Univerzita Karlova v Praze
Filozofická fakulta

Ústav informačních studií a knihovnictví

Studijní program: informační studia a knihovnictví
Studijní obor: informační studia a knihovnictví

Bakalářská práce

Tereza Heinová

Funkční testování softwarových aplikací
Functional testing of software applications

Praha 2011

Vedoucí práce: Doc. RNDr. Jiří Souček DrSc.

Oponent bakalářské práce:

Datum obhajoby:

Hodnocení:

Prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

V Praze, dne 10. 7. 2011 ……..

podpis studenta

Identifikační záznam:

HEINOVÁ, Tereza. *Funkční testování softwarových aplikací [Functional testing of software application]*. Praha, 2011-08-06.

91 s. Bakalářská práce (Bc.). Univerzita Karlova v Praze, Filozofická fakulta, Ústav informačních studií a knihovnictví. Vedoucí bakalářské práce Jiří Souček

Abstrakt:

Bakalářská práce popisuje průběh funkčního testování software. První část práce definuje místo a význam testování v procesu vývoje software. Vysvětluje základní druhy používaných testů a jejich účel. Pozornost je věnována rolím, které mají v procesu testování klíčovou úlohu. V druhé časti práce, je uveden podrobnější popis všech fází životního cyklu funkčního testování software. Stavebním kamenem v cyklu testování je plánování, zde je vysvětlen přístup k vypracovávání postupu testování a jeho harmonogramu. Druhou fází je příprava prostředí pro testování. Po realizaci testů následuje vyhodnocování výsledků testování. Zde jsou rozebírány zásady zadávaní chyb a jejich následné řešení. Součástí přípravy k testování je výběr vhodných podpůrných nástrojů a vytvoření testovacích skriptů. Závěr bakalářské práce hodnotí důležitosti testování v procesu vývoje software.

Abstract:

Bachelor thesis describes the course of functional software testing. The first part defines the place and importance of testing in software development process. Explains the basic types of tests used and their purpose. Attention is paid to the roles that are in the process of testing a key role. In the second part of the work, a more detailed description of all phases of the life cycle of functional testing software. The cornerstone of the testing cycle of planning, explained there is access to the drafting process and its testing schedule. The second phase is to prepare an environment for testing. Following the completion of testing followed by evaluation of test results. Here are discussed the principles of entering errors and their subsequent solutions. Part of the preparation for testing is the selection of appropriate support tools and the creation of test scripts. Conclusion The thesis evaluates the importance of testing in software development process.

Klíčová slova:

Analytik, aplikace, defect, manažer, software, tester, testování, tým, vývojář.

Keywords:.

Analytics, application, defect, developer, manager, software, tester, testing, team.

Obsah

Předmluva

O testování software je napsáno hodně publikací, ale jen některé lze aplikovat v praxi. Na úrovni teorie lze vymyslet spoustu pravidel a možností testování. I když je tester vybaven spoustou těchto teoretických znalostí, v praxi může být lehce v koncích. Testování software je specifická práce, která je velmi náročná. V této bakalářské práci bych ráda shrnula základní znalosti z oblasti testování software. Tyto znalosti vycházejí z mé několikaleté práce na pozici testera a projektového specialisty na projektech v České pojišťovně a.s. a v Softec CZ. I když se nyní věnuji převážně projektovému řízení a testování jen okrajově, můj zájem o testování dále trvá a stal se koníčkem. V této práci jsem čerpala ze svých znalostí a zkušeností, a proto ve většině práce nejsou uvedeny citace. Použité informační zdroje jsou citovány v souladu s normami ČSN ISO 690 a ČSN ISO 690-2. V závěrečném seznamu literatury jsou řazeny abecedně.

Literaturu jsem vybírala na základě rešerše, zpracované pro potřeby Ústavu informačních studií a knihovnictví v rámci předmětu PhDr. Evy Bratkové – Bibliografické rešeršní služby. Ta čítala celkem 80 záznamů a byla časově vymezena roky 1990–2010 [Heinová, 2010], nicméně zde je uvedeno jen třicet základních pramenů, které považuji za stěžejní. Z uvedené literatury jsem čerpala jen pro inspiraci, neboť jsem si jako cíl práce stanovila sestavení metodiky dle mých praktických zkušeností. Tato práce by měla být chápána jako malý praktický průvodce pro studenty, kteří začínají svou kariéru jako testeři a projektoví manažeři. A v neposlední řadě pro všechny, kteří se o testování zajímají a zároveň upřednostňují praktického pomocníka před čistě teoretickou publikací.

První kapitoly této práce popisují co je testování, jaká je jeho důležitost v procesu vývoje software. V dalších kapitolách se věnuji životnímu cyklu funkčního testování od plánování přes přípravu testů, samotné testování až po vyhodnocování testů a řešení nalezených chyb. V průběhu psaní práce jsem vyměnila kapitolu o vyhodnocování testů s kapitolou o testovacích nástrojích. Důvodem byl výzkum free testovacích nástrojů, který jsem pojala jako samostatnou kapitolu. V každé kapitole jsou obrázky, tabulky nebo grafy. Toto uspořádání je z čistě praktického hlediska.

Ráda bych poděkovala vedoucímu bakalářské práce panu Doc. Součkovi za podnětné připomínky a návrhy v průběhu zpracování tohoto textu.

1 Co je testování?

Testování je vlastně zkoumání testovaného produktu. Jedná se o proces ověřování kvality naimplementovaného software vůči zadání zákazníka, které je obvykle popsáno ve funkční dokumentaci. Obvyklé vnímání testování je takové, že se jedná o spouštění testů a software. Samotné testování je daleko rozsáhlejší, může být jak manuální tak automatické. Začíná před a končí dlouho po vykonání testů. K testování patří i aktivity jako je plánování, řízení, výběr testovacích podmínek, navržení testovacích případů, kontrola a vyhodnocování výsledků a samozřejmě reportování výsledků.

Testování má obvykle nějaký cíl. Buď tím cílem je nalezení defektů, nebo předcházení defektům. Každý tester a případně zodpovědný projektový manager ví, že testování odhalí přítomnost chyb v software, ale nedokáže, že v software chyby nejsou. Testování pouze snižuje pravděpodobnost chybovosti software, ale nezaručuje jeho stoprocentní bezchybnost. Nikdy totiž nelze otestovat všechny kombinace vstupních a výstupních prvků (výjimku tvoří triviální testovací případy). K odhalení co největšího počtu chyb software je nutné s testováním začít co nejdříve.

Každý kdo se věnuje testování, si zakládá na kvalitě. Většina profesionálů se řídí normami jako je ISO 9126 – mezinárodní standard pro vyhodnocování kvality SW, anebo metodou FURPS. O metodě FURPS vedl poutavou přednášku David Růžička v roce 2010. Metoda FURPS byla vytvořena společností Hewlett–Packard na základě potřeby definovat, jak poznat a ověřit kvalitu dodávaného software. První zmínky o této metodě pocházejí z roku 1986. Robert Grady a Deborah Caswell o této metodě psali v knize „Software Metrics: Establishing a Company–Wide Program“ už v roce 1987.

Jak se zmiňuje David Růžička (2009) ve svém článku, tak FURPS se dívá na kvalitu software nebo informačního systému z pěti základních hledisek: funkčnost, užitečnost, spolehlivost, výkon a rozšiřitelnost. „Metoda definuje na nejvyšší úrovni, co by mělo být hodnoceno, ale nespecifikuje, jakým způsobem mají být oblasti hodnoceny. V těchto jednotlivých oblastech musí být dále vytvářeny konkrétní metriky a jejich hodnoty.“

F (functionality) – funkčnost

Zaměřuje se na hlavní funkcionality a schopnosti programu, zda software podporuje byznys proces a bezpečnost systému.

- **U** (usability) – užitečnost

Hodnotí se zejména z pohledu lidského faktoru; jakým celkovým dojmem působí aplikace, dokumentace a školící materiály.

- **R** (reliability) – spolehlivost

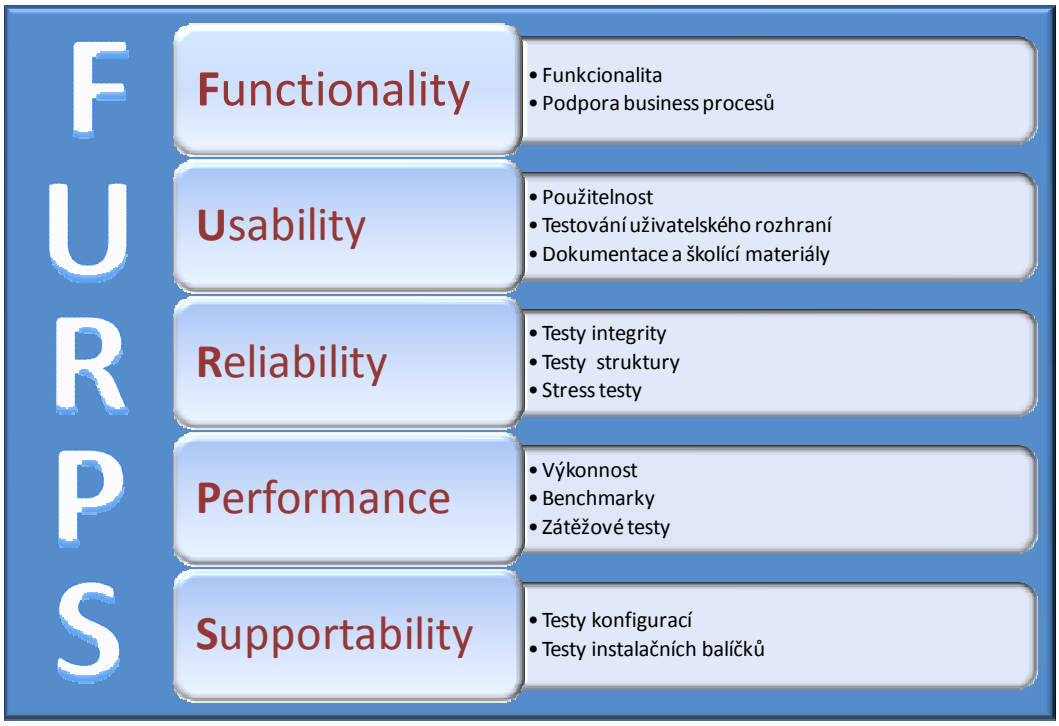
Jedná se o hodnocení četnosti a závažnost chyb, přesnosti zpracování vstupů a výstupů.

- **P** (performace) – výkon

Hodnocení celkové rychlosti odezev systému a zpracování klíčových byznys aktivit. Zároveň se sledují i technické parametry testovaného systému, např. vytížení zdrojů OS, zatížení síťového provozu, vytížení jednotlivých komponent systému.

- **S** (supportability) – rozšiřitelnost

Dalším hlediskem hodnocení je oblast rozšiřitelnosti aplikace v případě potřeby, možnosti údržby aplikace, její testovatelnosti. V této oblasti se taktéž hodnotí i přizpůsobitelnost a možnosti konfigurování. V zájmu není jen aplikace samotná, ale i její součásti pro konfiguraci a monitorování provozu.



Obrázek č. 1

Zdroj:www.testqa.cz

2 Důležitosti testování v procesu vývoje software

Ač by se někomu mohlo zdát, že testování je hračka a není až tak důležité, tak opak je pravdou. Vývoj software je velmi drahou záležitostí. Většina firem, které si nechávají „ušít“ software na míru, se také rozhoduje podle ceny služeb. Firmy, které se přihlašují do výběrových řízení, dávají takzvanou nabídku. Nabídka obsahuje odpovědi na základní otázky projektového řízení – co, jak, kdy, kde a za kolik. Také platí, že každá firma na trhu má know-how na určitou oblast. Některé firmy se zaměřují na software pro oblast bankovnictví, pojišťovnictví, jiná na oblast zdravotnictví a podobně. Takže pokud jde dodavatel s nabídkou k zákazníkovi, musí být perfektně připraven. Většina zákazníků má své „oblíbené“ dodavatele. Ovšem to neznamená, že nabídku dostane vždy jeden dodavatel. Ale pokud je dodavatel solidní, umí vyjít zákazníkovi vstříc a dodává kvalitní produkt, má velkou šanci získat další zakázku. Na tom aby byl produkt kvalitní, mají velkou zásluhu testeři. Testeři nejsou jen tak nějací členové projektového týmu. Musejí mít řadu znalostí.

2.1 Co by měl tester umět?

Na to není snadná odpověď. Tester by měl mít široký rozhled. Každý tester by měl být tak trochu hračička. Měl by si umět představit, co bude budoucí uživatel se software dělat, jak ho bude používat. Měl by samozřejmě dobře znát software, který testuje. K tomu mu může pomoci kvalitní funkční desing. Velmi přínosné je, když se testeři podílí na analýze. Obecně lze říct, že každý tester se může stát dobrým analytikem, ale dobrý analytik se často nestává dobrým testerem.

