoj. Většina hostovacích služeb nabízí vlastní issue tracker, takže obvykle vývojáři nepotřebují ještě při výběru hostingu přemýšlet jaký zařídit tracker – z hostingů popsaných v této práci obsahují vlastní issue tracking všechny. Díky této integraci lze navíc přiřazovat k problémům konkrétní commity či odkazovat na specifické řádky v souborech.

Issue tracker pravděpodobně není nejvhodněji zvoleným názvem, neboť kromě problémů (issues, bugs...) pomocí něj zaznamenáváme a sledujeme i žádosti o novou funkcionalitu, nevyžádané patche atp. Někteří vývojáři se proto možná setkali i s jinými názvy, například request tracker či defect tracker. V současnosti je ovšem nejpoužívanějším názvem issue tracker.

4.3 Continuous integration a build server

Představme si situaci, kdy jeden vývojář udělá změnu v kódu, ale protože je nedočkavý (nebo protože celý proces trvá příliš dlouho), vloží ji do repozitáře bez kompilace projektu nebo bez spuštění testů. Netuší tak, že někde udělal chybu. Druhý vývojář si kód z repozitáře stáhne a začne pracovat na jiné věci. Najednou mu neprocházejí testy, nebo projekt vůbec nelze zkompilovat. Druhý vývojář nyní ztrácí čas, který mohl věnovat práci, hledáním, kde se stala chyba.

V počátcích softwarového inženýrství znamenalo continuous integration pouhé slučování změn více vývojářů několikrát denně. Dnes už tento pojem nabral širšího významu, kdy CI většinou zahrnuje i použití tzv. build serveru. Právě build server značně usnadňuje práci vývojářů. Build server typicky bývá nejvýkonnější počítač, který máte k dispozici – musí totiž provádět spoustu výpočetně náročných úkolů. Jakmile někdo z vývojářů uloží změny do repozitáře, build server si automaticky stáhne kód, zkompiluje jej, spustí testy a případně ještě uloží ke stažení binární soubor či provede statickou analýzu kódu.

Pokud v některé fázi nastane chyba, vytvoří server podrobný log a zainteresovaným stranám pošle email. Vývojáři pak vědí, že v nejnovější verzi kódu je chyba a nemají tedy prozatím s repozitářem pracovat, dokud nebude chyba opravena. Vyřešení chyb při kompilaci by mělo mít nejvyšší prioritu každého týmu – taková chyba totiž v přímém důsledku zastaví veškerou práci programátorů. V neposlední řadě se build serverem odstraní situace, kdy jeden vývojář, který odevzdal svůj kód, při chybě kompilace na jiných strojích jen pokrčí rameny, řka: „U mě se to zkompiluje.“ Pokud se totiž kód nezkompiluje na build serveru, musí tento fakt vývojář řešit.

Ze známějších existujících řešení lze zmínit například AppVeyor [24] od stejnojmenné společnosti, TeamCity [25] od JetBrains nebo Bamboo [26] od

společnosti Atlassian (tvůrci známého verzovacího systému Bitbucket). Všechny tři zmíněné produkty mají placené verze, ale také verze zdarma pro open-source projekty (s různými podmínkami, které je nutno pročíst – u TeamCity například nesmí váš projekt nabízet placené verze).

4.4 Changelog

Důležité je dávat svým uživatelů pravidelně a konzistentně vědět, co je ve vydaných verzích nového. Typická situace, kterou zná pravděpodobně každý z IT oboru, je, že se vydá nová verze programu, ale nikde není k nalezení žádný seznam změn. Co teď? Přináší nová verze něco nového? Zásadní opravy? Hrozí, že update rozbije nějakou funkcionalitu, kterou potřebujete? Má tedy vůbec cenu update provádět? Toto vše jsou otázky, na něž musí changelog přinést odpovědi.

Každý changelog by měl obsahovat minimálně tyto tři věci:

- Nekompatibility s předchozími verzemi: Rozbila se (ať už úmyslně, či neúmyslně) nějaká funkcionalita? Stala se některá část veřejného API zastaralou? Pokud aplikace umožňuje ukládat stavy, budou stavy z předchozích verzí kompatibilní s novou? To vše je potřeba jasně uvést na pravou míru. Rozbitím uživatelova nastavení nebo výrazným narušením jeho zvyků s danou aplikací jej můžete velmi snadno ztratit; lze zdravě předpokládat, že konkurence ve vaší oblasti existuje a že danou činnost odvádí minimálně stejně kvalitně.

- Nové vlastnosti: Jakou novou funkcionalitu aplikace poskytuje? Byl zásadně vylepšen výkon? Byla přidána nová funkce do programovacího API? Na tyto otázky je dobré stručně a jasně odpovědět.

- Opravy chyb: Jaké chyby byly opraveny? Nejlepší je tento seznam uvést i s referencemi na příslušné položky issue trackeru. Pokud chybu nahlásil uživatel, je slušností jej v changelogu uvést s poděkováním (stačí obyčejné: „Díky Karlu N. za objevení chyby.“)

Je vhodné si uvědomit, že obyčejný seznam commitů není changelogem. Changelog by měl být psán čitelně – je určen pro uživatele, nikoliv pro stroje. Pakliže nechcete changelog zanést stovkami změn, stačí do něj uvést ty nejvýraznější a zveřejnit odkaz například na Wiki projektu, kde bude popsán i zbytek.

4.5 Mailing list

V dnešní době lze pravděpodobně namítnout, že mailing listy jsou již poněkud přežitek, a lze tedy dát přednost diskuznímu fóru. Diskuzní fóra mají

ovšem oproti mailing listům tu nevýhodu, že nutí uživatele aktivně je sledovat, zatímco mailing listy umožňují nastavit si snadno emailové notifikace – jejich periodicitu, témata... Zřejmě nejznámějším mailing listem jsou Google Groups, které dokonce umí poskytovat obsah z usenetových skupin.

K čemu ale mailing listy jsou dobré? Především se jedná o výchozí místo pro komunikaci s uživateli. Založením mailing listu dáváte světu najevo, že to s projektem myslíte vážně a máte starost o komunitu. Nikdo ve světě open-source nemá rád projekty, jejichž tvůrci veškeré diskuze provádějí soukromě v kanceláři. Karl Fogel ve své knize Tvorba open-source softwaru [27] na tento nešvar ostatně sám upozorňuje: Jakkoliv je lákavé a snadné probrat návrhy změn v uzavřené skupině pár vývojářů, měly by tyto porady probíhat v mailing listu. Ze začátku může být taková diskuze poněkud nešikovná, po čase si ale vývojáři zvyknou a zjistí, že komunita je schopná dívat se na záležitosti z úhlů, které nikoho z tvůrců nenapadly.

4.6 Readme

Dalším problémem je vytvoření souboru `readme`. `Readme` bývá typicky první informací, kterou uživatel vidí poté, co otevře stránku projektu. Měl by proto obsahovat stručný popis projektu (ideálně jeden odstavec – nechcete uživatele unudit haldou prozatím zbytečných informací; ve fázi čtení `readme` se uživatel teprve obvykle rozhoduje, zda mu aplikace vůbec k něčemu bude), jména a emailové adresy autorů, URL projektu, URL hostovacího webu, kde najít mailing list a kde najít issue tracker. Vhodné také je uvést v jednom odstavci licenci – většina zkušenějších vývojářů má v licencích přehled a zmínka již v `readme` jim usnadní nutnost klepnutí na licenční soubor.

Ve chvíli, kdy už je ke stažení použitelná verze softwaru, je taktéž vhodné do `readme` uvést postup stažení a instalace; pokud uživatelé ještě váš produkt nevyzkoušeli a nevědí jak ho jednoduše nainstalovat, je velmi pravděpodobné, že ztratí zájem a raději se poohlédnou po jiném, podobném, produktu.

4.7 Dokumentace

Psaní dokumentace sice nebývá snem každého vývojáře, jedná se nicméně o velmi podstatnou část projektu. I pokud se nechcete dokumentací nijak zvlášť zabývat, měla by obsahovat alespoň návod, jak produkt používat (třebas formou FAQ). Dokumentace zprvu nemusí být příliš rozsáhlá, ale je potřeba uživatelům dát do začátku alespoň nějaká vodítka – jinak je můžete ztratit ještě předtím, než se váš produkt naučí používat.

Pokud již máte nějaké zdrojové kódy, je možné alespoň k nim nechat dokumentaci vygenerovat automaticky. Programů, které dokumentaci na základě zdrojových kódů vytvoří, existuje několik desítek. Jedním z nejznámějších je Doxygen, [28] který lze spustit na operačních systémech Linux, OS X a Win-

dows a umí pracovat s jazyky C++, C, C#, Objective-C, Java, Perl, Python, IDL, VHDL, Fortran, Tcl a PHP.

4.8 Webová stránka (prezentace)

U tohoto bodu lze namítat, že webová prezentace je k ničemu, když už je projekt včetně `readme` uložen na webovém hostingu. Je ovšem dobré si říci, že například projekt na GitHubu s jedním textovým souborem na úvod není webová prezentace. Webová prezentace má sloužit k zaujmutí uživatele – musí mu během několika desítek vteřin být schopná vysvětlit, proč váš produkt potřebuje. Zkušenější autoři dovedou potenciálnímu zákazníkovi i vysvětlit nejen proč produkt potřebuje, ale už jen to, že ho vůbec potřebuje.

Ideálním místem, kde stránky založit, je dedikovaná doména (například `www.projectxyz.com` nebo `project.company.com`), pokud ovšem vývojář takovou doménu nemá k dispozici, některé hostovací weby poskytují možnost vytvořit webovou prezentaci u nich (GitHub a jejich GitHub Pages [29]). Nejlepší možností je dokonce nechat vytvoření prezentace na podobných nástrojích a svou dedikovanou doménu úpravou DNS záznamu na tuto prezentaci přesměrovat.

Tvorba atraktivní webové prezentace nicméně není tématem této práce, rád bych proto odkázal na kvalitní článek i s názornými příklady. [30]

4.9 Ohlášení (announcement)

Na závěr je potřeba projekt ohlásit. Pošlete odkaz na projekt svým známým, které by projekt mohl zajímat, vložte jej do mailing listů, kde je váš příspěvek tematický. Dejte si pozor, abyste nepřekročili hranici, za níž již budete vyhodnoceni jako spammeři. Ohlášení také dělejte v okamžiku, kdy již máte co prezentovat.

Kapitola 5

Užitečné nástroje a SW doporučení

V této kapitole si ukážeme některé nástroje a doporučení, které sice nejsou nezbytné, ale jejich správné použití dokáže usnadnit komunikaci v týmu, či dokonce přímo pracovní výkon.

5.1 Git-flow

Git-flow [31] je velmi užitečný nástroj pro týmy pracující s Gitem. Jedná se o sadu rozšíření, která zjednodušuje práci s tímto nikoliv zcela triviálním verzovacím systémem. Git-flow lze nainstalovat na počítače používající OS X, Linux a Windows.

Práce s Gitem se pak stává skutečně jednoduchou – kupříkladu novou funkcionalitu lze začít vyvíjet příkazem `git flow feature start MYFEATURE`, který vytvoří novou větev (branch) štěpící se z větve `develop` a přepne se na ni. Poté, co vývojář dokončí práci, lze novou větev odstranit a změny začlenit zpět do `develop` příkazem `git flow feature finish MYFEATURE`.

Vývojáři pracující na jedné funkcionalitě v týmu také ocení jednoduchý příkaz `git flow feature publish MYFEATURE`, která v podstatě nahrazuje commit a push do vytvořené větve. Analogicky k tomuto existuje samozřejmě i příkaz `git flow feature pull origin MYFEATURE`.

