Sem vložte zadání Vaší práce.

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

Bakalářská práce

Aplikační podpora pro správu projektů

Jan Vlnas

Vedoucí práce: Ing. Tomáš Kadlec

18. května 2012

Prohlášení

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

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

V Praze dne 18. května 2012 .....

Odkaz na tuto práci

Jan Vlnas. *Aplikační podpora pro správu projektů: Bakalářská práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

Abstract

This thesis focuses on modification of *ChiliProject* (forked from *Redmine*) through a set of plug-ins. The thesis evaluates open source web applications for project management, analyses some specific aspects of *Ruby on Rails* platform and describes design, implementation, and testing of plug-ins for the existing application. The resulting plug-ins provide decentralized user roles management, access restriction for particular resources, and more.

Keywords project management, redmine, chiliproject, ruby, rails, plugin, test

Abstrakt

Práce se zabývá modifikací systému *ChiliProject* (navazujícího na systém *Redmine*) prostřednictvím modulů. Porovnává některé open source webové aplikace pro správu projektů, analyzuje specifika platformy *Ruby on Rails* a zabývá se návrhem, implementací a testováním *rozšíření* existující aplikace bez zásahu do jejího kódu. Výsledkem práce je několik rozšíření, která mj. decentralizují správu uživatelských rolí a umožňují omezit přístup ke konkrétním zdrojům.

Klíčová slova správa projektů, redmine, chiliproject, ruby, rails, rozšíření, testování

Obsah

Seznam obrázků

Úvod

Oddělení ICT Fakulty informačních technologií má na starost správu softwarového i hardwarového vybavení. Zajišťuje funkčnost fakultních informačních systémů i různých zařízení (tiskárny, počítače v učebnách). To znamená sledovat a řešit velké množství různých požadavků, od přidání opravy do systému Edux po výměnu toneru v tiskárnách. Správu požadavků by měl usnadnit jednotný systém pro správu projektů. Takový systém může podporovat i sdílení znalostí (např. v rámci wiki) a správu kódu.

Pro správu projektů oddělení používá systém Redmine,[1] který splňuje většinu jejich potřeb. Redmine má však určité nedostatky z hlediska správy většího množství projektů a uživatelů.

Cíl práce

Cílem práce je zvolit a upravit takový systém, který bude nejlépe vyhovovat potřebám ICT oddělení. Preferované je takové řešení, které bude vyžadovat nejméně zásahů do originálního systému pro zachování kompatibility a udržitelnosti budoucího vývoje systému.

Struktura práce

Text práce je členěn do čtyř kapitol. První kapitola rozebírá současný systém, nakolik jeho funkce odpovídají potřebám ICT oddělení. Druhá kapitola porovnává alternativy k systému Redmine; zároveň zde odůvodním, proč jsem zvolil přechod na systém ChiliProject. Třetí kapitola řeší některá specifika zvoleného systému a platformy Ruby on Rails. Hlavní část textu tvoří čtvrtá kapitola, kde rozebírám návrh, implementaci a testování jednotlivých úprav. Závěr pak shrnuje výsledky a naznačuje možnosti budoucího vývoje.

[1] `http://www.redmine.org/`

Kapitola 1

Systém Redmine a požadované vlastnosti

Při určování požadavků je nutné zohlednit vlastnosti stávajícího systému, proto nejprve popíšu nejdůležitější vlastnosti Redmine s ohledem na funkce vyžadované ICT oddělením.

1.1 Popis Redmine

Redmine je webová aplikace napsaná v jazyce Ruby s použitím frameworku Rails, dostupná pod open source licencí GNU General Public License (GPL) verze 2.

Základem struktury Redmine je *projekt*, ke kterému je možné přiřadit uživatele. Každý projekt představuje částečně oddělenou jednotku s vlastní sadou úkolů, repozitářem kódu a wiki. Specifikem projektů v Redmine je, že mohou mít hierarchickou strukturu. Kromě lepší orientace při velkém množství projektů lze tuto hierarchii využít i pro částečné sdílení některých zdrojů – například je možné vypsat úkoly ze všech podprojektů.

Uživatele v Redmine je možné zařadit do *skupin*. V rámci projektu je pak uživateli či skupině přiřazena role (či více rolí), která slouží pro autorizaci uživatele. Skupiny i role jsou definované globálně, nicméně rozhodující je, zda má daný uživatel v projektu členství.

Issue tracker představuje systém správy úkolů – hlášení chyb, požadavků na změny, návrhy na zlepšení atp. Redmine nabízí několik možností jak úkoly třídit – fronty (například chyby, vylepšení) a kategorie (například podle komponent). Správce však může úkolům definovat další vlastnosti – pro jednotlivé projekty je pak možné zvolit, zda budou mít příslušnou frontu, kategorii či vlastnost k dispozici. I úkoly v Redmine mohou být hierarchicky uspořádané, navíc na sebe mohou vzájemně odkazovat – zde je výhodou, že každý úkol má

v rámci instance Redmine unikátní číslo, lze proto odkázat na úkol ve zcela jiném projektu.

Úkoly mají vždy nějaký stav – například otevřený, vyřešený či pozastavený. S tím souvisí podpora workflow – tato funkce umožňuje určit, jaký stav může určitá role nastavit úkolu s ohledem na stávající stav. Například role *vývojář* může změnit úkol, který je *ve vývoji* na *uzavřený*, nemůže jej však nastavit jako *nový*.

Wiki v Redmine se příliš neliší od jiných implementací. Jediným specifikem je, že stránky mohou být uspořádány v hierarchii – nejedná se však o kompletní podporu namespace, jak ji chápe software MediaWiki[2] či DokuWiki.[3] Hierarchie má význam pouze pro navigaci v rozsáhlejší struktuře stránek.

Redmine, jakožto systém určený především pro správu softwarových projektů, zahrnuje i Source Code Management (SCM). Systém podporuje řadu nástrojů pro správu kódu včetně Subversion, Git a Mercurial. Výhodou této integrace je propojení kódu s úkoly – revize může odkazovat na úkol a naopak. Zároveň je možné prohlížet zdrojový kód z webového rozhraní Redmine a procházet jednotlivé změny.

V prostředí organizace je důležitá podpora jednotné autentizace uživatelů – Redmine standardně podporuje Lightweight Directory Access Protocol (LDAP).

Podstatnou vlastností Redmine je i rozšiřitelnost a existence komunity uživatelů i vývojářů, kteří udržují obrovský systém rozšíření. Redmine je aktivně vyvíjený od roku 2006 čili lze říci, že se jedná o software do jisté míry prověřený časem.

1.2 Požadavky

Požadované vlastnosti jsou tyto:

- issue tracking,
- SCM,
- hierarchická wiki,
- hierarchická struktura projektů,
- modulární autentizace uživatelů (např. přes LDAP),
- podpora uživatelských skupin,
- konfigurovatelná autorizace.

[2] `http://www.mediawiki.org/`
[3] `http://www.dokuwiki.org/`

Výsledný systém by měl tyto požadavky splňovat, případně by měl nabízet adekvátní alternativu nebo by chybějící funkce mělo být možné doplnit.

Redmine tyto funkce splňuje, některé z nich mají však určité nedostatky.

Největší problém představuje „globálnost“ – role a skupiny může definovat pouze administrátor. Pokud je v systému větší množství projektů s vlastními správci, každý správce může potřebovat jednotnou organizaci uživatelů ve své hierarchii projektů, stejně tak může vyžadovat roli se zvláštní sadou oprávnění, která dosud neexistuje. Systém pro správu projektů by měl usnadňovat organizaci práce, nikoliv práci přidělávat, proto by měli mít správci projektů možnost definovat skupiny a role v rámci svých projektů.

Dalším problémem je viditelnost konkrétních stránek wiki a úkolů. Dejme tomu že v rámci jednoho projektu vidí všichni jeho členové všechny úkoly – všichni jsou si v tomto ohledu rovni. Někteří členové však mohou řešit třebas bezpečnostní problémy, které by měly být viditelné pouze pro úzkou skupinu uživatelů. Redmine umožňuje tyto „soukromé záležitosti“ řešit pomocí podprojektů – vytvoří se podprojekt, který bude přístupný pouze vybraným uživatelům. Není však jisté kdy (a zda vůbec) potřeba privátních úkolů či stránek nastane, navíc potenciálně každý uživatel může mít zájem vytvořit privátní úkol – třeba právě v případě bezpečnostní chyby, kterou chce uživatel nahlásit pouze povolaným osobám. Zároveň by však měl mít tento úkol nadále přístupný, aby byl informován v případě, že je problém vyřešen.

Posledním požadavkem, kterým se v rámci této práce budu zabývat, je podpora přidávání uživatelů z LDAP. Redmine pracuje pouze s lokální databází uživatelů (kromě autentizace) – uživatel se musí do systému alespoň jednou přihlásit, aby jej bylo možné přiřadit do projektu. V menších organizacích by tento problém byl řešitelný kompletní synchronizací s LDAP. Adresář Fakulty informačních technologií však čítá několik tisíc osob, tudíž by se nejednalo o příliš schůdné, ani užitečné řešení – se systémem ICT oddělená potřebuje pracovat pouze zlomek uživatelů. Jako lepší možnost se nabízí selektivní přidání uživatele ve chvíli, kdy je s ním zapotřebí začít pracovat.

Tyto funkce je možné do Redmine doplnit, nicméně jsem nejprve prozkoumal alternativy – není vyloučené, že existuje systém, který je funkčně kompletnější nebo snadněji rozšiřitelný než Redmine.

Kapitola 2

Porovnání alternativních systémů

Pro správu projektů existuje velké množství aplikací. Některé z nich řeší obecnou komunikaci v týmu se správou úkolů a dokumentů, řada aplikací má navíc specializované funkce pro správu softwarových projektů.

Některé nástroje se orientují výhradně na jeden aspekt – typickým příkladem je samostatný bug tracker, který nabízí pouze správu úkolů, [4] pro další funkce je nutné použít externí aplikace. Řada takových nástrojů má unikátní vlastnosti a jejich kombinací lze dosáhnout zajímavých možností. Nevýhodou je pak samozřejmě roztříštěnost konfigurace a rozhraní, z čehož vyplývá náročnější správa a vyšší nároky na uživatele. V řadě případů je proto lepší mít jednodušší nástroje, zato dostupné v rámci jediného systému.

Pro zúžení výběru jsem stanovil základní kritéria, která každá posuzovaná aplikace musí splňovat.

Webová aplikace Ačkoliv distribuované nástroje[5] mají své kouzlo, nelze po uživatelích vyžadovat instalaci software, aby mohli hlásit chyby.

Samostatně hostovaná Software musí být možné provozovat v rámci fakultní infrastruktury, nemůže se jednat o Software as a Service (SaaS) řešení.

Open source Aplikaci je možné modifikovat a kompletní zdrojový kód je možné zveřejnit.

Aktivní vývoj Poslední vydaná verze by neměla být starší než rok, větší komunita uživatelů a vývojářů může být výhodou.

[4] Například Bugzilla – `http://www.bugzilla.org/`
[5] Jako je třeba Ditz – `http://ditz.rubyforge.org/`

Nezávislost na proprietálních systémech V rámci integrace do fakultní infrastruktury by aplikaci mělo být možné provozovat na unixových serverech bez závislosti například na MS SQL Server.

Následující aplikace tato kritéria splňují, posoudil jsem proto jejich funkčnost a rozšiřitelnost s ohledem na požadavky.

2.1 Trac[6]

Jeden z nejpopulárnějších systémů svého druhu; svojí podobou inspiroval řadu dalších projektů, včetně Redmine [23]. Trac je napsaný v Pythonu (kompatibilní s řadou 2.x).

Trac je velice podobný Redmine, zakládá si na menším počtu komponent a jejich silnější provázanosti. Důraz je kladený především na wiki, jejíž vlastnosti prostupují celým systémem – zejména možnost snadno odkazovat na konkrétní informace v systému pomocí wiki syntaxe.

Zajímavý je i systém oprávnění, který nerozlišuje mezi skupinami uživatelů a uživateli samotnými – uživatelé tak mohou dědit práva od jiných uživatelů, což nabízí velkou flexibilitu.