Klíčové znalosti testovacího týmu:

Test manager

- Motivace
- Zkušenosti
- Vedení a řízení týmu
- Organizační schopnosti
- Znalost byznysu zákazníka
- Soft skills

Test analytik

- Znalost testovacího procesu
- Znalost technik pro test design
- Kreativita, analytické a koncepční myšlení
- Znalost byznys procesů
- Zkušenosti s řízením menších týmů
- Znalost nástrojů pro podporu testování
- Znalost SQL
- Základní zkušenosti s programováním

Tester

- Znalost procesu testování
- Technologické znalosti
- OS, DB, aplikační servery atd.
- Síťové prostředí
- Používání podpůrných nástrojů
- Znalost SQL
- Komunikační schopnosti
- Základní zkušenosti s programováním

Konzultant pro testování

- Výborná znalost různých metodik
- Testování a vývoj
- Znalost testovacích nástrojů, včetně implementace
- Znalost podnikových procesů
- ITIL, ITSM a Best Practise
- Dlouholeté zkušenosti s testováním

- Mentoring
- Koučing
- Komunikace
- Školení

Specialista automatizace testování

- Znalost SW pro automatizaci testování
- Funkční i zátěžové
- Zkušenosti z realizovaných projektů
- Znalost základních architektur systémů
- Znalost programování
- DB a SQL
- Zkušenosti s testováním obecných platforem

3 Význam testování v procesu vývoje software

3.1 Základní druhy testů

V literatuře se uvádí testování bílé a černé skříňky. Téma bílá/černá skříňka je podrobně popsáno v knize Rona Pattona (2002), pro potřeby této práce se o testování bílé a černé skříňky budu zmiňovat jen okrajově. Bílá skříňka, neboli white-box je pohled na testování se znalostí vnitřní architektury aplikace. Tedy z pohledu zdrojového kódu. Pro testování bílé skříňky, je důležité, aby měl tester znalosti v oblasti programování a programovacích jazyků. Naopak testování černé skříňky, black-boxu, vychází z předpokladu, že tester nezná vnitřní architekturu aplikace. Tester testuje z pohledu koncového uživatele. Zná vstupy a zadává vstupní hodnoty do aplikace a kontroluje výstupy. Tester tedy kontroluje skutečné chování aplikace oproti očekáváním. Výhodou je, že tester nemusí znát architekturu a programovací jazyk. Druhou výhodou je, že testy mohou provádět i méně zkušení testeři, kteří postupují podle testovacích scénářů. U testování černé skříňky je „výhodné“, že tester neví co se uvnitř aplikace děje, není tak sváděn k tomu, aby testy přizpůsoboval tomu, co je napsáno ve zdrojovém kódu. Tester by se měl držet funkčního designu a kontrolovat práci programátorů.

V této práci se zaměřím na funkční testování software, které lze dále dělit do několika kategorií. Těmto druhům testů se věnují i nadšenci na internetových stránkách. Většinou se jedná o profesionální testery, kteří se rozhodli se podělit o své znalosti. Testování se dá rozdělit také podle stádií, ve kterých se zrovna testování nachází.

Testy se dělí na tři základní kategorie:

1) Funkční testy
2) Zátěžové testy
3) Automatické testy

Funkční testy ověřují, zda všechny naimplementované funkčnosti a části fungují, jak mají. V rámci funkčních testů jsou i další testy, které vždy ověřují funkci. Tyto testy se provádějí v různých fázích testování. Unit testy jsou testy, které většinou provádějí programátoři. Obecně lze říct, že po sobě zkontrolují kód, který naprogramovali. Není dobré, aby byl programátor zapojen do ostatních druhů testů. Pokud něco vyrobíte, tak jen zřídka

odhalíte vlastní chyby. Po Unit testech následují Assembly testy, kterými pokračují programátoři v kontrolování. Integrační testy ověřují, zda všechny části aplikace spolu správně komunikují. Následují nejdůležitější testy ze všech, a sice Systémové testy. Tyto testy ověřují, zda aplikace jako celek funguje správně. Systémové testy probíhají v několika kolech, ve kterých se ověřuje, zda aplikace odpovídá všem požadavkům zákazníka. Testují se detailně a jednotlivě její části, ale také se testuje jako celek. V každém kole se tester snaží najít chyby, aby byly co nejdříve opraveny a mohly by být včas retestovány. Po systémových testech následují testy akceptační, což je druhá důležitá fáze. Akceptační testy provádí sám zákazník a kontroluje, zda jsou všechny podmínky pro akceptaci splněny a zda byla aplikace vyvinuta přesně podle zadání.

Zátěžové testy jsou velmi specifické a v této práci se jim nebudu věnovat podrobně. Zátěžové testy ověřují použité platformy, technologie a infrastruktury systému. Ověřují chování celého systému při předpokládané zátěži a při měsíčních či ročních špičkách. Dále mohou ověřovat chování systému při výpadku jednotlivých komponent systému.

Automatické testy lze použít jak pro funkční tak pro zátěžové testování. Tyto testy se provádějí pomocí nástrojů pro automatické testování jako je OpenSTA , Borland SilkPerformer , IBM Rational Performance Tester, Compuware QA Load a nejznámější HP LoadRunner .

3.2 Role v procesu testování

Důležitou úlohu v procesu testování software hrají role. Pro správné řízení testování je podstatné, aby každý z členů projektu a testovacího týmu znal svou roli a věděl, jaké jsou jeho povinnosti a za co odpovídá. Tyto role jsou v rámci každého projektu rozdílné v pojmenování. Rozdělení rolí bývá externí a interní. Záleží na tom, zda se projekt uskutečňuje v rámci jedné organizace nebo zda se na projektu podílí externí dodavatelé.

Začněme obecnými rolemi v rámci celého projektu.

3.2.1 Sponzor

Sponzor stanovuje celkový cíl projektu a obchodní požadavky. Rozhoduje o podstatných změnách směru projektu. Odpovídá za business case projektu. Odpovídá

za celkové plnění harmonogramu, cílů projektu a rozpočtu. Odpovídá za vyhodnocení přínosů projektů ve stanoveném období po jeho dokončení.

3.2.2 Projektový manager

Projektový manager odpovídá za přípravu návrhu projektu. Odpovídá za řízení rizik na projektu, za zajištění lidských i materiálních zdrojů. Výkonně řídí projekt. Odpovídá za čerpání lidských, materiálních a finančních zdrojů dle schválené výše a struktury a včas indikuje riziko jejich překročení. Odpovídá za aktualizaci harmonogramů a plánů projektu. Odpovídá za zpracování potřebné projektové dokumentace dle metodiky, připravuje pravidelné reporty o stavu projektu. Je zodpovědný za řádné formální ukončení projektu.

3.2.3 Projektové týmy

Mezi projektové týmy patří Analytický tým, Technický tým a Testovací tým. Tyto týmy se spolu podílejí na vzniku software.

Analytický tým zpracovává funkční dokumentaci a podílí se na vývoji software v jeho počátku. Analytik navrhuje spolu s Businessem, jak by software mělo vypadat, jaké budou jeho základní funkčnosti, odpovídá na první otázky. Funkční analytik je zodpovědný za funkční návrh vyvíjeného systému, případně i za High-level technický design. Poskytuje konzultace Test koordinátorovi a Test designerovi.

Technický tým pracuje na vlastním vytvoření software. V technickém týmu pracují programátoři, kteří vytvářejí samotný software na základě specifikace analytiků. Také jsou důležitou podporou testovacímu týmu.

Testovací tým je konečným článkem při vývoji software. Je také podstatným členem projektového týmu, neboť na něm závisí bezchybné fungování software. Členové testovacího týmu kontrolují vzniklý produkt a odpovídají za jeho bezchybné fungování. Ale i role v rámci testování jsou rozděleny.

3.2.4 Role v rámci testování

3.2.4.1 Test koordinátor

Test koordinátor zodpovídá za přípravu, koordinaci a řízení celého testování. Sleduje stav řešených chyb, organizuje retesty , připravuje protokoly a podklady na jednání. Vyhodnocuje testy a prezentuje výsledky testování businessu. Zkušený tester je zpravidla

skvělým test koordinátorem. Má znalosti z dané problematiky, má kontakty v organizaci i u zákazníka a co je nejdůležitější, má bohaté zkušenosti s testováním. Bohužel se často setkáváme se situací, kdy jsou do pozice test koordinátora přijímáni lidé „ z ulice“, kteří se teprve seznamují s celým chodem projektu. Takovýto test koordinátor má možná znalosti z jiných profesí, ale méně zkušeným testerům není vůbec nápomocen.

3.2.4.2 Test designer

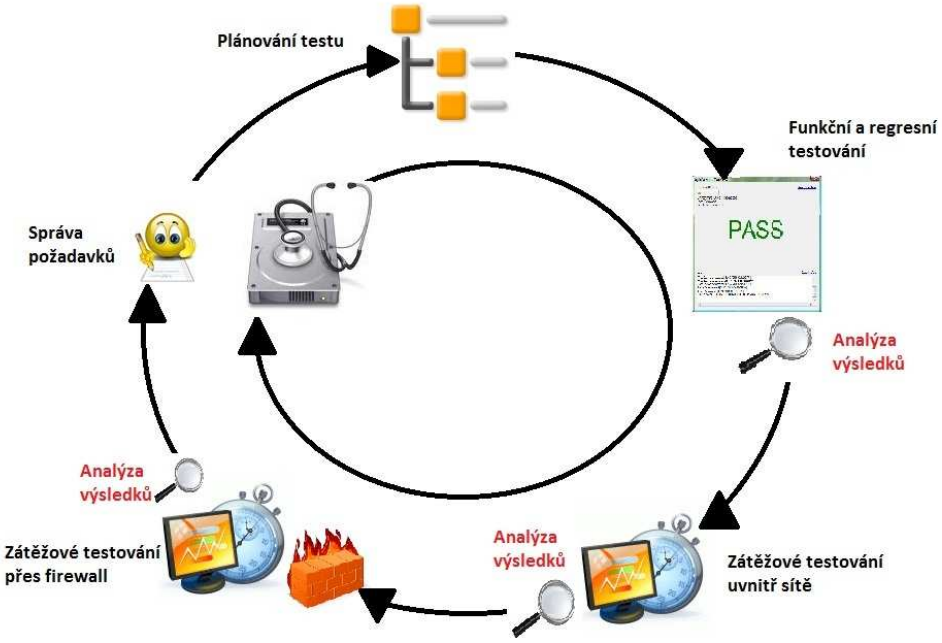
Test designer je někdy také nazýván Test Analytikem. Test designer vytváří kompletní testovací dokumentaci pro funkční testy, které jsou realizované v rámci systémových testů. Připravuje nejen testovací data, ale také připravuje testovací případy. Společně s Test koordinátorem vyhodnocuje testy a prezentuje výsledky testů.

3.2.4.3 Tester

Tester provádí testy dle připravené testovací dokumentace, zapisuje výsledky a eviduje zjištěné defekty. Rozdíl mezi test designerem a testerem je většinou nepatrný nebo nulový. Každý tester by měl umět vytvářet testovací dokumentaci a skripty. Rozdíly pak jsou u testerů juniorů a seniorů. Ze zkušených testerů seniorů se většinou stávají Test koordinátoři.

4 Životní fáze cyklu funkčního testování software

Životní fáze testování lze vyjádřit jednoduše. V praxi se ovšem setkáváme i s takovým průběhem, kdy nejsou všechna stádia dodržena. Největší problém bývá se změnovými požadavky, které přicházejí již ve fázi funkčních testů. V takovém případě se mění funkční desing a testovací tým musí udělat změny v testovacích scénářích. Příhodné je otestovat regresně celou aplikaci, protože i malá změna může ovlivnit celý proces chování aplikace. Níže uvedený obrázek z přednášky pana Davida Růžičky je učebnicovým příkladem životní fáze software.