Git-flow pochopitelně umí také vytvářet release z vývojové větve pomocí příkazu `git flow release start RELEASE [BASE]`, kde `[BASE]` je SHA-1 hash commitu nacházejícího se na `develop` větvi, z něhož chceme release vytvořit. Tento parametr je však nepovinný – Git-flow jinak vyrobí release z posledního commitu, který pak uživatel uvolní příkazem `git flow release publish RELEASE`. Posledním krokem pak je začlenit release do větve `master` – `git flow release finish RELEASE`.

Pro více příkazů je vhodné navštívit oficiální dokumentaci. [31]

5.2 Gitter

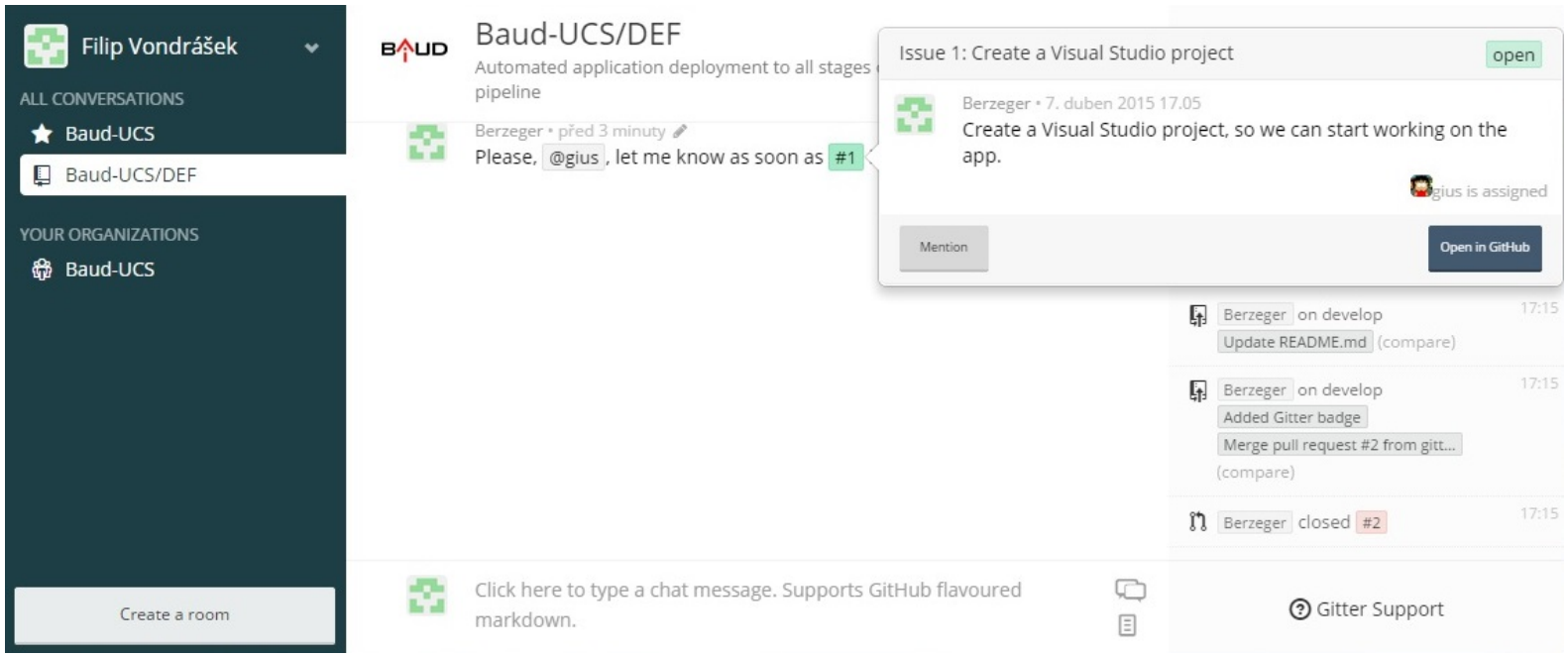
Gitter je chatovací nástroj určený pro uživatele hostovacích služeb GitHub a Bitbucket. Častým problémem u issue trackerů na hostovacích službách bývá, že se často zanesou množstvím komentářů, které pak i občas vybočují z původního tématu. Gitter si klade za cíl tento problém řešit integrací samostatného chatu. Názornou ukázku integrace Gitteru s GitHubovým projektem lze vidět na obrázku 5.3.

V placené verzi Gitteru (5 dolarů za uživatele za měsíc) lze navíc získat neomezeně dlouhou historii chatu, neomezený počet místností, uživatelů v chatech a integrací se službami.

5.3 HipChat

Podobným nástrojem, jako je Gitter, je také HipChat. [32] Tato služba se dokáže napojit nejen na hostovací weby, ale také na projekty typu Twitter atp. HipChat navíc nabízí desktovou a mobilní aplikaci a ve verzi zdarma skupinový chat, IM, sdílení souborů (s pětigigabajtovým úložištěm) a neomezený počet uživatelů a integrací.

Placená verze, jež stojí 2 dolary za uživatele za měsíc, k tomu poskytuje sdílení obrazovky, video chat a další možnosti, mezi něž patří i neomezený diskový prostor pro soubory.



Obrázek 5.1: Označování záznamů z issue trackeru na Gitteru.

5.4 Semantic Versioning (SemVer)

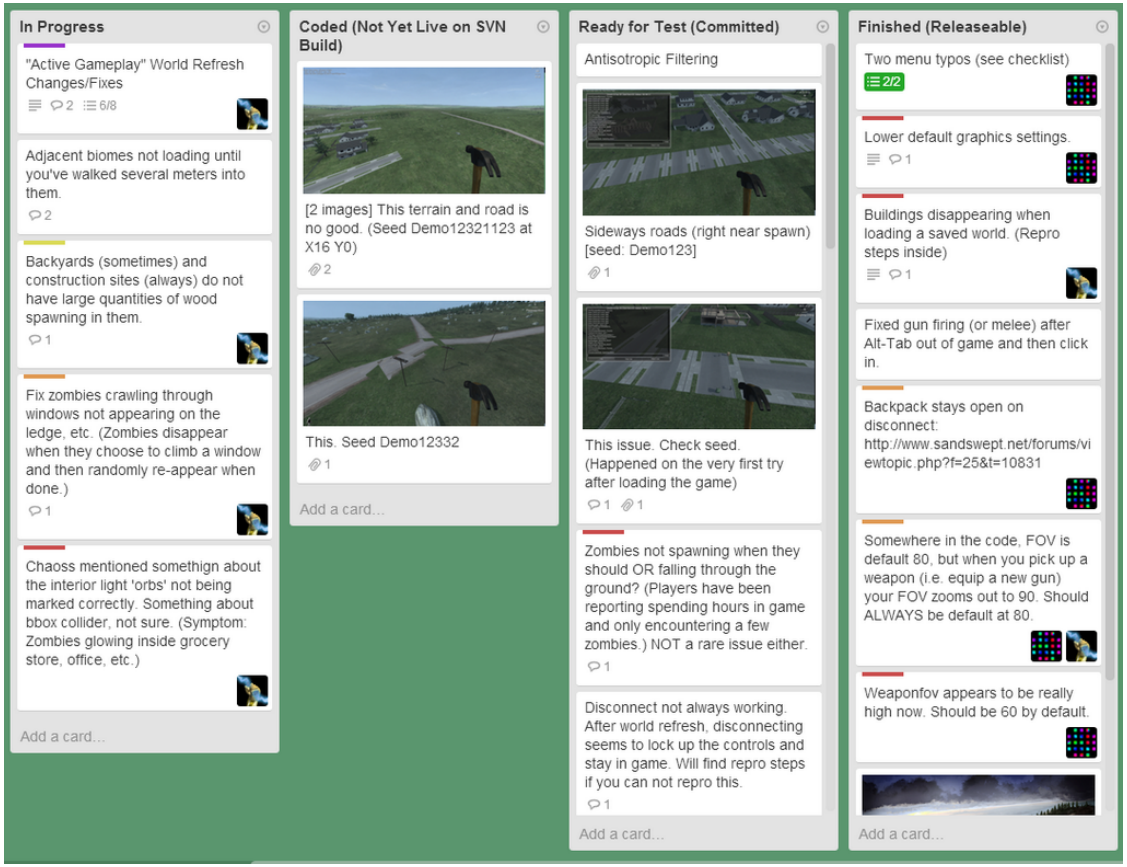
Semantic Versioning není nástrojem v pravém slova smyslu, jedná se o specifikace, jakým způsobem by se měly označovat verze programů – v podstatě se snaží definovat jakýsi standard. Semantic Versioning ve své verzi 2.0.0 udává, že jednotlivé verze programů by se měly označovat x.y.z, kde x označuje major verzi, y minor verzi a z patch.

Major verze znamená nekompatibilní změny v API – tj. pokud vyjde nová verze programu s jinou major verzí, při aktualizaci na straně uživatele bude třeba aktivně provést změny. Minor verze označuje změny API, které ovšem nezpůsobí zpětnou nekompatibilitu. Nakonec patch je patch ve skutečném slova smyslu – opravuje chyby, ale nemění nijak stávající API. Dále platí, že pokud je zvýšena major verze, je nutné vynulovat minor verzi a patch. Stejně tak pokud se zvýší minor verze, je třeba vynulovat patch. Ve verzi 2.0.0 navíc přibyla možnost označovat ještě uvolněné verze dalším řetězcem alfanumerických znaků, které označují například prerelease (i samotný SemVer se těchto pravidel drží, a tak lze nalézt verzi těchto pravidel pod označením 2.0.0.-rc.2). Oficiální popis je však mnohem podrobnější a obsahuje další významné detaily, více informací viz [33].

Dokonce existuje i open-source nástroj GitVersion [34], který je schopen ve vašem Git repozitáři verze číslovat podle pravidel SemVer zcela automaticky dle historie repozitáře.

5.5 Trello

Trello je užitečným nástrojem pro ukládání poznámek všech druhů. Poznámky lze rozdělit do několika „bloků“, například je lze tedy třídit dle jednotlivých projektů, na nichž uživatel pracuje. Ihned je patrné, na čem se zrovna pracuje, kdo danou věc dělá a obecně kde se něco děje. [35] Ke každé poznámce lze přiřazovat komentáře, štítky a různé barvy pro přehlednost, dále je možné je řadit i do různých kategorií. Přiložený snímek obrazovky ukazuje již zařízený Trello účet od vývojářů počítačové hry The Dead Linger (obrázek 5.2).



Obrázek 5.2: Trello účet hry The Dead Linger.

Zdroj: `https://twitter.com/thedeadlinger/status/488503917249323008/photo/1` [36]

Kapitola 6

Vedení vývojového procesu

V této kapitole probereme možnosti vedení vývojového procesu, když už jsou zařízeny počáteční náležitosti. Vedení lze rozdělit na dvě fáze – první, uzavřenější, a druhou, směrovanou již více na komunitu.

6.1 První fáze vývoje

Při přechodu od uzavřeného softwarového projektu k open-source bude mít hlavní slovo ve vývoji společnost, jíž projekt patří, a nikoliv komunita. Tato společnost má připravenou nějakou vizi, návrh a architekturu, které je potřeba realizovat podle stanoveného plánu. Je sice možné navrhnout ideu, co by produkt měl zhruba dělat a začít rovnou programovat s komunitou, pravděpodobně ale vývoj pak nebude probíhat zcela optimálně a z kódu se po chvíli stane nepřehledný zmatek. První fáze vývoje se tedy neliší příliš od klasického stylu vývoje v softwarové společnosti. S nejvyšší pravděpodobností stejně první kroky bude provádět pouze společnost, nejedná-li se o příliš známou firmu, nebo pokud neudělala nějaké výjimečně dobré ohlášení projektu.