Navzdory jednoduchosti systému má výchozí webové rozhraní řadu nedostatků. Jako největší slabina se jeví navigace – není zcela jasné, které odkazy se vztahují ke které komponentě, zda vedou na stránku wiki, na úkol, nebo třeba na stránku milníku. Tento problém však lze řešit pomocí alternativních vzhledů a pokročilejší konfigurace.

Nejzásadnější nedostatek systému Trac představuje nemožnost provozovat více projektů nad stejnou databází. Systém umožňuje poměrně jednoduše nakonfigurovat nové instance, které sdílí zdrojový kód i uživatele. Projekty jsou však vzájemně izolované; například není možné jednoduše odkazovat z úkolu v projektu A na úkol v projektu B. Není možné ani vytvářet hierarchii projektů.

Přidání těchto funkcí by znamenalo zásadní zásah do architektury systému a přepis velké části komponent.

2.2 GitLab[7]

Jedná se o velice mladý projekt (první verze vyšla v říjnu 2011) inspirovaný oblíbenou službou GitHub. GitLab je postavený na frameworku Ruby on Rails 3.2.

Je vhodný především pro malé vývojářské týmy. Každý projekt představuje repozitář kódu s vlastní wiki a úkoly. Mimo to má projekt k dispozici i *wall*

[6] `http://trac.edgewall.org/`
[7] `http://gitlabhq.com/`

(nástěnku), kam je možné přidávat soubory, a také *merge requests* – užitečná funkce pro revidování nového kódu.

Projekty (potažmo repozitáře) není možné nijak organizovat, systém oprávnění je příliš jednoduchý a issue tracker velice minimalistický. Je to však záměrná inspirace GitHubem, který si na maximální jednoduchosti nástrojů zakládá [8, snímky 55-71].

Logicky je tak GitLab méně vhodný pro velké týmy, velké množství projektů a nemusí být nejpřívětivější pro netechnické uživatele vzhledem k tomu, že klade důraz na přístup ke kódu.

S ohledem na zaměření i filozofii projektu jsem usoudil, že tento systém nebude pro potřeby ICT oddělení nejvhodnější. GitLab se však velice aktivně vyvíjí, tudíž je pravděpodobné, že postupem času bude představovat stále vitálnější alternativu k tradičním nástrojům (jako je právě Redmine). Pro menší týmy se už nyní jedná o zajímavou volbu.

2.3 The Bug Genie[8]

The Bug Genie se profiluje jako nástroj pro agilní vývoj. V praxi se to projevuje několika specifickými funkcemi navíc (např. burndown chart), jinak se jedná o běžný systém s issue trackerem a wiki. Implementačním jazykem je PHP.

Po funkční stránce je systém srovnatelný s Redmine. Navíc umožňuje podrobně definovat uživatelské role na úrovni jednotlivých projektů.

Velké množství funkcí však často vede k horší přehlednosti, a to i přesto, že The Bug Genie je – dle oficiální prezentace – krásný a (uživatelsky) přívětivý. Z praktických zkušeností je mi známo, že s ním měli problém i zkušenější uživatelé. Některé problémy s použitelností by šlo vyřešit zjednodušením kategorizace úkolů, odstraněním přebytečných funkcí a vylepšením vzhledu, případně zaškolením uživatelů.

Zajímavým způsobem The Bug Genie řeší integraci repozitářů kódu. Není zde samostatná komponenta pro procházení repozitáře, ale pracuje se s existujícími systémy, jako ViewVC (pro SVN, CVS), gitweb či GitHub. Uživatel je odkázán na odpovídající stránku v příslušném systému. Nevýhodu tohoto řešení představuje správa řízení přístupu ke kódu, protože nezávislý prohlížeč kódu má samostatně řešená oprávnění. Pravděpodobně by tak bylo nutné spravovovat autorizaci separátně, nebo ji synchronizovat mezi více systémy.

Funkce The Bug Genie pokrývají většinu požadavků a zmíněné problémy jsou řešitelné. Sympatická je i modularita systému, kde i standardně dodávané funkce jsou řešeny jako nezávislé moduly.

[8] `http://www.thebuggenie.com/`

2.4 Teambox[9]

Tato aplikace nemá součásti pro podporu vývoje software, spíše nabízí řešení pro obecnou spolupráci a komunikaci v týmu. Přestože autoři nabízí hostované řešení, zdrojový kód je dostupný pod licencí Affero General Public License (AGPL) verze 3.[10] Teambox je postavený na Ruby on Rails 3.0.

Funkce Teamboxu se točí kolem komunikace mezi spolupracovníky; z diskuzí lze vytvořit úkoly a dlouhodobé informace je možné zaznamenat na *stránkách* (Pages), které však mají ke klasické wiki poměrně daleko.[11]

Teambox je zpočátku velice jednoduchý a přehledný. S rostoucími nároky týmu a s velkým množstvím projektů se situace však rychle mění. Velké týmy se brzy utopí v záplavě diskuzí, softwarovým projektům bude vadit minimalistická správa úkolů a chybějící integrace repozitářů kódu. Pages není možné hierarchicky třídit a správa oprávnění je velice omezená.

Pro malé týmy a nesoftwarové projekty však Teambox představuje výborné řešení a v současné době se začíná prosazovat i na FIT.[12]

2.5 Open Atrium[13]

Jedná se o distribuci systému Drupal (PHP) s řadou rozšíření pro správu projektů. Vedle issue trackeru a wiki (Notebook) obsahuje systém i kalendář, blog (který může plnit funkci diskuzního fóra) a mikroblog (Shoutbox).

Jestli v něčem Open Atrium skutečně vyniká, tak je to konfigurovatelnost. Většinu vlastností systému je možné nastavit z webového rozhraní – i mé dosavadní zkušenosti se systémem Drupal nasvědčují tomu, že krédem vývojářů na této platformě je zřejmě „configuration over convention“. Síla Drupalu tvoří současně i největší slabinou systému Open Atrium – každá pokročilejší úprava obnáší jistý risk, že se něco rozbije. Konfigurace je však uložená v databázi, tím pádem je obtížné sledovat změny a systém se obtížně testuje.

Na samotném Open Atriu je znát jistý „puzzle efekt“ dané platformy – jednotlivé moduly jsou izolované a obtížně propojitelné; například není možné se odkazovat z wiki na konkrétní úkol a naopak. Jsou k dispozici rozšíření pro Drupal, která tyto problémy řeší. Stejně tak je možná i integrace procházení repozitáře. Ne všechna rozšíření jsou však kompatibilní se systémem Open Atrium, některá se mohou i vzájemně ovlivňovat.

Teoreticky neexistuje nic, co by se na platformě Drupal / Open Atrium nedalo postavit. Komplexnost a jistá křehkost celé platformy však činí tento systém poměrně neatraktivní.

[9] `http://teambox.com/`
[10] `https://github.com/teambox/teambox`
[11] Tato funkce je inspirovaná aplikací Backpack – `http://backpackit.com/`
[12] `http://teambox.fit.cvut.cz/`
[13] `http://openatrium.com/`

2.6 mtrack[14]

mtrack je klonem systému Trac v jazyce PHP.

I přes to, že se nejedná o zcela novou aplikaci (historie sahá do roku 2009), řada funkcí působí poměrně nedotaženým dojmem. Je zde například jakási koncepce projektů a komponent, avšak kromě jejich administrace se nezdá, že by s nimi bylo možné dělat cokoliv jiného – wiki a issue tracker sdílejí jediný prostor. Sympatická je správa repozitáře kódu, na rozdíl od systému Trac je možné přidávat nové repozitáře přes webové rozhraní a výchozí instalace podporuje, kromě Subversion, také Git a Mercurial. Wiki je také uložená v repozitáři, což lze využít k přímé editaci dokumentů bez potřeby webového rozhraní.

V ostatních ohledech se aplikace chová stejně jako Trac, rozdíly jsou spíše kosmetického charakteru.

Aplikace mtrack má před sebou ještě dlouhou cestu, aby se stala plnohodnotným řešením, přesto je zde znát jistý potenciál. Pokud se do vývoje zapojí širší komunita, mohlo by se brzy jednat o velice atraktivní volbu v rámci PHP aplikací.

2.7 ChiliProject[15]

ChiliProject vznikl jako fork Redmine [22]. Některé důvody pro odštěpení od Redmine byly nesystematičnost a uzavřenost vývoje původního systému. ChiliProject se proto snaží být otevřenější (z hlediska přístupnosti přispěvatelům) a stabilnější alternativou k Redmine.

Ačkoliv na první pohled vypadá ChiliProject odlišně, jedná se skutečně o pouhé kosmetické rozdíly. Po funkční stránce jsou systémy srovnatelné – občas si i koneckonců vyměňují kód. ChiliProject existuje teprve přes rok, což je v případě zavedeného a rozsáhlého projektu krátká doba pro vznik zásadních změn. Je však zřejmé, že oba projekty směřují jiným směrem; Redmine stále „nabaluje“ další funkce a opravy, zatímco vývojáři ChiliProjectu otevřeně diskutují o nutnosti radikálních změn v kódu. Pomyslné nůžky mezi těmito dvěma projekty se budou nadále rozevírat.

2.8 Zhodnocení

Z celkových 23 systémů (Příloha C]) jsem jich podrobněji prozkoumal šest. Nejvíce požadovaných vlastností splňoval systém The Bug Genie. Usoudil jsem však, že se na tento systém migrovat nevyplatí – oproti Redmine má z hlediska webové aplikace velice specifické chování a strmější křivku učení. Další nevýhodou je použitá platforma – pominu-li syntaktické a ideové rozdíly PHP a Ruby

[14] `http://mtrack.wezfurlong.org/`
[15] `https://www.chiliproject.org/`

(což je spíše otázka osobní preference), PHP nenabízí srovnatelnou dynamiku, které se věnuji v následující kapitole. V neposlední řadě je rozdíl v použitých knihovnách – Redmine je postavený na Ruby on Rails, což je časem prověřený a dobře zdokumentovaný framework. The Bug Genie má vlastní framework, kde prakticky jediným zdrojem informací je oficiální dokumentace.

Rozhodl jsem se však pro přechod na systém ChiliProject. Struktura databáze je u obou systémů – až na změny v nejnovějších verzích – stejná, migrace z Redmine tak nepředstavuje žádný problém. Jak jsem již zmínil, oba systémy se od sebe liší hlavně svým směřováním, ačkoliv ChiliProjectu chybí jedna z požadovaných funkcí (viz 4.6.2). Nové verze Redmine vychází „až když jsou hotové.“ ChiliProject má pevně vymezený časový rámec – *minor* verze, která je zpětně kompatibilní, vychází každý měsíc. *Major* verze, která může zahrnovat rozsáhlejší změny, vychází každý půlrok. V případě potřeby vyjde *patch* verze (například je nalezena chyba v zabezpečení). Pravidelné vydávání nových verzí zajišťuje, že se hotové funkce dostanou k uživatelům co nejdříve, vývojáři zase mají jasně vymezený časový rámec, ve kterém mohou pracovat. Je to podobný model, byť zjednodušený, jaký používá například Linux a Mozilla Firefox [18].

Kapitola 3

Analýza platformy

Pro implementaci úprav systému je nutné zvážit specifické vlastnosti platformy, na které je systém postavený. V této kapitole proberu některé charakteristiky platformy používané systémem ChiliProject, které určí formu výsledné práce.

3.1 Vlastnosti platformy

ChiliProject je napsaný v jazyce Ruby s použitím frameworku Ruby on Rails.

Samotný jazyk Ruby má řadu specifických vlastností, které značně rozšiřují možnosti implementace. Zaměřím se proto na některé dynamické vlastnosti jazyka Ruby, základy architektury Ruby on Rails aplikace, systém rozšíření a strukturu testování.

3.1.1 Dynamické vlastnosti jazyka Ruby

Ruby je dynamický, silně typovaný[16] jazyk, inspirovaný vlastnostmi jazyka Smalltalk a syntaxí jazyka Perl [26] – ačkoliv podle autora Ruby, Yukihiro „Matz“ Matsumoto, podobu jazyka ovlivnil i editor Emacs [15].

