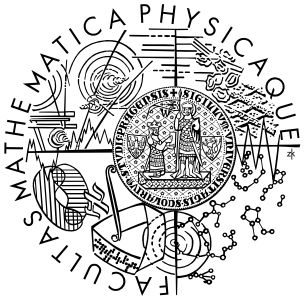
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Pitka

Software pro evidenci chyb v software

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Pavel Cejnar
Studijní program: Informatika, programování

2006

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 30. 5. 2006 Petr Pitka

Obsah

Název práce: Software pro evidenci chyb v software
Autor: Petr Pitka
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: Mgr. Pavel Cejnar
e-mail vedoucího: Pavel.Cejnar@mff.cuni.cz

Abstrakt: V předložené práci analyzujeme existující a navrhujeme nový software pro evidenci chyb v software. V první a druhé kapitole se věnujeme analýze existujících programů pro evidenci chyb, zjišťování jejich výhod a nevýhod. V první kapitole analyzujeme volně dostupný software, ve druhé pak komerční software. Ve třetí kapitole analyzujeme a navrhujeme architekturu vytvářeného systému. Ve čtvrté kapitole studujeme úzká místa vytvářeného systému. Popisujeme, proč jsme konkrétní místo implementovali daným způsobem a jak by se tato implementace dala v budoucnosti případně vylepšit.

Klíčová slova: evidence chyb, software, návrh

Title: Bug registration software
Author: Petr Pitka
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Pavel Cejnar
Supervisor's e-mail address: Pavel.Cejnar@mff.cuni.cz

Abstract: In the present work we analyze the existing bug registration software and we plan the new one. In the first and second chapter we test the existing bug registration applications and we probe their advantages and disadvantages. In the first chapter we analyze free accessible software and in the second chapter commercial software. In the third chapter we analyze and plan the architecture of the new system. In the fourth chapter we study narrow places in the system. We explain why we selected the concrete implementation of the narrow place and how this implementation can be improved in the future.

Keywords: bug registration, software, suggestion

Úvod

Cílem předkládané bakalářské práce bylo vytvořit software pro evidenci chyb, které vznikly v průběhu vývoje jiných aplikací a softwarových systémů. Produkt byl navrhován pro vnitrofiremní použití ve středních a velkých softwarových firmách majících desítky uživatelů a evidujících řádově tisíce chyb ve vyvíjených produktech.

Vytvořený software se skládá ze tří základních částí:

- Databázového serveru, který slouží k ukládání všech informací o vytvářených projektech a zadaných chybách.
- Webového rozhraní umožňujícího přístup k systému z libovolného počítače, který má přístup k vnitrofiremní síti a je vybaven webovým prohlížečem. Webové rozhraní umožňuje i přístup z lokací, které se nacházejí mimo firmu (například z domova), aniž by bylo nutno povolovat přístup přes speciální porty, stačí pouze povolený provoz protokolu http, což bývá většinou splněno.
- Speciálního klienta pro operační systémy Microsoft Windows, který umožňuje jednodušší a komfortnější přístup k zadávaným datům.

Obě vytvořená rozhraní pro přístup k datům (webové a klient pro Windows) jsou rovnocenná, tj. téměř všechny akce proveditelné v jednom z těchto rozhraní je možné provádět i v druhém. Podrobnosti o instalaci, ovládání rozhraní a implementaci je možné najít v dokumentaci, která se nachází na CD přiloženém k této práci.

Tato práce se skládá ze čtyř kapitol. První dvě kapitoly jsou věnovány analýze existujících softwarových produktů, které slouží k evidenci chyb. U popisovaných produktů byly vyhledány jejich přednosti a nedostatky. Přednosti jsem se snažil promítnout do vytvářeného systému, naopak nedostatkům jsem se snažil vyhnout. První z těchto kapitol popisuje volně dostupné a druhá komerční produkty.

Třetí kapitola je věnována analýze a návrhu architektury vytvářeného systému. V kapitole jsou shrnuty nejdůležitější důvody, proč uchovává zvolená data a proč je aplikace naimplementována navrhovaným způsobem.

Poslední kapitola rozebírá úzká místa vytvářeného systému. V krátkosti popisuje implementaci uzkých míst a porovnává ji s jinými možnostmi, které se před zahájením prací na projektu nabízely. V některých částech nabízí i možnosti optimalizace do případných budoucích verzí.

Kapitola 1

Analýza existujících volně dostupných produktů

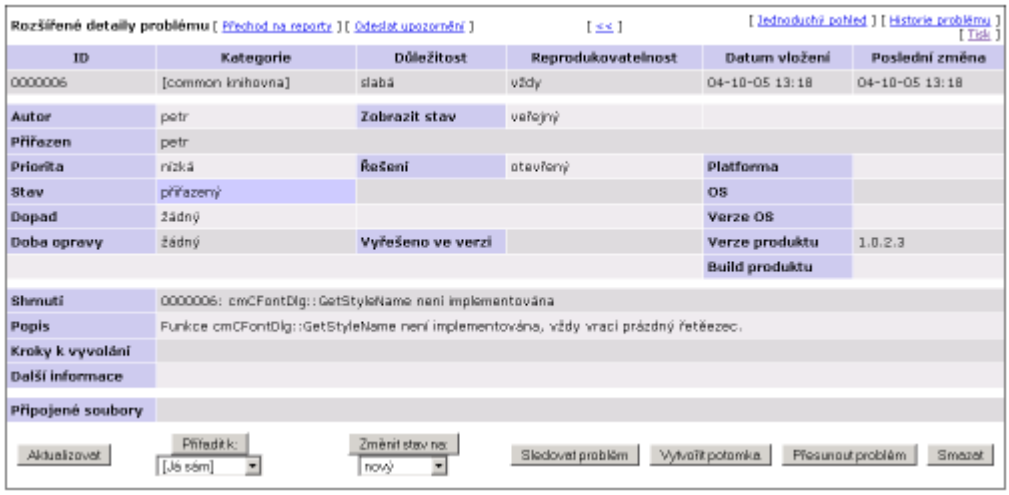
V této kapitole popisuji několik nejznámějších a nejrozšířenějších volně dostupných systémů pro evidenci chyb. Většina z produktů popisovaných v této kapitole je k dispozici pod licencí GNU a všechny jsou dostupné včetně zdrojových kódů. U všech popisovaných produktů se snažím vyzdvihnout jejich přednosti a nevýhody ve srovnání s ostatními produkty.

1.1 Mantis

Jedním z nejznámějších produktů pro evidenci chyb je **Mantis**. Je dostupný pouze jako webová aplikace vytvořená v PHP a přistupující do MySQL databáze. Projekt je dostupný na adrese *www.mantisbt.org* pod GNU licencí verze 2. Grafické prostředí projektu je jednoduché, ale i celkem hezké. Na jednotlivých stránkách se nachází poměrně velké množství údajů, což způsobuje mírnou nepřehlednost některých stránek. Systém umožňuje editovat informace o uživatelích systému, o vyvíjených projektech a nalezených chybách. K jednotlivým chybám je možné vytvářet poznámky, připojovat soubory, sledovat jejich historii a další.

Mezi hlavní výhody projektu Mantis patří následující:

- Jednoduché, hezké a docela intuitivní uživatelské rozhraní
- Dostupnost pod licencí GNU, široká podpora od existujících uživatelů a neustálé zlepšování systému



Obrázek 1.1: Část stránky zobrazující informace o problému (Mantis)

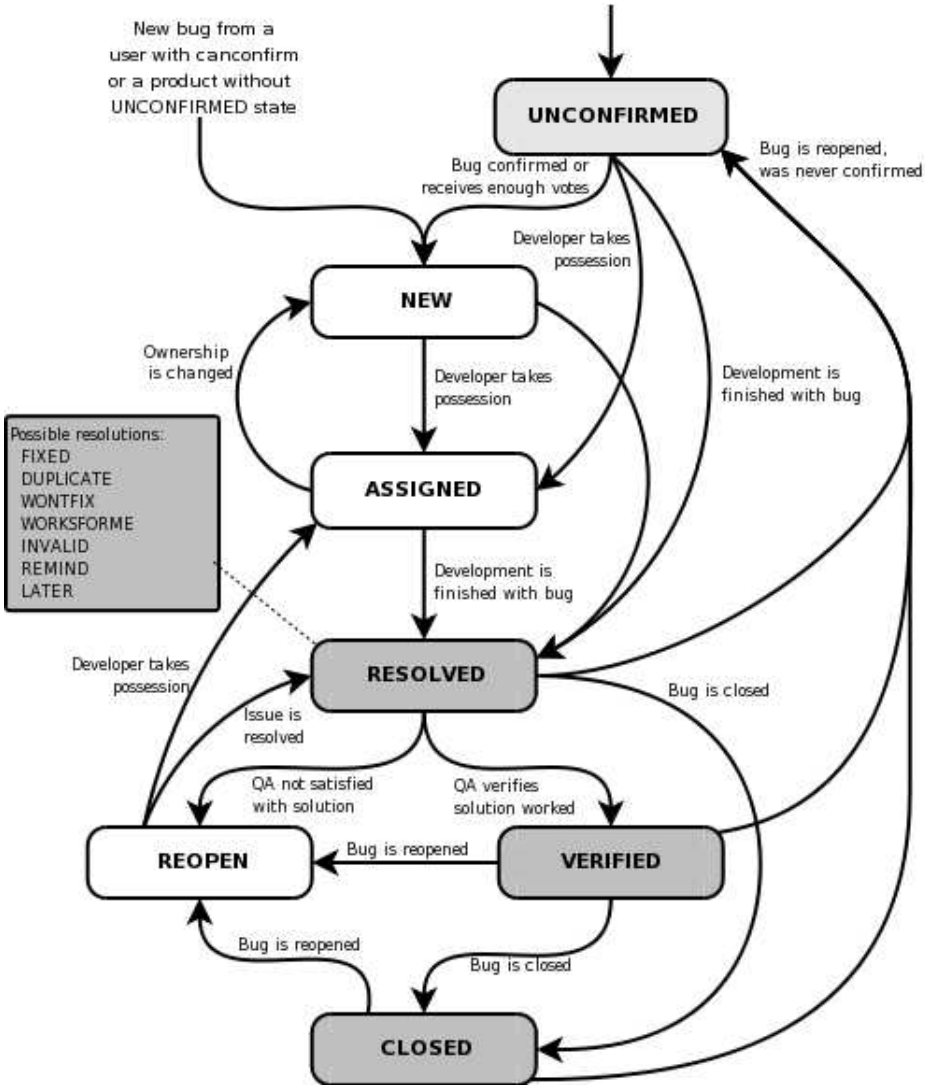
- Možnost definovat uživatelské atributy, které lze připojit k evidovaným problémům
- Možnost definovat barvy uživatelského rozhraní pomocí pomocných PHP skriptů

Nejdůležitější nevýhody projektu Mantis:

- Mnoho informací na jedné stránce, což vede k jejich nepřehlednosti a nutnosti používat posuvník
- Neexistence speciálních klientů pro různé operační systémy, kteří by umožňovali komfortnější a rychlejší práci se systémem
- Neexistence podpory pro skupiny, pouze omezená možnost přiřazovat uživatelům určité role

1.2 Bugzilla

Projekt **Bugzilla** je jedním z dalších velmi známých projektů založených na myšlence open-source softwaru. Podobně jako předchozí produkt obsahuje pouze webové rozhraní, které je tvořeno z CGI skriptů psaných ve skriptovacím jazyce Perl. Jako databázový backend nabízí na výběr mezi volně



Obrázek 1.2: Životní cyklus problému (Bugzilla)

dostupnými databázemi MySQL a PostgreSQL. Projekt je dostupný včetně podrobné dokumentace na adrese *www.bugzilla.org*.

V současné době se tento produkt používá pro evidenci chyb v celé řadě projektů, hlavně těch, které jsou založeny na myšlence open-source softwaru. Používá se např. v projektech Mozilla, GNOME, Apache, Open Office a mnoha dalších.

Projekt Bugzilla eviduje podobné informace o uživatelích, projektech a skupinách jako výše uvedený projekt Mantis. Životní cyklus chyby, který je podrobně popsán v dokumentaci tohoto systému, je také hodně podobný zmíněným produktům, pouze má méně stavů. Životní cyklus je velmi hezky vysvětlen obrázkem 1.2, který jsem převzal z uživatelské dokumentace projektu Bugzilla (viz [1]).