První fáze přechází v druhou zhruba v okamžiku, kdy už je komunita natolik rozrostlá, že je schopna bez větších sporů a problémů fungovat. V tuto chvíli již bude muset společnost brát názor komunity mnohem vážněji a k lidem se chovat rovnocenně, jinak se může stát, že o ně rychle přijde.

6.2 Druhá fáze vývoje

Na celém vedení vývojového procesu open-source projektu je nejdůležitější především management lidských zdrojů. Rozdíly mezi prací s týmem v jedné kanceláři a s týmem, který se skládá z jedinců mnoha kultur jsou výrazné. Dalším významným rozdílem je skutečnost, že zatímco v jedné společnosti na projektu pracují lidé typicky proto, že jsou k němu přiřazeni zaměstnavatelem (či zadavatelem), na open-source projektu lidé pracují hlavně ze samotného

zájmu. Přístup manažera projektu tedy musí být odlišný – navíc je třeba si uvědomit, že přispěvatelé, a to bohužel i ti nejkvalitnější, mohou z projektu vystoupit kdykoliv budou mít pocit, že jejich práce není dostatečně ceněna, nebo že se projekt začal ubírat cestou, s níž nejsou spokojeni.

Co ovšem jen málokdo přizná je skutečnost, že zájem o projekt je sice důvod velký, ovšem nesmí se opomenout i důvody lidské – každý člověk, ať je jakéhokoliv charakteru, touží po uznání od ostatních, po tom, aby jeho hlas měl nějakou váhu. Vedení vývojového procesu se tak stává napůl psychologickou prací; je nutné vnímat silné a slabé stránky spolupřispěvatelů, jejich charakterové rysy a umět na tyto lidi vhodně reagovat. Každý člověk vnímá kritiku jinak. Podobným uměním je i přidělování úkolů jednotlivým lidem.

6.2.1 Druhy vedení

Záleží samozřejmě na tom, jakým stylem je vývoj veden. Nejčastějšími možnostmi je autoritativní a demokratický přístup a různé prolínání těchto dvou přístupů. Každý přináší určité výhody a nevýhody, nicméně praktický vývoj často začíná právě autoritativním přístupem a postupně se přeléva do demokratického.

6.2.1.1 Autoritativní přístup

Minimálně ze začátku projektu by jej měla vést nějaká zvolená autorita, která by nikoliv přímo rozhodovala, co se má či nemá dělat, ale ideálně by měla dohlížet na dodržování daných postupů a mít poslední slovo v situaci, kdy se vývoj začíná rozštěpovat do dvou či více směrů a nelze nalézt shodu. V případě softwarové společnosti, která se rozhodla přejít na open-source vývoj a typicky mívá nějaký předem daný plán, je situace o něco jednodušší, protože přispěvatelé už předem věděli, do čeho jdou, a tedy jen těžko začnou vymýšlet vlastní pravidla či postupy práce.

6.2.1.2 Demokratický přístup

Postupně, jak práce pokračují, je typický pomalý přesun od autoritativního modelu k demokratickému. Není to proto, že by vedoucí nutně dělal svou práci špatně, ale jak čas plyne, skupina si zvykne na zaběhnutý model a vytvoří jakýsi ekosystém, který sami, ať už vědomě, či nevědomě, dodržují a snaží se v něm udržet rovnováhu. Problém nastává v situaci, kdy se zjistí, že se skupina začíná trhat a začíná docházet k neshodám – potom je potřeba autoritativní přístup obnovit.

6.2.2 Rozdělování úkolů

Rozdělování úkolů není jen o prostém přiřazení práce člověku, o němž si myslíte, že je vhodným kandidátem na daný problém. Hlavním důvodem, proč

vůbec úkoly delegovat, je úspora vašeho vlastního času – někdo jiný udělá úkol, který by jinak ležel na vašich bedrech. Vedlejším efektem je pak to, že člověk, na něhož úkol delegujete, ví, že jste v něj vložili důvěru; o to více, pokud jste jej o splnění požádali prostřednictvím veřejné diskuze (Gitter, Trello, fórum, mailing list...). Je ovšem velice důležité formulovat požadavek tak, aby nevyzněl jako rozkaz, ale jako prosba, jíž je možno odmítnout – typicky nechcete, aby se někdo stresoval s úkolem, kterému pořádně nerozumí, jen proto, že měl pocit, že jej udělat musí.

Čas od času se vyplatí dokonce i přidělit úkol nějakému vývojáři, i přestože víte, že byste ho sami zvládli rychleji nebo dokonce lépe – stydlivého člověka s potenciálem tak více zapojíte do veřejného dění, čímž aktivně zlepšujete jeho schopnosti. Jak bylo uvedeno na začátku kapitoly, management lidských zdrojů je z velké části o psychologii.

Velikou drzostí je vyplnit do bug trackeru novou funkcionalitu nebo chybu a bez předchozí dohody rovnou k úkolu někoho přidělit. Naznačujete tím, že je jasně jejich odpovědností se o úkol postarat. Nikdo ale nemá rád příkazy, o to méně u open-source projektu, do něhož lidé přispívají nejčastěji ze zájmu ve svém volném čase. Lidé obvykle sami dobře vědí, kdo je na co expert, a když přibude v trackeru nějaký záznam, sami se k němu rádi přiřadí. Nastane-li ovšem situace, že se k úkolu nikdo příliš nemá, je vhodné nechat jej nepřiřazený, ale přidat k němu poznámku typu: „Této oblasti dobře rozumí XYZ, chtěl byste na tomto úkolu pracovat? Pokud na něj nemáte čas, klidně jej odmítněte (pokud byste byl rád, kdybyste do budoucna podobné žádosti nedostával, dejte prosím vědět).“ Vývojář pak necítí povinnost úkol přijmout a navíc mu lehce zalichotíte. Pokud komentář zůstane bez odezvy, je dobré pár dní počkat a pak se připomenout. Lidé tak uvidí, že máte o projektu přehled a víte, co se neustále děje.

6.2.3 Kritika a chvála

Přestože by se mohlo zdát, že se jedná o přímé protiklady, kritika a chvála mají ve skutečnosti mnoho společného – obojí jsou formy pozornosti a jsou efektivnější, pokud jsou konkrétní a nikoliv obecné. Obě tyto formy mají tendence k inflaci – chvalte příliš často a vaše chvála po chvíli nebude příliš znamenat. Totéž platí pro kritiku, ta je ovšem k inflaci poněkud odolnější.

6.2.3.1 Kritika

Je bezpodmínečně nutné, aby kritika splňovala dvě kritéria – musí být detailní a bez emocí. Udělá-li například vývojář zbytečnou chybu, je bezvýsledné (a dokonce i kontraproduktivní) připsat k ní kritiku: „Tak toto byla zbytečná chyba.“ Tímto způsobem totiž nekritizujete chybu, ale spíše přímo onu osobu; jenže kritika se musí zaměřovat na práci, ne na lidi. Je nutno lidsky a slušně popsat, co je na odevzdané práci špatně a jak se příště podobné chybě vyhnout.

Pokud se ovšem jedná již o několikátou chybu, jíž se dalo jednoduše předejít, v řadě od jednoho člověka a nezdá se, že by detailní kritiku bral v potaz, je nejlepší ho buď přeřadit na jinou oblast projektu, či se s ním zcela rozloučit. Zde bohužel pravděpodobně neexistuje způsob, kterým to udělat, aniž byste se jej nedotkli.

6.2.3.2 Chvála

Jak už bylo zmíněno v sekci 6.2.3, kritika a chvála mají mnoho společného. Dokonce i detailní kritika bez emocí bývá často vnímána jako druh chvály, neboť implikuje, že se vývojářovým kódem někdo zabýval, někdo si s ním dal čas a námahu. Je ovšem nutné, aby taková kritika splňovala oba popsané body, tj. detailnost a absenci emocí.

Dále bylo zmíněno, že se má chvála, stejně jako kritika, užívat opatrně (u chvály zejména kvůli inflaci). Rozmyslete si, proč vlastně chcete chválu použít. Pravidlem palce je, že byste neměli chválit za samozřejmé věci, u nichž očekáváte, že je vývojář vykoná, nebo za věci, které jsou považovány za normální. Například nebudete chválit vývojáře, že dodržel pravidla čitelnosti kódu. Pokud byste takovou věc dělali, bylo by navíc velmi těžké poznat, kdy přestat – měli byste pak chválit všechny ve skupině za standardní věci? Chvála by naopak měla být používána jako reakce na neobvyklé či neočekávané (v pozitivním slova smyslu) jednání. Lidé si toho všimnou, což podpoří další nadstandardní výkony, to vše ale samozřejmě pouze, pokud to s chválou nebudete přehánět.

6.2.4 Postupné převzetí „nadvlády“ nad částí projektu

Může se stát, že se k projektu připojí expert (ať už opravdový, nebo si to o sobě jen myslí) na konkrétní oblast. Je v pořádku, pokud se tento člověk stará výhradně o svou oblast, problém však nastává v okamžiku, kdy začne agresivně přebírat práci cizích lidí. Ostatní vývojáři pak vycítí, že nejsou tímto člověkem v dané části projektu vítáni, a agresor zůstane sám, čímž nejenže se rozpadá komunitní duch, ale především se část projektu takto může stát slabinou, neboť tam přestává existovat kontrola kódu.

S tímto fenoménem je obtížný, ale nikoliv nemožný boj. Kupříkladu v Apache Software Foundation zavedli systém, kdy se do zdrojových kódů neuvádějí jména autorů. Člen ASF Sander Striker se k problému vyjádřil takto:

> „V Apache Software Foundation nechceme mít autorské značky ve zdrojových kódech. Mimo právních důsledků existuje více důvodů. Komunitní vývoj spočívá v práci na projektech ve skupině a ve starosti o projekt jakožto skupina. Uznání práce je dobrá věc, která by měla být dělána, ale způsobem, který neumožní přisouzení zásluh nesprávné osobě, i když jen náznakem. Neexistuje žádná jasná hranice, kdy přidat, či odebrat autorskou značku. Přidáte

> své jméno, když jen upravíte komentář? Když přidáte jednořádkovou opravu? Odstraníte autorskou značku, když refaktorujete kód a z 95 % vypadá jinak? Co děláte s lidmi, kteří sáhnou na každý soubor a změní přesně tolik, aby mohli přidat své jméno, takže se pak jejich jméno vyskytuje úplně všude?
>
> Jsou lepší způsoby, jak ocenit práci, a my se rozhodli využít je. Z technického hlediska jsou autorské značky zbytečné; pokud chcete zjistit, kdo napsal konkrétní kus kódu, řekne vám to verzovací systém. Autorské značky navíc mají tendence zastarávat. Opravdu si přejete být kontaktováni soukromou cestou kvůli kusu kódu, který jste napsali před pěti lety a jste rádi, že jste jej zapomněli?“
>
> (Sander Striker, Apache Software Foundation [37])

Pravděpodobně některé přispěvatele odradí, že jejich jméno nebude viditelné v souborech, ale jsou vývojáři, kteří pracují především kvůli slávě, ti, které opravdu u projektu chcete mít?

Kapitola 7

Zpracování zdrojového kódu

Zpracování zdrojového kódu v open-source projektu se příliš neliší od zpracování kódu v proprietárním softwaru. Je ale potřeba jasně uvést pravidla, kterými se psaní kódu bude řídit (narozdíl od proprietárního vývoje, u něhož jsou vývojáři s určitými nařízeními seznámeni globálně v rámci pravidel společnosti).