Obrázek č. 2

Zdroj: www.testqa.cz

4.1 Plánování testů

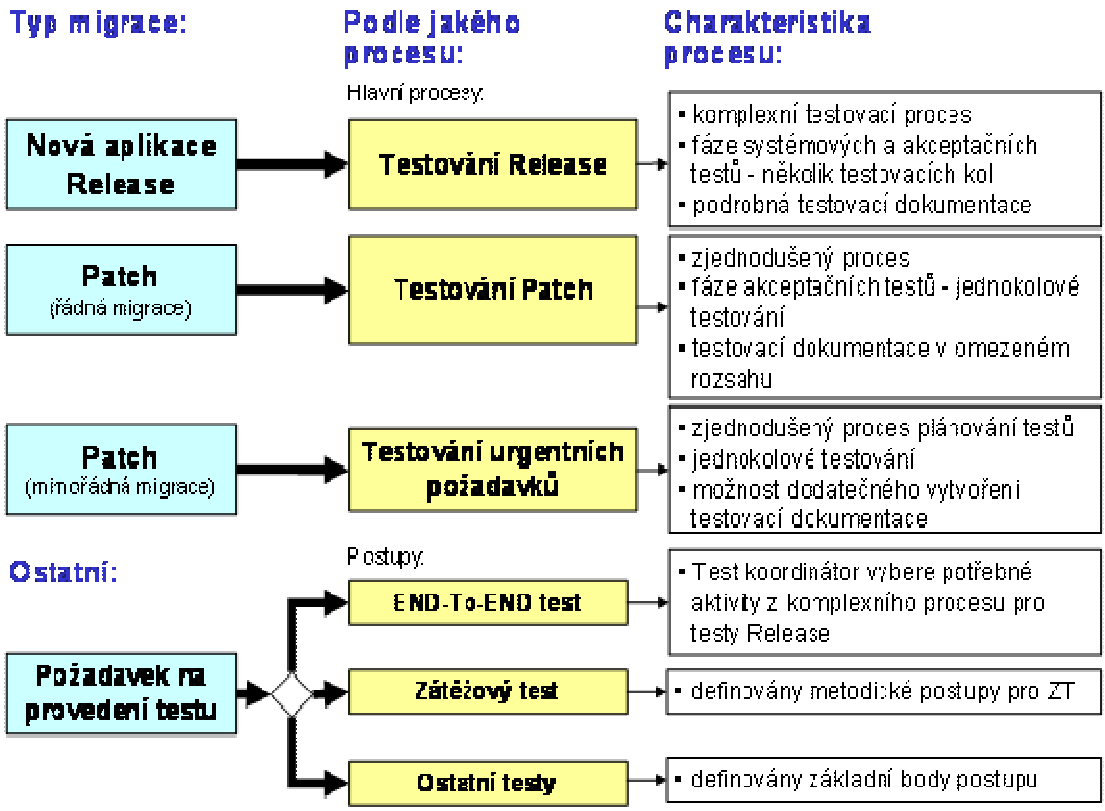
Plánování testů je důležitým začátkem testování. Jak uvádí Elfriede (2002) plánování by mělo být co možná nejdříve v životním cyklu software, protože pak může být testování úspěšnější. Plánování je součástí harmonogramu. Při plánování testů se definují požadavky testů, požadavky na zdroje- jak na lidské tak na technické. Také se v této fázi definují kritéria pro akceptaci. Po fázi plánování vzniká funkční desing, který je základem pro přípravu skriptů. „Plánování je většinou ovlivněno pravidly testování organizace, rozsahem testů, cíli,

riziky, omezeními. Čím pokročilejší je projektové plánování a plánování testů, o to více informací je k dispozici a je mohou být zahrnuty do plánu.“ (ISTQB).

Při plánování testů je také potřeba myslet na odhady kapacit (MD=man day). Zkušený test koordinátor většinou odhadne, kolik kapacit bude potřeba na vykonání testů. Existuje několik kritérií, podle kterých se lze orientovat. Takovým kritériem je například velikost a složitost produktu. Pokud se bude jednat o velký projekt nad 500MD je odhad jiný než u drobného vývoje na stávající aplikaci.

Nikdo ale nemá zaručený návod jak odhadnout pracnost na projektu. Velmi to záleží na zkušenosti a znalostech test koordinátorů. V praxi je běžné překračování kapacit, kdy se na počátku projektu chybně odhadne pracnost. Setkala jsem se s překročením kapacit i o 150%. Tato situace následně vrhá špatný stín na celý projektový tým (firmu), ale také snižuje výdělek firmy (a tím i projektové odměny pro členy projektového týmu). Zároveň se pak projekt hodnotí jako neúspěšný.

Níže je uveden příklad plánování testů dle rozsahu software.



Obrázek č. 3

Zdroj: Přednáška na téma Bezpečnost testování 2010

Plánování můžeme shrnout do několika kroků:

1) Zadání požadavků na testování
2) Příprava plánování testů na základě podkladů
3) Definice přístupů k testování
4) Shrnutí požadavků na testování
5) Plánování systémových testů
6) Plánování integračních testů
7) Plánování akceptačních testů

4.2 Příprava pro testování

4.2.1 Testovací skript

Testovací skripty neboli scénáře jsou nepostradatelnou pomůckou testera. Jejich vytvoření je součástí příprav pro testování. Je jim věnován čas a počítá se s jejich vytvořením v harmonogramu projektu. Podle testovacích skriptů se testuje ve všech kolech testování. Testovací skript by měl být připraven tak důkladně, aby podle něj mohl testovat kdokoli, i ten kdo o testovaném software nic neví. Do testovacího skriptu tester zanese vše podstatné z funkčního designu, aby se dobral co nejlepšího otestování software. Testovací skripty se mohou dělit na několik částí. Z mé několikaleté zkušenosti uvádím základní druhy testovacích skriptů a základní návod jak testovací skripty připravit. Základem pro vytvoření kvalitních skriptů je kvalitní funkční desing. Proto je velmi vhodné, aby při dokončení funkčního designu měli možnost členové testovacího týmu připomínkovat vzniklý dokument. Některé připomínky testerů k funkčnímu designu mohou být velmi přínosné a mohou často zamezit pozdějším nejasnostem. Pohled testera na aplikaci bývá rozdílný než pohled analytika. Analytik často neuvažuje jako koncový uživatel, a proto může přehlédnout nedostatky aplikace. Při testování softwarových aplikací se zaměřujeme několik podstatných věcí, a sice na vzhled, funkčnost a komunikaci s ostatními aplikacemi. Proto i testovací skripty musíme rozdělit podle oblastí, které testujeme. Pokud je software tvořeno několika obrazovkami, vytvoříme skripty na jednotlivé obrazovky.

Na jednotlivých obrazovkách testujeme jejich vzhled, tzn., věnujeme se zobrazení GUI. Dle funkčního designu tester ví, jak má obrazovka vypadat, jaké je rozložení polí na obrazovce a jaká tlačítka nebo obrázky jsou na obrazovce umístěny. Popíšeme tedy v testovacím scénáři jednotlivé součásti obrazovky. Věnujeme se i takovým detailům jako je velikost písma. Vše musí přesně odpovídat funkčnímu designu. Dále se tester zaměřuje na validace jednotlivých polí. Tyto validace jsou obvykle popsány ve funkčním designu. Pokud tomu tak není, je potřeba upozornit vedoucího funkčního týmu a žádat nápravu. Může se stát, že pro jednotlivá pole validace nejsou uvedeny a tudíž je na domluvě test koordinátora s vedoucím funkčního týmu a managerem projektu, zda se funkčnost bude testovat. V případě, že se netestuje, vzniká riziko chyb. Další podstatnou částí testování je otestování komunikace software s ostatními aplikacemi. Ovšem není podmínkou, že software musí být navázáno na jiné software. Pokud navázání je, je potřeba tuto funkčnost otestovat. Informace o provázanosti by měly být opět uvedeny ve funkční dokumentaci. Obecně lze

říct, jak kvalitní má tester testovací skripty, tak kvalitní je otestování software. Níže uvádím příklady testovacích scénářů.

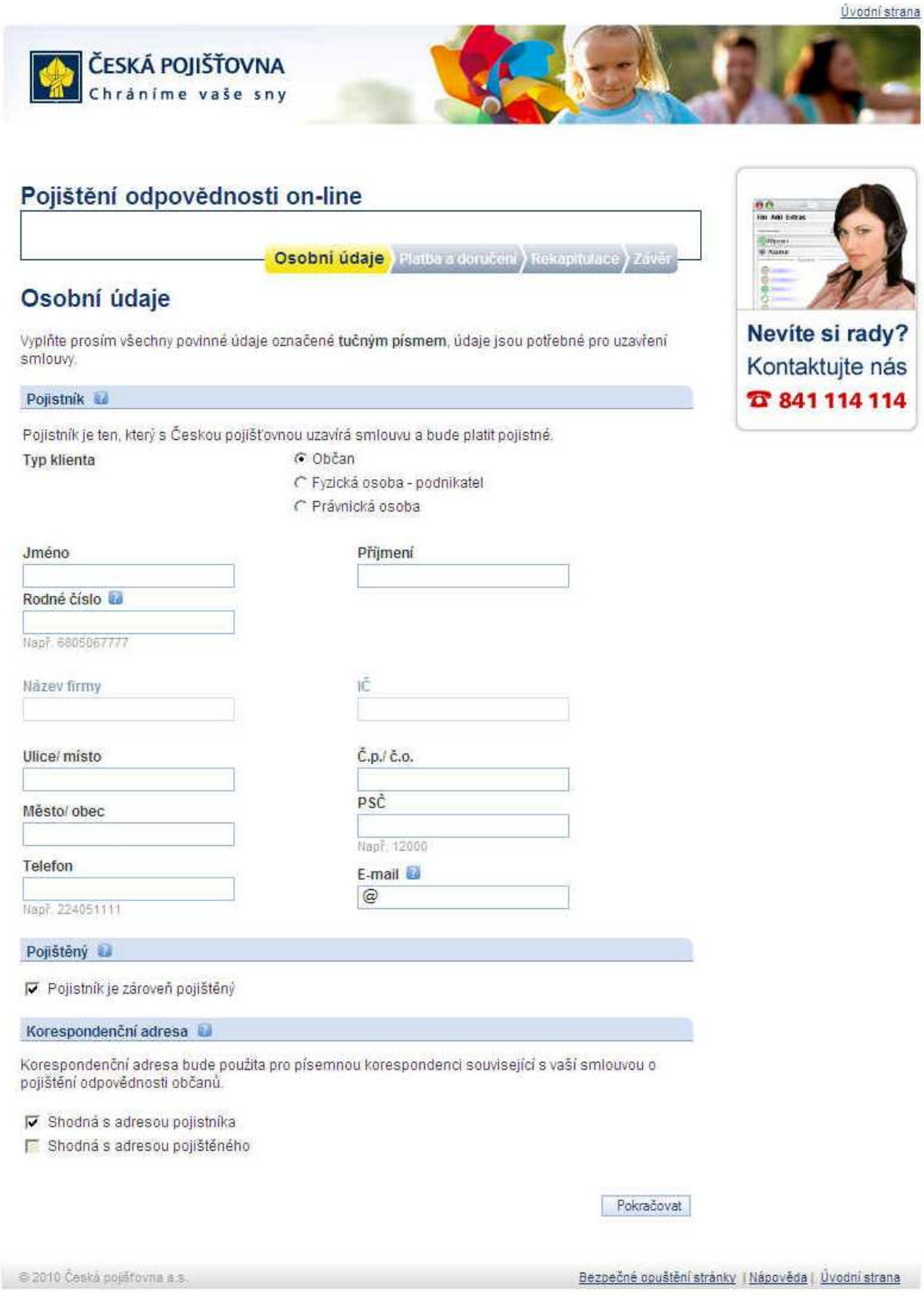
Jak tedy takový skript napsat a připravit? To je práce pro test analytika, nebo zkušeného testera.

Skripty pro vzhledy obrazovky neboli GUI.

Skripty pro kontrolu validace polí na obrazovce

Skripty pro kontrolu procesů

Níže jsou uvedeny dva příklady, jak by testovací skript měl vypadat. Jedná se o skript pro vzhled a validaci polí. Obrazovka, kterou jsem si pro ukázku vybrala, je jedna z obrazovek přímého pojištění po internetu České pojišťovny a.s., na jejímž vývoji jsem se podílela.



Obrázek č. 4

Zdroj: Česká pojišťovna a.s.

Níže je popsán skript na otestování vzhledu obrazovky.

Step Name	Description	Expected
Step 1	Zkontrolujte název obrazovky	Název obrazovky je "pojištění odpovědnosti on-line"
Step 2	Zkontroluje umístění a název obrázku	Obrázek je v pravém sloupci, umístěn nahoře, název obrázku je "box_kontakt.jpg"
Step 3	Zkontrolujte název záhlaví	Název záhlaví obrazovky je "Osobní údaje"
Step 4	Zkontrolujte staticky text pod záhlavím Osobní údaje	Vyplňte prosím všechny povinné údaje označené tučným písmem, údaje jsou potřebné pro uzavření smlouvy.
Step 5	Zkontrolujte sekci název sekce "Pojistník"	Název sekce je "Pojistník"
Step 6	Zkontrolujte statický text v sekci Pojistník	Pod nadpisem Pojistník je statický text : Pojistník je ten, který s Českou pojišťovnou uzavírá smlouvu a bude platit pojistné.