Základní výhody projektu Bugzilla spatřuji v následujících bodech:

- Velmi pokročilé vyhledávání pomocí tzv. *boolean charts*
- Spolupráce s externími nástroji jako je např. CVS
- Velmi silná podpora od existujících uživatelů

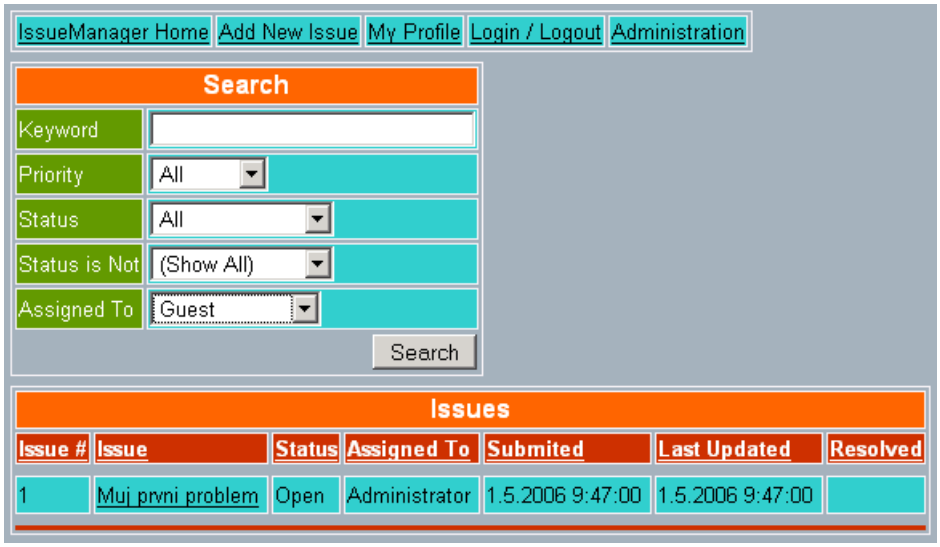
Naopak nevýhody projektu Bugzilla spatřuji v následujících bodech:

- Závislost na jazyku Perl, která komplikuje instalaci na některé operační systémy, jako je např. Microsoft Windows
- Neexistence speciálních klientů, kteří by umožňovali komfortnější práci se systémem
- Komplikovaná práce s přístupovými právy

1.3 Issue Manager

Produkt **Issue Manager** vytvořený společností Ultra Apps je dostupný na adrese *www.ultraapps.com*. Produkt je šířen pod interní licencí včetně zdrojových kódů. Produkt využívá technologie ASP.NET, tedy je ho možné provozovat pouze na serverech IIS společnosti Microsoft.

Uživatelské rozhraní je jednoduché a přehledné. Veškeré zobrazení se vejde do levého horního rohu a tak na poměrně běžném rozlišení monitoru (1024x768) zůstává velká část obrazovky nevyužita. Také volba barev mi



Obrázek 1.3: Seznam nalezených problémů (Issue Manager)

přijde trochu moc křiklavá (viz obrázek 1.3), ale vzhledem k dostupnosti zdrojových kódů patrně není problém barevné ladění aplikace upravit.

Jako úložiště dat aplikace využívá databázi MS Access, přičemž je možné aplikaci nastavit tak, aby používala databázový server MS SQL. K problémům je možné zadat (oproti předchozím testovaným produktům) podstatně méně informací. Aplikace žádným způsobem neumožňuje rozdělení chyb podle projektů nebo dokonce podle částí projektů. V administrativní části je možné editovat pouze uživatele. Aplikace nemá žádnou podporu pro skupiny uživatelů. Podpora přístupových práv spočívá v tom, že každý uživatel má přiřazenu roli, ve které vystupuje vzhledem k programu (administrátor, vývojář, manager).

Tento produkt za ostatními zaostává, ale na druhou stranu se velmi snadno používá a někomu může dostačovat (vzhledem k jeho ceně). Produktu bych vytknul následující věci:

- Neexistující podpora pro více projektů
- Neexistující podpora pro skupiny uživatelů a podrobnější přístupová práva
- Nevyužitost prostoru okna prohlížeče
- Závislost na platformě Microsoft Windows

- Neexistence speciálního klienta pro komfortnější práci s programem

Naopak, co bych u tohoto produktu mohl vyzdvihnout je konfigurovatelný obsah emailového upozorňování, bohužel upozorňovat lze pouze na obecné akce vložení nového problému a aktualizace existujícího problému.

1.4 ITracker

Produkt ITracker, který byl vytvořen Jasonem Carrollem, je dostupný na adrese *www.cowsultants.com*. Produkt je šířen pod licencí GNU, proto jsou dostupné i jeho zdrojové kódy. Software je vytvořen pomocí technologie Java Server Pages, díky které je víceméně nezávislý na platformě. Ke svému běhu potřebuje aplikační server JBoss a také databázový server. Na výše uvedené adrese je k dispozici také tzv. *express verze*, která v sobě obsahuje integrovaný aplikační i databázový server.

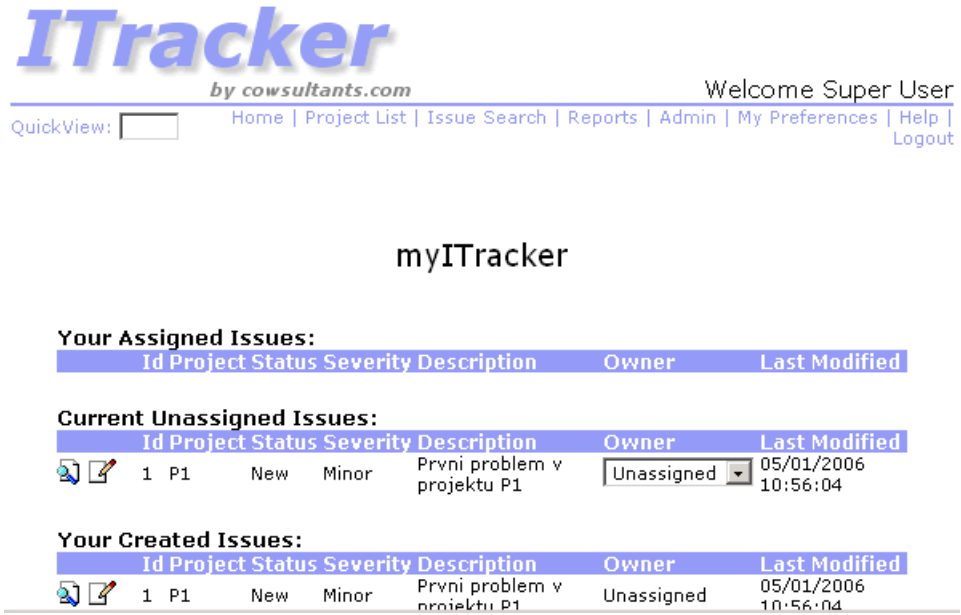
Vzhledem k použité technologii je přístup možný pouze přes webové rozhraní (viz obrázek 1.4). Podle webových stránek autora existuje aplikační rozhraní, které je možné zakomponovat do jiných programů a vytvořit tak speciálního klienta. Webové rozhraní je jednoduché a poměrně přehledné. Existuje ovšem několik maličkostí, které mi připadají matoucí. Například v seznamu projektů nemohu vytvořit nový projekt, musím jít do sekce administrace.

Jako úložiště dat je možné použít celou řadu relačních databázových systémů. V express instalaci je zahrnut HSql server, doporučovaný je MySQL server, ale aplikace obsahuje podporu i pro DB2, Firebird, MS SQL Server, Oracle, Postgre SQL a Sybase. Při změně databáze je nutno relativně složitě konfigurovat aplikační server JBoss.

Aplikace umožňuje zadávat pouze nejzákladnější údaje k jednotlivým chybám. Možnost přidávání poznámek buď zcela chybí nebo je skryta v historii, podle toho, jak se na to zrovna dívám. Podobně u vytvořených projektů není možné definovat kategorie, verze, přidávat soubory apod. K projektu je možné definovat uživatelské atributy, bohužel tato možnost již není pro jednotlivé problémy.

Mezi hlavní výhody programu bych zahrnul následující:

- Platformová nezávislost
- Možnost výběru z velkého množství databázových systémů



Obrázek 1.4: Úvodní stránka po přihlášení (ITracker)

- Existence aplikačního API pro integraci produktu do jiné aplikace
- Podpora pro export do různých formátů

Naopak mezi hlavní nevýhody bych zařadil následující:

- Chybějící podpora pro kategorie a verze projektu
- Chybějící možnost zadávání poznámek
- Náročnost konfigurace při instalaci (s výjimkou express edice)

Kapitola 2

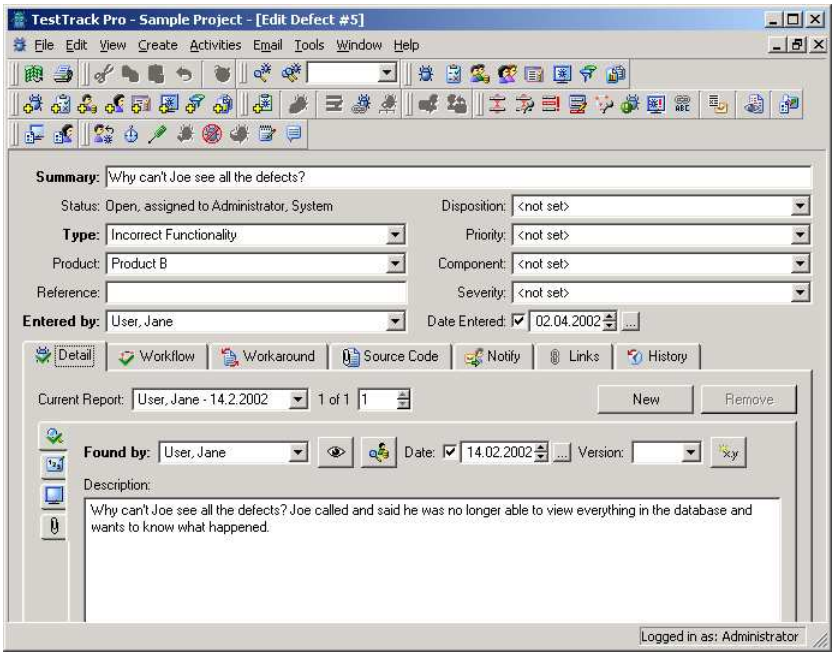
Analýza existujících komerčních produktů

V této kapitole popisuji několik nejznámějších a nejrozšířenějších komerčních systémů pro evidenci chyb. Většina z produktů popisovaných v této kapitole je dispozici pouze pro operační systémy Microsoft Windows, podpora pro ostatní operační systémy je poměrně malá. U všech popisovaných produktů se snažím vyzdvihnout jejich přednosti a nevýhody ve srovnání s ostatními produkty.

2.1 TestTrack

Produkt **TestTrack** pochází od společnosti Seapine Software a jeho demoverze je dostupná na adrese *www.seapine.com*. K dispozici je v různých edicích a pro různé operační systémy. Jeho cena se pohybuje v závislosti na edici a počtu uživatelů od 295$. Produkt je možné používat na různých operačních systémech – Microsoft Windows, MacOS X, Linux a Solaris. Verze pro Linux a Solaris je navíc k dispozici ve dvou variantách – založená na skriptech v jazyce Perl a založená na Javě. Program umožňuje i přístup přes webové rozhraní, bohužel není dostupná samostatná varianta pouze pro web. To je dáno tím, že používá jako CGI skripty zkompilované programy určené pro daný cílový operační systém. Jako úložiště dat je implicitně po instalaci využita interní databáze, ale produkt je možné nakonfigurovat i pro použití s relačními databázemi Oracle nebo SQL Server.

Z hlediska uchovávaných informací program eviduje podobné informace jako většina produktů uvedených v předchozí kapitole. Aplikace eviduje uži-



Obrázek 2.1: Editace problému (TestTrack)

vatele systému, rozděluje je do skupin, eviduje informace o projektech, jejich částech a verzích a přiřazuje jim chyby. K chybám je možné přidávat poznámky, soubory a další. Navíc umožňuje provázat chybu přímo se zdrojovým souborem a evidovat tzv. *work flow*, neboli průběh práce na chybě. V podstatě se jedná o zobecnění mechanismu přidávání poznámek, který je částečně provázán s historií.

Hlavní výhody tohoto produktu bych viděl v následujících bodech:

- Široká přizpůsobitelnost, od popisků sloupců, přes hodnoty (u výčtových položek) až po vytváření vlastních atributů
- Možnost vytvářet vlastní tiskové sestavy
- Podpora pro nejrozšířenější operační systémy
- Velké množství přístupových práv až na úroveň jednotlivých sloupců

Naopak hlavní nevýhody produktu bych viděl v následujících bodech:

- Nutnost instalace dvou serverů – vlastního aplikačního a licenčního, v případě použití relační databáze (což je asi častý případ) ještě databázového

- Nelze vytvářet a editovat projekty přímo v klientovi, je nutné spustit samostatného (tzv. administrativního) klienta
- Použití MDI rozhraní, které mi přijde (v tomto konkrétním případě) neintuitivní na ovládání aplikace
- Zanořenost některých údajů v několikanásobném systému záložek (viz např. obrázek 2.1)
- Při zmenšení okna se některé informace skryjí a není je možné zobrazit jinak než zvětšením okna

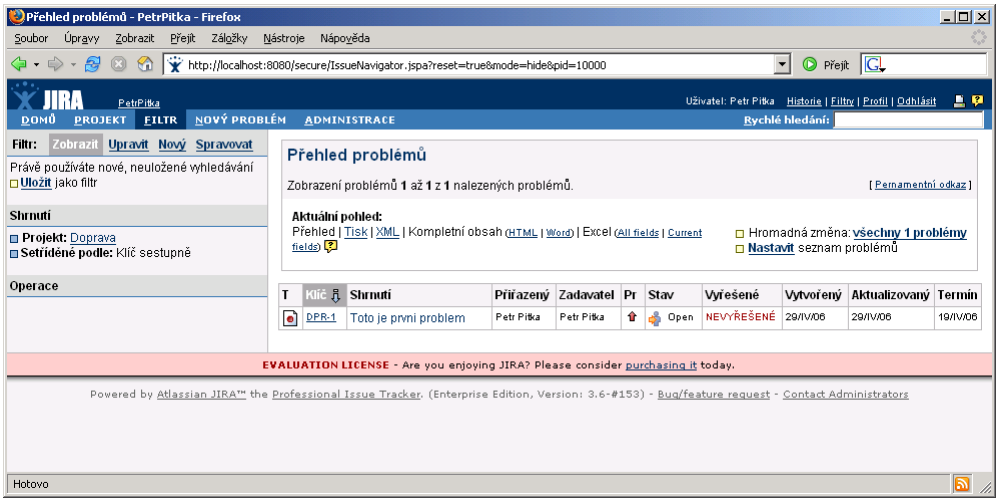
Na závěr popisu tohoto produktu vyzdvihnu ještě podporu tohoto produktu. Po stažení zkušební verze jste kontaktováni odpovědnou osobou, která s vámi komunikuje, diskutuje o problémech a na přání i spustí online prohlídku systému. Samozřejmě tohle vše je proto, aby přesvědčili zákazníka ke koupi jejich produktu, ale pokud mají stejný přístup i v případě technické podpory, pak bych technickou podporu určitě zařadil mezi klady tohoto produktu.