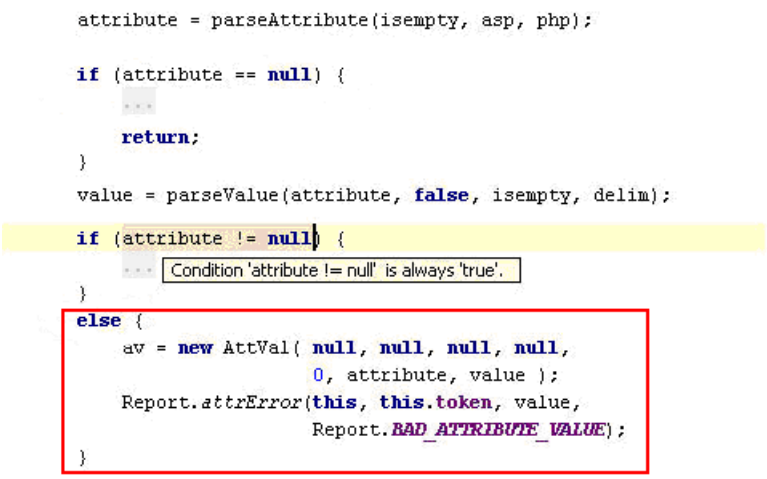
V této kapitole si řekneme, jakým způsobem by vývojáři měli s pravidly být seznámeni, dále jak kód testovat a proč a jak je vhodné odstranit proprietární moduly.

7.1 Kvalita kódu

Zdrojový kód v celém projektu musí dodržovat uniformní pravidla. K tomu může dopomoci například volitelný soubor `CONTRIBUTING` umístěný v kořeni repozitáře. V tomto souboru by administrátor projektu měl uvést, jakými pravidly se kód řídí a co vše je možné udělat, aby se zvýšila šance na akceptování uživatelských změn. Konkrétní příklad takového souboru lze nalézt v sekci 10.4.

Nedílnou součástí zdrojového kódu by měly být i testy. Není možné ručně testovat vše, testy proto musí být automatizované. Představte si situaci, kdy změníte kus kódu a otestujete ručně modul, který byl změněn (například otestování konkrétní webové stránky v prohlížeči, programujete-li webovou aplikaci). Nemůžete ale vědět, že jste změnou v kódu nerozbili nějakou zdánlivě nesouvisející část. Není v lidských silách po každé úpravě ručně testovat všechny možnosti – takové testování by zabralo více času než samotné programování.

Pokud je testů tolik, že už by zabraly při každé menší změně v kódu příliš mnoho času, lze testování nechat na build server (sekce 4.3).



Obrázek 7.1: Detekce nedosažitelného kódu v IntelliJ IDEA

Zdroj: `https://www.jetbrains.com/idea/documentation/static_code_analysis.html` [38]

7.1.1 Statické testování

Statické testování znamená analýzu kódu (v některých případech i objektového kódu, tj. binárního souboru, bytekódu atp.) bez nutnosti jeho spuštění.

Statické analyzátory umí nalézt pravděpodobné chyby (možnost `null` parametru tam, kde by neměl být), odhalit nedosažitelný kód (`else` větev podmínky, která ovšem bude vždy splněna – viz obrázek 7.1), výkonnostní problémy (řetězení stringů namísto využití třídy `StringBuilder`, použití rekurze, obvykle koncové, v situacích, kdy by iterace byla výhodnější), ale také problémy, které mají vliv na čitelnost (nedodržování standardů) či udržovatelnost kódu (duplicitní úseky).

Jistou formu statického analyzátoru dnes již implementuje většina moderních IDE (NetBeans, Visual Studio, IntelliJ Idea…), tyto analyzátory však obvykle s výchozím nastavením jen kontrolují vyložené chyby, které by jinak vrátil zpět kompilátor. U zmíněných NetBeans lze zapnout podrobnější statický analyzátor pro Javu, [39] pro Microsoft Visual Studio pak existuje kvalitní, leč bohužel placený, plugin ReSharper od JetBrains fungující pro jazyky C#, VB.NET, XAML, XML, ASP.NET, ASP.NET MVC, JavaScript, HTML a CSS. [40]

7.1.2 Dynamické testování

Dynamické testování poté naopak se spuštěním kódu počítá. Mezi prostředky dynamického testování patří typicky jednotkové testy, integrační testy, systémové testy a akceptační testy, v open-source vývoji jsou pak klíčové zejména

první dva jmenované druhy.

7.1.2.1 Jednotkové testy (unit testy)

Jednotkové testy testují odděleně jednotlivé části zdrojového kódu prostřednictvím předem vytvořených zkušebních (odborně „mock“ nebo „mocking“) dat. Tento druh testů ověřuje zejména správnost algoritmů – programátor například napsal matematickou knihovnu a chce zjistit, zda mu správně pracuje s maticemi. Tak vymyslí, či odněkud opíše pár matic a zkouší, zda se matice mezi sebou správně násobí, zda se správně vypočítá determinant, inverzní matice atp.

K jednotkovým testům se také váže pojem *assert*. Právě asserty jsou základní složkou těchto testů – jedná se o predikát (tvrzení typu pravda-nepravda), který na daném místě kódu musí být splněn. Pokud splněn není, program vrátí chybu. Zůstaňme u příkladu s maticemi a za predikát zvolme třeba existenci inverzí matice nebo hodnotu determinantu jedna.

Asserty lze ovšem použít nejen v testech, ale i v kódu jako takovém, typicky pro debugovací účely. Není nicméně vhodné asserty testovat části závislé na uživatelském vstupu – pokud nastane chyba na vstupu, měl by být uživatel upozorněn například dialogovým oknem a nikoliv hláškou: „Assertion failed in xyz.cpp at line 123.“

7.1.2.2 Integrační testy

Integrační testy následují po jednotkových testech. Jejich účelem je ověřit funkčnost nikoliv oddělených částí, jako je tomu právě u testů jednotkových, ale zda všechny části správně fungují, když se propojí dohromady. Integrační testy kupříkladu budou využívat vlákna, databázi, či jiné prostředky pro zajištění, že kód a změny prostředí správně fungují.

Vezměme si například serializaci. Jednotkovým testem ověříme, že serializovaný objekt se správně deserializuje (a naopak), integračním testem pak ověříme, že se serializovaný objekt správně zapíše na disk a také z něj zpět načte.

7.2 Odstranění proprietárních modulů

Při migraci proprietárního softwaru na open-source projekt může nastat situace, kdy některé moduly vývojář nechce uvolnit. Důvodů může být několik:

- Jejich zveřejněním by až příliš ukazoval firemní know-how či tajné údaje (přístupové údaje, URL interních serverů atp.).
- Moduly mohou být licencovány jinou společnosti/vývojářem (typická situace, pokud vývojář používá nějaký middleware) a není tedy právně možné je uveřejňovat, a to ani ve formě zkompilované knihovny.

- Modul je příliš orientovaný na konkrétní prostředí a kupříkladu vzhledem k jeho stáří je jednodušší jej nahradit nežli modifikovat – jeho původní vývojáři už dávno mohou pracovat jinde, nebo zkrátka již zapomněli, jak kód funguje (a chybí kvalitnější dokumentace).

Nejjednodušeji se proprietární moduly nahradí open-source variantami, pokud je aplikace psaná ve stylu *interface-oriented programming*; tento styl programování bývá obecně využíván v softwaru, který je určitým způsobem založen na modulech (například při programování pluginů pro Eclipse je potřeba implementovat dané rozhraní – i *Java Development Tools* pro Eclipse jsou pluginem). V takovém případě pak stačí najít open-source modul, nebo naprogramovat své nové řešení, které splňuje původní funkcionalitu, a napojit jej na stávající rozhraní.

Častými proprietárními moduly, které společnosti nechtějí uvolnit, jsou vlastní či upravené databázové systémy, mailovací systémy nebo knihovny, které používá celá firma a obsahují všelijaká vylepšení či užitečné metody.

Část II

Praktická část – Deployment Framework

Kapitola 8

Návrh a analýza

V předchozích kapitolách uvedený postup migrace na open-source je demonstrován na příkladě aplikace, jejíž zadání dodal externí zadavatel, firma Baud, spol. s.r.o. Následující specifikace a analýza byla provedena autorem práce a schválena společností Baud.

Rozhodnutí přejít na open-source předcházela dlouhá debata, kdy byla vyložena jasná pro i proti. V podstatě byly shrnuty důvody popsané v sekcích 1.2 a 1.3, ovšem naprosto legitimním důvodem, který zazněl i na zmíněné debatě, je i prosté „chtěli bychom si to vyzkoušet.“ Open-source dnes poměrně rychle nabývá takových rozměrů, že dokonce i společnosti zakládající si na svém proprietárním softwaru, začaly přecházet s některými svými projekty na open-source (například .NET framework od Microsoftu [41]).

8.1 Účel

Po vyvinutí softwarového produktu je třeba jej nasadit u zákazníka. Jelikož produkt bývá často řešen na míru, jeho instalace a aktualizace obvykle obnáší buď cestu technika za zákazníkem a manuální instalaci, případně instalaci přes vzdálenou plochu či jiný podpůrný software. To je sice řešení jednodušší, stále ale technik musí znát zákazníkovo prostředí a umět se orientovat v jeho zařízení. Situace se pak stává ještě komplikovanější, když vzroste počet zákazníků a/nebo počet vyvíjených produktů.

Účelem DEF je zjednodušit a zčásti zautomatizovat proces instalace jednotlivých produktů prostřednictvím centralizace zákazníků, produktů a instalací. Systém umožňuje automatickou instalaci libovolných verzí produktů dle zadaných skriptů na různá prostředí, kterými typicky bývají například testování, vývoj či produkce. K tomu bude dopomáhat i správa příslušných entit, jak je podrobněji popsáno v sekci 8.3.

Výhody jsou ovšem patrné i v případě, že se softwarová společnost stará o jediný produkt. Díky možnosti nastavovat jednotlivá prostředí se výrazně zjednoduší správa více verzí jednoho produktu a jejich posouvání na prostředí

další. DEF tedy usnadní cestu jednotlivých verzí produktu od vývoje přes testování až po produkci.

8.2 Přehled uživatelů

8.2.1 Vývojáři

Nasazují na developerská a testovací prostředí.

8.2.2 Osoby odpovědné za konkrétní produkty

Nasazují na UAT a produkční prostředí.

8.2.3 Administrátoři serverů

Nastavují servery a konfigurace.

8.2.4 Administrátoři DEF

Spravují uživatele a práva v DEF.

8.2.5 Správci produktů

Spravují produkty a aplikace.

8.2.6 Projektoví manažeři

Mají přehled o aktuálně nainstalovaných aplikacích pro konkrétní produkt.

8.3 Funkční požadavky

8.3.1 Dashboard

Dashboard je určen k přehledu veškerých produktů a jejich aplikací, prostředí a instalací. Jednotlivé entity přiřazené produktům jsou zobrazeny formou tabulek, kde řádky představují jednotlivé aplikace, sloupce představují prostředí a buňky (dlaždice) představují instalace. U jednotlivých instalací je možnost vybrat libovolnou verzi dané aplikace a tu na instalaci nasadit. Instalaci lze navíc odložit na vybrané datum a čas navíc nastavit, zda se akce má pravidelně opakovat. Dále lze z jednotlivých instalací zjistit, zda poslední nasazení skončilo úspěšně či neúspěšně a kdy se tak stalo, jaká je na ní současně nasazená verze, na kterém serveru se nachází a pokud je dostupná její URL, pak i odkaz.

8.3.2 Packages (balíčky)

Sekce Packages umožňuje zobrazení veškerých dostupných balíčků a informací k těmto balíčkům přidružených. Konkrétně se jedná o název, popis, dostupné verze, datum zveřejnění, závislosti na jiných balíčcích a informace o tom, kde je balíček v současnosti nasazen. Dále stránka obsahuje informaci pro uživatele, kam a jak nahrávat balíčky nové.