Step 7	Zkontrolujte obsah sekce "Pojistník"	Sekce obsahuje radionbuttony "Typ klienta"(Občan, Fyzická osoba - podnikatel, Právnická osoba), textové pole "Jméno", "Příjmení", "Rodné číslo", "Název firmy", "IČ", "Ulice/místo", "Č. p./č. o.", "Obec", "PSČ", "Telefon" a "Email"
Step 8	Zkontrolujte podsekci "Typ klienta"	V podsekci "Typ klienta", jsou tři radionbuttony , řazeny pod sebou, názvy zarovnané vpravo. Hodnoty radionbuttonů jsou: "Občan", "Fyzická osoba - podnikatel", "Právnická osoba"
Step 9	Zkontrolujte textová vstupní pole v sekci "Pojistník"	Pole mají názvy: "Jméno", "Příjmení", "Rodné číslo", "Název firmy", "IČ", "Ulice/ místo", "Č. p./č. o.", "Město/obec", "PSČ", "Telefon" a "Email",
Step 10	Zkontrolujte sekci "Pojištěný"	Název sekce je "Pojištěný"

Step 11	Zkontrolujte obsah sekce "Pojištěný"	Sekce obsahuje checkbox s labelem "Pojistník je zároveň pojištěný", který je umístěn pod nadpisem sekce.
Step 12	Zkontrolujte obsah sekce "Pojištěný" Zkontrolujte textová vstupní pole v sekci "Pojištěný"	Pokud není zaškrtnut checkbox Pojistník je zároveň pojištěný, jsou aktivní textová pole :"Jméno", "Příjmení", "Rodné číslo", "Ulice/ místo", "Č. p./Č. o.", "Město/obec", "PSČ",
Step 13	Zkontrolujte sekci "Korespondenční adresa"	Název sekce je "Korespondenční adresa"
Step 14	Zkontrolujte text v sekci Korespondenční adresa.	Pod nadpisem Korespondenční adresa je umístěn statický text: Korespondenční adresa bude použita pro písemnou korespondenci související s vaší smlouvou o pojištění odpovědnosti občanů.

Step 15	Zkontrolujte obsah sekce "Korespondenční adresa"	Sekce obsahuje dva checkboxy s labely "Shodná s adresou pojistníka" a "Shodná s adresou pojištěného", které jsou umístěny pod nadpisem sekce.
Step 16	Zkontrolujte obsah sekce "Korespondenční adresa" Zkontrolujte textová vstupní pole v sekci "Korespondenční adresa"	Pole mají názvy: "Ulice/ místo", "Č. p./č. o.", "Město/obec", "PSČ",
Step 17	Zkontrolujte tlačítko "Pokračovat"	Tlačítko má label "Pokračovat", je umístěno vpravo dole na stránce, je aktivní, po stisku přejde aplikace na stránku IPMO-201 platba a doručení

Ke stejné obrazovce uvádím pro ukázku skript na test validací polí:

Step Name	Description	Expected
Step 1	Zkontrolujte radionbutton Typ klienta	Jako výchozí hodnota je nastaven "Občan", vždy lze vybrat jen jednu z možností (občan, Fyzická osoba-podnikatel, právnická osoba)
Step 2	Vyberte hodnotu "Typ klienta" "Občan". Zkontrolujte přístupnost textových polí	Pole "Jméno" je aktivní Pole "Příjmení" je aktivní Pole "Rodné číslo" je aktivní Pole "Název firmy" není aktivní Pole "IČ" není aktivní Pole "Ulice" je aktivní Pole "Číslo popisné/evidenční" je aktivní Pole "Obec" je aktivní Pole "PSČ" je aktivní Pole "Telefon" je aktivní Pole "E-mail" je aktivní

Step 3	Vyberte hodnotu "Typ klienta" "Fyzická osoba podnikatel". Zkontrolujte přístupnost textových polí	Pole "Jméno" není aktivní Pole "Příjmení" není aktivní Pole "Rodné číslo" není aktivní Pole "Název firmy" je aktivní Pole "IČ" je aktivní Pole "Ulice" je aktivní Pole "Číslo popisné/evidenční" je aktivní Pole "Obec" je aktivní Pole "PSČ" je aktivní Pole "Telefon" je aktivní Pole "E-mail" je aktivní
Step 4	Vyberte hodnotu "Typ klienta" "Právnická osoba". Zkontrolujte přístupnost textových polí	Pole "Jméno" není aktivní Pole "Příjmení" není aktivní Pole "Rodné číslo" není aktivní Pole "Název firmy" je aktivní Pole "IČ" je aktivní Pole "Ulice" je aktivní Pole "Číslo popisné/evidenční" je aktivní Pole "Obec" je aktivní Pole "PSČ" je aktivní Pole "Telefon" je aktivní Pole "E-mail" je aktivní

Step 5	Zkontrolujte validaci pole "Jméno" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků (max. 60 znaky, číslice) 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 a 1 je zobrazena chybová hláška, maximální počet vložených znaků je 60
Step 6	Zkontrolujte validaci pole "Příjmení" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 a 1 je zobrazena chybová hláška, maximální počet vložených znaků je 60
Step 8	Zkontrolujte validaci pole Rodné číslo: 1. Pole Rodné číslo vyplňte tak, že	V horní části obrazovky se zobrazí chybová zpráva Pole se zvýrazní.

	bude obsahovat nenumerický znak.	
Step 9	Ověření hodnoty pole Rodné číslo: 1. Pole Rodné číslo vyplňte tak, že nebude splňovat následující podmínky: A: 9-místné číslo musí začínat dvoučíslím do 53, B: 10-místné číslo musí začínat šestičíslím, menším než RRMMDD podle dnešního data a musí být dělitelné 11 beze zbytku. C: zadejte RČ z budoucnosti	V horní části obrazovky se zobrazí chybová zpráva. Pole se zvýrazní.
Step 10	Zkontrolujte validaci pole Rodné číslo: 1. Vložte do pole maximální počet znaků	Do pole lze zadat 10 znaků
Step 11	Zkontrolujte validaci pole "Název firmy" 1. Vložte do pole maximální počet znaků 2. Ponechte pole prázdné	U bodu č. 2 je zobrazena chybová hláška, maximální počet vložených znaků je 60

Step 12	Zkontrolujte validaci pole "IČ" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 je zobrazena chybová hláška, maximální počet vložených znaků je 8
Step 13	Zkontrolujte validaci pole "Ulice/místo" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70

Step 14	Zkontrolujte validaci pole "Č. p./ Č. o." 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodů č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 15
Step 15	Zkontrolujte validaci pole "Město/Obec" 1. Vložte do pole znaky "AAA - aa / aaa\aaa" 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70

Step 16	Zkontrolujte validaci pole "PSČ" 1. Vložte do pole nepovolené znaky (=ˇ˘::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "41145" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a4" 9. Vložte do pole znaky "-12 A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 je zobrazena chybová hláška, maximální počet vložených znaků je 5
Step 17	Zkontrolujte validaci pole "Telefon" 1. Vložte do pole znaky "+421987654321" 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12 A" 10. Ponechte pole prázdné	- Minimální délka pole 9, maximální délka pole 12 (znaky, číslice) - Tvar pole kontrolován vůči masce. {telefonní číslo může obsahovat pouze číslice a nesmí začínat číslicí (číslicemi) 0} - Povinný údaj - Zadejte telefonní číslo (Patch PPO5.3)

Step 18	Zkontrolujte validaci pole "Email" 1. Vložte do pole znaky "<<<Vaše emailová adresa>>>" 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "pokus@pokus" 4. Vložte do pole znaky "pokus@/cz" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12 A" 10. Ponechte pole prázdné	U všech bodů mimo č. 1 je zobrazena chybová hláška, maximální počet vložených znaků je 50
Step 19	Zkontrolujte checkbox "Pojistník je zároveň pojištěný"	Checkbox je defaulně zaškrtnut
Step 20	Odškrtněte checkbox "Pojistník je zároveň pojištěný"	Zobrazí se skrytá textová pole v sekci pojištěný Jméno, Příjmení, Titul, Rodné číslo, Ulice, Číslo popisné/evidenční, Obec, PSČ
Step 21	Vyberte typ klienta "Občan" a zkontrolujte checkbox "Pojistník je zároveň pojištěný"	Checkbox je zaškrtnut, pole v sekci pojištěný jsou skryta
Step 22	Vyberte typ klienta "Fyzická osoba - podnikatel" a zkontrolujte checkbox "Pojistník je zároveň pojištěný"	Checkbox je odškrtnut a disablován, pole v sekci pojištěný jsou viditelná a enablovaná

Step 23	Vyberte typ klienta "Právnická osoba" a zkontrolujte checkbox "Pojistník je zároveň pojištěný"	Checkbox je odškrtnut a disablován, pole v sekci pojištěný jsou viditelná a enablovaná
Step 24	Zkontrolujte validaci pole "Jméno" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 a 1 je zobrazena chybová hláška, maximální počet vložených znaků je 60
Step 25	Zkontrolujte validaci pole "Příjmení" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A"	U všech bodů mimo č. 6 je zobrazena chybová hláška, maximální počet vložených znaků je 60

	10. Ponechte pole prázdné	
Step 27	Zkontrolujte validaci pole Rodné číslo: 1. Pole Rodné číslo vyplňte tak, že bude obsahovat nenumerický znak.	V horní části obrazovky se zobrazí chybová zpráva Pole se zvýrazní.
Step 28	Ověření hodnoty pole Rodné číslo: 1. Pole Rodné číslo vyplňte tak, že nebude splňovat následující podmínky: A: 9-místné číslo musí začínat dvoučíslím do 53, B: 10-místní číslo musí začínat šestičíslím menším než RRMMDD podle dnešního data a musí být dělitelné 11 beze zbytku. C: zadejte RČ z budoucnosti	V horní části obrazovky se zobrazí chybová zpráva. Pole se zvýrazní.
Step 29	Zkontrolujte validaci pole Rodné číslo: 1. Vložte do pole maximální počet znaků	Do pole lze zadat 10 znaků

Step 30	Zkontrolujte validaci pole "Ulice / místo" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70
Step 31	Zkontrolujte validaci pole "Č. p./ Č. o." 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 15

Step 32	Zkontrolujte validaci pole "Město/obec" 1. Vložte do pole znaky "AAA - aa / aaa\aaa" 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70
Step 33	Zkontrolujte validaci pole "PSČ" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "41145" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a3" 9. Vložte do pole znaky "-12 A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 je zobrazena chybová hláška, maximální počet vložených znaků je 5

Step 34	Zkontrolujte checkbox "Shodná s adresou pojistníka"	Pole je defaultně zaškrtnuto, pokud je checkbox zaškrtnutý, jsou na obrazovce skryta pole korespondenční adresy. Pokud je pojistník různý od pojištěného a je označeno, že korespondenční adresa je shodná s adresou pojištěného, checkbox se automaticky odškrtne
Step 35	Zkontrolujte checkbox "Shodná s adresou pojištěného"	Checkbox je zaškrtnut, pokud není v sekci "Pojištěný" zaškrtnut checkbox "Shodný s pojistníkem". Jinak je disablován a nezaškrtnut. Pokud je zaškrtnut checkbox "Pojistník je zároveň pojištěný" je checkbox neaktivní a nezaškrtnutý. Pokud je checkbox zaškrtnutý, jsou skrytá pole korespondenční adresy. V případě, že je pojistník různý od pojištěného a že je korespondenční adresa shodná s adresou pojistníka se checkbox automaticky odškrtne.

Step 36	Odškrtněte checkboxy "Shodná s adresou pojistníka" a "Shodná s adresou pojištěného"	Zobrazila se pole pro zadání korespondenční adresy Ulice, Číslo popisné/evidenční, Město/obec a PSČ
Step 37	Zkontrolujte validaci pole "Ulice / místo" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70

Step 38	Zkontrolujte validaci pole "Č. p./ Č. o." 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "aaa" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "12345678" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 15
Step 39	Zkontrolujte validaci pole "Obec" 1. Vložte do pole znaky "AAA - aa / aaa\aaa" 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "AAA" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12A" 10. Ponechte pole prázdné	U bodu č. 10 je zobrazena chybová hláška, maximální počet vložených znaků je 70

Step 40	Zkontrolujte validaci pole "PSČ" 1. Vložte do pole nepovolené znaky (=´ˇ::*+]) 2. Vložte do pole maximální počet znaků 3. Vložte do pole znaky "1234" 4. Vložte do pole znaky "12,34" 5. Vložte do pole znaky "-12" 6. Vložte do pole znaky "41145" 7. Vložte do pole znaky "1234A" 8. Vložte do pole znaky "12,a34" 9. Vložte do pole znaky "-12 A" 10. Ponechte pole prázdné	U všech bodů mimo č. 6 a 1 je zobrazena chybová hláška, maximální počet vložených znaků je 5

4.2.2 Testovací sady

Přípravou skriptů ale není práce zcela hotová. Testovací scénáře neboli skripty se musí seřadit do testovacích sad. Testovací sady se skládají z jednotlivých skriptů. Testeři si tak před zahájením testování mohou připravit skripty do jednotlivých kol testování. Sady se mohou řadit například podle různých kritérií. Například se sady dají dělit podle časové nebo logické návaznosti, nebo na základě testovacího prostředí. Například pokud máme aplikaci s několika obrazovkami, můžeme si vytvořit sady podle jednotlivých obrazovek a do těchto sad zahrnout jednotlivé skripty z různých oblastí.