2.2 Jira Bug Tracking

Produkt **Jira Bug Tracking** od společnosti Atlassian je dostupný na adrese *www.atlassian.com*. Produkt je dodáván ve třech edicích (standard, professional a enterprise). Jeho cena začíná v závislosti na edici a počtu uživatelů na 600$. Produkt je vytvořen pomocí technologie Java Server Pages, díky které je nezávislý na platformě a operačním systému. K dispozici je i standalone varianta, která v sobě přímo integruje potřebný aplikační server pro zpracování použité technologie a databázový server. Jedinou součástí, kterou je nutno instalovat zvlášť je runtimové prostředí jazyka Java.

Vlastní webové rozhraní je jednoduché a rozdělené do dvou sloupců (viz obrázek 2.2). První (menší) sloupec obsahuje různé akce, které lze provádět, filtry apod. Druhý (větší) sloupec obsahuje vlastní data a případné operace s nimi. Rozhraní je na první pohled nepřehledné, ale je možno si na ně poměrně rychle zvyknout.

Zadávaná data jsou ukládána do databáze. Verze standalone má databázi integrovanou, ale je doporučováno používat externí relační databázi. Podle informací na webových stránkách produktu systém spolupracuje s celou řadou relačních databázových serverů, mimo jiné je možné využít i volně dostupné databázové stroje jako např. MySQL nebo PostgreSQL.



Obrázek 2.2: Seznam problémů (Jira Bug Tracking)

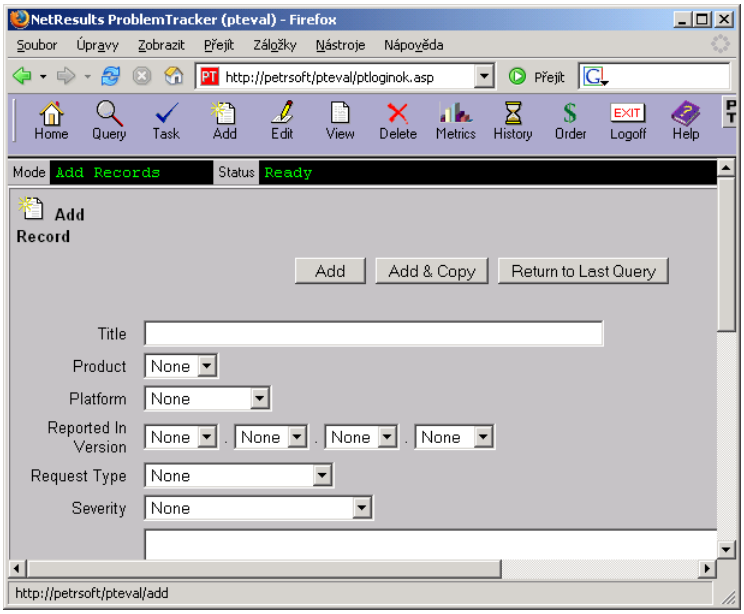
Z hlediska evidovaných údajů neobsahuje tento produkt některé méně významné atributy, což pro funkci programu vůbec nevadí. Naopak podporuje tzv. *work-flow*, neboli průběh práce na chybě, stejně jako předchozí produkt **TestTracker**.

Jako hlavní výhody produktu bych vyzdvihl následující:

- Podpora pluginů, např. pro integraci softwaru pro správu verzí (jako je CVS, ale i další)
- Import z jiných produktů (konkrétně Bugzilla, Mantis nebo i Excel)
- Nezávislost na platformě
- Možnost nakonfigurovat tak, že uživatelé si své účty vytvářejí sami, nebo účty uživatelům vytváří administrátor

Naopak jako hlavní nedostatky produktu bych označil následující:

- Nepřítomnost klienta, který by umožňoval efektivnější a komfortnější práci se softwarem
- Lokalizace do češtiny ve stažené verzi byla provedena pouze zčásti, proto výsledkem byl mix češtiny a angličtiny
- Nepříliš velké možnosti přizpůsobení aplikace



Obrázek 2.3: Vkládání nového problému (Problem Tracker)

- Použití levého sloupce, který ubírá místo pro zobrazení dat na obrazovkách s menším rozlišením

2.3 Problem Tracker

Produkt **Problem Tracker** pochází od společnosti NetResults Corporation a je dostupný na adrese *www.problemtracker.com*. Skládá se pouze z webového rozhraní, které je vytvořeno pomocí ASP skriptů. Cena tohoto produktu začíná (v závislosti na edici a počtu uživatelů) na 175$.

Webové rozhraní produktu je velmi jednoduché (viz obrázek 2.3). I přes tuto jednoduchost mi prostředí programu přišlo velmi neintuitivní, hlavně ve srovnání s ostatními testovanými produkty. Například se mi nijak nepodařilo zobrazit seznam všech chyb v databázi.

Zadávaná data program po nainstalování ukládá do databáze MS Access, ale je ho možno nakonfigurovat i pro použití s databázemi MS SQL Server a Oracle.

Program uchovává podobné údaje jako ostatní testované systémy. Pouze základem nejsou projekty, ale tzv. *workgroups*, tedy něco jako pracovní skupiny. Nevýhodou jejich implementace je fakt, že každá pracovní skupina má

jiné URL zadávané do prohlížeče. Navíc konfigurace pracovních skupin probíhá ve zvláštním programu. Celkově po přihlášení do pracovní skupiny můžu editovat pouze problémy a vlastnosti přihlášeného uživatele, jinak nic. Pro editaci uživatelů a pracovních skupin musím použít jinou aplikaci umístěnou na jiné URL adrese.

U tohoto programu jsem nenalezl žádné přesvědčivé výhody oproti ostatním produktům. Podle webových stránek program ve srovnání s jinými umožňuje *work-flow* a také *process-management*, ale mě osobně se nepovedlo tyto věci v produktu najít.

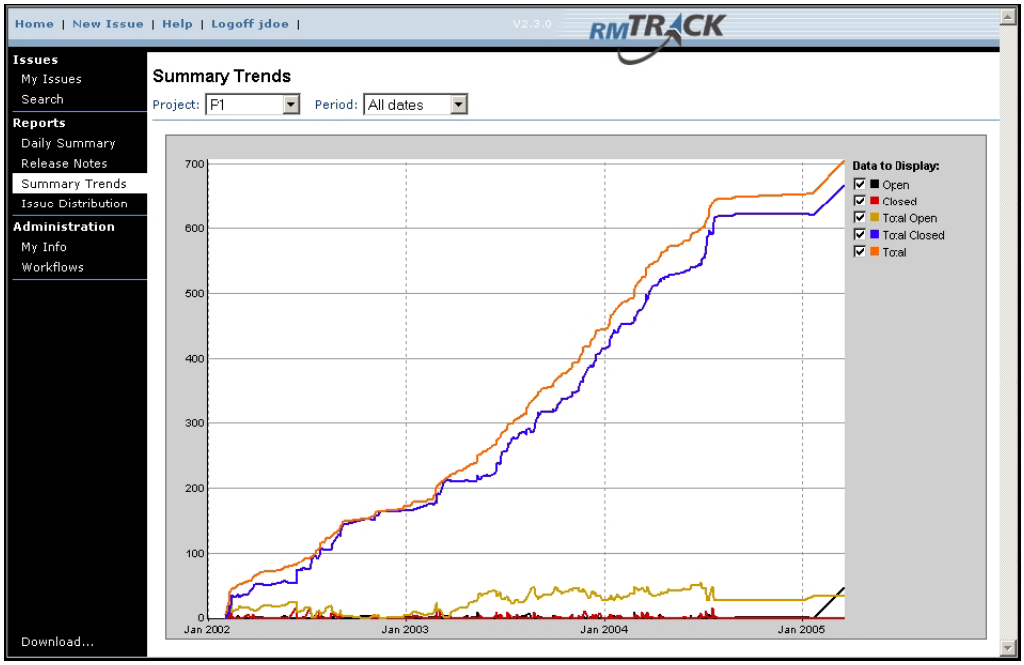
Naopak nevýhod jsem nalezl celou řadu:

- Neintuitivní uživatelské rozhraní
- Nutnost pro administraci spouštět speciální aplikace
- Webový server musí být IIS na Microsoft Windows
- Instalace požaduje vypnutí antivirového softwaru a restartování počítače (jako jediná u testovaných produktů)
- Verze je požadována ve tvaru čtyř číslic oddělených tečkami namísto libovolného řetězce tak, jak je to v ostatních programech

2.4 RMTrack

Produkt **RMTrack** od společnosti RMTrack Issue Tracking Solutions Inc. je k dispozici na adrese *www.rmtrack.com*. Tento produkt jako jediný z testovaných je k dispozici ve dvou variantách. První varianta je klasická, která se stáhne a lokálně nainstaluje. Druhá varianta využívá hostování na serverech společnosti RMTrack Issue Tracking Solutions Inc., kdy zákazník zaplatí poplatek za měsíční hostování a pak přes webové rozhraní využívá produkt. Výhodou tohoto postupu je fakt, že se zákazník nemusí starat o instalaci produktu, ani o jeho správu. Produkt je komerční, jeho cena se pohybuje od 149$ v lokální variantě a od 14,90$ měsíčně při hostování.

Produkt je založen na technologii ASP. Pro jeho provozování je proto vyžadován webový server IIS od společnosti Microsoft a tedy i operační systém Microsoft Windows. Produkt je primárně psán pro webový prohlížeč Internet Explorer, ale je možné použít i alternativní prohlížeče (Opera, Mozilla). V těchto prohlížečích budou některé funkce programu zčásti omezené.



Obrázek 2.4: Ukázka reportu ve formě grafu (RMTrack)

Konkrétně se jedná o prvky produktu založené na technologii ActiveX, jako je např. nástroj pro snímání obrazovky, grafové sestavy a grafický návrhář průběhu práce na chybě.

Pro ukládání zadaných dat aplikace využívá databázový stroj SQL Server (minimální verze 2000) společnosti Microsoft. Dále je požadován funkční SMTP server pro odesílání emailových upozornění. Ukládané informace jsou stejné jako u většiny výše zmíněných produktů.

Mezi nejdůležitější výhody tohoto produktu bych zahrnul následující:

- Široká konfigurovatelnost aplikace, lze např. vytvářet uživatelské vlastnosti problémů, přejmenovávat jednotlivé entity
- Pokročilé možnosti tvorby tiskových sestav v podobě textové i v podobě grafu (viz obrázek 2.4)
- Editovatelný průběh práce na chybě, to znamená, že je možné graficky stanovit pravidla, jak se bude s každou chybou zacházet od okamžiku jejího nahlášení až po okamžik jejího uzavření
- Možnost využít přeinstalovaného systému na serveru společnosti

Naopak nejdůležitější nevýhody spatřuji v následujících bodech:

- Velmi silná provázanost s produkty společnosti Microsoft
- Neexistence speciálního klienta, který by umožňoval komfortnější práci se systémem
- Nemožnost lokálního vyzkoušení softwaru bez nutnosti instalace placeného Microsoft SQL Serveru

2.5 Defect Manager

Produkt **Defect Manager** byl vytvořen společností TieraSoft a je dostupný na adrese *www.tierasoft.com*. Cena tohoto produktu začíná na 99$. Narozdíl od většiny ostatních testovaných produktů obsahuje jak webové rozhraní, tak speciálního klienta pro operační systém Microsoft Windows.

Uživatelské rozhraní obou prostředí je poměrně dobře přehledné. V případě klienta pro operační systém Windows jsou zadávaná data rozdělená do záložek. U webového rozhraní jsou všechna data vztahující se k problému uvedena na jedné stránce, nějaká emulace záložek by se v tomto případě určitě také vyplatila.

Zadávaná data jsou ukládána implicitně do databáze MS Access, ale produkt je možné nakonfigurovat i pro použití s relačními databázemi Oracle nebo MS SQL Server. Program umožňuje zadávání všech standardních údajů nutných pro evidování problému a souvisejících informací. Kromě těchto základních informací umožňuje provázání chyby se zdrojovým kódem a také k chybě přiřazovat systémové testy a klíčová slova, která jsou bohužel trochu nešťastně nazvána *Bookmarks*.

Tento produkt dále rozšiřuje mechanismus *work-flow* (průběh práce na chybě) o mechanismus *work-queue*, což je fronta problémů, které jsou v určité fázi a čekají na zpracování a posun do další fáze. Počet a typy front jsou plně konfigurovatelné, stejně jako vlastní průběh práce na chybě. Tento mechanismus je podrobně popsán v uživatelských manuálech programu (viz [2] od strany 13, [3] od strany 12).