8.3.3 Products (produkty)

Sekce Products slouží ke správě produktů a jim přiřazených entit. Správou se rozumí možnost přidávat, upravovat a mazat produkty a taktéž přidávat, upravovat a mazat aplikace, prostředí a instalace. U produktů uživatele zajímá název, popis, priorita a případné poznámky. Dále se u jednotlivých produktů zobrazují přidružené entity popsané níže.

8.3.3.1 Environments (prostředí)

Uživatelé mají možnost vidět a upravovat název, popis a prioritu.

8.3.3.2 Applications (aplikace)

Uživatelé mají možnost vidět a upravovat název, popis, prioritu a přidružený balíček.

8.3.3.3 Installations (instalace)

Uživatelé mají možnost vidět a upravovat přidruženou aplikaci, prostředí a server a dále umístění, URL a případné poznámky.

8.3.4 Servers (servery)

Sekce Servers umožňuje správu jednotlivých serverů, tzn. uživatelé mají možnost přidávat, upravovat a mazat servery. Také je zde možnost provést health check vybraného serveru. Mezi informace, které uživatele zajímají, patří název, URL, API klíč, datum a čas poslední synchronizace serveru, současně nainstalovaná verze skriptů, indikace, zda je tato instalovaná verze aktuální, a dále níže popsané entity.

8.3.4.1 Logs (logy)

Uživatelé mají možnost číst logy, v nichž jsou zaznamenány informace o nasazeních proběhlých na daném serveru. Jedná se zejména o datum a čas jednotlivých úkonů, stručný průběh a výsledek nasazení.

8.3.4.2 Sites (umístění)

Uživatelé mají možnost vidět seznam umístění dostupných na serveru.

8.3.4.3 Configuration (konfigurace)

Uživatelé mají možnost číst, přidávat, upravovat a mazat jednotlivé konfigurace serveru ve formátu klíč - hodnota. V případě, že je konfigurace již uložena, není možnost upravovat její klíč, pouze hodnotu. Sekci lze dále filtrovat dle jednotlivých umístění – příslušnost konfigurace k umístění se pozná podle toho, že klíč má jako předponu název umístění zakončený tečkou. Tedy například klíč Test1.ConnectionString má příslušnost k umístění Test1.

8.3.5 Security (zabezpečení)

Sekce Security je rozdělena na tři provázané části – uživatelé, pozice a role:

8.3.5.1 Users (uživatelé)

Podsekce Users slouží ke správě uživatelů. Umožňuje přidávat a upravovat uživatele, nikoliv však mazat. Zobrazovanými informacemi jsou indikace, zda je uživatel aktivní, jeho přihlašovací jméno, křestní jméno, příjmení, email, pozice, k nimž je uživatel přiřazen, datum a čas odkdy a dokdy je aktivní a případné poznámky. Všechny položky kromě první zmiňované jsou upravené. Indikace, zda je uživatel aktivní, je vypočítána z položek odkdy a dokdy je aktivní.

8.3.5.2 Positions (pozice)

Podsekce Positions slouží ke správě pozic. Umožňuje přidávat a upravovat pozice, nikoliv však mazat. Zobrazovanými informacemi jsou indikace, zda je pozice aktivní, název pozice a přiřazení uživatelé, role a prostředí. Na rozdíl od sekce 8.3.5.1, indikátor aktivity je zde ručně měnitelný.

8.3.5.3 Roles (role)

Podsekce Roles slouží ke správě rolí. Umožňuje přidávat a upravovat role, nikoliv však mazat. Zobrazovanými informacemi jsou indikace, zda je role aktivní, název role a přiřazené pozice a práva. Na rozdíl od sekce 8.3.5.1, indikátor aktivity je zde ručně měnitelný.

8.4 Nefunkční požadavky

8.4.1 Bezpečnost

V aplikaci je definována sada práv, do nichž lze přiřazovat role. Role lze přiřazovat pozicím, a ty na závěr uživatelům (bráno z opačného pohledu lze pozicím přiřazovat uživatele a role, rolím práva). Pozicím lze taktéž přiřazovat jednotlivá prostředí, která klient uvidí. Práva na prostředí lze však přepsat jednotným právem, které zajistí, že klient vidí všechna prostředí, včetně těch v budoucnu přidaných.

8.4.1.1 Přihlášení

K přístupu do DEFu je nutné být uložen jako uživatel a nastaven jako aktivní, tj. aktuální datum a čas musí spadat do stanovených mezí.

8.4.1.2 Menu

Pokud uživatel nemá žádné právo z konkrétní sekce, nebude na ni nikde ani vidět odkaz.

8.4.1.3 Přístup k funkcím

Práva se dělí na vytvářecí, editační a čtecí. Na vytváření nových produktů, serverů, uživatelů, rolí a pozic jsou potřeba vytvářecí práva v dané kategorii. Na úpravu vyjmenovaných entit a vytváření jejich podentit (například vytvoření nové aplikace v produktu) jsou potřeba práva editační. Na pouhé zobrazení detailů stačí práva čtecí.

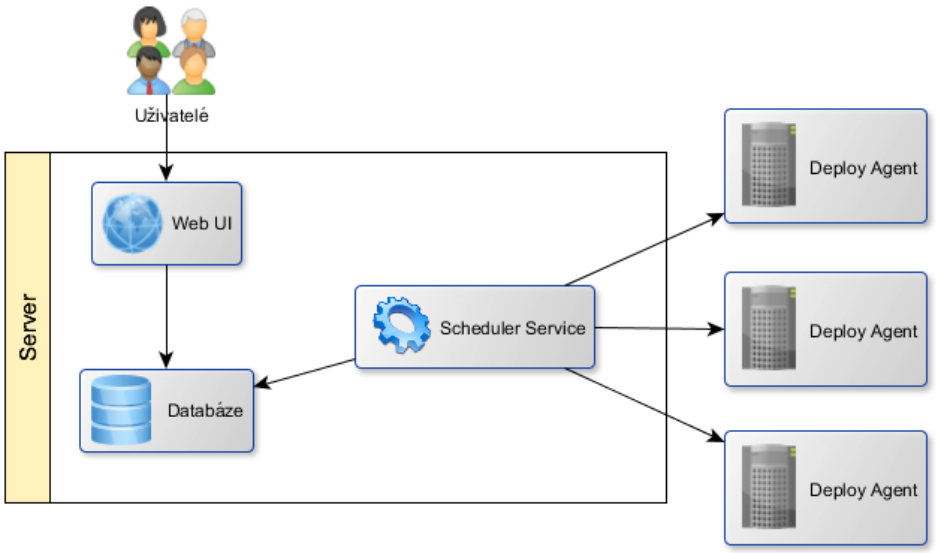
8.5 Architektura

Aplikace DEF je rozdělena na jednu serverovou část a neomezené množství Deploy Agentů. Serverová část se skládá z webového rozhraní (Web UI) a plánovací služby označené jako Scheduler Service.

8.5.1 Deploy Agents (Update Service)

Na jednotlivých klientských počítačích, [3] běží služba Update Service, která přijímá požadavky od Scheduler Service na instalaci balíčků. Služba tyto požadavky fyzicky vyřizuje, tedy instaluje balíčky na stroji, na němž je sama nainstalována. Výsledky instalací pak zapisuje do lokálního souboru – tyto výsledky pak čte Scheduler Service a ukládá do databáze. Samotní Deploy Agenti nemají do databáze žádný přístup.

[3] Počítačích, na něž jsou produkty fyzicky nasazovány.



Obrázek 8.1: Architektura aplikace DEF.

8.5.2 Serverová část (Web UI, Scheduler Service)

8.5.2.1 Web UI

Web UI je webové rozhraní, které slouží k uživatelskému ovládání všech podstatných funkcionalit - od přehledů až po samotné nasazování produktů, jak je popsáno ve specifikacích DEFu. Veškeré požadavky na nasazování se odesílají do databáze, z níž je pak načítá propojená serverová část – Scheduler Service. Ostatní požadavky (přidávání, editace a mazání různých entit) se vyřizují bez účasti Scheduler Service. Tyto entity se rovněž z databáze načítají.

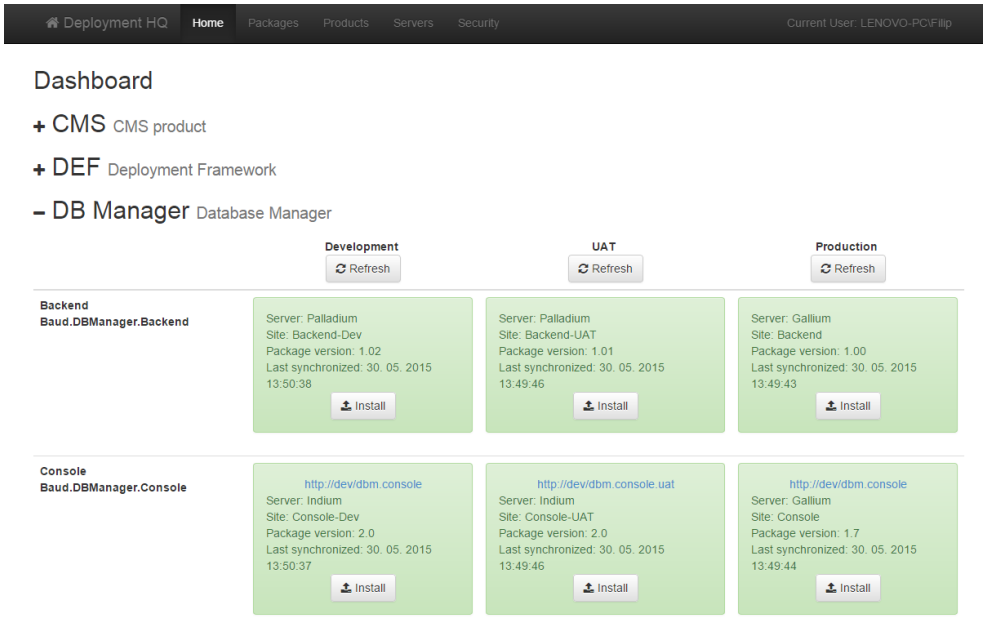
8.5.2.2 Scheduler Service

Scheduler Service požadavky přijaté z databáze, do níž je předtím zapsalo Web UI, ve vhodném čase (nastaveném uživateli) přeposílá do příslušných deploy agentů komunikací s jednotlivými US (je důležité uvědomit si, že deploy agentů typicky může být více a všichni se ovládají prostřednictvím jediného Web UI a Scheduler Service). Scheduler Service posléze kontroluje stav instalací a zapisuje tato data do databáze.

Kapitola 9

Implementace

V této kapitole jsou probrány implementační detaily DEFu – především důležité použité technologie v obou hlavních částech aplikace (od platformy až po práci se soubory) a způsob testování.



Obrázek 9.1: Hlavní stránka – Dashboard.

9.1 Zvolené technologie

9.1.1 Web UI

Pro napsání webové části byla použita platforma .NET 4.5 v kombinaci s MVC 5 frameworkem pro achitekturu aplikace, zobrazovacím enginem Razor (ASP.NET) a Entity Frameworkem 6 pro objektový přístup k databázi. V předchozí verzi DEFu bylo použité klasické HTML5 s javascriptovými knihovnami KnockoutJS (na zobrazování údajů, filtrování dat a obecně práci s uživatelem webu, to vše v reálném čase) a Durandal (jakožto logické jádro), práce s těmito knihovnami se ale ukázala být příliš nemotornou, navíc jsme se setkali s problémy při přechodech na novější verze. ASP.NET je dlouho zaběhnutý framework, který navíc ve společnosti Baud je používán i ve spoustě dalších projektů, tedy s ním vývojáři mají zkušenosti.