Mezi současnými dynamickými programovacími jazyky je Ruby do jisté míry extrémem. Prakticky jakoukoliv vlastnost programu je možné zkoumat a měnit „za běhu,“ se všemi pozitivy i negativy, která z toho vyplývají. Zpravidla to bývá využíváno k metaprogramování [27, 3c]. Framework Rails využívá metaprogramování mj. v podobě maker, která se hojně využívají u modelů, např. pro validaci atributů a definici asociací.

S dynamičností Ruby souvisí i otevřenost tříd a možnost redefinice metod, tzv. monkey patching [20, str. 293]. Každá třída může být opět otevřena a doplněna o další metody, případně existující metody mohou být zcela nahrazeny. Metody mohou být přidávány a měněny také vložením modulů (v tomto

[16] Přesněji řečeno používá duck typing, na rozdíl od slabě typovaných dynamických jazyků, jako je JavaScript či PHP, však Ruby neprovádí implicitní konverzi typů [13]

kontextu nazývanými mixins); Ruby tímto nahrazuje vícenásobnou dědičnost (multiple inheritance). Moduly však mohou být do třídy vloženy i zvenčí.

Pro představu uvedu ukázku. Mějme definovanou třídu Example s metodou hello:

```
class Example
  def hello
    "Hello"
  end
end
>> puts Example.new.hello #=> "Hello"
```

Nyní vytvoříme patch pro tuto třídu:

```
module ExamplePatch
  def greeting
    "Odelay!"
  end
end
```

Původní třídu rozšíříme o tento modul:

```
>> Example.send :include, ExamplePatch
>> puts Example.new.greeting #=> "Odelay!"
```

Pro redefinici existujících metod z modulu je vhodné použít makro `alias_method_chain` z knihovny ActiveSupport (z frameworku Rails).

```
module ExamplePatch
  def self.included(base)
    base.class_eval do
      alias_method_chain :hello, :patch
    end
  end

  def hello_with_patch
    hello_without_patch + " world!"
  end
end

>> Example.send :include, ExamplePatch
>> puts Example.new.hello #=> "Hello world!"
```

Toto makro zaručí, že interpret najde upravenou metodu a současně umožní zavolat původní metodu. Několik modulů tak může přepsat tutéž metodu a volání zřetězit.

Popsané vlastnosti Ruby dávají rozšířením mnoho možností jak pozměnit jádro aplikace bez potřeby zásahu do originálního kódu. Díky tomu je možné vůbec uvažovat o implementaci úprav v rámci rozšíření bez potřeby odštěpení od originální aplikace.

3.1.2 Architektura Ruby on Rails

Framework Rails má zajisté velký podíl na popularizaci jazyka Ruby i architektury Model–view–controller (MVC) pro webové aplikace. Jak poukazuje Krzywda [10], ve skutečnosti má MVC v pojetí Rails blíže k paradigmatu Model 2, který vznikl s ohledem na specifika webových aplikací [17]. Označení MVC používá řada webových frameworků inspirovaných Rails, nicméně rozlišení oproti architektuře Model 2 (někdy se hovoří o *MVC2*) a klasickému MVC nabývá na významu s popularizací klientských frameworků (jako například Backbone.js nebo Ember.js), které mají k původnímu návrhu MVC blíže.

Ruby on Rails, jakožto „opinionated“ framework [5, kpt. 4, Make Opinionated Software] předpokládá totožnou MVC strukturu u všech aplikací. ChiliProject v tomto směru není výjimkou, jádro aplikace je rozdělené do těchto tří vrstev:

Model
: Představuje doménovou logiku (business logic), aplikace a v případě Rails i zdroj dat.
Většinou se mapuje na odpovídající tabulku v databázi.

Controller
: Propojuje pohled a model.
Zpracovává přijatá data a připravuje data pro šablony.
Jednotlivé veřejné (public) metody controlleru představují akce.
V této vrstvě se většinou řeší autorizace.

View (pohled)
: Prezentuje data z controlleru.
Většina akcí controlleru má svůj pohled, současně tyto pohledu mohou pracovat s dílčími (partial) pohledy

Vedle toho zde existují tzv. helpers, což jsou pomocné funkce, které mohou sloužit ke sdílení funkcionality v pohledech.

Jedno z často opakovaných pravidel při vývoji Ruby on Rails aplikací je „Skinny controller; fat model“[1] – controller by měl obstarávat pouze jednoduché dotazování a přípravu dat. Většina logiky – ať už složitější dotazy nebo určité zpracování a reprezentace dat – by měla být obstarána v modelu, čímž se redukuje duplikace kódu. Bohužel, důsledné následování tohoto pravidla vede k jinému extrému – model bude mít příliš mnoho funkcí bez zjevné souvislosti [14]. ChiliProject tímto problémem místy trpí; rozdělení modelů na logicky související celky [6] by mohlo být předmětem refaktorování v budoucích verzích.

3.1.2.1 ActiveRecord

Pro objektově relační mapování (ORM) používá Rails knihovnu ActiveRecord, která implementuje stejnojmenný návrhový vzor [3, str. 160]. Důsledkem je, že model zajišťuje jak perzistenci, tak doménovou logiku aplikace. ActiveRecord v Rails, v souladu s paradigmatem „convention over configuration“ však předpokládá, že databáze bude splňovat specifické konvence pojmenování tabulek a sloupců. Tím pádem není ve většině případů nutné řešit mapování mezi databází a objekty. Samozřejmě má tento přístup svá úskalí: u nové aplikace od začátku vyvíjené pro Rails je to výhoda, v případě kompatibility s existující databází postavené na odlišných konvencích je to problém – pro takové případy však není ActiveRecord vhodné ani doporučené používat.

Konvence pojmenování v ActiveRecord jsou následující:

- Název modelu je v singuláru (`Project`), název odpovídající tabulky v plurálu (`projects`).
- Primární klíč záznamu je ve sloupci `id`, ActiveRecord nepředpokládá používání přirozených klíčů.
- Sloupce cizích klíčů mají název ve tvaru `model_id`, např. `project_id`.
- Některé názvy sloupců jsou rezervované pro ActiveRecord, především se jedná o sloupec `type` používaný pro dědění v rámci jedné tabulky (Single Table Inheritance [3, str. 278])

Většinu těchto konvencí lze u jednotlivých modelů změnit, v praxi je však jednodušší přizpůsobit se. ChiliProject tato pravidla také ve většině případů dodržuje.

Změny v databázi jsou řešené prostřednictvím migrací, což jsou krátké skripty, které provádí dílčí změny v databázi – přidávání / odebírání tabulek a sloupců, mohou také měnit existující záznamy. Migrace se provádí v pořadí, ve kterém byly vytvořeny. Každá má dvě metody: `up` a `down`, což je de facto dopředný a zpětný chod. Pokud migrace v metodě `up` vytvoří tabulku, metoda `down` tutéž tabulku smaže. Rails od verze 3 umí k některým takovým změnám automaticky přiřadit opačný postup, potom si migrace vystačí s jedinou metodou `change`.

Charakteristické pro Rails je, že databázi považuje jen za „hloupé“ úložiště dat. Ačkoliv je možné v migraci definovat některá omezení a unikátní klíče, Rails vyžaduje provádění kontroly dat na aplikační úrovni. ActiveRecord sám o sobě neřeší ani referenční integritu či kompozitní klíče; jsou-li však takové funkce vyžadovány, v řadě případů je doplní rozšíření. Případně lze ActiveRecord zcela nahradit jiným ORM frameworkem jako je například DataMapper nebo řada specializovaných knihoven pro nerelační databáze – například MongoMapper. ActiveRecord se zkrátka nesnaží být „stříbrnou kulkou“ pro všechny případy.

3.1.3 Struktura rozšíření, Rails Engines

ChiliProject používá systém Engines, který umožňuje rozšířit funkce aplikace prostřednictvím dílčích „miniaplikací“ – každý engine může mít vlastní modely, pohledy a controllery, které jsou v běžící aplikaci integrovány do jediného celku s vlastními komponentami aplikace. Engines je standardní součást Ruby on Rails od verze 3.1. Z toho důvodu už původní modul není vyvíjený, nicméně ChiliProject jej využívá z důvodu kompatibility s Rails 2.3.

Protože pouhé přidávání komponent však zpravidla nestačí, může rozšíření upravovat existující metody (viz 3.1.1) a nahrazovat výchozí pohledy – zde však samozřejmě nastává problém, pokud více rozšíření potřebuje nahradit stejný pohled. Proto má ChiliProject systém zpětných volání, tzv. hooks.[17] Ty se většinou volají z pohledů, ale v ojedinělých případech jsou použity i v controlleru či v modelu.

Vedle toho mají rozšíření k dispozici standardní mechanismy zpětných volání. Například filtry v controlleru, které se zavolají před provedením akce.[18] Ty se často používají pro autorizaci přístupu a k načtení dat pro pohled. Obdobný systém existuje i pro modely, rozšíření se může nechat zavolat například po vytvoření nového záznamu, uložení změn či před validací.[19]

Celkově je na systému rozšíření pro ChiliProject zajímavé, jak málo funkcí řeší samotná aplikace. Většinu potřeb rozšíření obstará samotné Ruby ve spolupráci s Rails. Je to příjemný rozdíl například oproti PHP, kde řada aplikací spoléhá čistě na explicitní systém událostí a zpětných volání – pokud událost není vyvolána na správném místě, jedinou možností je většinou zásah do jádra aplikace.

3.1.4 Struktura testů

Jak už jsem dříve naznačil, Ruby je dynamický jazyk ad absurdum. U složitějších aplikací je prakticky nemožné odhadnout, jak se budou chovat – nemalá část logiky vzniká až za běhu programu. I to je zřejmě jeden z důvodů, proč komunita Ruby investuje nemalé úsilí do testování kódu; není zde žádný překladač, který mnohdy odchytí nejhrubější chyby v logice programu. Jedinou zárukou funkčnosti programu je jeho úspěšný běh. Ve skutečnosti každý vývojář testuje svou aplikaci, byť manuálně. Automatizované testy vypadají jako zbytečná práce navíc, s narůstající komplexitou aplikace se však začne rychle vyplácet [25].

Ruby on Rails pro testování standardně používají framework Test::Unit, který patří do rodiny xUnit frameworků [2]. Každá třída v aplikaci má odpovídající test case. Metody test case se nazývají testy. V rámci testů se porovnává

[17] `http://www.redmine.org/projects/redmine/wiki/Hooks`

[18] Eventuálně po provedení či v obou případech, `http://apidock.com/rails/v2.3.8/ActionController/Filters/ClassMethods`

[19] `http://apidock.com/rails/v2.3.8/ActiveRecord/Callbacks`

očekávaný stav s reálným chováním aplikace pomocí různých `assert` metod (assertions). Pokud `assert` selže (např. nějaká metoda vrátí `false` namísto očekávaného `true`), test neprošel.

Rails řadí testy do několika kategorií – zjednodušeně lze říct, že kategorizace je dána typem testované třídy.

- Jednotkové (unit) testy slouží pro testování modelů a helperů.
- Funkcionální (functional) testy slouží pro testování controllerů a případně pohledů.
- Integrační (integration) testy prověřují aplikaci jako celek tím, že simulují chování uživatele.

Vedle toho někteří vývojáři pracují s akceptačními (acceptance) testy, které by měly ověřit, zda funkčnost aplikace vyhovuje požadavkům zadavatele. Zde se často používá framework Cucumber;[20] chování aplikace se popisuje v rámci tzv. features, kde jsou jednotlivé scénáře rozepsány v jazyce Gherkin – ten záměrně vypadá jako přirozený jazyk, ačkoliv se jedná o Domain Specific Language (DSL). Záměr je takový, že specifikace pro Cucumber je čitelná i pro netechnického uživatele, což by mělo usnadnit komunikaci s klientem. Jak však poukazuje [9], v praxi tato idea příliš nefunguje – výsledkem jsou často „integrační testy v přestrojení," zbytečně dlouhé, hůře strukturované, a přesto pro klienta nečitelné.