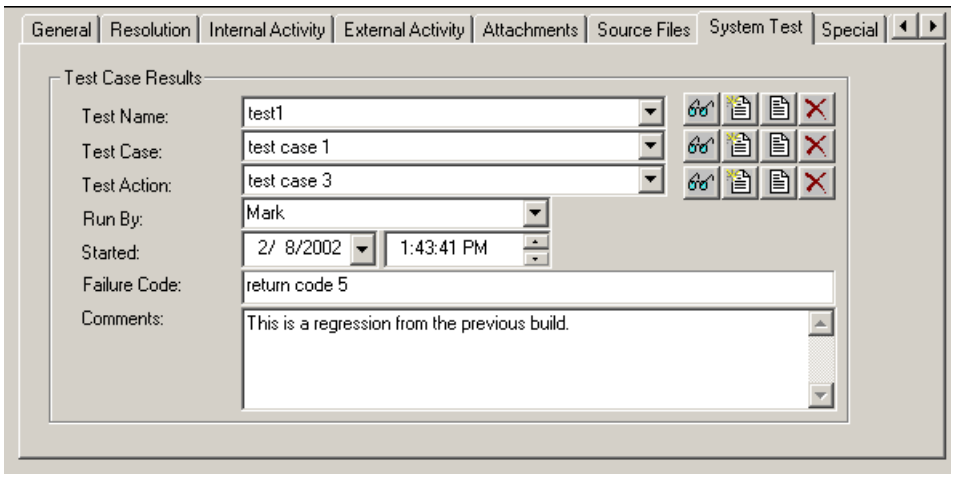
Tento produkt je vzhledem ke své ceně poměrně hodně pokročilý, proto uvedu pouze několik jeho nejdůležitějších výhod:

- Provázanost chyb se zdrojovým kódem, včetně integrace programu do vývojového prostředí Microsoft Visual Studio
- Možnost integrace se systémem pro správu verzí, např. s VSS
- Existence programátorského API umožňujícího vytvářet třetím stranám různé pluginy do tohoto produktu
- Možnost tvorby uživatelských vlastností problémů
- Možnost tvorby vlastních sestav
- Celá řada statistik (ve formě tabulek i grafů)
- Možnost vytvoření znalostní databáze pro technickou podporu

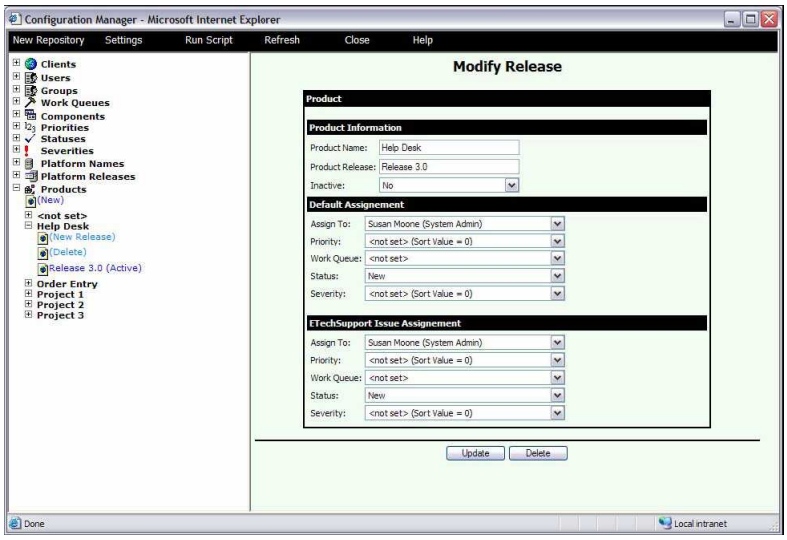
Naopak tomuto produktu se nedá téměř nic vytknout, přesto bych uvedl pár nevýhod produktu:

- Vázanost na prostředí MS Windows
- Program nabízí tolik možností, až to může začínající uživatele deprimovat, hodilo by se nějaké rozhraní pro méně pokročilé uživatele
- Oddělené aplikace pro administraci systému (opět ve formě webového i windows rozhraní), což ovšem vzhledem k počtu možností může být i výhodou

Obrázek 2.5 ukazuje část obrazovky pro editaci chyb s možnostmi pro zadávání informací o testech a obrázek 2.6 ukazuje hlavní okno webové aplikace pro administraci systému.



Obrázek 2.5: Vkládání informací o testování problému (Defect Manager)



Obrázek 2.6: Hlavní okno pro administraci systému (Defect Manager)

Kapitola 3

Architektura aplikace

V této kapitole navrhuji a analyzuji architekturu aplikace pro evidenci chyb v software. U jednotlivých částí architektury se snažím vysvětlit důvody, které mě vedly k popsanému návrhu.

3.1 Části aplikace

Aplikace by se měla skládat ze dvou základních částí. První částí je databázový server, který slouží k uchovávání zadávaných dat. Pro projekt byl zvolen server MySQL verze 4.1. Hlavním důvodem volby byla jeho rozšířenost a výborná spolupráce s různými webovými technologiemi. Oddělený databázový server navíc usnadňuje víceuživatelský přístup a zjednodušuje zálohování.

Druhou částí aplikace jsou rozhraní, která slouží pro zadávání dat a práci s nimi. Rozhodl jsem se pro dvě rozhraní. Prvním je webové rozhraní postavené na bezestavovém protokolu HTTP s využitím skriptovacího jazyka PHP. Webové rozhraní považuji u aplikací tohoto typu za samozřejmost, protože umožňuje práci i mimo pracoviště, například na služební cestě nebo z domova. Navíc přístup k webovému rozhraní je nezávislý na používaném operačním systému. Skriptovací jazyk PHP byl zvolen hlavně kvůli snadné integraci do většiny používaných webových serverů, jeho dostupnosti a rozšířenosti ve webovém prostředí.

Druhým rozhraním pro práci s daty je speciální klient pro operační systémy Microsoft Windows. Pro vytvoření tohoto klienta jsem rozhodl hlavně z toho důvodu, že webové rozhraní neumožňuje příliš efektivní práci s daty jako vlastní klientská aplikace. Z běžně používaných operačních systémů

jsem si vybral Microsoft Windows, protože mnoho vývojářů na tomto systému vyvíjí a nemají k dispozici podobný nástroj. Toto rozhraní by také mělo být jedinou součástí aplikace, která je přímo závislá na konkrétním operačním systému. Vytvoření klientů pro jiné operační systémy je možné, ale vzhledem k omezenému času nebude prozatím realizováno.

3.2 Informace v databázi

Základním úkolem celé aplikace je uchovávat data o nalezených chybách ve vytvářených projektech. Vzhledem k tomu, že většina softwarových společností vyvíjí souběžně více než jednu aplikaci, rozhodl jsem se, že vytvářený produkt musí mít možnost definovat projekty. Větší projekty se většinou rozdělují na menší části (např. vrstvy), proto je potřeba mít možnost ke každému projektu přidat seznam těchto částí. Vzhledem k tomu, že většina projektů je vyvíjena postupně, vznikají různé verze dané aplikace a tedy je potřeba pro každý projekt evidovat seznam existujících verzí. K vytvářené aplikaci vzniká celá řada dokumentů (specifikace, poznámky, dokumentace apod.), proto jsem do evidovaných informací zahrnul i soubory, které lze připojit k projektu. Ne všichni vývojáři pracující ve společnosti mohou mít přístup ke všem projektům, proto jsem se rozhodl poskytnout možnost přidělení přístupových práv uživatelům a skupinám uživatelů na úrovni jednotlivých projektů.

Informace o nalezených chybách jsou nejdůležitějšími údaji v celé databázi. Každá chyba patří k právě jednomu konkrétnímu projektu. Předpokládám, že situace, kdy jedna chyba bude patřit do více projektů, bude poměrně vzácná. Pokud ale nastane, je možné ji řešit vytvořením kopie dané chyby v jiném projektu.

Mezi základní informace o chybě jsem zahrnul různé druhy informací, které umožní chyby rozlišit na základě jejich důležitosti, priority, stáří, dopadu na projekt apod. Z těchto informací jsou nejdůležitější dvě – stav chyby a stav řešení. Různé stavy chyby odpovídají průběhu prací na opravě chyby. Naopak stav řešení říká, zda je nebo není chyba vyřešena a případně, že se nejedná o chybu nebo že se daná chyba nedá vyřešit. Postihuje i případné znovuotevřené chyby.

Vzhledem k tomu, že chyba prochází různými stavy, mění se její vlastnosti atd., rozhodl jsem se evidovat historii chyb. V této historii by se měly uchovávat nejenom změny stavů, ale i změny dalších údajů jako je priorita, důležitost, přiřazený uživatel apod. Tyto informace jednak upřesňují pohled

na vývoj chyby, jednak mi umožní implementovat další části aplikace, jako jsou statistiky nebo upozorňování na změny.

Nalezená chyba bývá často doprovázena nějakým chybovým hlášením na obrazovce, informace o ní je možné najít v logu apod., proto jsem umožnil k jednotlivým chybám připojovat různé soubory, jako například screenshoty. Podobně jako u souborů připojených k projektu je obsah souborů uchováván v databázi, proto je snadno dostupný ze všech klientů.

Tester, který zadá chybu do databáze, často nezná všechny potřebné informace k odhadu příčin nalezené chyby. Proto je možné přidávat k chybě dodatečně různé poznámky. Poznámky jsem se rozhodl implementovat hlavně proto, že přímo u chyby umožňují evidovat diskuzi nad příčinami a řešením chyby. To je obzvlášť užitečné, pokud více vývojářů diskutuje o chybě na větší vzdálenost.

Chyby většinou nebývají osamocené, ale často nějakým způsobem souvisí jedna s druhou. Proto jsem se rozhodl tuto závislost postihnout, aby programátor pracující na opravě jedné chyby věděl, že když jeho kolega opraví jinou chybu, tak bude moci snadněji opravit tu svoji.

Nejdůležitějšími testery bývají koncoví uživatelé. Ti také často přicházejí s různými náměty, jak konkrétně aplikaci vylepšit. Navrhovaný produkt by měl proto umožňovat evidenci těchto informací. Rozhodl jsem se, že pro tyto informace nebudu zakládat vlastní objekty, ale že náměty budu evidovat jako chyby. Pro odlišení od chyb jsem zavedl speciální kategorii závažnosti a to *návrh*.

Aplikace musí umožňovat přístup mnoha různých druhů osob. Jednak jsou to testeři, kteří nalezené chyby zadávají, pak vývojáři, kteří chyby opravují. Do databáze mohou přistupovat recenzenti, kteří zjišťují, jaké změny byly v aplikaci provedeny. Z tohoto důvodu je nutné, aby se každý uživatel identifikoval přístupovým jménem a heslem. Správu uživatelů a tedy i přístupových práv jsem se rozhodl dělat ve vlastní režii a ne na úrovni databáze. To mi umožnilo větší kontrolu nad přístupovými právy a možnost implementovat jednoduše nastavení uživatelů.

Protože aplikace musí být schopna pracovat s desítkami uživatelů, tak pro snadnější práci s uživatelskými účty jsem se rozhodl zavést mechanismus skupin uživatelů. Díky tomu není nutné měnit práva přímo jednotlivým uživatelům, ale pravomoci uživatelů je možné řídit zařazením do příslušné skupiny. Podrobnosti o přístupových právech jsou v následující podkapitole.

3.3 Přístupová práva

Jak již bylo zmíněno v předchozí podkapitole, aplikace by měla umožňovat nastavování práv na několika úrovních. Každý uživatel a každá skupina by měla mít svoji masku přístupových práv. Dále by mělo být možné přiřadit přístupová práva uživateli nebo skupině vzhledem ke konkrétnímu projektu. Tento přístup jsem zvolil kvůli možnosti nadefinovat skupiny vývojářů, které pracují na konkrétních projektech a nemají přístup k ostatním projektům.

Přístupová práva by měla být v zásadě dvojího druhu. Prvním druhem jsou přístupová práva týkající se celé aplikace. Mezi ně patří např. zobrazení a správa seznamu uživatelů, skupin a projektů. Druhá skupina se týká práce s chybami v konkrétním projektu, jako např. přidávání, editace a odstraňování chyb, zobrazování historie chyb apod. Počet a význam přístupových práv jsem se snažil zvolit tak, aby jich jednak nebylo moc (což by vedlo k nepřehlednosti při jejich nastavování) a aby jich nebylo ani málo (což by vedlo ke špatnému oddělení různých skupin uživatelů).

Výsledná přístupová práva jsou logickým součtem přístupových práv uživatele, všech skupin, do kterých uživatel patří a všech přiřazení k aktuálnímu projektu. To znamená, že zařazení uživatele do skupiny rozšíří přístupová práva uživatele o práva skupiny, ale žádná práva mu neodebere. Tento systém je pro uživatele snadno pochopitelný a také neklade vysoké časové nároky na implementaci.

3.4 Upozorňování na změny

Aplikace by měla být schopna evidovat stovky až tisíce chyb, proto by měla podporovat určité formy upozorňování na konkrétní změny v databázi. Díky tomu nemusí uživatel ručně prohlížet seznam chyb a zjišťovat, co se změnilo od jeho posledního přihlášení a co ne.

Prvním druhem upozorňování by mělo být sledování konkrétní chyby. Tento typ sledování je nejvíce omezen, upozorní pouze na změnu stavu vybrané konkrétní chyby.

Druhým typem by mělo být upozorňování na obecnější změny, jako je například vložení chyby do databáze, přidání poznámky k chybě, změna stavu libovolného problému. Jako základní mechanismus upozorňování jsem v tomto případě zvolil upozornění emailem. Pro toto řešení jsem se rozhodl proto, že hodně uživatelů pravidelně kontroluje svoji emailovou schránku, proto si upozornění s velkou pravděpodobností všimnou.

Jako alternativu k emailovému upozorňování jsem zvolil upozorňování přímo v rámci spuštěné aplikace. To má tu výhodu, že není závislé na emailovém systému. Tento typ upozorňování sice vyžaduje explicitní akci od uživatele k zobrazení změn, ale na druhou stranu by měl být přehlednější než procházení jednotlivých emailů. Navíc tento typ byl měl být schopen zobrazit i upozornění na blížící se termín dokončení opravy chyby, které by se pomocí emailu obtížně implementovalo.