Krom toho bylo rozhodnuto, že pro balíčkový systém [4] pro klientskou část nebude použit NuGet, který byl používán v první verzi DEFu. Místo něj bude použit Bower, a to i přesto, že NuGet je produktem Microsoftu, a tudíž by s ASP.NETem měl v pořádku nativně spolupracovat. Důvody proč použít Bower místo NuGetu skvěle popisuje Tim Jones. [42] Primárním, a v podstatě jediným, důvodem proč v případě větších webových projektů zatratit NuGet, je znečištění globálního prostoru. Standardem u NuGetu je rozdělení javascriptových souborů do složky `Scripts` a souborů s CSS styly do složky `Content`. Bohužel tento způsob je poněkud nešťastný, neboť dále soubory netřídí do žádných podsložek. Pojmenování souborů leží na tvůrcích té které knihovny, ale co se stane, když jeden vývojář pojmenuje svůj soubor například `scripts.js` a autor jiné knihovny, kterou chcete taktéž používat, pojmenuje svůj skript stejně? V tom případě pak vítězí ten soubor, který byl přidán později. Nepořádek, který NuGet umí ve složkách nadělat, lze vidět na obrázku 9.2.

Bower všechny knihovny třídí do samostatných složek, situace popsaná výše tedy nemůže nastat.

9.1.2 Deploy Agents

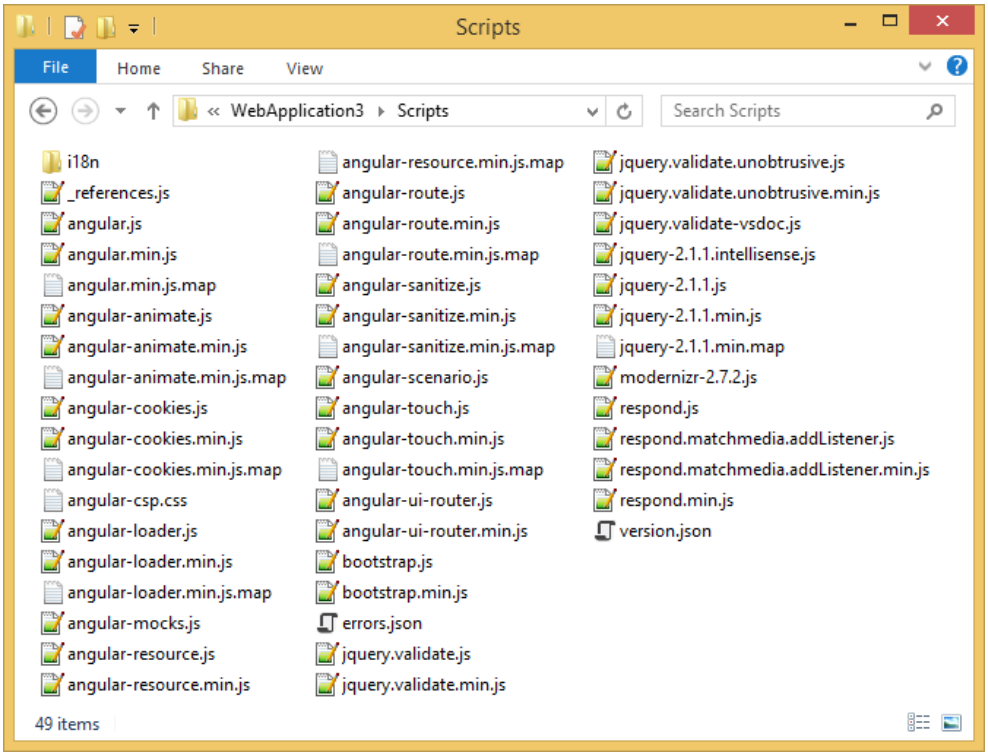
Platforma .NET 4.5 byla využita též pro Deploy Agenty. Pro ukládání umístění a logů byla zvolena knihovna Biggy, která veškerá data načítá a ukládá jako `json` soubory. Z oficiální dokumentace [43] si nyní uvedeme příklad použití:

Kód 9.1 nejprve vytvoří třídy pro práci s umělci.

Kód 9.1: Vytvoření pomocných tříd.

```
public class ArtistDocument {
  public ArtistDocument() {
    this.Albums = new List<Album>();
```

[4] Systém, jehož prostřednictvím lze snadno do projektu přidávat různé závilosti, kupříkladu různé javascriptové knihovny.



Obrázek 9.2: Znečištění globálního prostoru NuGetem.

Zdroj: `http://simplyaprogrammer.com/2014/06/why-bower-is-better-than-nuget.html` [42]

```
  }
  public int ArtistDocumentId { get; set; }
  public string Name { get; set; }
  public List<Album> Albums;
}

public partial class Album {
  public int AlbumId { get; set; }
  public string Title { get; set; }
  public int ArtistId { get; set; }
}
```

Nyní naplníme pár dat. Kód 9.2 vytvoří v kořenovém adresáři projektu soubor `Data/artistdocuments.json`.

Kód 9.2: Tvorba a naplnění souboru v kořenovém adresáři.

```
var newArtist = new ArtistDocument { ArtistDocumentId = 3, Name = "Nirvana"
     };
newArtist.Albums.Add(new Album { AlbumId = 1, ArtistId = 3, Title = "Bleach
    " });
```

```
newArtist.Albums.Add(new Album { AlbumId = 2, ArtistId = 3, Title = "
    Incesticide" });
```

Kód 9.3 nyní bez problémů může data načítat a aktualizovat.

Kód 9.3: Načtení a aktualizace dat.

```
// Nacte soubor z daneho umisteni:
var artists = new BiggyList<ArtistDocument>(store);

// Tento dotaz vubec nepracuje s HDD - nacita data z pameti:
var someArtist = artists.FirstOrDefault(a => a.Name == "Nirvana");
var someAlbum = someArtist.Albums.FirstOrDefault(a => a.Title.Contains("
    Incest"));

// Update:
someAlbum.Title = "In Utero";

// Okamzity zapis do souboru:
artists.Update(someArtist);
```

Biggy umí kromě souborů pracovat i s relačními databázemi, jelikož ale Deploy Agenti s databází vůbec nekomunikují, nebudeme tuto možnost v práci dále rozvádět.

Instalaci balíčků obdržených ze Scheduleru pak obstarává NuGet a případné PowerShell skripty dodané uživatelem.

9.1.3 Testovací framework

Pro testování byl zvolen framework Fluent Assertions, který umožňuje testování takřka lidským jazykem. Abychom si ukázali, co je myšleno lidským jazykem, uveďme pár příkladů z oficiální webové prezentace [44].

Představme si například řetězec `ABCDEFGHI`. Kód 9.4 ověří, že řetězec opravdu začíná `AB`, obsahuje `EF`, končí `HI` a je dlouhý přesně devět znaků.

Kód 9.4: Ověření řetězce.

```
string actual = "ABCDEFGHI";
actual.Should().StartWith("AB").And.EndWith("HI").And.Contain("EF").And.
    HaveLength(9);
```

Podobným způsobem se dají testovat také kolekce. A nejen u nich lze nastavit, jaké chybové hlášení se má vypsat v případě, dojde-li k chybě, takovou situaci ukazuje kód .

Kód 9.5: Ověření kolekce.

```
IEnumerable collection = new[] { 1, 2, 3 };
collection.Should().HaveCount(4, "because we thought we put three items in
    the collection"));
```

V tomto případě dostaneme zpět výpis:

> Expected <4> items because we thought we put three items in the collection, but found <3>.

Zajímavá je možnost testování reálných čísel, u nichž lze stanovit určitou přesnost. Například kód 9.6 ověří, zda proměnná `value` nabývá hodnoty mezi 3.139 a 3.141.

Kód 9.6: Ověření čísel s plovoucí desetinnou čárkou.

```
float value = 3.1415927F;
value.Should().BeApproximately(3.14F, 0.001F);
```

Pro pokročilejší příklad lze nahlédnout do testů, které jsou v Deploy Agentech. Kupříkladu kód 9.7 ověřuje, že se správně vrátí HTTP chyba 404 při pokusu o načtení neexistujícího umístění.

Kód 9.7: Ověření načtení neexistujícího umístění.

```
public async Task GetSingle_Returns404_IfNotPresent()
{
    var parameters = new Dictionary<string, string>
    {
        { "KeyOne", "First value" }
    };

    var context = new ApiTestContext();
    context.SitesService.GetSiteParameters("TestSite").Returns(new
        ReadOnlyDictionary<string, string>(parameters));

    using (var client = context.GetHttpClient())
    {
        var response = await client.GetAsync("http://localhost/api/sites/
            TestSite/parameters/UnknownKey");

        response.StatusCode.Should().Be(HttpStatusCode.NotFound);
    }
}
```

Kapitola 10

Migrace na open-source

Tato kapitola se zabývá praktickou stránkou migrace uzavřeného projektu, v případě této práce DEFu, jak je popsán v kapitole 8, na open-source. Teoretické aspekty uvedené v předchozích kapitolách uvádí v praxi konkrétními postupy práce.

10.1 Licence

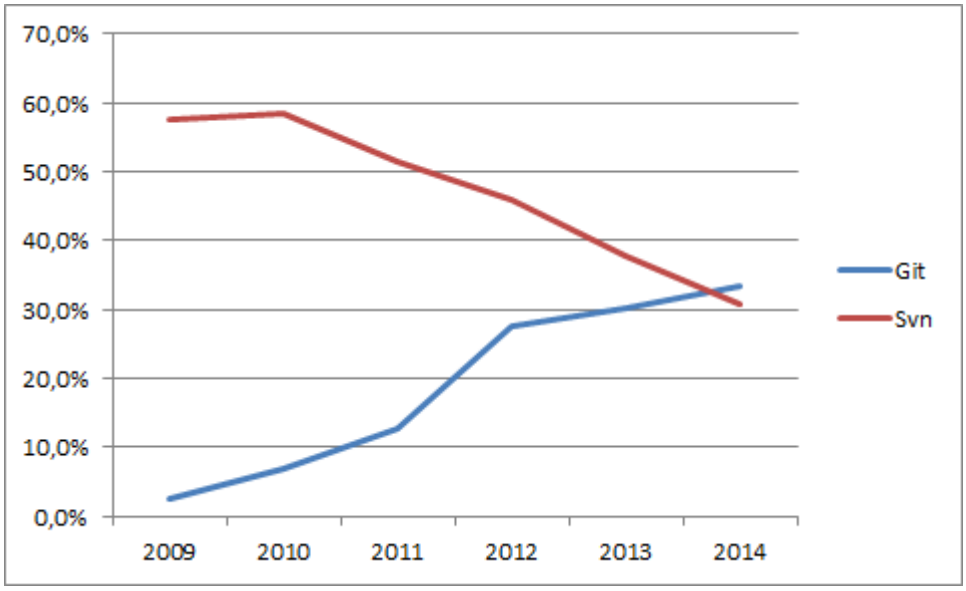
Na licenci byly ve společnosti požadavky, aby byla co nejméně omezující, ale na druhou stranu aby nebylo zcela jednoduché DEF vzít a začít kupříkladu prodávat. Studium licencí ukázalo, že s druhým požadavkem je poněkud problém, neboť prodej výslovně nezakazuje žádná licence – jen se staví různě ke způsobu, jakým prodej umožňuje.