ChiliProject používá Test::Unit v kombinaci s knihovnou Shoulda.[21] Ta zahrnuje dvě funkce: jednak přidává další užitečné `assert` metody, jednak přidává podporu pro kontext – ten umožňuje sdílení stavu mezi jednotlivými testy. Zatímco Test::Unit umožňuje připravit sdílený stav mezi všemi testy v rámci jednoho test case pomocí metody `setup`, kontext umožňuje test case dále rozčlenit do menších částí, jenž mají svůj vlastní stav a neovlivňují ostatní testy. Výhodou je, že jednotlivé testy mohou být kratší, a tím pádem je snazší identifikovat problematické chování. Kromě toho Shoulda nahrazuje metody test popisnějším makrem `should`. Pro praktický příklad viz Ukázka kódu 3.1.

Testy musí mít k dispozici data, na kterých bude prověřována funkčnost aplikace. Rails standardně podporují tzv. fixtures, což jsou serializovaná data (nejčastěji ve formátu YAML, jenž je činní snadno editovatelnými), která jsou před provedením testů načtena do databáze. Fixtures musí být průběžně udržovány a s rostoucí komplexností aplikace se komplikuje i jejich struktura. Proto se místo toho většinou používají generátory dat (data factories), které vytvoří požadované objekty – včetně závislostí – až když jsou zapotřebí. ChiliProject používá oba přístupy, pro generování dat používá knihovnu ObjectDaddy.[22]

[20] `http://cukes.info/`

[21] `https://github.com/thoughtbot/shoulda`

[22] `https://github.com/edavis10/object_daddy`

Ukázka kódu 3.1: Test case s frameworky Test::Unit a Shoulda

```
class UserTest < ActiveSupport::TestCase
  # Metoda setup je sdilena pro vsechny testy
  # v~tomto test case
  def setup
    @user = User.generate!
  end

  should "be a user" do
    assert @user.kind_of?(User)
  end

  context "administrator" do
    # Makro setup se provadi jen pro testy
    # v~tomto kontextu
    setup do
      @user.is_admin = true
    end

    # Konkretni test
    should "be allowed to do anything" do
      assert @user.allowed_to?(:anything)
    end
  end
end
```

Další technikou, byť v ChiliProjectu ne příliš používanou, jsou tzv. mock objekty. S tím často souvisí i tzv. stubs, pro podrobnosti viz [4]. Mock objekty i stubs umožňují vytvořit „testovacího náhradníka“ (test double) objektu, což je užitečné v případech, kdy použití skutečného objektu je příliš náročné nebo přímo nemožné (například při práci s externím API). Kromě toho je pro mock objekty možné určit, kolikrát má být zavolána určitá metoda a s jakými parametry – pokud předpoklad není splněný, test selže. Ačkoliv Ruby činní nasazení mock objektů velice jednoduché, jejich přílišné používání může vést k zavádějícím a mnohdy i špatně udržovatelným testům [21].

Na sadě testů pro ChiliProject je zřejmé, že se již dlouho vyvíjí společně s aplikací, nicméně už samotná existence a funkčnost testů je nesmírně důležitá. Představuje konvenci na které mohou stavět testy jednotlivých rozšíření. Zároveň je možné testovat integraci rozšíření, zda jejich nasazení nenaruší standardní funkce systému.

Ožehavá otázka: „Co všechno testovat?“ Obligátní odpověď „všechno“ v praxi neobstojí a ani by to neměl být hlavní cíl testů. Jak podotýká D.H.H. [7], testy představují kód, který se také musí udržovat, jinými slovy: nejsou zadarmo. Cílem by nemělo být 100% pokrytí kódu a rozhodně by psaní testů

nemělo zabírat více času než psaní samotné aplikace. Z hlediska dlouhodobé udržitelnosti je lepší mít malou a nekompletní, zato funkční a přehlednou sadu testů, ve které se nový vývojář snadno zorientuje. Testy jsou doplňovány s nalezenými chybami, což zmenší riziko regresí. Naopak, i pokud je poměr testů vůči samotné aplikaci 3:1, není to zárukou bezchybnosti.

3.2 Možnosti implementace funkcí

Požadované funkce mohou být implementované dvěma způsoby: úpravou samotné aplikace ChiliProject nebo v rámci individuálních rozšíření. Tyto možnosti se vzájemně nevylučují, do jádra by mohly být přidané pouze některé funkce, i tak by ale tato cesta měla jisté následky.

Modifikace jádra představuje odštěpení od hlavní vývojové větve, fork. Je nutné sledovat změny v originální aplikaci a hrozí, že dojde ke konfliktům mezi forkem a originální aplikací. Integrované funkce je obtížné vzájemně odlišit a vyextrahovat v případě, že by byla zapotřebí pouze jejich část. Výhodou je naopak potenciálně „čistší“ implementace změn s menší duplikací kódu.

Naproti tomu rozšíření se dají nasadit na oficiální verzi aplikace. Každé z nich představuje ohraničenou sadu funkcí, snadno přenositelnou a snadno odstranitelnou. To má i velkou výhodu pro udržitelnost – individuální rozšíření je velice přístupné pro externí přispěvatele, kteří mohou objevit chyby a doplnit další funkce. Fork upravený pro potřeby konkrétní organizace je interní řešení, za jehož osud zcela odpovídá daná organizace.

Mým cílem bylo veškeré požadované funkce implementovat jako rozšíření – jak jsem již rozebral, jsou to především vlastnosti jazyka Ruby, které umožnily tohoto cíle dosáhnout. Jen v jediném případě jsem se nevyhnul zásahu do originální aplikace, podrobněji viz 4.2.

3.3 Cílové verze prostředí

Prvotním předpokladem implementace byla kompatibilita s poslední verzí oficiálního interpreta Ruby, Matz's Ruby Interpreter (MRI) 1.9.3. Samotný ChiliProject podporuje také verze 1.8.7 a 1.9.2, avšak pokud je to možné, nevyplatí se je používat i proto, že oproti novější verzi poskytují horší výkon. Je tomu tak i v případě Ruby Enterprise Edition (REE) [12, End of Life], což je modifikovaná verze MRI řady 1.8, která by měla poskytovat vyšší výkon pro serverové aplikace (a oproti oficiální verzi MRI 1.8 se jí to skutečně daří).

Čísla verzí jsou u MRI velice zavádějící, aplikace napsané pro MRI verze 1.9.2 mohou mít problémy s verzí 1.9.3. ChiliProject používá mnoho knihoven, ktré byly cílené na MRI řady 1.8 a v současnosti už se nevyvíjí. Přitom se problémy s kompatibilitou poměrně obtížně analyzují, většinou jsou důsledkem drobné změny v chování interpreta.

Cílová verze frameworku Rails je určená aplikací ChiliProject – ta používá poslední verzi Rails řady 2 (2.3.14), která už není oficiálně podporovaná.

Rozšíření byla testovaná s vývojovou verzí ChiliProject 3.2.0, pravděpodobně by však s menšími úpravami mohla být kompatibilní jak se staršími verzemi, tak se systémem Redmine.

Kapitola 4

Návrh a implementace rozšíření

Na základě analýzy požadavků lze určit rozložení požadovaných funkcí do samostatných rozšíření. Samozřejmě by bylo možné modifikace seskupit do jediného celku – to by sice snížilo duplikaci některých částí kódu, ale výsledek by byl nepřehledný a obtížně udržovatelný. Proto jsem funkce rozdělil do dílčích částí tak, aby každé rozšíření bylo použitelné nezávisle na ostatních; sdílené či vzájemně se překrývající části kódu mohou být vyjmuté do samostatného rozšíření.

1. Správa uživatelských skupin na úrovni projektů (4.3);
2. správa rolí na úrovni projektů, úzce souvisí i s workflow, výsledné rozšíření zahrnuje i správu workflow pro role na úrovni projektů (4.4);
3. wiki stránky s omezeným přístupem (4.5);
4. úkoly s omezeným přístupem (4.6);
5. přidávání uživatelů z LDAP (4.7).

Tato rozšíření jsem zahrnul do hlavního repozitáře *chiliproject-fit* společně s aplikací ChiliProject, lehce přizpůsobenou pro testování (nejedná se však o funkční zásahy do jádra)

Každé rozšíření pracuje s odlišnými částmi systému ChiliProject, proto se návrhu a implementaci věnuji pro každé rozšíření zvlášť. Některé základní postupy přitom zůstávají pro každé rozšíření stejné.

4.1 Příprava testovacího prostředí

Psaní testů pro rozšíření bohužel není příliš časté, ačkoliv to ChiliProject přímo podporuje. Informace v této oblasti jsou poměrně střídmé a narazil jsem na problémy s provozováním testů pod Ruby 1.9.3 – zřejmě se nikdo nepokoušel testovat rozšíření pod touto verzí.

Prvním krokem tedy bylo „otestovat testovací prostředí“ a zajistit jeho stabilitu. Po vzoru metodiky Test-Driven Development (TDD) je tak možné psát testy současně s funkčním kódem. Zároveň lze v průběhu vývoje využívat Continuous Integration (CI).

Protože veškerý kód v této práci je pod open source licencemi, je od začátku hostovaný ve veřejných repozitářích na serveru GitHub. Není to pouze otázka osobní preference, existuje řada nezávislých služeb, které nabízí projektům na GitHubu užitečné funkce navíc. Konkrétně v tomto případě jsem se rozhodl využít službu Travis-CI,[23] která poskytuje hostované řešení CI pro open source projekty zcela zdarma.

Travis-CI funguje následovně: ze serveru GitHub dostane oznámení, že v cílovém repozitáři přibyl nový commit (přes tzv. post-commit hook). Travis spustí *build*: vytvoří virtuální stroj, stáhne do něj repozitář s kódem, nainstaluje potřebné knihovny a spustí sadu testů. Ty typicky vrací návratový kód – jak je zvykem v Unixu: `0` znamená úspěch, jiné číslo neúspěch. Travis kontroluje tento návratový kód a podle toho označí build jako úspěšný či neúspěšný. Průběh buildu je možné sledovat v reálném čase přes webové rozhraní.

Použití CI má několik výhod. Je zde jednodušší provádět časově náročné testy (provedení celé testovací sady ChiliProjectu trvá 15 minut!) Také je jednodušší kombinovat různé vlastnosti prostředí; testy lze provádět například s různými databázovými systémy či jinými verzemi Ruby – kombinací těchto variant vznikne matice, tzv. *build matrix*. Má-li aplikace podporovat tři různé databázové systémy a tři verze interpreta, představuje to devět různých buildů – testovat něco manuálně by bylo přinejmenším obtížné. Hostovaná služba je pak samozřejmě výhodná, neboť vyžaduje minimální konfiguraci – do repozitáře zpravidla stačí přidat konfigurační soubor a aktivovat post-commit hook. Výsledek buildu je veřejně přístupný, může tak sloužit jako základní indikátor pro vývojáře i uživatele, zda je aktuální vývojová verze aplikace použitelná.

Travis-CI jsem použil pro dva případy: testování ChiliProjectu jako celku s instalovanými rozšířeními a testování jednotlivých rozšíření.

První případ představoval přidání konfigurace pro Travis-CI do repozitáře *chiliproject-fit*. Každé přidané rozšíření je submodul pro Git, proto je nutné před testem provést jejich inicializaci. Současně je nutné připravit testovací databázi – poskytnout údaje pro připojení, databázi vytvořit a spustit migrace. Ve výsledku se de facto jedná o integrační test – v systému jsou zavedená všechna rozšíření a úspěšné provedení testů je minimální zárukou toho, že žádné z nich nenarušilo funkce systému.

Druhý případ je složitější – rozšíření bez aplikace, kterou rozšiřuje, nelze spustit, natož testovat. Tento problém řešili vývojáři systému Radiant [19]. Před testováním se provede skript, který stáhne samotnou aplikaci a přesune do ní (již stažené) rozšíření. Skript jsem přizpůsobil pro ChiliProject a zobecnil jsem jej tak, aby byl snadněji konfigurovatelný pro další rozšíření. Kostrou

[23] `http://travis-ci.org/`

tohoto řešení je rozšíření `chiliproject_test_plugin` které posloužilo jako prototyp k odladění.