4.3 Samotné testování

Testování se zahajuje dle harmonogramu a má několik fází. Níže popsané fáze testování vychází z mých zkušeností a mohou se tedy lišit v různých firmách a na různých projektech. Testování by mělo začít u programátorů, kteří provádějí Unit testy. Jedná se o kontrolu kódu. Tyto testy trvají týden či dva. Doba se liší podle velikosti aplikace. Většina programátorů odhalí chyby v kódu, ale neodhalí chyby v návaznosti na ostatní aplikace, neboť v této době ještě není aplikace navázána na další aplikace.

Následují kola systémových testů, které se také označují jako ST. Obvykle jsou čtyři a každé kolo trvá jeden týden. Velmi vhodné je testování v nultém kole systémových testů. Předejde se tak velkým chybám a chybám ve vzhledu aplikace. V dalších kolech systémových testů se pak testovací tým může soustředit hlavně na procesní testy. Testy v jednotlivých systémových kolech se opakují. Na konci systémového testování je dobré regresně aplikaci otestovat. Regresní testy jsou také velmi vhodné při drobném zásahu do aplikace. Velmi často se stává, že po implementaci drobné změny se zanese chyba do části aplikace, kde by ji nikdo nehledal. Proto je na testerech, aby pečlivě zkontrolovali celou aplikaci.

Po skončení systémových testů následují testy akceptační, zkratkou UAT. Akceptační testy provádí zákazník, nejčastěji určené osoby z businessu. V tomto období dělají testeři podporu a dá se říct i prostředníka mezi zákazníkem a programátorem. Ne všechny chyby nalezené zákazníkem jsou chyby. Velmi často se setkávám s chybami způsobenými neznalostí aplikace. Časté jsou také chyby typu „nefunguje aplikace“ nebo „aplikace padá“. Tyto chyby musí testeři vždy retestovat a určit zda se o chybu skutečně jedná. Po skončení akceptačních testů se aplikace migruje z testovacího prostředí do ostrého provozu. Migrace se provádí v noci nebo o víkendu, kdy je zaručeno, že nebude využívaná klienty. Přítomnost testera na migraci je bezesporu důležitá. Po nasazení aplikace do ostrého provozu je tester prvním uživatelem, který aplikaci vyzkouší. Při postmigračních testech je poslední možnost odhalení chyb.

4.4 Vyhodnocování výsledků testů

Vyhodnocování testů probíhá jak v jednotlivých kolech, tak na závěr celého testování a předání aplikace zákazníkovi.

Při hodnocení v jednotlivých kolech se hodnotí jednotlivé skripty a jednotlivé kroky ve skriptech. Ve chvíli kdy tester nalezne v jednom kroku chybu, zadá defekt. To ovšem neznamená, že nedotestuje celý skript. Může se totiž stát, že i při testování ostatních kroků nalezne další chyby. Kdyby v testu nepokračoval, mohly by se tyto chyby přenést i do dalších kol testování. Ron Patton (2002) uvádí, že většina selhání je docela malých a některé z nich jsou dokonce tak bezvýznamné, že není úplně vždycky jasné, jestli se vůbec jedná o skutečné selhání. Ale pokud je testovaný software určen zákazníkovi, tak nesmíme dopustit, aby naše aplikace obsahovala jen jednu maličkou chybu. V takovém případě je vhodné konzultovat nalezenou chybu s analytikem a následně s programátorem. Z praxe mám zkušenost,

že i drobná chybička může způsobit kolaps celého systému. Také se často stává, že tester zjistí nesrovnalost v aplikaci nebo navrhne drobnou změnu, která se ovšem nehodnotí jako chyba, ale jako change (drobná úprava, která se realizuje po skončení projektu).

I defekt má své životní fáze, ty se označují jako „bug lifecykle“. Jak se vlastně defekt zadává? Defekt se zadává do defekt-manageru, který je zvolen na počátku projektu. Je také dobré stanovit si pravidla pro zadávání defektů a jejich řešení. Tím se zlepší, zkrátí a zefektivní komunikace mezi testovacím týmem a technickým týmem. Zároveň tím odpadne nutnost přítomnosti testera, který defekt zakládal, při řešení defektu. Přesné popsání chyby není důležité jen pro technický tým, ale také pro další testery, kteří mohou zadanou chybu retestovat – tedy znovu otestovat.

Do defektu se popisují chyby, které tester našel. Při zadávání defektu je velmi důležité přesně popsat, jak k chybě tester přišel, jaká zadával data, jak se aplikace zachovala a jak se zachovat měla. Musíme mít na paměti, že po testerovi dostane defekt k řešení programátor, který zná kód, ale nemusí znát přesné chování aplikace. Proto je velmi důležité uvést některé základní údaje. Těmi může být datum a čas provedení testu, popis činností, při kterých došlo k chybě, očekávaný stav a skutečný stav. Nová chyba je vždy ve stavu NEW, čímž začíná její životní cyklus.

Důležitou součástí defektu je také závažnost chyby. Závažnost chyb se klasicky dělí na kritické chyby, vysoké chyby, střední a malé chyby. Každý tester v týmu by měl vědět, k jaké chybě jakou závažnost dát. Obecně se jako kritická chyba označuje chyba aplikace, která způsobuje její pád. Příkladem takové chyby je chybná validace, která je špatně naprogramovaná a tudíž vždy dojde k pádu aplikace. Za vysoké chyby považujeme chyby, které nám brání v testování, ale nezpůsobují pád aplikace. Jsou to chyby, které nám brání v přechodu mezi jednotlivými obrazovkami, nebo chyby kdy nelze vyplnit nějaké pole, nefungují tlačítka a podobně. Tedy jsou to chyby poměrně závažné, ale ne do takové míry, aby kriticky ohrožovaly fungování aplikace. Dále máme chyby střední závažnosti, které jsou specifické například pro chybné validace. Malé chyby jsou chyby týkající se chyb v grafickém zobrazení, chyby v textaci apod. Náklady na opravu chyb jsou značné a s každým kolem testování se zvyšují, proto je velmi důležité většinu velkých chyb najít velmi brzo.

Po stavu NEW se chyba dostává do stavu ASSIGNED. V tomto stavu se chyba dostává k programátorovi, který jí vyřeší. Pokud programátor zjistí, že se nejedná o chybu, dá chybu do stavu REJECTED. Tento stav se nestává často, ale je důležité o něm vědět. Může nastat třeba ve chvíli, kdy testerovi zlobí počítač nebo internetový prohlížeč a zdá se mu, že chyba je v aplikaci. Pokud se o chybu jedná, technik ji opraví a dá do stavu RESOLVED. Ze stavu RESOLVED jí odpovědná osoba předá k retestování ve stavu FIXED.

Po opravě chyby technikem se testerovi vrátí defekt k retestování, ve stavu FIXED. Po retestování se defekt uzavře v případě, že byla chyba správně opravena. Tester by neměl při uzavírání defektu zapomenout napsat verzi software, ve které defekt uzavírá. Ač se to může jevit jako nepodstatné, je velmi důležité vědět, v jaké verzi byl defekt uzavřen. Uzavřená chyba je ve stavu CLOSED. Pokud tester zjistí, že chyba není opravena, vrátí defekt technikovi, který chybu opravoval. Takovou chybu dá tester do stavu REOPEN. Vždy je důležité napsat poznámku k defektu, ať už je opraven správně nebo je potřeba chybu opravit znovu. Zároveň by poznámku měl vyplnit i technik.

Retestování defektů je velmi důležité a bez retestování defektů nemůže být ukončeno testovací kolo.

5 Testovací nástroje pro funkční testování software

Testovací nástroje jsou důležitým pomocníkem při celém procesu testování. Používají je test koordinátoři, test analytici, testeři, analytici i vývojáři. V dobrém testovacím nástroji nalezneme několik modulů. Jedním z nejvyužívanějších je modul pro zadávání a správu defectů. Oblíbeným modulem je i modul na správu a tvorbu testovacích scénářů a v neposlední řadě modul na samotnou realizaci testů. Čím více má testovací nástroj modulů, tím více je žádaný a tudíž i jeho pořizovací cena vyšší. Na trhu však existují i testovací nástroje, jejich pořízení stojí maximálně pár minut námahy na vyhledání a instalaci. Free nástroje většinou nebývají favority u velkých firem a profesionálů, ale i tak se mezi nimi najdou zajímavé nástroje, které stojí za povšimnutí. V této kapitole bych ráda popsala některé free testovací nástroje a také jeden komerčně dostupný a asi i nejvyužívanější nástroj mezi profesionály.

5.1 Placené nástroje

Mercury TestDirector for Quality Center

Aplikace Mercury TestDirector for Quality Center je asi nejznámější a nejvyužívanější nástroj pro testování.

Nástroj je určen pro správu a řízení všech fází procesu testování (správa požadavků, plánování testů, provádění testů, sledování řešení chyb, sledování pokrytí požadavků na systém). Podporuje současnou správu ručních i automatizovaných testů. Je přístupný přes webový prohlížeč a lze jej propojit s nástroji pro automatizované testování.

Ruční test je tvořen dílčími kroky s podrobným popisem jednotlivých činností a očekávaných reakcí aplikace. Automatizovaný test je tvořen skriptem, který lze spouštět pomocí nástroje pro automatizované testování. Umožňuje spouštět automatizované testy i na vzdálených počítačích. Testy nejsou prováděny jednotlivě, ale v sadách. Testovací sady jsou skupiny testů s určeným pořadím, resp. závislostí (počáteční podmínkou) pro provedení konkrétního testu.

Umožňuje vytvářet testovací plán ve vazbě na požadavky na testování, poskytuje prostředky pro jeho jednoduché a rychlé strukturování a sledování, jak a s jakými výsledky

testování probíhá. V repository jsou uchovávány výsledky všech předchozích běhů testů. V případě automatizovaných testů lze naplánovat i dávkové spouštění mimo pracovní dobu.

Umožňuje vytvoření uživatelských sestav ze všech etap procesu testování. K tomuto účelu vyvinutý Document Generator exportuje do formátu MS Word. Tabulky a grafy lze exportovat také do MS Excel a HTML.

Testování je týmová práce a nástroj zajišťuje sdílení dat. Veškeré informace o jednotlivých testech, včetně údajů o provedení, nalezených chybách v testované aplikaci a o stavu jejich řešení jsou uchovávány ve společné repository. Přístupy uživatelů jsou zabezpečeny heslem, oprávnění jsou definována na úrovni uživatelských rolí. Klientská část nástroje pracuje v prostředí MS Windows z prohlížeče MS Internet Explorer, jako databázi lze využít Oracle, MS SQL Server, Sybase nebo MS Access na platformách Unix či MS Windows.
Pro usnadnění přenosu dat z jiných etap životního cyklu software slouží rozhraní Open Test Architecture (OTA). Toto rozhraní umožňuje vazbu na nástroje pro vývoj IS jiných výrobců. V praxi je využíváno propojení se specializovanými nástroji pro správu požadavků nebo s nástroji pro evidenci chyb. Na bázi OTA lze začlenit do své správy TestDirectoru dokonce i testovací nástroje jiných výrobců, včetně programů vyvinutých vlastními silami. Mezi nástroje, pro které je propojení připraveno patří: Rational RequisitePro, Rational ClearQuest, Rational ClearCase 2002 a 2003, Merant PVCS Tracker, Microsoft Visual SourceSafe server 6.0c a 6.0d, Microsoft Word a Excel.