Vzhledem k různým požadavkům uživatelů by upozorňování na změny mělo být konfigurovatelné. Mělo by být možné si zvolit, na co a jak si uživatel přeje být upozorněn. Rozhodl jsem se, že konfigurace bude samostatná pro jednotlivé projekty, ke kterým má uživatel přístup.

3.5 Webové rozhraní

Webové rozhraní by mělo být tvořeno HTML stránkami interpretovanými webovým prohlížečem a generovanými skripty v jazyce PHP. Důvody pro volbu tohoto jazyka jsou uvedeny na začátku této kapitoly. Pro zachování jednotného vzhledu stránek jsem se rozhodl vzhled řídit pomocí kaskádových stylů umístěných v samostatných souborech. Pro zmenšení zátěže jsem se rozhodl všechny formuláře pro zadávání dat ošetřit na straně klienta pomocí javascriptu. Protože javascript nemusí být na všech prohlížečích k dispozici nebo může být vypnut, je potřeba řešit všechny kontroly údajů i duplicitně na straně serveru.

Vzhledem k velkému počtu údajů jsem se rozhodl, že zobrazení seznamu chyb bude stránkované. Na jedné stránce bude zobrazen pouze uživatelem omezený počet položek. Tento počet položek si uživatel může nastavit. Aby se uživatel nemusel probírat velkým množstvím chyb, měl by mít k dispozici jednak mechanismus filtrování podle nejdůležitějších informací (stav, priorita, důležitost, ...) a také možnost hledání v textech chyb.

Při editaci by měly být informace rozděleny do několika nezávislých stránek. Hlavním důvodem pro toto rozdělení je snaha udržet stránky co nejvíce přehledné. Vedlejším efektem je i snazší implementace přístupových práv, kdy na nepřístupnou stránku se uživateli nedovolí vstoupit.

Pro zjednodušení některých často prováděných operací (změna stavu, přiřazení uživatele, ...) by mělo webové rozhraní nabízet možnost spuštění dávkové akce nad vybranými chybami.

3.6 Klient pro Windows

Hlavní důraz při navrhování obou rozhraní je kladen na to, aby obě rozhraní byla rovnocenná. Díky tomu si uživatel může vybrat, které rozhraní mu více vyhovuje. Proto pro klienta platí většina informací uvedených v předchozí podkapitole.

Klient by měl být vytvořen tak, aby ve srovnání s webovým rozhraním umožňoval rychlejší a pohodlnější práci. To by měly umožnit hlavně klávesové zkratky. Všem příkazům, které klient bude podporovat, by mělo být možné přiřadit téměř libovolné klávesové zkratky.

Klient je primárně určen pro jednoho uživatele, ale může být používán i více osobami na jednom stroji. Jednotliví uživatelé mohou mít i vlastní nastavení, za předpokladu, že se do operačního systému přihlašují pod jinými uživatelskými jmény.

3.7 Škálovatelnost

Aplikace je navrhována pro použití desítkami uživatelů a měla by být schopna obsluhovat řádově tisíce chyb. Proto je potřeba se vypořádat se dvěma problémy – výkonností systému a konzistencí databáze. Výkonnosti je věnována celá následující kapitola, proto se v této podkapitole budu věnovat pouze konzistenci databáze.

V databázi by neměly být uloženy informace, které odkazují na již neexistující záznamy. Např. k chybě neexistujícího projektu se nikdo nedostane, chyba objevená v neexistující verzi mate uživatele apod. Abych se těmto chybám vyhnul, využívám speciální úložiště dat *InnoDb*, které podporuje referenční integrity. Díky tomu se odstraňování údajů stává bezpečným. Navíc většina aktualizačních dotazů se skládá pouze z jednoho příkazu, proto databázový stroj zajistí atomicitu vykonání této operace.

Druhým (méně vážným) problémem je možnost souběžné aktualizace jedné položky, což může vést k tomu, že změny provedené jedním uživatelem jsou přepsány změnami, které provedl druhý uživatel. Tento stav nemusí být vždy chybový, ale rozhodl jsem se ho detekovat. Pokud tento stav nastane, jeho řešení nechám na uživateli. Pro řešení tohoto problému musím zjistit z databáze, zda položka byla od jejího načtení do aplikace někým modifikována. Pokud nebyla, je možné ji aktualizovat. Obě operace musí být provedeny atomicky, proto je zde nutné explicitně použít transakcí (a v důsledku toho zamykání na úrovni řádků). Pokud uživatel tento konflikt vy-

řeší, musím znovu detekovat, zda nedojde k dalšímu konfliktu. Tato situace se může teoreticky mnohokrát opakovat. Ale vzhledem k tomu, že editace chyb neprobíhá tak často a většinou ji provádí jeden konkrétní uživatel, tak by ke konfliktům mělo docházet pouze výjimečně.

Kapitola 4

Analýza výkonnosti systému

V této kapitole se budu věnovat analýze úzkých míst výkonnosti v systému. Zdůvodním, proč jsem zvolil daná konkrétní řešení, proč jsem to neřešil jinak a případně načrtnu, co by se do budoucna dalo ještě vylepšit.

4.1 Databáze

V databázové části aplikace není příliš mnoho možností, jak zvyšovat výkon aplikace jinak než výměnnou hardwaru. Jedním ze základních obecných principů je optimalizace SQL dotazů. Vzhledem k tomu, že obě dostupná rozhraní aplikace generují dynamicky velmi jednoduché dotazy, tak na nich není příliš co optimalizovat. Filtrování probíhá většinou přes identifikační číslo záznamu, což je vždy primární klíč, indexy na těchto sloupcích jsou tedy implicitně definovány. Doplnil jsem pouze indexy na sloupce, podle kterých se častěji hledá (např. uživatelské jméno, název projektu). Aplikace obsahuje několik složitějších (pevně zapsaných) SQL dotazů, ale ty se vyskytují v částech, kde velmi rychlý přístup k datům není nezbytný, např. při generování statistik nebo protokolu o opravených chybách.

Nevýhodou použitého databázového stroje MySQL ve verzi 4.1 je neexistence podpory pro triggery a uložené procedury. Vyplatil by se přechod na verzi 5, ale ta bohužel není na většině volně dostupných serverech přístupná. Při použití triggerů a uložených procedur by se zjednodušila (a tedy i zrychlila) obě rozhraní. Kód přesunutý do databáze (např. logování historie) by tak byl napsán pouze jednou a nebylo by ho nutno ladit v každém rozhraní zvlášť. Navíc přesunutí části kódu na úroveň databáze (např. výše zmíněné logování do historie) by také zrychlilo běh aplikace, neboť by ubylo SQL

příkazů posílaných ke zpracování a navíc zpracování uložené procedury je rychlejší než zpracování sekvence SQL příkazů.

4.2 Komunikace PHP s databází

Jedním z úzkých míst webového rozhraní je získávání dat z databáze. PHP si musí zobrazovaná data vyžádat z databáze, což stojí jednak čas procesoru pro získání sady výsledků, jednak se prostředky spotřebovávají při komunikaci PHP s databází (ať je webový a databázový server umístěný na jednom stroji nebo na více). Proto je potřeba, aby část získaných dat zůstala uchována na webovém serveru, aby se nemuselo tolik přistupovat do databáze. Na jednu stránku by měl připadat v průměru jeden SQL příkaz pro získání dat nebo jejich aktualizaci. Různé pomocné informace by měly být již přednačteny. U některých stránek jsem se nevyhnul použití více SQL dotazů (např. při generování statistik). Na základě předpokladu o menší používanosti těchto stránek jsem se rozhodl, že tyto stránky nebudu zatím optimalizovat.

Na druhou stranu uchovávání položek na webovém serveru například v session proměnných vyžaduje poměrně velké množství paměti, resp. diskového prostoru. Výchozí implementace session proměnných v PHP používá diskových souborů. Pokud pro jednoho uživatele takto uchováváme 100KB dat, pak pro tisíc uživatelů (současně přihlášených) potřebujeme přibližně 100MB diskového prostoru (nejčastěji někde v odkládací oblasti disku).

4.3 Session proměnné a překlady informací

Mnou vytvořená implementace uchovává na webovém serveru kromě nezbytných informací (jako např. informace o přihlášeném uživateli, informace o aktuálním projektu, aktuální filtr, aktuální řazení,...) pouze informace určené pro překlad dat. Jedná se nejčastěji o dvojice údajů číslo a odpovídající text. Příkladem může být překlad kategorie, kdy číslo kategorie (které je obsaženo v záznamu o problému) se přeloží na název kategorie (který je v samostatné tabulce a je zobrazen uživateli). Mezi další podobné údaje patří číslo verze (ať již pro objevení nebo pro vyřešení chyby), jméno uživatele, který chybu nahlásil, jméno uživatele, kterému je problém přiřazen atd. Tyto překládací údaje (tj. dvojice číslo a název) jsou uchovávány v objektu typu `Dataset`. Naplněné datasety jsou ukládány do jednotlivých session pro-

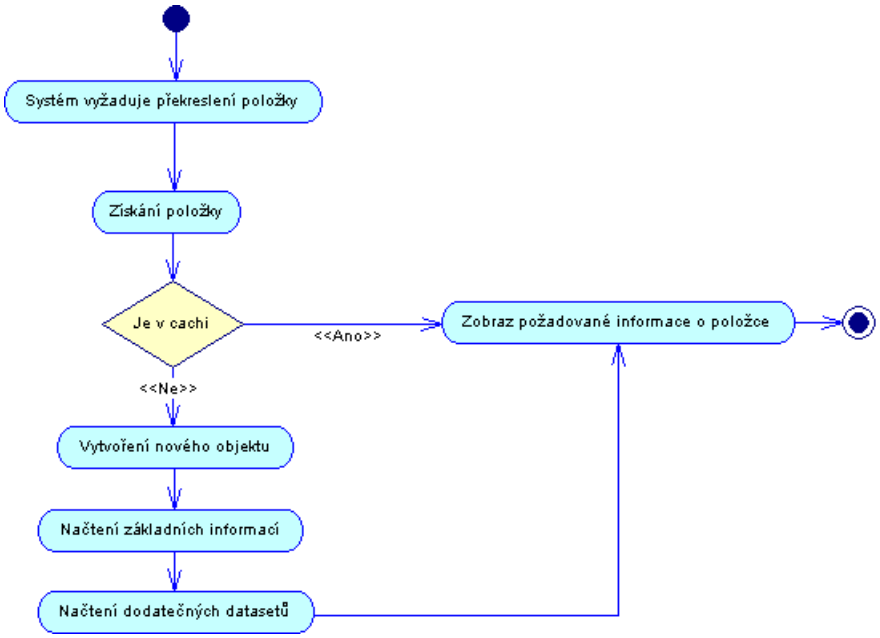
měnných a informace z nich jsou používány při zobrazování dat uživateli. Kvůli zajištění aktuálnosti zobrazovaných dat jsem se rozhodl, že jednou za pevně daný počet přístupů k datasetu znovu načtu data z databáze.

Jinou možností, jak překlady řešit, je využití spojení tabulek již na úrovni databáze. Tedy například při zobrazení seznamu chyb včetně názvu kategorie bych spojil tabulku chyb s tabulkou kategorií (pravým spojením). Pokud bych chtěl zobrazit i jména uživatelů nebo identifikace verzí, musel bych připojit další tabulky. Tento přístup jsem zamítl z několika důvodů. Prvním důvodem byl komplikovaný kód pro vygenerování SQL dotazu a zobrazení výsledné sady záznamů. Druhým důvodem je fakt, že spojování většího množství tabulek je náročnější operace než pouhé vyfiltrování záznamů z jedné tabulky. Třetím a nejdůležitějším důvodem pro zvolený přístup je ten fakt, že mechanismus překladů je univerzálnější a lze ho využít i tam, kde SQL dotazy nepotřebujeme nebo nemůžeme použít. Jedním příkladem je vyplňování editačních formulářů. Například pokud při zadávání dat chceme využít roletku, pak pro vygenerování jejího obsahu nemusíme data získávat z databáze, ale můžeme využít přednačteného datasetu. Druhým příkladem může být zobrazování historie, kde překládáme hodnoty sloupců, jejichž název je uveden v jiném sloupci. Takový SQL dotaz, který by umožnil získaná data ihned přívětivě zobrazit, nejde sestavit.

Dalším vylepšením by mohlo být ukládání celých záznamů o problému do session proměnné. To by vyžadovalo naprogramování nějaké cache, která by omezila počet takto ukládaných záznamů na nějakou rozumnou hodnotu. Osobně jsem se rozhodl položky nekešovat. Už stávající soubor pro session proměnné má velikost přes 50KB a podle výše provedené úvahy pro větší počet současně přihlášených uživatelů tak aplikace vyžaduje poměrně mnoho místa. Navíc by v podstatě došlo k nahrazení zisku dat z databáze ziskem dat ze souboru na lokálním disku. Takže výhoda získaná takovýmto typem kešování by toho tolik nepřinesla. Navíc by bylo nutné řešit problémy s občerstvováním nakešovaných dat a pokud by cache nebyla určena pouze pro čtení, pak by bylo nutné řešit i problémy spojené se zpožděným ukládáním dat do databáze a vznikem případných konfliktů.