Při testech reálných rozšíření jsem však narazil na nepříjemný problém.

4.2 Oprava Engines

Rozšíření pro ChiliProject jsou postavená na modulu Engines a využívají jej i testy. Rozšíření zpravidla používá fixtures originální aplikace, zároveň může soubory fixtures přidávat nebo i nahrazovat. Engines to řeší tak, že postupně zkopíruje soubory z původní aplikace a z rozšíření do dočasného adresáře, cestu k němu pak přidá na začátek globální proměnné `$LOAD_PATH`. To zaručí, že se fixtures načtou z tohoto adresáře.

V Ruby 1.9 se však změnila specifikace standardní funkce pro vytvoření adresáře `FileUtils.mkdir_p` – původně vracela řetězec, nyní vrací pole. Engines (v modulu `Engines::Testing`) přidávají výstup této funkce bez další kontroly do proměnné `$LOAD_PATH` – pod Ruby 1.9 se tak v této proměnné ocitne pole, což zcela zmate systém načítání závislostí. Volání funkce `require` (pro načtení souboru) pak způsobí pád testů.

Oprava této chyby byla poměrně banální, největším problémem bylo ji nalézt a otestovat – testy Engines jsou pod Ruby 1.9 zcela nefunkční. Výsledkem je opravená verze modulu `engines`, kterou jsem zahrnul do distribuce *chiliproject-fit*. Problém byl nahlášený vývojářům ChiliProject, bohužel oprava dosud nebyla přenesena do oficiální verze.[24]

S funkčním testovacím prostředí už bylo možné začít implementovat jednotlivá rozšíření.

4.3 Správa uživatelských skupin na úrovni projektů (ChiliProject Project Group)

4.3.1 Specifikace

Cílem tohoto rozšíření je decentralizovat správu uživatelských skupin tak, aby se jejich vytváření a modifikace obešla bez zásahu administrátora. Jako nejvhodnější možnost se nabízí navázání skupin na projekt, kde už s nimi bude moct pracovat správce projektu.

Správce projektu s oprávněním pro modifikaci projektových skupin může:

1. Vytvořit novou či smazat existující projektovou skupinu,
2. přidat a odebrat členy projektové skupiny,
3. změnit název skupiny

[24] `https://www.chiliproject.org/issues/952`

Správce projektu s oprávněním pro správu členů projektu může:

4. Přiřadit skupině roli v projektu; příslušnou roli dostanou všichni členové skupiny.

Projektové skupiny jsou k dispozici všem podprojektům (lze jim přiřadit roli – požadavek 4), nelze je ale v rámci podprojektu modifikovat (požadavky 1 až 3).

Nadále bude možné používat i standardní funkci globálních skupin, které jsou k dispozici všem projektům a může je modifikovat pouze administrátor.

4.3.2 Existující řešení

Některé z požadavků částečně splňuje rozšíření Group Assignee,[25] které umožňuje přiřadit skupině zodpovědné osoby, jenž mohou přiřazovat a odebírat členy. Ačkoli to částečně řeší decentralizaci správy, stále je nutné, aby administrátor skupiny nadefinoval. Navíc všechny skupiny stále existují v globálním kontextu, což při jejich větším množství vede k nepřehlednosti.

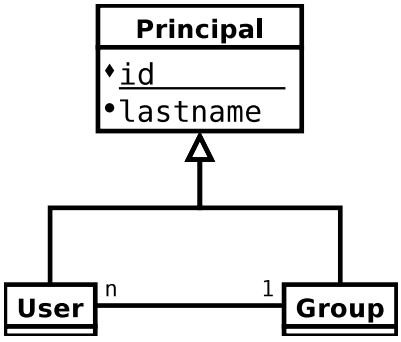
4.3.3 Systém uživatelů, skupin a rolí

ChiliProject řeší správu uživatelů prostřednictvím tří modelů:

Principal je abstraktní třída, která seskupuje společné metody uživatele a skupin.

User je podtřída Principal a představuje konkrétního uživatele.

Group je podtřída Principal, která může mít více uživatelů.

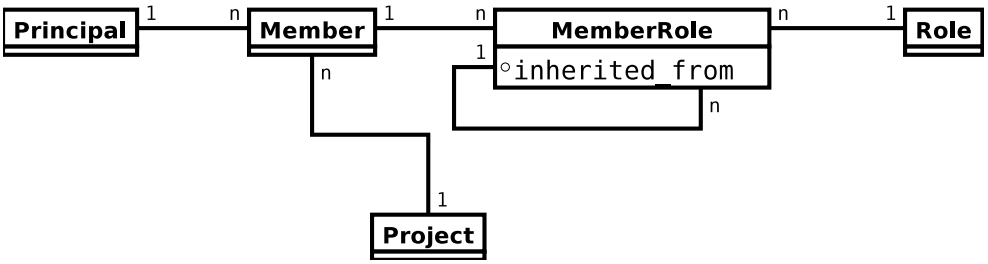


Obrázek 4.1: Vztah modelů Principal, User a Group

Model Principal se používá v případech, kdy nezáleží na tom, zda pracujeme s uživateli či skupinami, například při přidávání členů projektu.

Přidáním uživatele (skupiny) do projektu vzniká člen (Member). Každý člen projektu může mít více rolí; dekompozicí této relace je model MemberRole.

[25] `http://www.redmine.org/plugins/groupsassignee`



Obrázek 4.2: Vztah uživatelů, rolí a projektů

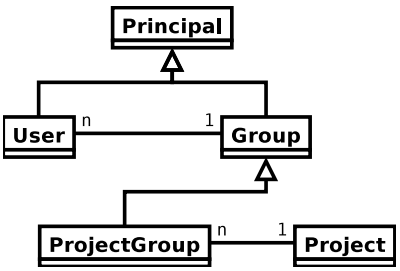
Při přidání skupiny do projektu se reálně přidají členové skupiny jako členové projektu. ChiliProject mezi takto „zděděnými“ uživateli a individuálně přidanými uživateli rozlišuje na úrovni modelu MemberRole prostřednictvím atributu `inherited_from`, který odkazuje na původní záznam MemberRole skupiny. Tento model nevypadá příliš elegantně, ale na druhou stranu tím, že je dědičnost určená pro každou členskou roli, je pak možné řešit i poměrně složité situace. Například když je do projektu přidán tentýž uživatel jednou přímo a podruhé v rámci skupiny. V tomto systému si zachová jak svou původní roli, tak zdědí roli skupiny. Pokud bude skupina z projektu později odebrána, uživatel přijde pouze o svou roli. Pokud člen nemá v projektu žádnou roli, je jeho záznam po odebírání poslední role odstraněn.

4.3.4 Model projektové skupiny

Co potřebujeme je speciální typ skupiny, která bude vázaná na konkrétní projekt. K tomu lze využít dědičnost, nový model ProjectGroup je podtřídou modelu Group. Zároveň patří (belongs to) rodičovskému projektu (asociace `parent_project`, model Project).

Díky dědičnosti nemusí rozšíření řešit přidávání a správu členství v projektech – o to se postará standardní implementace třídy Group a polymorfismus.

Model Project v tomto vztahu má mnoho (has many) projektových skupin. Také je zde nutné upravit asociaci `member_principals` která představuje



Obrázek 4.3: Model ProjectGroup

všechny uživatele a skupiny, jenž jsou členy daného projektu. ChiliProject zde vybírá konkrétní typy záznamů prostřednictvím sloupce `type` – do podmínky je proto nutné zahrnout i typ ProjectGroup.

```
# module ProjectGroupPlugin::ProjectPatch
has_many :member_principals, :class_name => 'Member',
:include => :principal,
:conditions =>
   "#{Principal.table_name}.type='ProjectGroup' OR
   #{Principal.table_name}.type='Group' OR
   (#{Principal.table_name}.type='User' AND
   #{Principal.table_name}.status=#{User::STATUS_ACTIVE})"
```

Vícenásobné volání makra `has_many` nahradí předchozí definici, proto bude asociace definovaná v rozšíření brána v potaz.

Větší problém je s validátory, které se řetězí – vícenásobná definice validace na stejný atribut pouze přidá další zpětné volání na konec seznamu validací. To však vadí, pokud chceme z rozšíření nastavit méně přísné podmínky platnosti záznamu – v tomto případě se jedná o unikátnost názvu skupiny. Standardně musí mít každá skupina unikátní název; u projektové skupiny však stačí, aby její název byl unikátní pouze v rámci daného projektu. Naštěstí jsem nebyl první, kdo podobný problém řešil [16] – výsledek není moc pěkný, ale je ukázkou síly metaprogramování v Ruby.

Na model Group se při inicializaci aplikuje toto volání:

```
# module ProjectGroupPlugin::GroupPatch
@validate_callbacks.reject! do |c|
  begin
    if Proc === c.method && eval("attrs",
       c.method.binding).first == :lastname &&
       c.options[:case_sensitive] == false
      true
    end
  rescue
    false
  end
end
```

Z třídní proměnné `validate_callbacks` se odstraní ta validace pro atribut `lastname`, která má možnost `case_sensitive` nastavenou na **`false`** – pokud je pro tento atribut definováno více validací, poslední kontrola zaručuje, že se odstraní pouze kontrola unikátnosti (jiné validace s volbou `case_sensitive` nepracují). Následně se definuje nová validace omezená na konkrétní projekt; globální skupiny nemají projekt určený, a tím pádem se budou také validovat se stejným omezením – je tak zachována podmínka unikátnosti názvů globálních skupin.

4.3.4.1 Dědění skupin v hierarchii projektů

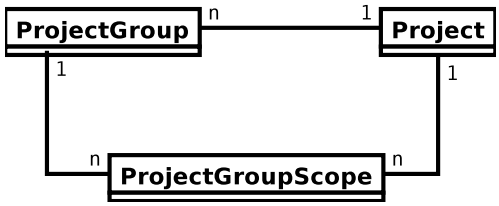
Pro splnění požadavku dědění skupin do podprojektů se nabízí dvě řešení. Buď se bude relace mezi skupinami a podprojekty udržovat v databázi, nebo se zděděné skupiny určí spojením projektů se skupinami a určí se z hierarchie projektů.

První možnost je jednodušší z hlediska Rails a potenciálně nabízí více možností, například pokud by se projektové skupiny dědily mimo hierarchii podprojektů nebo by měly mít v rámci různých projektů různé vlastnosti. Nevýhodou je pak riziko vzniku nekonzistence v důsledku jisté duplikace dat a náročnost některých operací – při manipulaci s projekty či projektovými skupinami (např. přesun projektu, vytvoření nového projektu či skupiny) se musí udržovat vazby v hierarchii podprojektů.

Druhá možnost je čistší z hlediska databáze, informace o dědičnosti jsou na jediném místě.

Pro demonstraci jsem se rozhodl implementovat obě řešení v rámci dvou rozšíření – skupiny používají první možnost, role používají možnost druhou.

Mezi modely ProjectRole a Project vznikne nová M:N vazba, tu jsem dekomponoval do modelu ProjectGroupScope (konvence ActiveRecord nabízela název ProjectGroupProject, což mi nepřišlo moc rozumné).

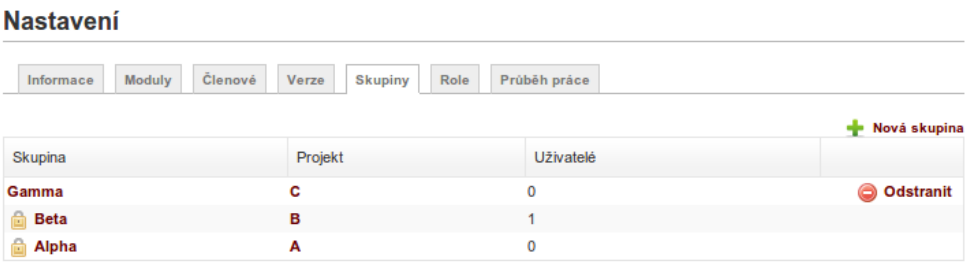


Obrázek 4.4: Vazba ProjectGroup–ProjectGroupScope–Project

Projektové skupiny se budou vždy vybírat přes tuto vazbu, budou zde proto zahrnuty i vazby mezi rodičovskými projekty a jejich vlastními potomky.