5.2 Free nástroje

Na počátku jsem stanovila kritéria pro výběr vhodného free testovacího nástroje. Hlavním kritériem byla free dostupnost. Nedefinovala jsme si však žádná kritéria týkající se služeb, které má nástroj obsahovat. Bylo důležité najít vyhovující nástroj s jakoukoli funkcí, která je potřebná při testování software. Dle tohoto kritéria jsem provedla rešerši, po jejímž vypracování jsem získala přehled free testovacích nástrojů. Po vypracování rešerše a podrobném studiu jednotlivých nástrojů, došlo k selekci (do selekce byla zahrnuta potřeba operačního systému a podrobnější informace o nástroji), na jejímž základě bylo vybráno 11 nástrojů. Důvodem vyřazení většiny nástrojů, bylo nutné zakoupení licence, i když se na první pohled nástroj představoval jako freeware. Následovalo reálné testování nástrojů, přičemž u většiny byly zjištěny značné potíže při instalaci. Z jedenácti nástrojů jsem

nakonec úspěšně otestovala pět. Jejich podrobnější popis je níže. Vybraný nástroj by měl mít takové vlastnosti, aby bylo možno ho nabídnout klientovi. Tato nabídka by měla probíhat formou prezentace nástroje. Ideálním výsledkem je nalezení jednoho nástroje, který by byl uživatelsky přívětivý, a po funkční stránce bezchybný.

5.2.1 Systém hodnocení

Prvním krokem, který jsem se rozhodla udělat, bylo stanovit si systém hodnocení., vybrala jsme si tyto hodnotící prvky:

Administrátorské rozhraní – Hlavním kritériem je jednoduchost. Nástroj by měl umožňovat měnit nastavení. Uživatel nemusí mít znalosti DB. Jednoduché změny lze realizovat přes front end .

Grafická část – Nástroj musí být jednoduchý, jasný, musí mít intuitivní orientaci, příjemný vzhled a na první dojem by měl být přehledný.

Funkční část – Nástroj musí bezchybně pracovat dle popisu. Měl by mít možnost rozšíření o pluginy cílem není detailní rozpracování funkčnosti nástroje. Detailní popis jednotlivého nástroje by mohla být další fáze RP, kde by byl vypracován uživatelský manuál.

Použitelnost –. Nástroj by měl sloužit pro testovací tým a technický tým v českém prostředí a pro malé firmy, které si placené nástroje nemohou z finančních důvodů dovolit.

V každé z těchto 4 částí bylo možné udělit maximálně po 25 bodech. Celkový součet bodů je zobrazen na konci každého popisu testovacího nástroje. Maximální možný počet dosažených bodů je 100.

Hodnotící tabulka jednotlivých částí:

Počet bodů v jednotlivých částech	Hodnocení jednotlivé části
25 - 20 bodů	Vynikající bez výhrad
19 - 15 bodů	Výborné s drobným nedostatkem
14 - 10 bodů	Dobré s nedostatky
9 – 5 bodů	Dostačující
5 – 0 bodů	Nevyhovující

Hodnotící tabulka součtu bodů:

Počet celkových bodů	Vhodnost nástroje
100 – 70 bodů	Vynikající
69 – 41 bodů	Dostatečný
40 – 0 bodů	Nevyhovující

5.2.2 Forma popisu jednotlivých nástrojů

Nástroje jsou hodnoceny jednotlivě ve třech základních bodech.

V první části je uveden stručný popis nástroje, nároky na OS a popsány jsou role, kterými nástroj disponuje.

Druhou část tvoří ukázky obrazovek s popisem, přičemž je kladen důraz na uvedení všech funkcí, které je nástroj schopen poskytnout. Každý nástroj však funguje jinak a proto není možné detailní srovnání jednotlivých modulů.

V posl ední, třetí části je hodnocení nástroje. Hodnocení je nejprve slovní a bodové. Následuje závěr a doporučení, kde byl kladen důraz na plusy a mínusy jednotlivých nástrojů.

5.2.3 Jednotlivé otestované nástroje

5.2.3.1 TRAC

Jedním z výsledně vybraných nástrojů je TRAC. Tento nástroj funguje jako webová aplikace, která je napojena na vlastní databázi. Samostatný nástroj slouží pouze pro zadávání, evidenci a následné řešení chyb. Nástroj umožňuje vytváření reportů a grafů. To je jednou z výhod nástroje, dají se tak připravit výstupy stavu defektů. Nástroj je limitován Pythonem a Apache serverem, v té míře, na kolik je architektura založená na těchto dvou technologiích. Python je možné nasadit jak na Apache server, tak i na jiné. Je možné ho zpřístupnit i přes internet. Testovaná byla aplikace v Internet Explorer 7 a FireFox2. Disponuje vlastní databází SQL lite, ale podporuje MySQL, PostgreSQL. Vyžadovaným OS je WIN32, UNIX, MAC (v případě, že není použitý APACHE server).

Použité pluginy:

Create new divisions in roadmap and milestone progress bars, Clients support for Trac Tickets, AccountManagerPlugin, Batch modification of tickets, TestCaseManagement a A plugin that creates a feed of all changes to all of a report's tickets.

Samotný nástroj rozlišuje několik rolí:

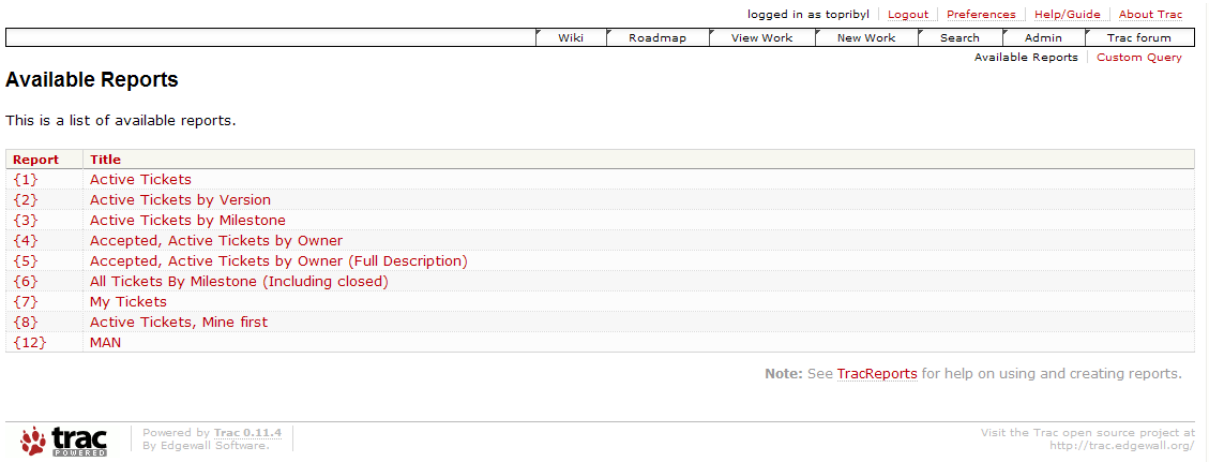
Tester, TestKoordinátor, Manager

Všechny role jsem částečně prozkoumala a konstatovala, že chybí ještě role Solvera. Také přidání uživatelů a jejich rolí probíhá mimo prostředí, přímo v databázi, což je trochu nevýhodné.

Jednoduchá ukázka prostředí a jednoduchých úkonů

Záložka View Work

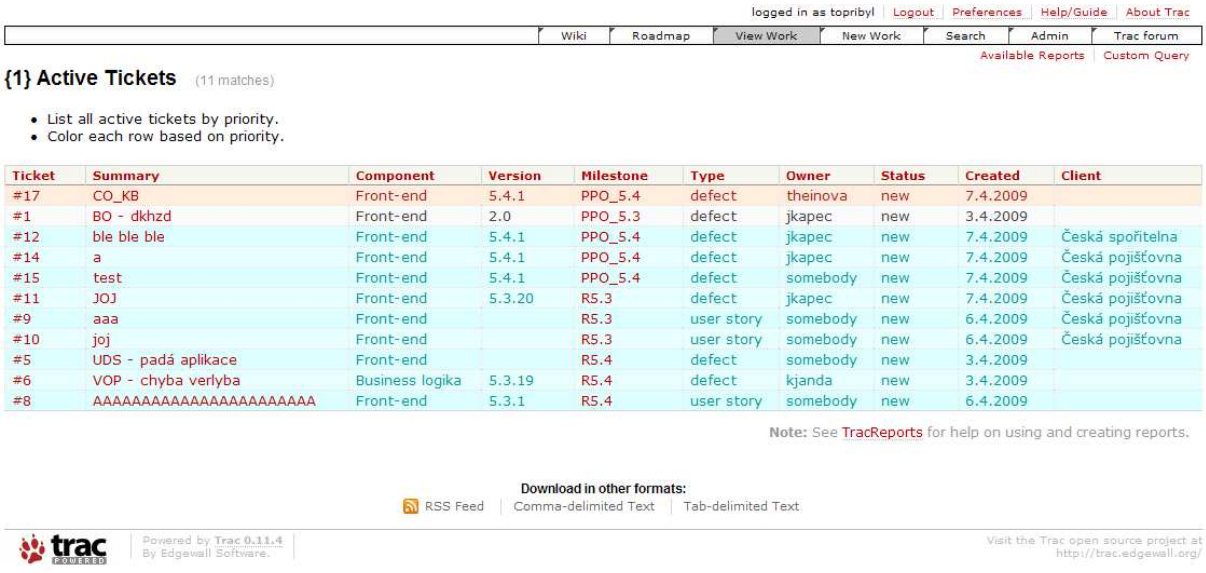
Po otevření záložky se zobrazí stránka s nadpisem **Available Reports**



Obrázek č. 5

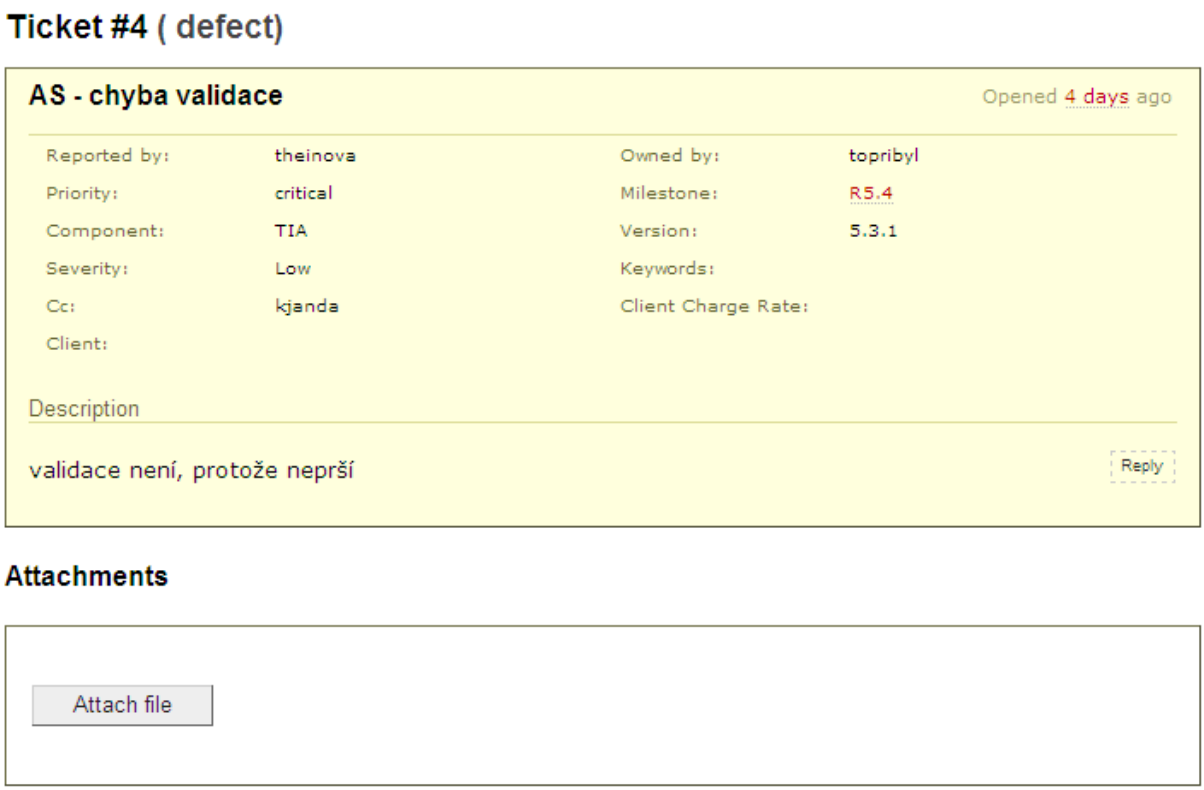
Active Tickets

Zobrazují seznam všech otevřených defektů.