4.4 Načítání záznamů do klienta

Tato podkapitola se věnuje získávání záznamů chyb, projektů, uživatelů a skupin. Při načítání jednoho záznamu se načítají i související informace. Například při načítání projektu se zároveň načítá i seznam kategorií a verzí



Obrázek 4.1: Obecný mechanismus získání jedné položky pro zobrazení

definovaných v daném projektu, seznam souborů připojených k projektu a seznam uživatelů přiřazených k projektu. Načtení jedné položky tedy může vyžadovat několik SQL dotazů a patří proto mezi pomalé akce. Při testování se průměrný čas načtení jedné položky pohyboval okolo 10 ms. Zde se otevírá prostor pro budoucí optimalizaci, kdy by se nějakým mechanizmem zajistilo, že seznam verzí apod. se načte až v okamžiku, kdy bude doopravdy k něčemu potřeba. Tento mechanismus jsem do aktuální verze neimplementoval, neboť jeho příspěvek k rychlosti není až tak zásadní. Celkový průběh načítání jedné položky je znázorněn na obrázku 4.1.

Při zobrazování dat, potřebuji znát identifikátory položek patřících na jednotlivé řádky. Proto z databáze načítám seznam identifikátorů položek k zobrazení (na základě aktuálního filtru a třídění). Načtení tohoto seznamu vyžaduje pouze jeden SQL dotaz. Vlastní záznamy jsou z databáze načítány až v okamžiku, kdy jsou skutečně potřeba (například pro zobrazení na obrazovce nebo pro editaci).

Výhodou tohoto přístupu oproti načítání všech položek je fakt, že se načtou pouze ty položky, které jsou doopravdy potřeba. To znamená, že po spuštění aplikace se načte pouze prvních přibližně 30 položek, které jsou

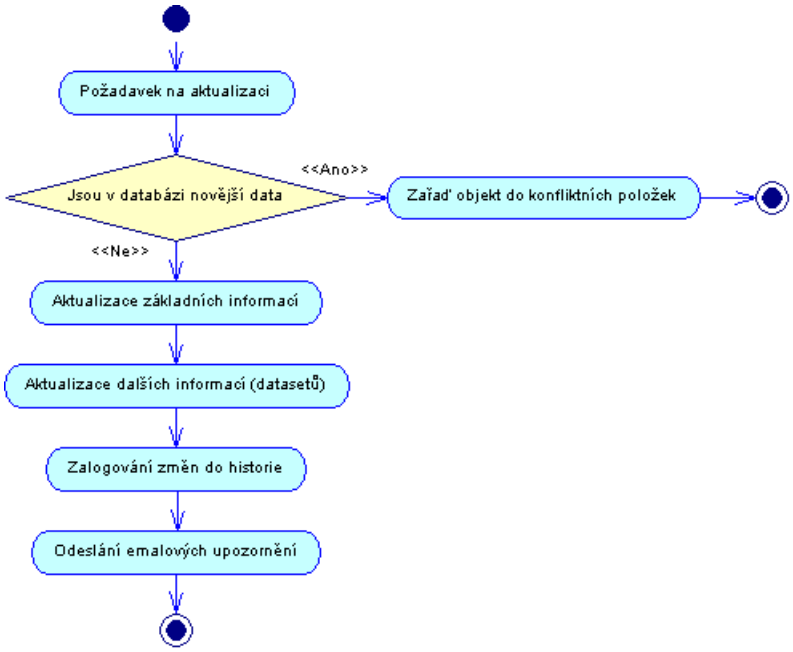
zobrazeny na obrazovce, ostatní, ke kterým se dostaneme až posuvníkem, prozatím načteny nejsou (a pokud je uživatel nezobrazí, tak ani nikdy nebudou). Naopak nevýhodou současné implementace je postupné zobrazování jednotlivých položek, což působí mírně rušivě. Tento problém je možné vyřešit načítáním více položek (např. deseti) najednou. Tento postup jsem zatím odložil, neboť pro jeho snadnou implementaci bych potřeboval alespoň částečnou implementaci odloženého načítání jednotlivých částí položek (viz výše). Navíc prováděné kešování položek (viz následující podkapitola) redukuje tento problém pouze na první načtení zobrazených záznamů. Načtení všech maximálně čtyřiceti položek (při rozlišení 1024x768 a maximalizovaném okně) se při testování pohybovalo pod jednu sekundu.

4.5 Kešování položek a občerstvování dat

Všechny základní položky načtené z databáze (problémy, projekty, uživatelé a skupiny) jsou jako celek kešovány. Po prvním načtení jsou tedy uloženy do cache a při dalším zobrazení (např. při překreslování okna nebo při požadavku na editaci) jsou získány z cache a nemusí se provádět dotazování databáze. Velikost cache je nastavena řádově na desítky pro projekty, uživatele a skupiny a na stovky pro problémy. To znamená, že při předpokládaném provozu je možné všechny projekty, uživatele a skupiny držet v cache. Problémy nebude možné v cache udržet všechny, ale vzhledem k charakteru práce, kdy se pracuje pouze s částí problémů (otevřené a nejnovější), by nemělo docházet k nutnosti vyhazovat položky z cache příliš často.

S kešováním položek souvisí dva základní problémy. Prvním je občerstvování dat. Vytvořená implementace je velmi jednoduchá. Z databáze načtu nový seznam identifikátorů záznamů k zobrazení a zneplatním všechny položky v cache. Díky tomu při následujícím přístupu k libovolné položce dojde k načtení položky z databáze. Pokud bych tento postup uplatnil i na cache s chybami, došlo by ke zbytečnému zneplatnění celé řady nemodifikovaných položek. Proto při načítání seznamu identifikátorů záznamů si také načtu datum poslední modifikace jednotlivých položek a v cache zneplatním pouze ty záznamy, které byly modifikovány. Tento postup by šel samozřejmě aplikovat i na ostatní typy záznamů, ale vzhledem k jejich předpokládanému množství není získané vylepšení až tak velké.

Druhým problémem souvisejícím s kešováním je odložené uložení změn do databáze. Tomuto problému jsem věnoval následující podkapitolu.



Obrázek 4.2: Obecný průběh aktualizace jedné položky

4.6 Aktualizace dat v klientovi

Ze začátku bylo plánováno, že změny se budou ukládat ihned po jejich provedení. Nevýhodou tohoto přístupu je drobné zpoždění mezi ukončením editace a uložením změn, během kterého uživatel nemůže s aplikací pracovat. Toto zpoždění je úměrné počtu ukládaných položek, proto při dávkových operacích by zpoždění mohlo dosáhnout řádově jednotky až desítky sekund.

Uložení změn v jednom záznamu do databáze se skládá z několika kroků. Průběh ukládání je schematicky znázorněn na obrázku 4.2. Mezi nejužší místa aktualizace patří logování změn do historie a hlavně odesílání emailových upozorňování. Při logování do historie je vytvořeno a vykonáno několik SQL příkazů (jejich počet je úměrný počtu změněných informací o chybě). Zde by nejvíce pomohl přechod na databázi podporující triggery. Díky tomu by se logování do historie přesunulo do databáze a z průběhu aktualizace by ho bylo možno odstranit.

Odesílání emailových upozorňování patří mezi nejpomalejší části aplikace. Při testování jsem zjistil, že odeslání samostatné zprávy zabere přibližně 100 ms. Pokud je během jednoho spojení odesláno více zpráv, pak na

odeslání jedné zprávy připadá přibližně 70 ms. Pokud se nepodaří vytvořit spojení se SMTP serverem, běh programu se přibližně na jednu sekundu zablokuje ve funkci provádějící připojení k SMTP serveru. Jako jednu možnost zrychlení provádění aktualizací bych viděl přesunutí kódu odesílajícího emailová upozornění na pozadí. Uživatel tedy nebude zdržován čekáním na odesílání emailů. Díky tomu bude také možné počkat s odesláním do té doby, než se shromáždí větší počet zpráv.

Kvůli prodlevě při ukládá změn jsem se rozhodl, že ukládání budu provádět odloženě. Vlastní uložení změn probíhá buď na pozadí, tedy uživatele nezdržuje od práce nebo po explicitním příkazu uživatele. V posledním případě jsem uživatele pomocí speciálního okna vyzval k trpělivosti.

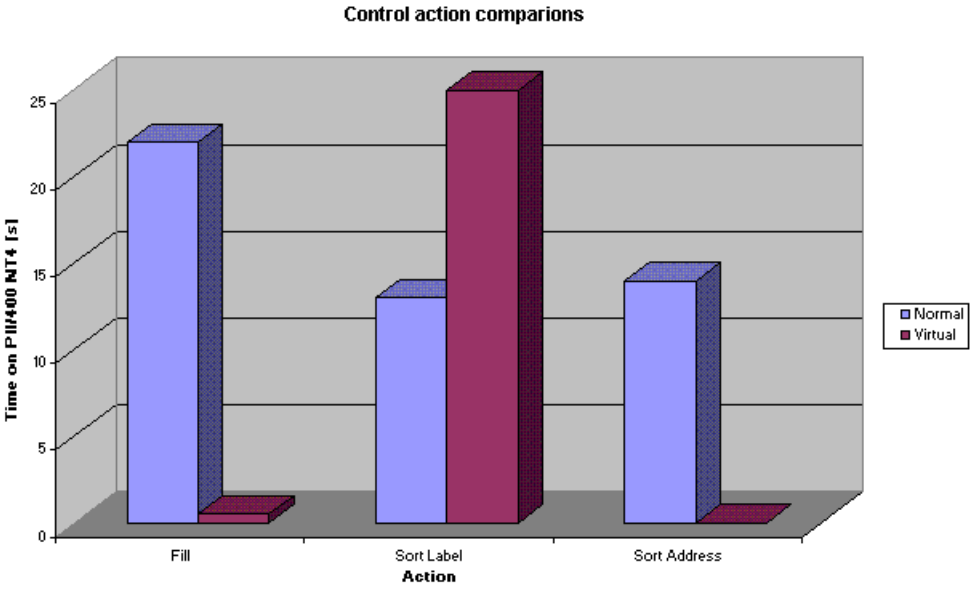
4.7 Překladové informace v klientovi

Kromě dosud popisovaných „hlavních“ dat aplikace (problémy, projekty, uživatelé, skupiny a jejich části) existují v aplikaci i pomocná data načtená z databáze. Mezi tato data patří informace o aktuálním uživateli a aktuálním projektu. Kromě těchto informací se trvale uchovávají tzv. *překladové objekty*, které obsahují mapování číselných identifikátorů na zobrazitelné texty. Díky tomu odpadá dotazování databáze na tyto informace, resp. konstrukce složitých dotazů do databáze. Navíc tyto informace se používají i pro jiné účely, které bych pomocí SQL realizoval pouze velmi složitě.

4.8 Technika zobrazování dat v klientovi

Poslední mechanismus, kterému bych se chtěl v této kapitole věnovat, je princip zobrazování dat. Většina seznamů dat (problémů, projektů, uživatelů, verzí, kategorií atd.) je zobrazena v ovládacím prvku *ListView*. Ovládací prvek ListView klasicky pracuje tak, že se nadefinují sloupce, které se mají zobrazit a pak se všechny položky pomocí metody `InsertItem` postupně vloží do ovládacího prvku. Tento postup má několik nevýhod, které téměř znemožňují jeho využití ve vytvořené aplikaci. Před vložením musí být již položka načtena z databáze, při refreshi je nutno všechny položky znovu vložit a informace jsou ukládány duplicitně. Hlavní nevýhodou je ale rychlost vkládání, která je rozebrána později v této podkapitole.

Vzhledem k nevýhodám uvedeným v předchozím odstavci jsem se rozhodl pro alternativní přístup podporovaný ovládacím prvkem ListView. Při



Obrázek 4.3: Srovnání výkonnosti dvou přístupů k prvku ListView

tomto přístupu, pracovně ho nazývám režim `OwnerData`, zajímá ovládací prvek jediný údaj a to počet položek, které má zobrazit. O ostatní se musí postarat aplikace. Ovládací prvek pak vykresluje pouze údaje o těch položkách, které aktuálně uživatel vidí. Položky, které uživatel nevidí, ovládací prvek nezajímají a tedy ani nemusí být k dispozici. Celý tento mechanismus navíc zapadá do mechanismu získávání dat z databáze popsaného dříve v této kapitole.

V závěrečném odstavci se chci podívat na srovnání výkonu obou výše uvedených přístupů. Toto porovnání vychází z internetového článku (viz [4]). Ovládací prvek byl naplněn 50 000 položkami. Graf na obrázku 4.3 zobrazuje výsledek měření. Pro vytvořený projekt je nejdůležitější první dvojice sloupců ukazující rychlost naplnění ovládacího prvku. Při klasickém přístupu trvá vložení 50 000 položek přes 20 sekund, v režimu `OwnerData` (na obrázku označen jako `virtual`) trvá pod jednu sekundu. Porovnání rychlosti třídění je pro projekt Bugáček nepodstatné, neboť o třídění se stará databáze. Obrázek pouze ukazuje, že ovládací prvek `ListView` je optimalizován na třídění řetězců, ale třídění čísel mu dělá problémy (neboť je interně převádí na řetězce). Pro třídění v režimu `OwnerData` využil autor testu algoritmu QuickSort.

Závěr

V prvních dvou kapitolách jsem představil devět existujících systémů pro evidenci chyb v software. První kapitolu jsem věnoval volně dostupným produktům, druhou komerčním. Nejde samozřejmě o všechny existující produkty. Snažil jsem se vybírat nejznámější a také nejlepší existující software. Při testování produktů jsem si všiml, že většina z nich umí (až na pár drobností) to samé. To znamená nadefinovat uživatele, zařadit je do skupin, nadefinovat projekty a jejich části či verze a k jednotlivým projektům přidat nalezené problémy. Pokročilejší nástroje umožňují definovat i průběh práce na problému, vytvářet konfigurovatelné atributy či vytvářet různé pokročilé sestavy pro tisk a export.