Pro udržování vazeb lze použít zpětná volání ActiveRecord – konkrétně když je projektová skupina vytvořena, musí se propagovat do podprojektů rodičovského projektu – k tomu poslouží zpětné volání `after_create`, ve kterém skupina vytvoří vazby na následníky projektu. Při odstranění skupiny se vazby odstraní automaticky.

Zvláštní případ je však přesun projektu. Zde se nejedná o zpětné volání ActiveRecord, ale o metodu `set_parent!` v ChiliProjectu, která se volá při každé změně hierarchie. V takovém případě se přebudují vazby pro přesunutý projekt a jeho potomky (projekt lze přesunout pouze jako celek se všemi potomky, nikoliv však do svého podstromu). Přesunutý projekt nahradí původní zděděné skupiny skupinami svého rodiče, a to se rekurzivně opakuje až k listům. Tato operace se provede i při vytvoření nového projektu, nejvýše však jednou; nový projekt nemůže mít žádného potomka.



Obrázek 4.5: Správa projektových skupin

4.3.5 Manipulace se skupinami, autorizace

Aby bylo možné projektové skupiny spravovat, je pro ně nutné vytvořit administrační rozhraní. Zde lze vyjít ze standardní správy skupin, kterou ChiliProject obsahuje.

Správa skupin je umístěná v nastavení projektu – zde je možné prohlížet projektové skupiny, které má daný projekt k dispozici, a vytvářet nové. Patří-li skupina tomuto projektu, je možné ji upravovat (přidávat uživatele, měnit název). Pokud ne, může si správce pouze zobrazit seznam členů této skupiny.

Na záložce Členové, v nastavení projektu, se projektové skupiny zobrazí stejně jako globální skupiny a je možné mezi nimi vyhledávat. K tomu však bylo nutné nahradit odpovídající pohled `projects/settings/members` – protože totéž bylo nutné později provést i pro projektové role, vyextrahoval jsem tento pohled do samostatného rozšíření *Members View*, které funguje také jako gem.

Z hlediska autorizace přidává rozšíření jedno oprávnění: *Manage project groups* – má-li uživatel roli s tímto oprávněním (společně s možností spravovat projekt), bude moct pracovat se skupinami v daném projektu. K oprávnění se vážou specifické akce controlleru ProjectGroupsController, kde se navíc kontroluje, zda daná skupina patří projektu a je ji tím pádem možné měnit.

4.3.6 Testy

Testy pokrývají především funkčnost controllerů (včetně autorizace), patchů a složitějších operací s modely, jako je například zmíněná změna hierarchie projektů.

Dochází zde k určitému překryvu mezi funkcionálními a jednotkovými testy, nicméně jsem se snažil klást důraz na funkcionální testy, které lépe vystihují konečné chování rozšíření.

Pouze jedna chyba se projevila až při nasazení na produkční server: administrace globálních skupin fungovala do té doby, než uživatel jednou otevřel správu projektových skupin. Systém se v administraci globálních skupin

nepokoušel načíst z databáze projektové skupiny, pokud jejich třída nebyla předtím použita. To je důsledek cachování tříd, ovšem tato funkce je ve vývojovém režimu Rails vypnutá, aby nebylo nutné restartovat aplikační server při každé změně kódu.[26]

4.3.7 Zhodnocení

Rozšíření splňuje všechny požadavky a bylo otestováno jak automatickými testy, tak manuálně. Jedna z funkcí, která by mohla být v budoucnu implementována, je kopírování skupin, což by bylo užitečné zejména v kontextu projektových skupin (mohu si zkopírovat zděděnou skupinu do svého projektu a dále ji upravovat).

4.4 Správa rolí a workflow na úrovni projektů (ChiliProject Project Roles)

4.4.1 Specifikace

Cíl tohoto rozšíření je podobný jako u projektových skupin – poskytnout správcům projektů více možností a více decentralizovat správu. Toto rozšíření však může mít mnohem větší dopad na práci se systémem ChiliProject.

Uživatel s oprávněním pro modifikaci projektových rolí může:

1. Vytvářet či mazat projektové role v kontextu daného projektu,
2. upravovat práva projektových rolí,
3. změnit práva role anonymous či non-member v kontextu projektu.

Kromě rolí definovaných administrátorem má ChiliProject navíc dvě vestavěné role: *anonymous* (pro nepřihlášeného uživatele) a *non-member* (pro uživatele, který v kontextu projektu není jeho členem).

S funkcemi rolí je úzce spjatý systém workflow – pokud by tato funkce nebyla zahrnuta, projektové role by mohly používat issue tracker pouze omezeně.

Uživatel s oprávněním pro modifikaci projektového workflow může:

4. Měnit a kopírovat workflow pro projektové role.

[26] Novější verze Rails navíc detekují změny v souborech a obnovují jen ty třídy, které byly změněny.

4.4.2 Existující řešení

Část požadovaných funkcí nabízí rozšíření Role Shift,[27] které funguje na principu *posunu* role – jedna role se tak v určitém projektu může chovat jako jiná. Ačkoli to nezní moc užitečně, částečně to řeší požadavek 3, protože je tím možné předefinovat i vestavěné role. Právě pro řešení vestavěných rolí se tato strategie ukázala jako nejschůdnější.

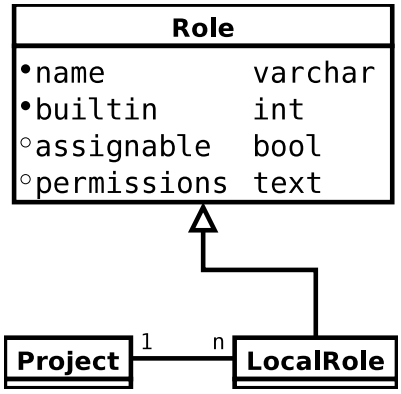
4.4.3 Systém rolí a oprávnění

Jaký je vztah rolí vůči uživatelům a projektům už jsem naznačil (4.3.3). Oprávnění nemají v systému samostatný model, většinou se s nimi pracuje na úrovni rolí jako s polem symbolů.

Pokud je zapotřebí zajistit autorizaci uživatele, použije se instanční metoda `Principal#allowed_to?` která dostane symbol představující určité oprávnění (například `:manage_project_roles`), kontext (tím může být projekt, pole projektů, nebo `nil`) a volbu, zda se má kontrolovat globální kontext – ten se používá například u vytváření nových projektů, které nemají rodiče, tj. tam, kde nelze vyjít z kontextu existujícího projektu. Pro ověření autorizace se projdou všechny role, které daný uživatel má v daném kontextu – projektu (či v projektech). Pokud se u nějaké role najde hledané oprávnění, je autorizace úspěšná.

4.4.3.1 Model projektových rolí

Obdobně jako u projektových skupin je projektová role (LocalRole) podtřídou standardního modelu Role a přidává vazbu na rodičovský projekt.



Obrázek 4.6: Model LocalRole

Projektová role se z hlediska autorizace chová stejně jako normální role, s výjimkou autorizace v globálním kontextu – v takovém případě nesmí být

[27] `http://projects.andriylesyuk.com/projects/role-shift`

projektové role brány v potaz a z kontroly práv jsou zcela vyloučeny. V kontextu projektu se pak kontrolují pouze ty role, které má uživatel v tomto projektu přiřazené, není zde tedy nutné dodatečně kontrolovat, zda prověřovaná role do projektu skutečně patří. Nicméně tato kontrola je prováděná u člena projektu (model Member), kde validace zabrání vzniku členství s rolemi, které nepatří do hierarchie projektu.

Pro dědění rolí do podprojektů jsem z popsaných možností (4.3.4.1) použil druhou variantu. Jádrem tohoto řešení je scope v modelu Role, který vybere projektové role z předchůdců rodičovského projektu společně s globálními rolemi:

```
named_scope :available_for_project, lambda { |project|
{:joins => "LEFT OUTER JOIN projects ON
   roles.local_role_project_id = projects.id",
 :conditions => ["roles.type = 'Role' OR (roles.type =
    'LocalRole' AND projects.lft <= ? AND projects.rgt >=
    ?)", project.left, project.right] }
}
```

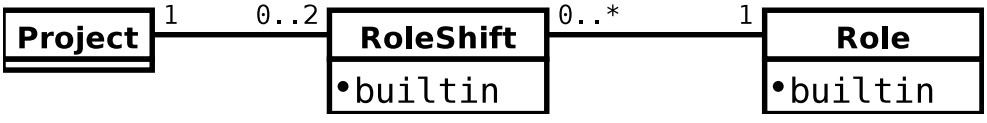
Obdobné řešení lze použít, pokud takto chceme vybrat pouze lokální role.

Tímto lze vyřešit vytváření a aplikování projektových rolí, jak si však poradit se speciálními rolemi?

4.4.3.2 Posun rolí

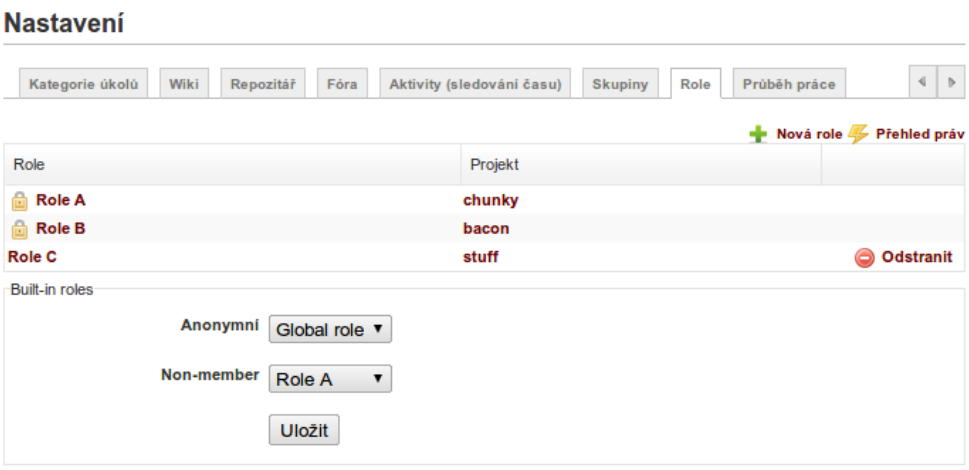
Uvažoval jsem nad různými variantami, jak v rozšíření zpracovat speciální role *non-member* a *anonymous*. Většina návrhů měla problém s děděním rolí. Nebylo možné je vyřešit dostatečně transparentně a přívětivě pro správce projektu, nebo narážela na prostý fakt, že ChiliProject počítá s tím, že v databázi existuje pouze jediná speciální role daného druhu. Nakonec jsem použil modifikaci řešení od Andriye Lesyuka ze zmíněného rozšíření Role Shift.

Do aplikace je přidán nový model RoleShift, který je dekompozicí M:N relace mezi projektem a rolí.



Obrázek 4.7: Model RoleShift

Každý projekt může mít nejvýše dva záznamy RoleShift: `role_non_member` a `role_anonymous` (typ je určený pomocí atributu `builtin`). Pokud tyto záznamy existují, budou s nimi asociované role použity namísto odpovídajících globálních rolí.



Obrázek 4.8: Správa projektových rolí

Správce tak má pod kontrolou, zda chce výchozí globální roli nahradit či nikoliv, navíc je možné použít jednotnou správu projektových rolí a jejich dědičnost.

4.4.3.3 Manipulace s projektovými rolemi

Pro controller a pohledy bylo opět možné vyjít ze standardní správy rolí.

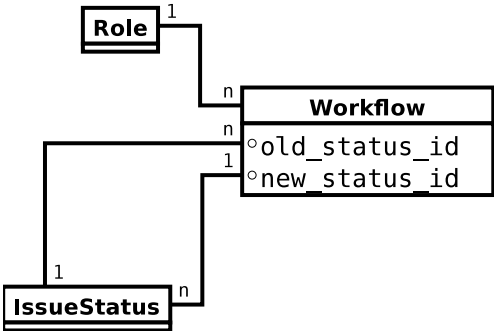
Autorizace je rozdělená do dvou oprávnění – *Manage project roles* a *Manage role shifts*. Důvodem je, že o projektové role a o posuny rolí se starají dva různé controllery a ChiliProject nenabízí způsob, jak sloučit autorizaci k více controllerům pod jediné oprávnění. Návrh však předpokládá, že tato dvě oprávnění budou používaná společně.