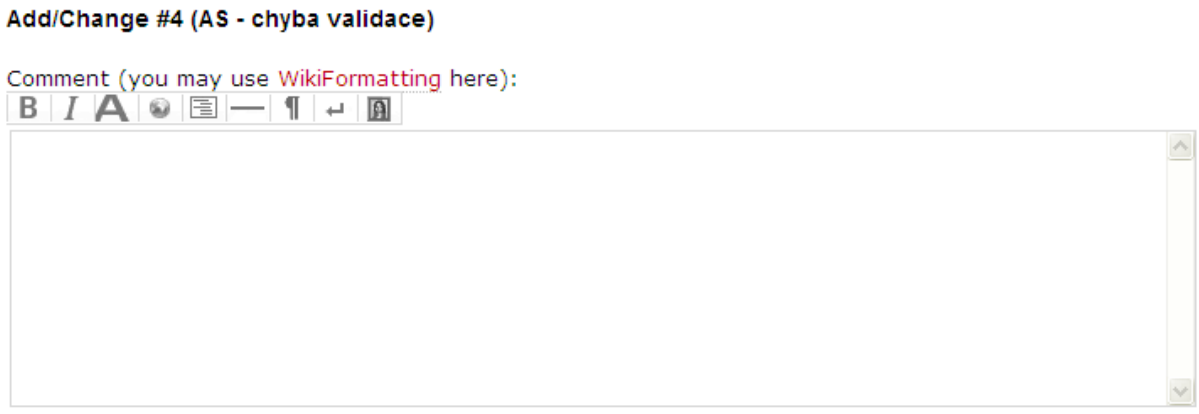


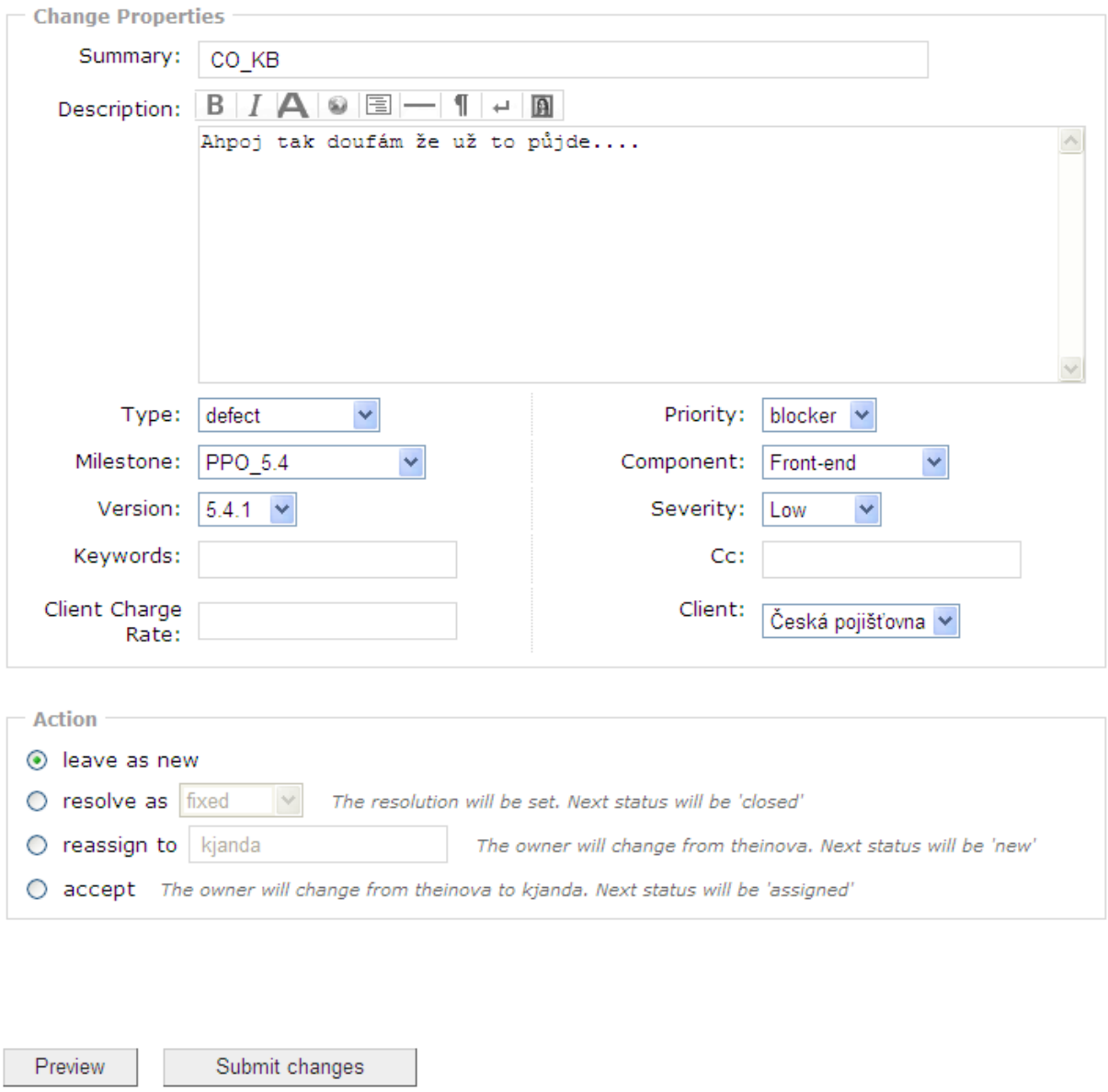
Obrázek č. 6

U každého defektu se dá přejít na detail:



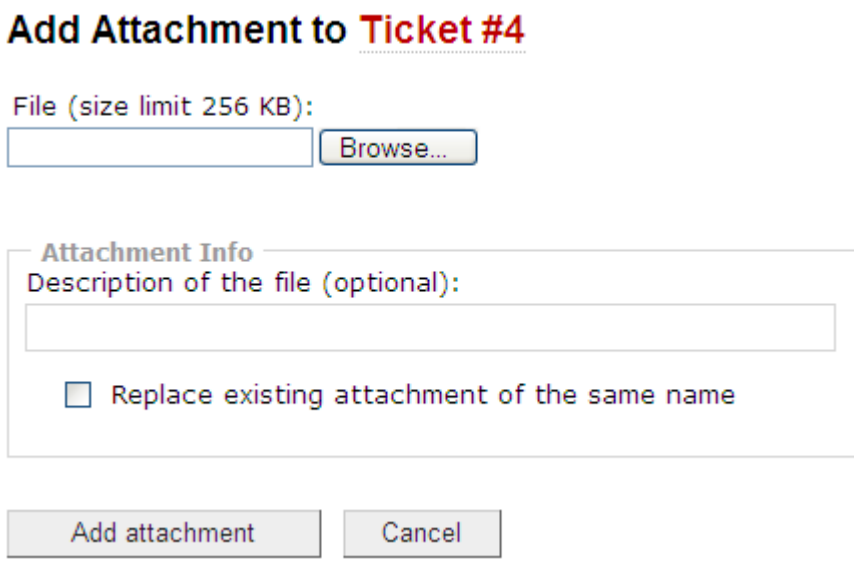
Obrázek č. 7



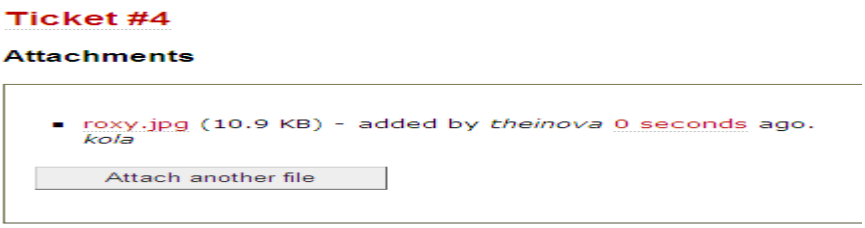

Change Properties
Summary:
CO_KB
Description:
Ahpoj tak doufám že už to půjde....
Type:
defect
Priority:
blocker
Milestone:
PPO_5.4
Component:
Front-end
Version:
5.4.1
Severity:
Low
Keywords:
Cc:
Client Charge Rate:
Client:
Česká pojišťovna
Action
leave as new
resolve as
fixed
The resolution will be set. Next status will be 'closed'
reassign to
kjanda
The owner will change from theinova. Next status will be 'new'
accept
The owner will change from theinova to kjanda. Next status will be 'assigned'
Preview
Submit changes

Obrázek č. 8

Stiskem tlačítka „**Attach file**“ se uživatel ocitne na obrazovce Add Attachment, kde může přiložit přílohy k defectu.



Po úspěšném přiložení se zobrazí potvrzovací obrazovka.



Obrázek č. 9

Při neúspěšném přiložení se zobrazí obrazovka

Obrázek č. 10

Active Tickets by Version

Zde se zobrazují defekty po jednotlivých verzích. Opět se může požadavek rozkliknout k editaci (vše jako v předchozích krocích).

{2} Active Tickets by Version (7 matches)

This report shows how to color results by priority, while grouping results by version.

Last modification time, description and reporter are included as hidden fields for useful RSS export.

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#5	UDS - padá aplikace	Front-end		defect	somebody	new	3.4.2009
#7	AS - chyba v komunikaci	TIA		defect	kjanda	new	3.4.2009

2.0 (2 matches)

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#2	a	Front-end	2.0	defect	theinova	assigned	3.4.2009
#1	BO - dkhzd	Front-end	2.0	defect	jkapec	new	3.4.2009

5.3.1 (2 matches)

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#4	AS - chyba validace	TIA	5.3.1	defect	topribyl	new	3.4.2009
#8	AAAAAAAAAAAAAAAAAAAAAA	Front-end	5.3.1	user story	somebody	new	6.4.2009

Obrázek č. 11

Active Tickets by Milestone

Zde si uživatel může prohlídnout defekty dle vývojového stádia. Opět se může požadavek rozkliknout k editaci (vše jako v předchozích krocích).

{3} Active Tickets by Milestone (7 matches)

This report shows how to color results by priority, while grouping results by milestone.

Last modification time, description and reporter are included as hidden fields for useful RSS export.

Milestone (1 match)

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#2	a	Front-end	2.0	defect	theinova	assigned	3.4.2009

Milestone R5.4 (6 matches)

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#4	AS - chyba validace	TIA	5.3.1	defect	topribyl	new	3.4.2009
#5	UDS - padá aplikace	Front-end		defect	somebody	new	3.4.2009
#6	VOP - chyba verlyba	Business logika	5.3.19	defect	kjanda	new	3.4.2009
#7	AS - chyba v komunikaci	TIA		defect	kjanda	new	3.4.2009
#8	AAAAAAAAAAAAAAAAAAAAAA	Front-end	5.3.1	user story	somebody	new	6.4.2009

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#1	BO - dkhzd	Front-end	2.0	defect	jkapec	new	3.4.2009

Obrázek č. 12

Hodnocení:

Slovní:

Administrátorské rozhraní – složité

Grafická část – nástroj Trac je uživatelsky nepřívětivý

Funkční část – funkčnost nástroje je bez chyb

Použitelnost – dá se použít pro malé projekty i velké projekty

Bodové:

Administrátorské rozhraní – 12 bodů

Grafická část – 13 bodů

Funkční část – 25 bodů

Použitelnost – 15 bodů

Celkové hodnocení program je **65** bodů.

Závěr testování Tracku:

Záporem nástroje je vzhled a skutečnost, že nástroj je uživatelsky komplikovaný a nepohodlný, zejména při prohlížení defektů. Další nevýhodou je nutnost technické podpory při instalaci nástroje a také při jeho používání. Pozitivně lze hodnotit velké množství pluginů. Tím se však opět zvyšuje náročnost na instalaci a přípravu vhodného prostředí. Nástroj je sice velmi dobře nastavitelný a rozšiřitelný, ale jeho pracnost a ovladatelnost je opravdu v porovnání s jinými velmi složitá. Bez podpory technického týmu je práce s tímto nástrojem značně komplikovaná a není tedy vhodný pro projekty malého typu, kde v testovacím týmu jsou ne IT pracovníci. Track lze doporučit pro technické týmy, zejména pro vývojáře, kteří v nástroji mohou sledovat životní cyklus svého software.

5.2.3.2 JTrac

Mezi další vybrané nástroje patří webová aplikace JTrack, která je napojena na vlastní databázi. Nástroj JTrac lze používat pro zadávání, evidenci a také pro generování reportu chyb. Jedním z kladů tohoto nástroje je intuitivní ovládání nástroje, přehlednost zadaných defektů a také jednoduchá instalace JTracku. Mezi další výhody patří jednoduchá úprava a tvorba nových rolí. Založený je na Java, to znamená, že je omezený na možnosti distribucí Javy (v současné době pro Linux, Solaris a Win32). Na spuštění je potřeba webový server s Java web kontejnerem, je tedy možné zpřístupnit funkce i přes internet. Na komunikaci s DB využívá Hibernate – objektově relační mapovač. Je možné použít jakoukoli DB, ke které existuje JDBC ovládač.