Nalezené výhody jednotlivých produktů jsem se snažil zahrnout do svého produktu. Faktem je, že některé pokročilé vlastnosti jsem do svého softwaru nezahrnul, neboť jejich implementace by vyžadovala neúměrně mnoho času. Naopak nalezeným nevýhodám jsem se snažil vyhnout, což se mi ve většině případů (alespoň podle mého vlastního mínění) podařilo.

Ve třetí kapitole jsem se zabýval analýzou a návrhem architektury aplikace. Popsal jsem důvody, které mě vedly ke řešení, které jsem pro implementaci aplikace zvolil.

Ve čtvrté kapitole jsem se věnoval analýze úzkých míst výkonnosti systému. Krátce jsem popsal mechanismus fungování aplikace v okolí úzkých míst a jeho vliv na výkon systému. V některých případech jsem také nastínil možnosti, jak výkon případně dále zvýšit.

Příloha A

Specifikace programu

V této příloze je zobrazen text specifikace, na základě které bylo vytvořeno popisované softwarové dílo. Jedná o kopii originálního dokumentu, který je možné najít v podadresáři `specification` adresáře projektu **Bugáček**.

Úvod

Projekt **Bugacek** (dále nazývaný pouze aplikace) je software určený pro evidenci chyb a námětů na vylepšení v softwarových projektech.

Bugacek se bude skládat ze tří základních částí:

1) databázový stroj
2) webové rozhraní přístupné prostřednictvím webového prohlížeče
3) speciální klient pro operační systémy Windows

Úkolem databázového stroje je ukládat data a zajišťovat k nim přístup. Pro tento úkol bude využit open source databázový stroj *MySQL*, hlavně vzhledem k jeho rozšířenosti v prostředí nekomerčních webových sídel. Podrobnosti o ukládaných datech jsou dále v části nazvané *Databázový stroj*.

Druhá část aplikace – webové rozhraní – bude vytvořena jako systém skriptů v jazyce *PHP* a *HTML* stránek. PHP bylo zvoleno kvůli jeho rozšířenosti a schopnosti běžet na většině dostupných a používaných webových serverech. Podrobnosti o této části aplikace najdete níže v části nazvané *Webové rozhraní*.

Poslední část aplikace tvoří speciální klient určený pro operační systémy Windows. Tento klient bude poskytovat přístup ke všem funkcím, které jsou dostupné i pomocí webového rozhraní. Výhodou oproti webovému rozhraní

bude snadnější a efektivnější ovládání. Podrobnosti hledejte níže v části nazvané *Klient pro Windows*.

Databázový stroj

Základním úkolem databázového stroje je ukládání dat o chybách v softwaru, námětů na vylepšení softwaru, informací o přistupujících uživatelích a jejich nastavení aplikace, případně určité statistické informace. Jako databázový stroj bude využit existující databázový server *MySQL* v aktuální verzi 4.1 (www.mysql.com).

Níže uvedené údaje, které je potřeba v databázi ukládat tvoří pouze nutný základ. Ukládané údaje je možné v rozumné míře během vývoje rozšiřovat (pokud tím nedojde ke změně v návrhu programu). Ukládaná data lze rozdělit do několika skupin.

Informace o projektu

Projekt je určitým základním kamenem, ze kterého vychází další informace. Při práci s daty (ať již pomocí webového rozhraní nebo klienta pro Windows) musí být vybrán jeden konkrétní projekt, se kterým se bude dále pracovat. Mezi informace uchovávané o projektu patří následující:

- Název projektu
- Popis projektu
- Stav projektu – ve vývoji, stabilní, ukončený
- Povolený – zda se lze do projektu momentálně přihlásit nebo ne
- Zobrazení – veřejný nebo soukromý (určuje, zda budou mít k projektu přístup externí osoby jako recenzenti apod.)
- Operační systém (popř. i hardwarová platforma)
- Hlavní programovací jazyk

Jeden projekt se může obsahovat několik verzí. Součástí informací o verzích by mělo být číslo verze, datum vytvoření dané verze, popis této konkrétní verze a její stav (ve vývoji, ukončená).

Jeden projekt se může skládat z několika částí jako např. jádro aplikace, uživatelské rozhraní, přístup k datům apod. Aplikace bugacek by měla být schopna tyto části definovat. Ke každé části je možno přiřadit jednoho uživatele (typicky vývojáře), který je zodpovědný za tuto část aplikace.

Ke každému projektu je možno přiřadit určité uživatele a skupiny uživatelů (ať již vývojáře, vedoucí, recenzenty a další). Tito uživatelé (skupiny) budou mít k tomuto projektu přístup v rámci svých oprávnění, která jsou mu k tomuto projektu přiřazena. Z těchto uživatelů je jeden uživatel speciální a tím typicky je zakladatel projektu (typicky nějaký vedoucí projektu), který má na starosti správu projektu, jako např. přidávání uživatelů do systému, tvorbu nových verzí.

Ke každému projektu je možno přidat externí soubory (např. s dokumentací). U každého souboru si aplikace bude navíc pamatovat datum vložení tohoto souboru, výstižný název souboru (titulek) a popis obsahu souboru.

Informace o chybách

Každá chyba je přiřazena určitému projektu. Není možno vytvořit chybu, která by se nevztahovala na určitý projekt. Kromě vazby na projekt je možno chybu svázat s určitou částí projektu (viz výše v informacích o projektu). Mezi informace, které je nutno o chybě uchovávat, patří následující:

- Číslo chyby
- Reprodukovatelnost (možné varianty jsou vždy, občas, náhodně, nevyzkoušeno, nelze duplikovat, není zadáno)
- Důležitost (návrh, triviální, malá, střední, velká, pád aplikace, blokující)
- Priorita (např. žádná, nízká, normální, vysoká, urgentní, okamžitě)
- Platforma
- Operační systém (včetně verze)
- Verze produktu, ve které byla chyba objevena (viz informace o projektu)
- Verze produktu, ve které byla chyba opravena
- Uživatel, který chybu zadal do systému
- Uživatel, který má za úkol chybu vyřešit
- Shrnutí problému (krátký popis do titulku)
- Podrobnější popis problému
- Kroky vedoucí k vyvolání problému
- Případné další informace k problému
- Otevřenost problému (veřejný nebo soukromý)
- Datum vložení problému

- Datum poslední změny problému
- Stav problému (nový, potvrzený, schválený, přiřazený, vyřešený, uzavřený)
- Dopad na projekt (žádný, slabý, malá změna, velká změna, změna designu)
- Doba opravy (symbolicky určitá rozmezí, např. jeden den, jeden týden, dva až tři týdny, více než tři týdny)

Základní informací o chybě je stav chyby. Při opravě bude konkrétní chyba postupně procházet různými stavy. Při vložení do systému bude ve stavu *nová*. Zodpovědný uživatel pak chybu musí *potvrdit*. Tento krok je zde z toho důvodu, že chyby mohou vkládat i např. uživatelé zvenku a tímto se zabraňuje nechtěnému vícenásobnému vložení stejné chyby. Dalším volitelným krokem je *schválení chyby*, např. jestli se bude opravovat nebo ne. Potom se chyba přiřadí konkrétnímu uživateli (vývojáři), který má za úkol problém vyřešit. Při přiřazení je možno specifikovat datum, do kdy musí vývojář chybu opravit. Po vyřešení se může problém uzavřít. Některé informace o chybě mají smysl pouze při dosažení určitého stavu, např. verze produktu, ve které byla chyba opravena, má smysl až po vyřešení problému. Chyba nemusí projít všemi stavy ani nemusí dodržovat určité pořadí v jakém stavy bude procházet, vše záleží na konkrétním uživateli (společnosti), jak se k tomu postaví.

Protože jednotlivé problémy postupně procházejí určitými stavy, aplikace si bude pamatovat historii vývoje jednotlivých chyb (tato historie odpovídá v podstatě přechodům mezi jednotlivými stavy a případným změnám hodnot v záznamu chyby). Kromě času modifikace si bude pamatovat uživatele, který modifikaci provedl, název pole, které bylo změněno, a jeho starou i novou hodnotu. Za změnu se považuje i přiřazení (případně znovupřiřazení) určitého uživatele k chybě. Historie musí být udělána tak, aby se z ní šlo zjistit určité statistiky (viz webové rozhraní).

Podobně jako k celému projektu, je možno i ke konkrétnímu problému přidat nějaké soubory (např. nějaké velmi podrobné informace o chybě, výpis logu apod.). U každého souboru si aplikace bude navíc pamatovat datum vložení tohoto souboru, výstižný název souboru (titulek) a popis obsahu souboru.

Každý uživatel se může přihlásit ke sledování konkrétní chyby. Sledování znamená, že při změně stavu této chyby bude uživatel informován elektronickou poštou o této události.

Chyby a problémy nemusí být osamocené. Aplikace musí postihnout následující relace mezi problémy. Základní závislostí je fakt, že dva problémy jsou spolu spojeny, tj. jednou opravou je možno vyřešit oba problémy. Druhou relací je, že jeden problém je závislý na jiném, tj. není ho možno vyřešit dříve, než bude vyřešen ten jiný problém. Opačným případem je fakt, že jeden problém blokuje druhý problém. Závislost má spíše informativní charakter, nebude mít velký vliv na práci aplikace. Jediným důsledkem bude, že při změně stavu jednoho problému se nabídne změna stavu i druhého problému u závislých problémů.

Poslední vymožeností související s chybami je možnost přidání libovolného počtu poznámek k chybě. Kromě vlastního textu poznámky bude aplikace evidovat autora poznámky, zobrazitelnost (soukromá nebo veřejná poznámka), datum vložení a datum poslední modifikace poznámky.

Kromě chyb je možné také zadávat náměty na vylepšení. Informace o námětu na vylepšení se skládá z těchto částí – krátký popis vylepšení (titulek), podrobnější popis vylepšení, případné přiložené soubory, datum vložení a datum poslední modifikace. K námětu musí být také možno přidávat poznámky. Vzhledem k celkové podobnosti námětu s chybou by se námět mohl specifikovat jako speciální chyba.

Informace o uživatelích

Základní informace uchovávané o všech uživatelích aplikace jsou následující:

- Přihlašovací jméno
- Přihlašovací heslo
- Skutečné jméno
- E-mailová adresa
- Datum vytvoření
- Datum poslední návštěvy
- Povolenost (aby bylo možno uživatele dočasně zakázat)
- Výchozí přístupová práva (viz dále)

Aplikace povinně vyžaduje přihlášení. Po instalaci aplikace musí být k dispozici jeden základní účet (administrator). Pro vkládání chyb od neregistrovaných uživatelů je možno vytvořit jeden speciální anonymní účet. Tento účet ale nebude speciální v aplikaci (tam se bude chovat jako normální účet), ale ve tom smyslu, že přihlašovací jméno a heslo bude mít veřejné.

Každý uživatel má možnost si vytvořit několik profilů, které mu usnadní vkládání nových chyb. Profil bude obsahovat základní informace o uživatelově systému - platformu, operační systém a jeho verzi.

Informace o skupinách

Jednotlivé uživatele je možno vkládat do jednotlivých skupin, což usnadňuje správu uživatelských účtů. Informace o skupině musí obsahovat název skupiny (např. správce, manager, vývojář, reportér) a přístupová práva přidělená dané skupině (viz níže). Dále každá skupina obsahuje seznam všech svých členů (jeden uživatel může být členem více skupin).

Při práci se skupinami (týká se ale i práce s uživateli) by mělo být možno vytvářet kopii existující skupiny, aby se dalo snadno vytvořit podobná skupina.

Přístupová práva

Přístupová práva jsou následující (ve výsledku mohou některá práva přibýt):

- Vytvořit nový problém
- Aktualizovat problém (kromě změny stavu problému)
- Sledovat problém (e-mailové upozorňování)
- Přesunout problém do jiného projektu
- Odstranit problém
- Znovu otevřít uzavřený problém
- Změnit stav problému
- Zobrazit seznam uživatelů sledujících daný problém
- Přidávat poznámky k problému
- Aktualizovat poznámky k problému
- Odstraňovat poznámky
- Zobrazování seznamu souborů připojených k problému
- Stahování souborů připojených k problému
- Odstranění souborů připojených k problému
- Přidávání souborů k problémům
- Vytvoření projektu
- Odstranění projektu

- Správa projektu
- Přidávání uživatelů a skupin do projektu
- Zobrazení dokumentů připojených k projektu
- Vložení dokumentů k projektu
- Správa uživatelů a skupin uživatelů
- Prohlížet soukromé objekty (projekty, poznámky, ...)

Práva konkrétního uživatele se skládají z celé řady údajů. Jednak každý uživatel má svoje práva, pak má práva spojená s každou skupinou, ve které je zařazen. Tato práva jsou obecná a slouží pouze pro určení oprávnění k akcím, které nijak nesouvisí s konkrétním projektem (jako je vytváření projektů, správa uživatelů a skupin). Celková práva uživatele se určí logickým součtem práv uživatele a práv všech skupin, ve kterých je uživatel zařazen.

Pro přístup ke konkrétnímu projektu musí být k tomuto projektu přiřazen buď přímo jako uživatel, nebo musí být v některé skupině, která je k tomuto projektu přiřazena. Jeho práva se určí následovně. Vezmou se všechny skupiny, ve kterých se daný uživatel nachází, přiřazené k danému projektu a také práva uživatele, pokud je k projektu přiřazen, a všechna tato práva se logicky sečtou.

Z uvedeného principu plyne, že pokud uživatel je zařazen v nějaké skupině, pak má vždy přidělena všechna práva této skupiny, není možno konkrétnímu uživateli práva omezovat, lze je pouze rozšiřovat.