4.4.4 Správa workflow

V případě workflow není nutné přidávat další modely – je možné využít standardní asociace s modelem Role.

V rámci rozšíření tak stačí zajistit možnost správy workflow ve vlastním controlleru (LocalWorkflowsController). Ten je téměř stejný jako standardní controller (WorkflowsController), rozdíl je však v rozsahu rolí, se kterými pracuje. V rámci projektu je možné editovat pouze workflow těch rolí, které bezprostředně patří danému projektu.

I workflow mají samostatné oprávnění *Manage project workflows*.



Obrázek 4.9: Vazby Workflow

4.4.5 Testy

Logika testů pro práci s rolemi a workflow byla velice podobná standardním testům odpovídajících funkcí v ChiliProject, nicméně jsem testy přepsal do konvencí Shoulda a rozšířil je o kontrolu autorizace a dalších omezení.

4.4.6 Zhodnocení

Požadavky byly splněny a funkčnost rozšíření byla otestována. Podobně jako u projektových skupin se pro další vývoj nabízí implementace kopírování rolí.

Nabízí se zde také jistý prostor pro optimalizaci dotazování databáze – zejména posun rolí se provádí při téměř každé akci bez ohledu na to, zda nějaká náhradní role existuje – to však vyžaduje podrobnější určení scopes vhodných pro optimalizaci.

Vzhledem ke komplexitě a množství metod, které rozšíření nahrazuje v jádru aplikace, může dojít ke konfliktům s jinými rozšířeními. Takové případy však bude nutné řešit individuálně úpravou tohoto nebo konfliktního rozšíření.

4.5 Wiki stránky s omezeným přístupem (ChiliProject Private Wiki)

4.5.1 Specifikace

Rozšíření by mělo umožnit vytváření soukromých stránek na wiki, které budou přístupné pouze uživatelům s odpovídajícím oprávněním. Zcela původní návrh hovořil o komplexní správě přes Access Control List (ACL) s individuální správou konkrétních akcí (prohlížení, úprava, přístup k historii atp.), ale nakonec bylo upřednostněno jednodušší řešení. Mimo jiné s ohledem k tomu, že správa projektových rolí může tyto funkce celkem dobře nahradit. Role však pracují s wiki v projektu jako celkem, proto toto rozšíření umožňuje jemnější řízení přístupu ke konkrétním stránkám. Rozšíření by mělo respektovat hie-

Obrázek 4.10: Soukromá stránka na wiki

rarchii stránek – pokud je soukromý některý z předchůdců, bude daná stránka také soukromá.

Uživatelé budou moct získat dvě nová oprávnění: správa viditelnosti stránky a přístup na soukromé stránky. První oprávnění dává uživateli možnost nastavit stránku jako soukromou, druhé umožňuje aby vůbec mohl vidět obsah stránky.

4.5.2 Existující řešení

V tomto případě už existuje poměrně funkční řešení v podobě rozšíření Redmine Private Wiki,[28] které napsal Oleg Kandurov. Autor ovšem nespecifikoval licenci. Po ujištění, že je kód dostupný pod permisivní MIT licencí (která byla zachována), jsem vytvořil fork rozšíření, obligátně nazvaný ChiliProject Private Wiki.

4.5.3 Úprava rozšíření Private Wiki

Rozšíření přidává k wiki stránkám dodatečný booleovský atribut `private`. Pro zajištění dodatečné autorizace se provede `before_filter` v controlleru wiki stránek (WikiPagesController), který po standardní autorizace zkontroluje, zda je stránka soukromá a zda daný uživatel má oprávnění k ní přistupovat. Mimo to zajišťuje rozšíření změnu atributu `private` prostřednictvím nové akce v controlleru – tu může oprávněný uživatel vyvolat prostřednictvím odkazu na wiki stránce. Uživatelské rozhraní je upraveno pomocí JavaScriptu – pohledy bohužel nemají dostatek zpětných volání, aby je bylo možné rozšířit na straně serveru. O přístupnosti pro uživatele s vypnutým JavaScriptem však beztak nelze hovořit, protože samotný framework Rails vyžaduje JavaScript i pro běžnou uživatelskou interakci.[29]

Rozšíření má však některé nedostatky – nerespektuje hierarchii stránek a nemá testy.

Na rozdíl od projektů je hierarchie wiki stránek v databázi řešena jako jednoduchý strom, kdy stránka odkazuje pouze na svého bezprostředního rodiče přes sloupec `parent_id`. Při procházení ke kořeni se pro každého předka provede nový dotaz do databáze – na druhou stranu, Rails v rámci jedné akce implicitně cachuje výsledky totožných dotazů, tudíž se tato operace provede

[28] `https://github.com/f0y/redmine_private_wiki`

[29] Rails 3 používá „unobtrusive JavaScript“ což tento problém z velké části řeší.

pouze jednou. Kontrola soukromých stránek v rámci hierarchie bude mít na výkon aplikace jen minimální dopad – ChiliProject beztak vypisuje seznam předků stránky (tzv. drobečková navigace), takže databáze dostane tuto sadu dotazů i bez tohoto rozšíření.

V tomto případě, vzhledem k tomu, že jde o rozšíření, které zpřísňuje autorizaci, jsou nejdůležitější funkcionální testy controlleru. Použil jsem zde (podobně jako u ostatních rozšíření) hromadný test, zda soukromá stránka zablokuje všechny akce pro neautorizovaného uživatele:

```
{:rename => :get, :edit => :get, :update => :put,
   :protect => :post, :history => :get,
 :diff => :get, :annotate => :get, :add_attachment =>
    :post, :destroy => :delete}.each do |action, verb|
  context "#{verb.to_s.upcase} #{action}" do
    setup do
      self.send verb, action, :project_id => @project,
         :id => @page.title
    end
    should_respond_with 403 # access denied
  end
end
```

Tento fragment vytvoří devět testů pro různé akce s příslušnými metodami a očekává, že ve všech případech controller vrátí HTTP kód 403 (přístup zakázán). Není to samozřejmě neprůstřelná metoda; pokud do controlleru přibude nová akce, je nutné test aktualizovat – zde by bylo zajímavé využití reflexe pro automatické určení metod pro testování.

4.5.4 Zhodnocení

Toto rozšíření je poměrně jednoduché jak z vývojářského, tak z uživatelského hlediska. Zajímavé bude sledovat jeho využívání v praxi – v kombinaci s projektovými rolemi se jedná o zajímavou alternativu k plnohodnotnému ACL, která jen minimálně zatěžuje uživatele.

Další vývoj by mohl například přidat přehled soukromých stránek a případně jejich hromadnou správu. Užitečná by byla i možnost přímého vytvoření soukromé stránky – volba by byla dostupná hned ve formuláři.

4.6 Úkoly s omezeným přístupem (ChiliProject Private Issues)

4.6.1 Specifikace

Cíl je podobný jako u soukromých wiki stránek – některé úkoly v issue trackeru budou označené jako soukromé, a uvidí je pouze oprávnění uživatelé. Jsou zde

však jistá specifika; uživatel by měl vidět úkol, který sám vytvořil, nebo mu byl přiřazen, a to bez ohledu na to, zda má příslušné oprávnění, nebo ne.

Opět jsou zde dvě oprávnění: nastavení viditelnosti úkolu a přístup k soukromým úkolům. Uživatel má přístup k soukromému úkolu, pouze pokud má příslušné oprávnění, je autorem úkolu a nebo byl k němu přiřazen – obdobně jako s přístupem je to i s viditelností úkolu v přehledu úkolů.

4.6.2 Existující řešení

V této oblasti je Redmine napřed – viditelnost úkolů je možné určit od verze 1.2.[30] ChiliProject tuto funkci neimplementoval s odvoláním na to, že systém oprávnění bude beztak zapotřebí kompletně přepracovat,[31] bohužel se tato iniciativa zatím nedostala moc daleko. Implementace v Redmine alespoň může posloužit jako reference pro samostatné rozšíření.

4.6.3 Implementace rozšíření

Základní idea je stejná jako u wiki stránek – controller zakáže přístup, pokud uživatel neprojde autorizací. Oříškem je však rozhraní pro vybírání úkolů z databáze – model Query. Kromě jednoduchého výpisu úkolů umožňuje i předdefinovat specifické kombinace dotazů. Uvnitř se jedná o velký generátor SQL, který občas může vrátit klauzuli `WHERE 0=1`. Rozšíření nemá mnoho možností, jak ovlivnit generování dotazu. Jedině odchytit vygenerovaný řetězec s dotazem a připojit k němu vlastní podmínky:

```
# module PrivateIssues::QueryPatch
def statement_with_private_issues
  filters_clauses = statement_without_private_issues

  user = User.current
  unless project &&
      user.allowed_to?(:view_private_issues, project)
    filters_clauses << " AND (issues.private =
        #{connection.quoted_false} OR issues.author_id =
        #{user.id} OR issues.assigned_to_id = #{user.id})"
  end
  filters_clauses
end
```

Testy pak kontrolují, zda se úkol zobrazí ve výpisu uživateli s oprávněním pro prohlížení soukromých úkolů, stejně tak i pro uživatele, který je k úkolu přiřazený. Obdobně je tomu i s přístupností controlleru.

[30] http://www.redmine.org/issues/7414

[31] https://www.chiliproject.org/issues/189

Obrázek 4.11: Soukromý úkol

4.6.4 Zhodnocení

Tato implementace sice není tak funkční jako vestavěná funkcionalita v Redmine, nicméně svoji úlohu plní. V budoucích verzích by zřejmě bylo dobré rozdělit přidružená oprávnění, což by umožnilo podrobněji definovat, co uživatelé mohou, a co ne. Bylo by možné také hlouběji propracovat integraci s vyhledáváním úkolů – mohlo by se jednat o kritérium, což by umožnilo snadno nalézt všechny soukromé úkoly.

4.7 Přidávání uživatelů z LDAP (ChiliProject Add LDAP Users)

4.7.1 Požadavky

Pro přidání uživatele do systému ChiliProject je nutné projít registrační procedurou – vyplnit jméno, heslo, e-mail... Přitom všechny tyto údaje jsou k dispozici přes LDAP, který systém již umí používat. I když se uživatel poprvé přihlásí s údaji z LDAP, je mu automaticky vytvořen účet bez nutnosti cokoliv vyplňovat. Je tedy celkem logické, aby existovala funkce, která umožní administrátorovi importovat uživatele zadáním uživatelského jména.

Rozšíření přidá novou stránku do administrace, kde je možné zadat uživatelská jména – pokud je takto nalezen uživatel, který doposud není v databázi ChiliProjectu, bude mu automaticky vytvořen účet, stejně jako by se zaregistroval.

4.7.2 Implementace rozšíření

Autentizace uživatelů je řešena přes abstraktní model AuthSource. ChiliProject standardně obsahuje autentizaci přes LDAP – model AuthSourceLdap. Z toho

je zřejmé, že autentizace je modulární; lze snadno definovat nové autentizační zdroje a použít je v projektu, přičemž postačí, když budou dodržovat definované rozhraní.

Toto rozšíření však pracuje s modelem AuthSourceLdap. Ten umí autentizovat uživatele dle zadané dvojice jméno-heslo a vrátit údaje pro další zpracování. Nejprve z adresáře stáhne údaje o uživateli, čímž také ověří, že vůbec existuje (`AuthSourceLdap#get_user_dn`), následně se pokusí s předanými údaji k adresáři připojit. V rozšíření potřebujeme provést pouze první část – získat z adresáře údaje o uživateli – a následně vytvořit záznam uživatele. Patch pro AuthSourceLdap přidává metodu `get_user`, se kterou pak pracuje samostatný controller `LdapUsersController` dostupný přes administraci.