První požadavek, tedy aby licence potenciální uživatele příliš nesvazovala, efektivně z výběru odstranil GNU GPL, která nutí nejen odvozená díla, ale jakákoliv jiná díla, která využívají byť jen části programů pod touto licencí, aby byla dále vydávána pod licencí GNU GPL. Na toto téma ostatně byly napsány články i v češtině. [45]

Licence MIT byla zamítnuta takřka ihned, neboť je paradoxně až příliš svobodná a volně řečeno říká: „Dělejte si s naším projektem, co chcete.“ Apache licence byla zamítnuta taktéž. Přestože není tolik diktátorská jako GNU GPL, nepatří mezi nejjednodušší licence. Bylo proto rozhodnuto pro BSD 3-Clause (tedy starší verzi BSD), která po uživatelích vyžaduje zachování copyrightu a narozdíl od novější BSD 2-Clause ještě zakazuje propagaci odvozených děl s využitím jmen původních autorů bez jejich svolení.

10.2 Hosting projektu

Již od začátku, ještě před samotným schválením startu projektu, jsme věděli, že chceme používat verzovací systém Git. Přestože práce s Gitem není



Obrázek 10.1: Využití Gitu oproti SVN.

Zdroj: `http://programmers.stackexchange.com/questions/136079/are-there-any-statistics-that-show-the-popularity-of-git-versus-svn` [47]

nejjednodušší, jedná se v dnešní době pravděpodobně o nejmocnější nástroj. Nespornou výhodou oproti dnes stále velmi používanému SVN je decentralita – veškerá historie commitů je uložena na lokálním počítači. Pokud se někdy stane situace, že vývojář není připojen k Internetu (nebo například na nějakou dobu vypadne hosting projektu), stále je s Gitem možno vrátit se k předchozím verzím nebo vytvářet nové commity bez nutnosti komunikace se serverem.

Z výsledků průzkumu společnosti The Eclipse Foundation [46] provedeného v červnu 2014 vyplývá, že dnes již Git překonal SVN v počtu projektů (obrázek 10.1).

Už jen toto byl pro nás dostatečný důvod, proč zvolit Git – čím větší je vývojářská základna, tím pro nás samozřejmě lépe. Dále tedy šlo o to zvolit vhodný hostovací web; zde bylo rozhodnutí ještě jednodušší – GitHub. Kromě mnoha výhod jako issue tracking, dokumentace, webová prezentace atp. je tento hosting dnes nejpoužívanější. Čili podobný důvod, z jakého jsme zvolili vůbec Git. Všechna tato rozhodnutí byla tedy činěna s ohledem na co největší základnu potenciálních přispěvatelů.

10.3 Build server

Při rozhodování, jaký bude zvolen build server, bylo relativně rychle určeno, že bude použit AppVeyor. Pro licenci zdarma u TeamCity nebyly splněny

požadavky (alespoň tři měsíce starý projekt, nebylo ještě rozhodnuto, zda nebudou nabízeny placené doplňky atp.), Bamboo bylo zavrženo pro svou až přílišnou složitost, ostatní build servery, které byly nalezeny, měly různé nedostatky; občas se jednalo pouze o placené produkty, některé produkty zase neuměly pracovat s .NET frameworkem. Nakonec byl zvolen AppVeyor, který se umí jednoduše propojit s GitHubem bez zdlouhavého nastavování.

Postupem času se nicméně ukázalo, že problémem u AppVeyoru je jeho pomalost, po vyřešení důležitějších starostí se tedy pravděpodobně bude ještě volit jiný build server, který by lépe vyhovoval našim požadavkům.

10.4 Zakládání nezbytných dokumentů

Po založení projektu na GitHubu nám odpadla starost s tvorbou textového souboru obsahujícího licenci, neboť GitHub umožňuje licenci předvolit a pak se o vytvoření souboru s licencí postará automaticky (jen je potřeba do něj dopsat jméno autora, případně další detaily, jsou-li třeba).

Byl tedy vytvořen `readme` soubor

README.md

Deployment Framework

Automated application deployment to all stages of deployment pipeline. Deploy anything, anywhere, anytime!

Authors: Augustin Šulc (augustin.sulc at baud.cz), Filip Vondrášek (filip.vondrasek at baud.cz)

GitHub URL: https://github.com/Baud-UCS/DEF

Issue tracker: https://github.com/Baud-UCS/DEF/issues

Wiki: https://github.com/Baud-UCS/DEF/wiki

Project URL: http://def.baud.cz

Gitter Chat: GITTER JOIN CHAT →

Build status: build passing

Mailing list: TBD

Obrázek 10.2: Readme soubor pro DEF.

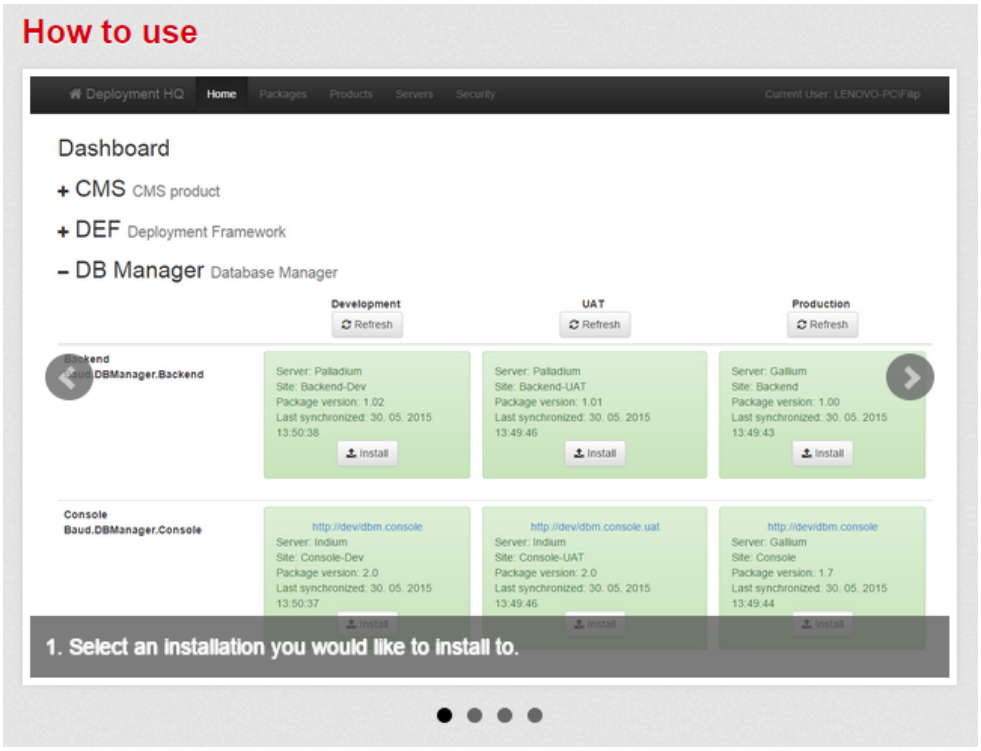
Po `readme` byl ještě vytvořen soubor `contributing`:

> Oceňujeme pull requesty od každého. Pokud chcete přispět, prosím proveďte následující:
>
> Nejprve forkněte a potom naklonujte repozitář:
>
> `git clone git@github.com:your-username/DEF.git`
>
> Abyste dodržovali naše (povinné) pravidla pro zpracování zdrojového kódu, budete potřebovat Microsoft Visual Studio 2013 a plugin StyleCop.
>
> Serverový kód se nachází v `/src/Server/DeploymentFramework`
>
> Kód Deploy Agentů se nachází v `./src/Agent/DeployAgent`
>
> Poté, co dokončíte a otestujete svou změnu, uložte ji do svého forku a vytvořte pull request.
>
> Nyní čekáte pouze na nás.
>
> Některé věci zvýší pravděpodobnost, že váš pull request bude akceptován:
>
> - Napište testy.
> - Dodržujte naše StyleCop pravidla.
> - Napište dobrou commit zprávu.
>
> Pokud si něčím nejste jisti, ujistěte se, že jste si přečetli náš README a neostýchejte se zeptat v našem Gitter chatu!

10.5 Webová stránka (prezentace)

Pro webovou prezentaci jsme využili GitHub Pages. Zakládání této prezentace se provádí přes jednoduchý webový nástroj, v němž si vývojář nakliká grafickou šablonu a veškeré nastavení (včetně podpory Google Analytics). Drobnou nevýhodou je nemožnost šablonu jakýmkoliv způsobem upravovat – zde se vyskytl problém, kdy jsme přemýšleli, zda spíše zvolit šablonu, která připomíná grafický styl DEFu, nebo šablonu, která více používá firemní barvy. Nakonec jsme se rozhodli pro druhý případ.

Webová prezentace hostovaná na GitHub Pages dále používá poměrně krkolomnou URL (v našem případě `http://baud-ucs.github.io/DEF`). Byla proto založena subdoména `http://def.baud.cz` a vytvořen `CNAME` záznam ukazující na zmíněnou konkrétní `github.io` adresu. Na GitHubu jsme dále do branche `gh-pages`, která je vytvořena automaticky po založení prezentace, přidali soubor `CNAME` s jediným řádkem: `def.baud.cz`.



Obrázek 10.3: Javascriptový plugin bxSlider použitý ve webové prezentaci

Tento postup zajistil, že je nyní možno přistoupit na adresu `http://def.baud.cz`, která zrcadlí veškerý obsah z našich GitHub Pages. Uživatel je tak kompletně odstíněn od nepěkné adresy a využívá pouze naší snadněji zapamatovatelnou URL. Je důležité, aby se soubor `CNAME` jmenoval přesně takto, včetně velkých písmen, a aby obsahoval jeden jediný řádek s adresou, na níž se obsah má zrcadlit.

Prvních pár týdnů, kdy byl autorem této práce teprve vytvářen návrh s analýzou, byl na webové prezentaci původní „lorem ipsum“ text. Posléze byla na stránku přidána (lehce upravená) sekce „Účel DEFu“ (v anglickém překladu „The purpose of DEF“), která v době chybějících pokročilejších informací, jako je stručný popis použití aplikace, poskytuje nejdůležitější informace.

Nicméně hlavním cílem webové prezentace je prezentovat, nebylo proto možno, aby na stránce zůstal jen strohý (a nudný) popis. Bylo tedy využito existující první verze DEFu, z níž byly vytvořeny screenshoty. Tyto screenshoty byly posléze použity s javascriptovými (využívajícími jQuery) pluginy bxSlider [48] a Magnific Popup [49], aby poutavou obrazovou formou předvedly možnosti DEFu.

10.6 DEF Wiki

Zbytek dokumentace, který potenciálním přispěvatelům k něčemu je (tj. zbytek specifikací a architektura), byl přidán na Wiki poskytnutou GitHubem. [50] Dále byly na této Wiki založeny stránky pro instalaci a konfiguraci DEFu, které, v okamžiku, kdy DEF začne být více funkční, budou poskytovat podrobné informace o instalaci a konfiguraci takovým způsobem, aby uživatelé nebyli odrazeni složitostí ještě předtím, než vůbec produkt vyzkouší. Důležité je uvědomit si, že zrovna tyto dvě sekce budou sloužit nejenom přispěvatelům, ale také běžným uživatelům, je proto potřeba je psát jednodušším způsobem.

Dalším důležitým dokumentem, který by na Wiki neměl chybět, je programátorská příručka. V době psaní práce nicméně projekt ještě nebyl v takové fázi, aby bylo příliš o čem psát, programátorská příručka tedy byla odložena na později.

10.7 Metodika vývoje

Pro počáteční fázi vývoje, tj. fázi, kdy ještě není vytvořena komunita a celý proces vývoje je spíše v rukou společnosti, bylo nutno zvolit určitý postup, kterým se bude vývoj ubírat.

Pro první fázi vývoje bylo voleno z klasičtějších přístupů, nakonec byl zvolen přístup inkrementální, a to zejména pro jeho flexibilitu – v open-source vývoji nikdy nemůžete spoléhat na pevně stanovená data (nikdy si nemůžete být jisti svým týmem), což by rozbilo metodiky spoléhající právě na „pečlivější“ naplánování (například zmíněný vodopád).