Upozornění na změny

Kromě možnosti sledování konkrétního problému nabídne aplikace možnost sledování níže uvedených událostí. Sledováním rozumíme upozornění o výskytu události s některými důležitými informace o vzniklé události. Události jsou následující:

- Přidání nového problému
- Přiřazení problému nějakému uživateli
- Přiřazení problému přímo mě
- Znovuotevření problému
- Odstranění problému
- Přidání poznámky k problému
- Změnu vztahu konkrétního problému

- Změna stavu na ...
- Upozornění na blížící se termín splnění

Poslední událost je schéma událostí, jedna událost pro každý možný stav, ve kterém se problém nachází.

Upozornění o výskytu události se bude buď posílat na e-mail nebo se bude zobrazovat varovné okénko v aplikaci. V případě webového rozhraní se při přihlášení zobrazí upozorňující zpráva. Klient pro Windows může být v tomto ohledu trochu pružnější, může danou událost zobrazit v nějakém okénku v okamžiku, kdy ji zjistí. Hlavně by měl být schopen upozornit uživatele na blížící se (případně prošlý) termín splnění daných úkolů.

Upozornění na blížící se termín splnění nebude patrně odesílán automaticky elektronickou poštou.

Webové rozhraní

Webové rozhraní tvoří základní základ pro přístup k uloženým datům. Webové rozhraní bylo zvoleno jako primární rozhraní hlavně pro jeho nezávislost na operačním systému klientského počítače.

Webové rozhraní bude vytvořeno jako sada HTML stránek generovaných skripty v PHP. Vzhled systému bude určen CSS soubory. Jednotlivé formuláře budou mít vstupy ošetřeny nejenom na straně serveru (PHP), ale i na straně klienta s využitím JavaScriptu.

Před započetím práce s webovým rozhraním je nutné se přihlásit k systému. Po přihlášení se uživateli zobrazí přehledová stránka, na které je přehled několika posledních změn v systému. Mezi tyto změny patří kromě problémů naposledy změněných, uzavřených, vytvořených také problémy přihlášeným uživatelem sledované a přihlášenému uživateli přiřazené. Na většině stránek by měla být možnost změny aktuálního projektu. Pokud není žádný projekt vybrán, pak při přechodu na další stránky se zobrazí stránka nabízející výběr projektu.

Jádrem webové části bude zobrazování vlastních chyb v softwaru. Chyby musí jít stránkovat a hlavně filtrovat podle následujících kritérií – stavu, autora, verze produktu, priority, důležitosti, kategorie, přiřazenosti (tj. uživatele, který má za úkol problém vyřešit) a také podle data vytvoření nebo změny záznamu. Implicitně budou vyfiltrovány uzavřené problémy. Kromě vlastního zobrazování musí být aplikace schopna vyfiltrované záznamy exportovat do souborů CSV a zobrazit je ve stavu vhodném k vytištění z webo-

vého prohlížeče. Uživatel musí být také schopen nechat si zobrazit historii problému (tj. jeho vývoj v čase – viz výše).

Kromě filtrování by aplikace měla umožňovat řazení podle zobrazených sloupců. Zobrazené by měly být pouze základní sloupce umožňující identifikovat záznam a zobrazující pouze ty nejdůležitější informace (kvůli přehlednosti, aby se jeden záznam vešel na jeden řádek). Mezi tyto informace zcela určitě patří číslo chyby, její titulek, kategorie, důležitost a stav. Dále aplikace musí umožňovat vyhledávání ve všech chybách daného projektu.

Uživatel může (pokud má na to oprávnění) přidávat nové záznamy, editovat stávající (včetně změny stavu) a také záznamy chyb odstraňovat. Vzhledem k většímu množství údajů zadávaných o jednotlivých chybách, by přidávání nového záznamu (a tím spíše i editace stávajícího) mělo být rozděleno kvůli přehlednosti do několika stránek.

Součástí stránky zobrazující jednotlivé záznamy by měla být možnost rychlé změny stavu a přiřazení uživatele několika záznamů najednou. Uživatel by také měl být schopen záznamy přesunovat či kopírovat do jiného projektu, hromadně je rušit a uzavírat.

Součástí aplikace by také měla být možnost generovat protokol o opravených chybách, který by šel přiložit k aplikaci. Výstupním formátem tohoto protokolu by měl být obyčejný textový soubor.

Uživatelé s odpovídajícími oprávněními (tj. správci) mohou samozřejmě přes webové rozhraní provádět správu projektů (vytvářet nové, upravovat existující a odstraňovat existující), uživatelů (vytváření nových, měnit oprávnění, zakazovat a povolovat účty, odstraňovat staré účty).

Aplikace by si měla pamatovat některá nastavení uživatelů. Určitě si musí pamatovat poslední projekt, který měl uživatel vybrán, poslední použité řazení a filtrování a výchozí profil pro zadávání nových chyb (profil uchovává informace o operačním systému uživatele).

Součástí webového rozhraní by měla být i stránka (stránky) zobrazující *statistiky* o chybách a vývojářích. Statistiky jsou dvou druhů. Jedna statistika se koná rozložení chyb podle různých projektů. Aplikace musí umět zobrazit rozložení počtu chyb otevřených, vyřešených a uzavřených podle následujících kritérií – podle projektu, podle stavu, podle důležitosti, podle kategorie, podle priority a také podle stáří (v nějaké rozumné škále, např. jeden den, dva dny, týden, měsíc, dva měsíce, čtvrtletí, rok).

Druhý typ statistik zobrazuje informace o uživatelích (především vývojářích). Aplikace by měla být schopna zobrazit počty chyb nahlášených jednotlivými uživateli (i s rozdělením na chyby otevřené, vyřešené a uza-

vřené, popř. s rozdělením na kategorie uvedené níže u vývojářů), počty chyb přiřazených jednotlivým vývojářům rozložené do následujících kategorií – otevřené, vyřešené, znovuotevřené, nelze vyřešit, nelze nasimulovat, duplicitní, nejde o problém, odložené, neopravované) včetně procentního zobrazení opravených úkolů vzhledem k přiřazeným.

Klient pro Windows

Poslední částí aplikace by měl být speciální klient určený pro operační systémy Windows. Tento klient bude vytvořen s využitím Windows API za podpory knihovny MFC. Klient bude pokrývat *stejné funkce* jako webové rozhraní. To znamená, že bude umožňovat nejenom veškerou práci se záznamy chyb a námětů na vylepšení, ale bude umožňovat i správu projektů a uživatelů (tj. činnosti správce). Hlavní výhodou klienta by mělo být snadnější a efektivnější ovládání. Většina příkazů by měla mít přiřazeny klávesové zkratky (s možností jejich přizpůsobení). Dialogy pro zadávání dialogů by měly být co nejvíce přehledné. Protože při práci s chybami se pracuje s velkým množstvím údajům, je vhodné rozdělit dialog na několik částí např. využitím tzv. „záložek“.

Klient bude orientován na jednoho uživatele, proto zde bude možnost lokálně si uložit přihlašovací údaje k systému včetně výchozího projektu, to znamená, že aplikace provede při spuštění přihlášení k datovému zdroji. Nastavení aplikace, které bude společné s webovým rozhraním (výchozí projekt, počet záznamů na stránku apod.) bude možno ukládat buď lokálně (tj. nezávisle na webovém rozhraní) nebo do stejného úložiště jako webové rozhraní. První přístup umožňuje rozdílná nastavení pro windows klienta a webové rozhraní, druhý přístup naopak toto nastavení sjednocuje.

Klient by si měl pamatovat nastavení uživatelského rozhraní (jako např. velikosti okna, rozložení panelu nástrojů apod.) tak, aby si uživatel nemusel uživatelské prostředí při každém spuštění programu znovu nastavovat. Nastavení by se mělo ukládat tak, aby pokud daný počítač a tedy i klienta používá více uživatelů, tak aby každý uživatel měl vlastní nastavení.

Škálovatelnost

Aplikace by měla být vytvořena tak, aby byla schopna obsluhovat řádově desítky uživatelů a uchovávat řádově tisíce chyb, resp. námětů na vylepšení.

S tím souvisí dva problémy – rychlost a konzistence databáze. Oba problémy se týkají spíše klienta pro Windows než webové části aplikace, kde kvůli bezstavovosti HTTP není příliš možností, jak oba problémy řešit.

Co se týká rychlosti, klient pro Windows by měl být schopen v relativně rozumné době načíst i poměrně velké množství položek. Pro urychlení jeho práce budou položky (buď přímo nebo za využití driveru databáze) krátkodobě cachovány. Uživatel by měl mít možnost nějakým příkazem znovunačíst všechna data z databáze. Úpravy budou probíhat lokálně a teprve na přání uživatele budou změny přeneseny do databáze. Přenesení do databáze může být i automatické, např. v definovaných časových intervalech. Kvůli tomuto přístupu je ovšem nutno řešit následující problém.

Při aktualizaci dat musí aplikace (webové rozhraní i klient pro Windows) řešit problém, pokud aktualizuje data, která už před ní aktualizoval někdo jiný (aktualizoval je mezitím, co došlo k načtení dat a byl vznesen požadavek na aktualizaci). Potom aplikace při aktualizaci neuspěje a zobrazí uživateli seznam neaktualizovaných řádků. U každého řádku zobrazí seznam původních hodnot, aktuálních hodnot v databázi a navrhované změny. Potom je na uživateli, zda aktualizaci vynutí nebo zahodí. Tato situace se může opakovat několikrát za sebou. Ovšem takováto situace by neměla typicky nastávat, protože jednotlivé položky bude aktualizovat uživatel, který je má přiděleny a toto přidělení položek by mělo být disjunktní.

Vlastní aktualizace (přidávání, odstraňování) položek musí být uděláno atomicky a to buď vhodným napsáním SQL příkazů nebo jiným prostředkem, který databáze nabízí (krátkodobé zamknutí tabulek, transakce). První způsob by měl být upřednostněn všude, kde to jde.

Historie

Změny ve verzi 0.03

Přiřazení uživatele k projektu:

V původní verzi je psáno, že jeden z přiřazených uživatelů je speciální. Tato speciálnost uživatele byla vypuštěna, doporučujeme ji ale zachovat tím, že jednomu uživateli (vedoucímu projektu) přiřadíme větší přístupová práva.

V části věnované přístupovým právům se píše, že pokud se uživatel chce přihlásit k projektu, musí být k němu přiřazen. Tato podmínka byla vypuštěna. Uživatel se může přihlásit ke kterémukoliv projektu, pouze to zda uvidí

nějaká data závisí na tom, zda na to má oprávnění. Tento typ oprávnění je doporučeno dávat při přiřazování k projektu, ale není to podmínkou.

Webové rozhraní:

V původním dokumentu je zmiňováno, že po přihlášení uživatele se mu zobrazí stránka obsahující několik posledních změn. Tato stránka byla vypuštěna jako nežádoucí, uživatel má možnost nechat si zobrazit pouze ty změny v systému, o které má zájem pomocí upozorňování (buď emailem nebo speciální stránkou).

Stránka zobrazující upozornění na změny v systému byla přesunuta proti původnímu dokumentu z pozice po přihlášení. Nyní je možné nechat si ji zobrazit z menu. Důvodem přesunu byla zmatenost v ovládání aplikace, pokud stránka byla po přihlášení.

Klient pro Windows:

V původním dokumentu je možnost volby ukládání společného nastavení s webovým rozhraním buď do databáze nebo samostatně do lokálního souboru. Byla zvolena první možnost, hlavně kvůli tomu, že to představuje chování, které se uživateli bude více zamlouvat.

Příloha B

Obsah přiloženého CD

V závěrečné části bych se rád krátce zmínil o obsahu CD přiloženého k této práci. Přiložené CD obsahuje kompletní software **Bugáček** včetně zdrojových kódů a dokumentace. Projekt bugáček je umístěn v podadresáři `Bugacek`. Jednotlivé podadresáře v tomto adresáři mají následující význam:

- Podadresář `Specification` obsahuje specifikaci, na základě které program vznikl
- Podadresář `Dtb` obsahuje skripty pro vygenerování databázových tabulek pro MySQL databázový stroj
- Podadresář `Web` obsahuje zdrojové kódy vytvořeného webového rozhraní aplikace
- Podadresář `Win` obsahuje zdrojové kódy klienta pro operační systém Windows včetně potřebných knihoven
- Podadresář `Win/Release` obsahuje přeloženého win klienta
- Podadresář `Doc` obsahuje programovou a uživatelskou dokumentaci pro všechny součásti systému

V adresáři `Servers` jsou v samostatných adresářích poslední verze webového serveru, interpretu PHP a databázového serveru, které je možné použít při zprovozňování projektu **Bugáček** na cílovém počítači.

Adresář `Standalone` obsahuje projekt **Bugáček** připravený ke spuštění bez nutnosti instalovat databázový a webový server. Podrobnosti hledejte v přiloženém souboru `readme.txt`.

Literatura

[1] The Bugzilla Team: *The Bugzilla Guide* (http://www.bugzilla.org/docs/2.22/html/lifecycle.html)

[2] TieraSoft: *User Guide - Defect Manager for Windows* (http://www.tierasoft.com/products/defectmanager/manuals/User%20Guide%20-%20Web.pdf)

[3] TieraSoft: *User Guide - Defect Manager for Web* (http://www.tierasoft.com/products/defectmanager/manuals/User%20Guide%20-%20Windows.pdf)

[4] Andrew Small: *MFC Virtual ListView Control* (http://www.codeguru.com/cpp/controls/listview/advanced/article.php/c4151/)