Součástí rozšíření nejsou testy – v praxi se ukázalo jednodušší vyzkoušet funkčnost s fakultním LDAP serverem, než server simulovat. Pravděpodobně by ani testy nepomohly objevit problém s kódováním řetězců získaných přes LDAP. Vypsání těchto řetězců (typicky se jednalo o plné jméno přidávaného uživatele) způsobilo chybu ve vykreslování stránky. Mock server by tuto chybu neodhalil.

4.7.3 Zhodnocení

Toto rozšíření je velice jednoduché a považuji jej spíš za funkční prototyp, který by se mohl stát součástí rozsáhlejší integrace služeb LDAP serveru. Při přidávání uživatelů by ChiliProject mohl, například při přidávání uživatele do projektu, „našeptávat“ uživatele nalezené v adresáři podle zadaných údajů, a následně jim vytvořit účty – částečně by se tak smazala hranice mezi lokální databází a externími zdroji.

Alternativou by bylo dále rozvíjet systém přidávání uživatelů, který by umožnil vyhledávání a hromadný import uživatelů na základě určitých kritérií – například uživatelské skupiny v LDAP.

Jako minimální implementace však může sloužit toto rozšíření už nyní.

Závěr

Cílem práce bylo analyzovat a zvolit nejvhodnější řešení pro správu projektů v Oddělení ICT Fakulty informačních technologií. Pro vybraný systém pak byla implementována rozšíření, která ho dále přizpůsobila potřebám oddělení. Funkčnost rozšíření byla otestována manuálně i prostřednictvím automatických testů, dále byla ověřena vzájemná nekonfliktnost a integrace všech rozšíření – systém jako celek prošel všemi automatickými testy. Z tohoto hlediska lze práci hodnotit jako úspěšnou.

Mým cílem bylo implementovat veškeré přidané funkce jako rozšíření, bez zásahu do jádra aplikace – tento cíl se mi také podařilo splnit a věřím, že to bude mít pozitivní vliv jak pro ICT oddělení v podobě snadnější správy aktualizací, tak pro budoucí vývoj jednotlivých rozšíření, která mohou být zapojena do komunitního vývoje.

Rozšíření jsou v současnosti provozována na testovacím serveru; nepředpokládám, že nový systém bude ihned uveden do provozu, bude vhodné jej nejprve otestovat s reálnými uživateli a daty. Jako jedno z řešení pro postupnou migraci se nabízí provoz testovacího serveru nad replikou databáze současného systému. Testovací provoz je i vhodnou příležitostí pro nasazení moderního software, ať už je to Ruby 1.9.3 či poslední generace aplikačního serveru Passenger [11].

Modifikace uvedené v této práci představují pouhý zlomek funkcí, které mohou dále zjednodušit správu projektů a zlepšit uživatelský komfort.

Literatura

(1) Buck, J.: Skinny Controller, Fat Model. 2006. Dostupné z WWW: <http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model>

(2) Fowler, M.: Xunit. Dostupné z WWW: <http://www.martinfowler.com/bliki/Xunit.html>

(3) Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2003, ISBN 0321127420, 533 s. Dostupné z WWW: <http://books.google.com/books?id=FyWZt5DdvFkC&pgis=1>

(4) Fowler, M.: Mocks Aren't Stubs. 2007. Dostupné z WWW: <http://martinfowler.com/articles/mocksArentStubs.html>

(5) Fried, J.; Hansson, D. H.; Linderman, M.: *Getting Real: The smarter, faster, easier way to build a successful web application*. 37signals, 2009, ISBN 0578012812, 194 s. Dostupné z WWW: <http://gettingreal.37signals.com/toc.php>

(6) Grimm, A.: Objects on Rails. 2012. Dostupné z WWW: <http://objectsonrails.com/>

(7) Hansson, D. H.: Testing like the TSA. Duben 2012. Dostupné z WWW: <http://37signals.com/svn/posts/3159-testing-like-the-tsa>

(8) Holman, Z.: How GitHub Uses GitHub to Build GitHub. Říjen 2011. Dostupné z WWW: <http://zachholman.com/talk/how-github-uses-github-to-build-github>

(9) Kinsella, J.: Why Bother With Cucumber Testing? 2011. Dostupné z WWW: <http://www.jackkinsella.ie/2011/09/26/why-bother-with-cucumber-testing.html>

(10) Krzywda, A.: Rails is not MVC. Září 2011. Dostupné z WWW: <http://andrzejonsoftware.blogspot.com/2011/09/rails-is-not-mvc.html>

(11) Lai, H.: A sneak preview of Phusion Passenger 3.2. 2012. Dostupné z WWW: <http://blog.phusion.nl/2012/04/13/a-sneak-preview-of-phusion-passenger-3-2/>

(12) Lai, H.: Ruby Enterprise Edition 1.8.7-2012.02 released; End of Life imminent. 2012. Dostupné z WWW: <http://blog.phusion.nl/2012/02/21/ruby-enterprise-edition-1-8-7-2012-02-released-end-of-life-imminent/>

(13) Lamontagne, F.: Ruby is dynamically AND strongly typed. 2007. Dostupné z WWW: <http://www.rubyfleebie.com/ruby-is-dynamically-and-strongly-typed/>

(14) Libbery, B.: "Skinny Controller; Fat Model" is misleading. 2011. Dostupné z WWW: <http://qualityonrails.com/archives/33>

(15) Matsumoto, Y.: How Emacs changed my life. Březen 2012. Dostupné z WWW: <http://www.slideshare.net/yukihiro_matz/how-emacs-changed-my-life>

(16) McAlpin, L.: Overriding Rails validations metaprogramatically. Květen 2011. Dostupné z WWW: <http://www.lmcalpin.com/post/5219540409/overriding-rails-validations-metaprogramatically>

(17) McCallister, B.: MVC, Model 2, Java WebApps, (and callcc, why not) . 2004. Dostupné z WWW: <http://kasparov.skife.org/blog/src/java/mvc.html>

(18) Mozilla: Mozilla Firefox: Development Process. 2011. Dostupné z WWW: <http://mozilla.github.com/process-releases/draft/development_overview/>

(19) Muhl, J.: How to enable Travis CI for an extension. Leden 2012. Dostupné z WWW: <http://radiantcms.org/blog/archives/2012/01/06/how-to-enable-travis-ci-for-an-extension/>

(20) Olsen, R.: *Eloquent Ruby*. Upper Saddle River NJ: Addison-Wesley, 2011, ISBN 9780321584106.

(21) Parsons, C.: Your tests are lying to you. 2011. Dostupné z WWW: <http://chrismdp.github.com/2011/10/your-tests-are-lying-to-you/>

(22) Přispěvatelé ChiliProject: Why Fork? 2011. Dostupné z WWW: <https://www.chiliproject.org/projects/chiliproject/wiki/Why%_Fork?version=11>

(23) Přispěvatelé Trac: RedMine. 2011. Dostupné z WWW: <http://trac.edgewall.org/wiki/RedMine?version=4>

(24) Přispěvatelé Wikipedie: Wiki. 2012. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Wiki&oldid=8421194>

(25) Rappin, N.: *Rails Test Prescriptions*. Raleigh N.C.: Pragmatic Bookshelf, 2011, ISBN 9781934356647, 350 s.

(26) Stewart, B.: An Interview with the Creator of Ruby. 2001. Dostupné z WWW: <http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html>

(27) why the lucky Stiff: why's (Poignant) Guide to Ruby. 2008. Dostupné z WWW: <http://mislav.uniqpath.com/poignant-guide/>

Příloha A

Slovník

Glossary

ACL Seznam uživatelů a jejich oprávnění pro přístup k určitému zdroji.

bug tracker Systém hlášení úkolů a chyb zpravidla, lze považovat za speciální případ issue tracker.

commit Sada změn přidaných do repozitáře kódu.

duck typing „Pokud to kváká jako kachna, budu tomu říkat kachna“ – není kontrolován typ objektu, postačí že ví, jak odpovědět na zaslan zprávy.

fixtures Metoda fixování testovacích dat, viz subsection 3.1.4.

fork Odštěpení od hlavní vývojové větve, často s úmyslem dalšího nezávislého vývoje.

gem V kontextu jazyka Ruby: standardní formát balíčků pro aplikace a knihovny v Ruby.

Git Distribuovaný systém správy kódu.

issue tracker Systém pro správu úkolů, problémů, požadavků i chyb, viz také bug tracker.

LDAP Adresářová služba.

Mercurial Distribuovaný systém správy kódu.

mikroblog Zjednodušený a minimalizovaný blog, například Twitter.

namespace Jmenný prostor.

rozšíření Plug-in, zásuvný modul – kód který rozšiřuje funkčnost aplikace bez potřeby její úpravy.

SaaS Software poskytovaný formou on-line služby.

SCM Souhrnné označení software pro správu kódu.

scope V kontextu Rails: makro které umožňuje definovat podmínky výběru z databáze, volání scopes lze řetězit.

submodul V kontextu systému Git: odkaz na externí repozitář z lokálního repozitáře.

Subversion Centralizovaný systém správy kódu.

wiki Systém správy obsahu [24].

workflow Průběh práce.

YAML Textový formát pro serializaci dat, zaměřený na uživatelskou přívětivost; viz `http://www.yaml.org/`.

zpětné volání Callback – reference na funkci předaná za účelem jejího dalšího volání; v Rails se zpětná volání používají k obsluze událostí.

Příloha B

Seznam použitých zkratek

Acronyms

ACL Access Control List.

AGPL Affero General Public License.

API Application Programming Interface.

CI Continuous Integration.

DSL Domain Specific Language.

FIT Fakulta informačních technologií.

GPL GNU General Public License.

HTTP Hypertext Transfer Protocol.

ICT Information and communication technologies.

LDAP Lightweight Directory Access Protocol.

MRI Matz's Ruby Interpreter.

ORM Object-Relational Mapping.

REE Ruby Enterprise Edition.

SaaS Software as a Service.

SCM Source Code Management.

SQL Structured Query Language.

TDD Test-Driven Development.

YAML YAML Ain't Markup Language.

Příloha C

Seznam systémů uvažovaných pro hodnocení

Trac http://trac.edgewall.org/

The Bug Genie http://www.thebuggenie.com/

Teambox http://www.teambox.com/

Launchpad https://dev.launchpad.net/

Open Atritum http://openatrium.com/

Redmine http://www.redmine.org/

ChiliProject https://www.chiliproject.org/

GitLab http://gitlabhq.com/

Collabtive http://collabtive.o-dyn.de/

mtrack http://mtrack.wezfurlong.org/

GLPI http://www.glpi-project.org/spip.php?lang=en

Request Tracker http://bestpractical.com/rt/

OTRS Help Desk http://www.otrs.com/en/products/otrs-help-desk/

Roundup http://www.roundup-tracker.org/

LibreSource http://dev.libresource.org/

PHProjekt http://www.phprojekt.com/

TeamLab http://www.teamlab.com/

Retrospectiva `https://github.com/dim/retrospectiva`

Plancake `http://www.plancake.com/`

Project HQ `http://projecthq.org/`

Streber PM `http://www.streber-pm.org/`

Project Pier `http://www.projectpier.org/`

Todoyu `http://www.todoyu.com/`

Příloha D

Obsah přiloženého média

U jednotlivých rozšíření je připojený textový soubor README.md s instrukcemi a popisem funkcí.

- `chiliproject-fit` ..... distribuce ChiliProject včetně rozšíření
- `plugins` ..... jednotlivá rozšíření
  - `engines` ..... opravená verze Engines
  - `chiliproject_add_ldap_users` ..... přidávání uživatelů z LDAP
  - `chiliproject_members_view` .. pomocné rozšíření pro sdílené pohledy
  - `chiliproject_private_issues` ..... soukromé úkoly
  - `chiliproject_private_wiki` ..... soukromé stránky na wiki
  - `chiliproject_project_group` ..... projektové skupiny
  - `chiliproject_project_roles` ..... projektové role
  - `chiliproject_test_plugin` ..... prototyp rozšíření pro testování
- `text` ..... text práce
  - `src` ..... zdrojové soubory práce pro LaTeX
  - `BP_Vlnas_Jan_2012.pdf` ..... text práce ve formátu PDF