V počáteční fázi byl vytvořen milník Skeleton (kostra), který obnášel vytvoření projektů ve Visual Studiu, instalace vhodných knihoven, vytvoření základní dokumentace (například DEF Wiki popsaná výše), vytvoření pravidel pro zdrojový kód, založení build serveru atp. Zatímco se pracovalo na tomto milníku, byly naplánovány další dva – Proof of concept (prototyp) a First release (první verze).

V milníku Proof of concept byla implementována nejpodstatnější funkcionalita – požadavky na nasazení aplikace z Web UI, nasazení na cílový server a zápis logů.

V milníku First release by mělo být dosaženo splnění popsané funkcionality – tedy by mělo existovat funkční webové rozhraní správně propojené se Scheduler Service, a dále Scheduler Service správně propojená s Deploy Agenty.

Nakonec, po určitém čase práce na prototypu, byl ještě vytvoření milník Future improvements (budoucí vylepšení), v němž dostane hlavní slovo komunita, zde se vývoj plynule přesune do druhé fáze. U budoucích vylepšení, které nebyly již zpočátku plánovány ve společnosti, by komunita mohla přispět nejvíce – vnáší do vývoje pohled, z něhož jsme se na projekt nedívali.

Závěr

Cílem této práce byl popis charakteristik a postupu open-source vývoje a předvedení těchto postupů na vzorové aplikaci, jejíž zadání bylo dodáno externím zadavatelem. Hlavní důraz byl kladen na teoretickou část, po jejímž přečtení by čtenář měl mít dostatek znalostí, aby byl schopen sám a kvalifikovaně začít vyvíjet open-source software – měl by zvládnout projít bez větších problémů všemi důležitými aspekty open-source vývoje, a to od takových věcí jako je výběr vhodné licence, přes volbu webového hostingu až po zdánlivé detaily, jakým je třeba napsání správného readme souboru.

Byly popsány dvě fáze vývoje – první, uzavřenější, a druhá, převážně komunitní. Bylo vysvětleno, jak vést tým dobrovolníků; jakým způsobem rozdělovat práci, kritiku i chválu a jak řešit například fenomén, kdy jeden člen týmu agresivně přebírá nad některou částí projektu sám vedení.

V diplomové práci bylo dále popsáno zpracování zdrojového kódu. Byly řečeny podstatné detaily o kvalitě kódu, testování projektu, odstranění proprietárních modulů a bylo vysvětleno, jak vynutit dodržování předepsaných pravidel.

V rámci praktické části práce byla navržena, analyzována a implementována aplikace Deployment Framework. Na této aplikaci byla předvedena migrace na open-source dodržující postupy popsané v části teoretické.

Aplikace Deployment Framework se po napsání této práce nacházela na přelomu dvou vysvětlených částí vývoje – podstatná funkcionalita byla implementována a probíhal začátek ohlášení projektu, a tím i nábor dobrovolných přispěvatelů. Bylo tedy splněno zadání a dílo se nyní může stát příručkou pro vývojáře, kteří chtějí přejít na open-source vývoj.

Literatura

[1] Open Source Initiative: *The Open Source Definition [online]*. [cit. 2015-03-15]. Dostupné z: http://opensource.org/osd/

[2] *The Heartbleed Bug [online]*. [cit. 2015-04-13]. Dostupné z: http://heartbleed.com/

[3] *Shellshock Bash Vulnerability Tester [online]*. [cit. 2015-04-13]. Dostupné z: https://shellshocker.net/

[4] *Open Source Software Licensing Trends [online]*. [cit. 2015-06-21]. Dostupné z: http://osswatch.jiscinvolve.org/wp/2015/02/05/open-source-software-licensing-trends/

[5] Creative Commons: *Creative Commons [online]*. [cit. 2015-03-18]. Dostupné z: http://creativecommons.org/

[6] Creative Commons: *Creative Commons – Choose a License [online]*. [cit. 2015-03-18]. Dostupné z: http://creativecommons.org/choose/

[7] Open Source Initiative: *The MIT License [online]*. [cit. 2015-03-20]. Dostupné z: http://opensource.org/licenses/MIT/

[8] jQuery Foundation: *Creative Commons Wiki – FAQ [online]*. [cit. 2015-03-19]. Dostupné z: https://jquery.org/license/

[9] Open Source Initiative: *The BSD 3-Clause License [online]*. [cit. 2015-03-20]. Dostupné z: http://opensource.org/licenses/BSD-3-Clause/

[10] Open Source Initiative: *The BSD 2-Clause License [online]*. [cit. 2015-03-20]. Dostupné z: http://opensource.org/licenses/BSD-2-Clause/

[11] Free Software Foundation: *The GNU General Public License v3.0 [online]*. [cit. 2015-03-20]. Dostupné z: https://www.gnu.org/copyleft/gpl.html

[12] Free Software Foundation: *GNU Lesser General Public License v3.0 [online]*. [cit. 2015-03-20]. Dostupné z: https://www.gnu.org/copyleft/lgpl.html

[13] The Apache Software Foundation: *Apache License, Version 2.0 [online]*. [cit. 2015-03-20]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0/

[14] Open Source Initiative: *Open Source Licenses [online]*. [cit. 2015-03-22]. Dostupné z: http://opensource.org/licenses/

[15] Vesperman, J.: *Essential CVS*. O'Reilly Media, 2003, ISBN 0-596-00459-1, 336 s.

[16] *Concurrent Versions System Bugs [online]*. Dostupné z: http://savannah.nongnu.org/bugs/?group=cvs

[17] *Linus Torvalds on GIT and SCM [online]*. [cit. 2015-04-16]. Dostupné z: http://codicesoftware.blogspot.com/2007/05/linus-torvalds-on-git-and-scm.html

[18] SmartBear: *How to Turn Your Pile of Code into an Open Source Project [online]*. [cit. 2015-03-25]. Dostupné z: http://blog.smartbear.com/open-source/how-to-turn-your-pile-of-code-into-an-open-source-project/

[19] Atlassian: *Atlassian Bitbucket passes one million users [online]*. [cit. 2015-03-22]. Dostupné z: https://blog.bitbucket.org/2013/06/04/atlassian-bitbucket-passes-one-million-users/

[20] GitHub, Inc.: *GitHub Press [online]*. [cit. 2015-03-22]. Dostupné z: https://github.com/about/press/

[21] Google: *Bidding farewell to Google Code [online]*. [cit. 2015-03-25]. Dostupné z: http://google-opensource.blogspot.cz/2015/03/farewell-to-google-code.html

[22] *The Linux Kernel on Launchpad [online]*. [cit. 2015-03-25]. Dostupné z: https://launchpad.net/linux/

[23] SourceForge: *About SourceForge [online]*. [cit. 2015-03-25]. Dostupné z: http://sourceforge.net/about/

[24] Appveyor Systems Inc.: *Continuous Integration and Deployment service for Windows developers – Appveyor [online]*. [cit. 2015-06-06]. Dostupné z: http://appveyor.com/

[25] JetBrains s.r.o.: *Continuous Integration for Everybody – TeamCity [online]*. [cit. 2015-06-06]. Dostupné z: https://www.jetbrains.com/teamcity/

[26] Atlassian: *Continuous Integration & Build Server – Bamboo [online]*. [cit. 2015-06-06]. Dostupné z: https://www.atlassian.com/software/bamboo

[27] Fogel, K.: *Tvorba open source softwaru*. Praha: CZ.NIC, 2010, ISBN 978-80-904248-5-2, 312 s.

[28] *Doxygen: Generate documentation from source code [online]*. [cit. 2015-06-22]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/

[29] GitHub, Inc.: *GitHub Pages [online]*. [cit. 2015-03-31]. Dostupné z: https://pages.github.com/

[30] ConversionXL: *8 Things That Grab and Hold Website Visitor's Attention [online]*. [cit. 2015-03-25]. Dostupné z: http://conversionxl.com/how-to-grab-and-hold-attention/

[31] Kummer, D.: *Git-Flow Cheatsheet [online]*. [cit. 2015-04-07]. Dostupné z: http://danielkummer.github.io/git-flow-cheatsheet/

[32] *HipChat [online]*. [cit. 2015-06-23]. Dostupné z: https://www.hipchat.com/integrations

[33] *Semantic Versioning 2.0.0 [online]*. [cit. 2015-04-14]. Dostupné z: http://semver.org/spec/v2.0.0.html

[34] *GitVersion [online]*. [cit. 2015-05-20]. Dostupné z: https://github.com/ParticularLabs/GitVersion

[35] *What is Trello? [online]*. [cit. 2015-04-14]. Dostupné z: http://help.trello.com/article/708-what-is-trello

[36] *The Dead Linger Trello [online]*. [cit. 2015-06-18]. Dostupné z: https://twitter.com/thedeadlinger/status/488503917249323008/photo/1

[37] *GOP c – Authorship in source files [online]*. [cit. 2015-06-15]. Dostupné z: http://lilypond.org/~graham/gop/gop_15.html

[38] JetBrains, s.r.o.: *IntelliJ IDEA: Static code analysis [online]*. [cit. 2015-06-18]. Dostupné z: https://www.jetbrains.com/idea/documentation/static_code_analysis.html

[39] *Static Code Analysis in the NetBeans IDE Java Editor [online].* [cit. 2015-06-11]. Dostupné z: https://netbeans.org/kb/docs/java/code-inspect.html

[40] JetBrains, s.r.o.: *ReSharper: Unique code analysis [online].* [cit. 2015-06-11]. Dostupné z: https://www.jetbrains.com/resharper/features/code_analysis.html

[41] .NET Blog: *.NET Core is Open Source [online].* [cit. 2015-06-06]. Dostupné z: http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx

[42] Jones, T.: *Why Bower is better than NuGet [online].* [cit. 2015-06-18]. Dostupné z: http://simplyaprogrammer.com/2014/06/why-bower-is-better-than-nuget.html

[43] *Biggy – A File-based Document Store for .NET [online].* [cit. 2015-06-18]. Dostupné z: https://github.com/xivSolutions/biggy

[44] *Fluent Assertions [online].* [cit. 2015-06-18]. Dostupné z: http://www.fluentassertions.com/

[45] Věštník misantropizace: *Proč nepoužívám GPL? [online].* [cit. 2015-04-06]. Dostupné z: http://www.misantrop.info/proc-nepouzivam-gpl

[46] *Eclipse Community Survey 2014 Results [online].* [cit. 2015-04-06]. Dostupné z: http://eclipse.dzone.com/articles/eclipse-community-survey-2014

[47] Programmers Stack Exchange: *Are there any statistics that show the popularity of Git versus SVN? [online].* [cit. 2015-06-18]. Dostupné z: http://programmers.stackexchange.com/questions/136079/are-there-any-statistics-that-show-the-popularity-of-git-versus-svn

[48] *jQuery Content Slider | bxSlider [online].* [cit. 2015-06-04]. Dostupné z: http://bxslider.com/

[49] *Magnific Popup: Responsive jQuery Lightbox Plugin [online].* [cit. 2015-06-04]. Dostupné z: http://dimsemenov.com/plugins/magnific-popup/

[50] *DEF Wiki [online].* [cit. 2015-05-25]. Dostupné z: https://github.com/Baud-UCS/DEF/wiki

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ASF Apache Software Foundation

CI Continuous Integration

CSS Cascading Style Sheets

CVS Concurrent Versions System

DEF Deployment Framework

FAQ Frequently Asked Questions

GPL General Public License

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

LGPL Lesser General Public License

OSI Open Source Initiative

SVN Subversion

UAT User Acceptance Testing

UI User Interface

URL Uniform Resource Locator

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Vondrasek_Filip_2015.pdf` ..... text práce ve formátu PDF