V nástroji byly používány tyto role:

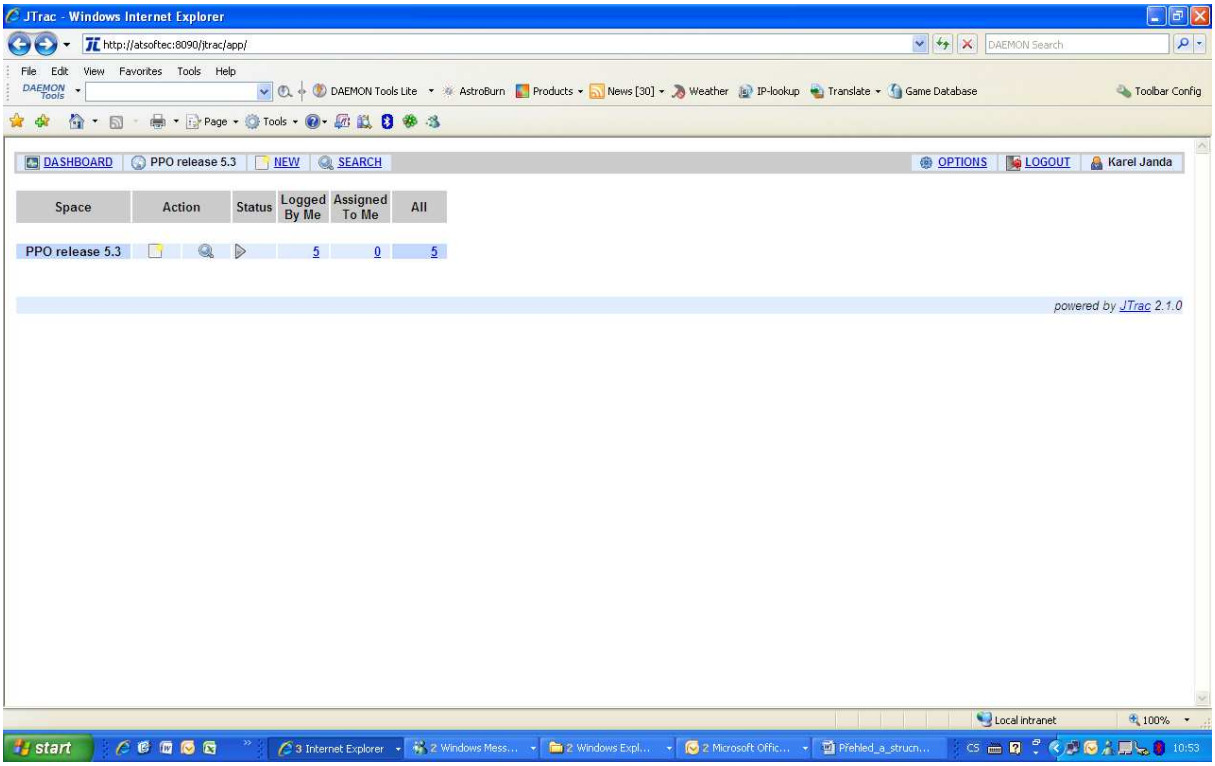
Tester, Solver, TestKoordinátor, Administrátor

Výše zmíněné role byly samostatně vytvořeny a prozkoumány (jejich funkčnost).

Jednoduchá ukázka prostředí a jednoduchých úkonů

Záložka **Dashboard**

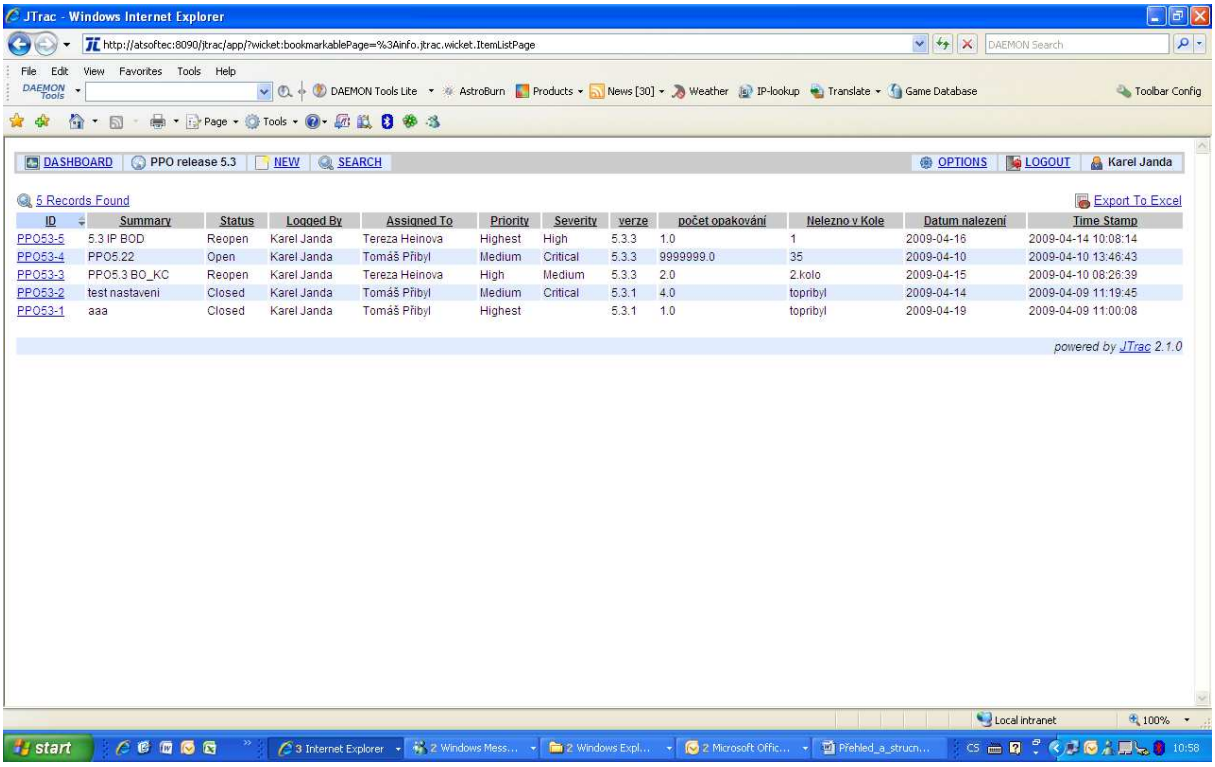
Po otevření záložky se zobrazí počet zadaných chyb rozdělených podle releasu



Obrázek č. 13

Link ve sloupci ALL v záložce **Dashboard**

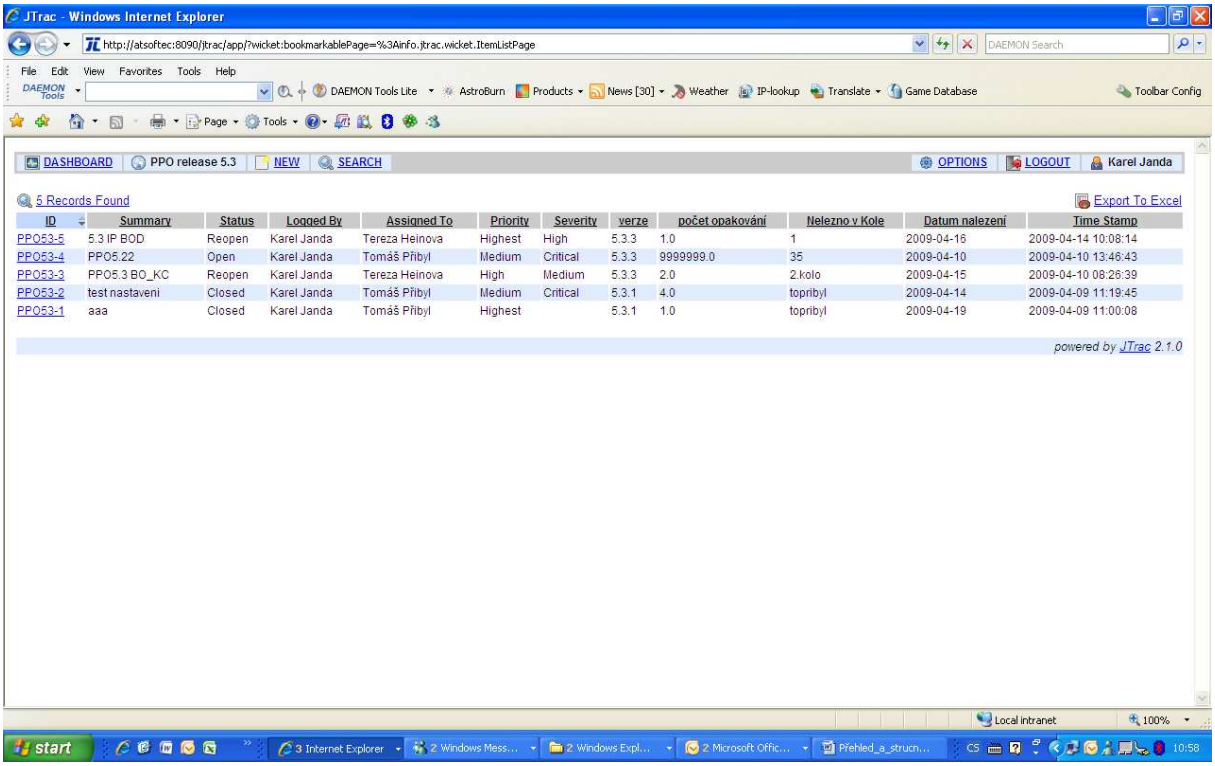
Po otevření linku ve sloupci All bude zobrazen přehled všech chyb



Obrázek č. 14

Link Export To Exel (je umístěn na pravé horní části obrazovky) v záložce **Dashboard**

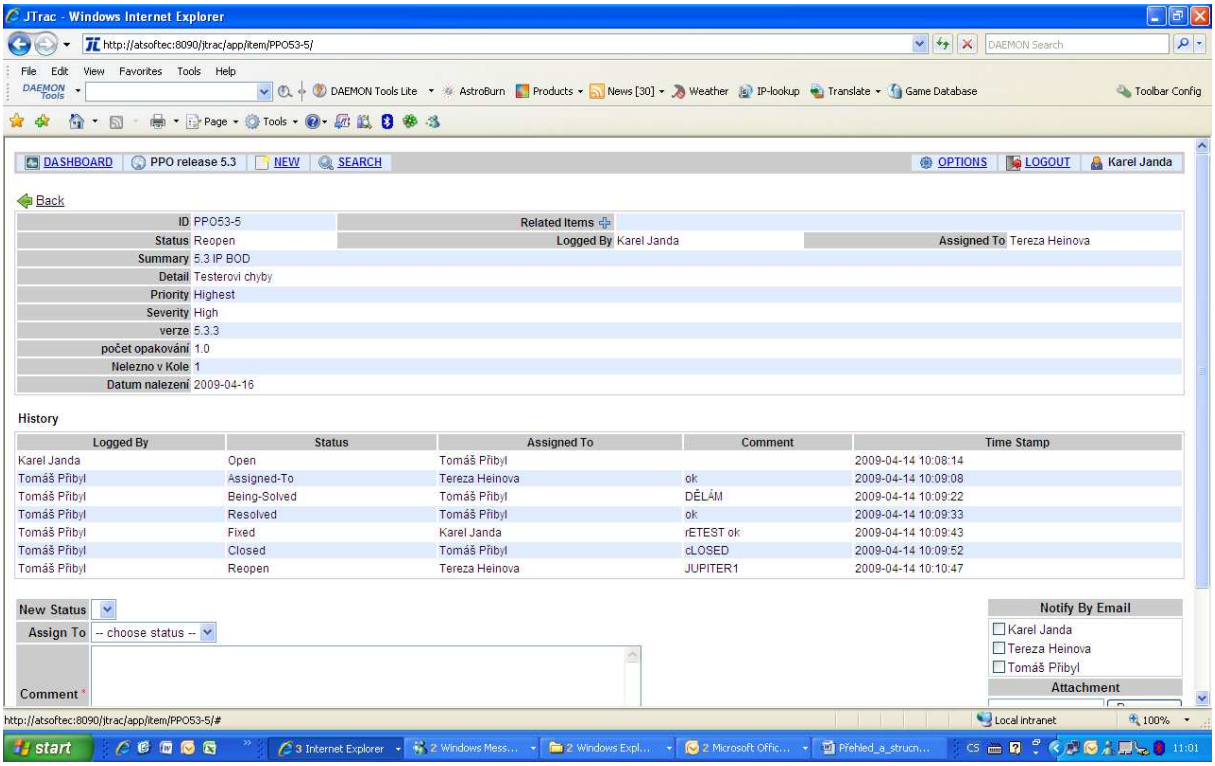
Po kliknutí se zobrazí, zda chcete report otevřít nebo uložit



Obrázek č. 15

Link ve sloupci ID v záložce **Dashboard**

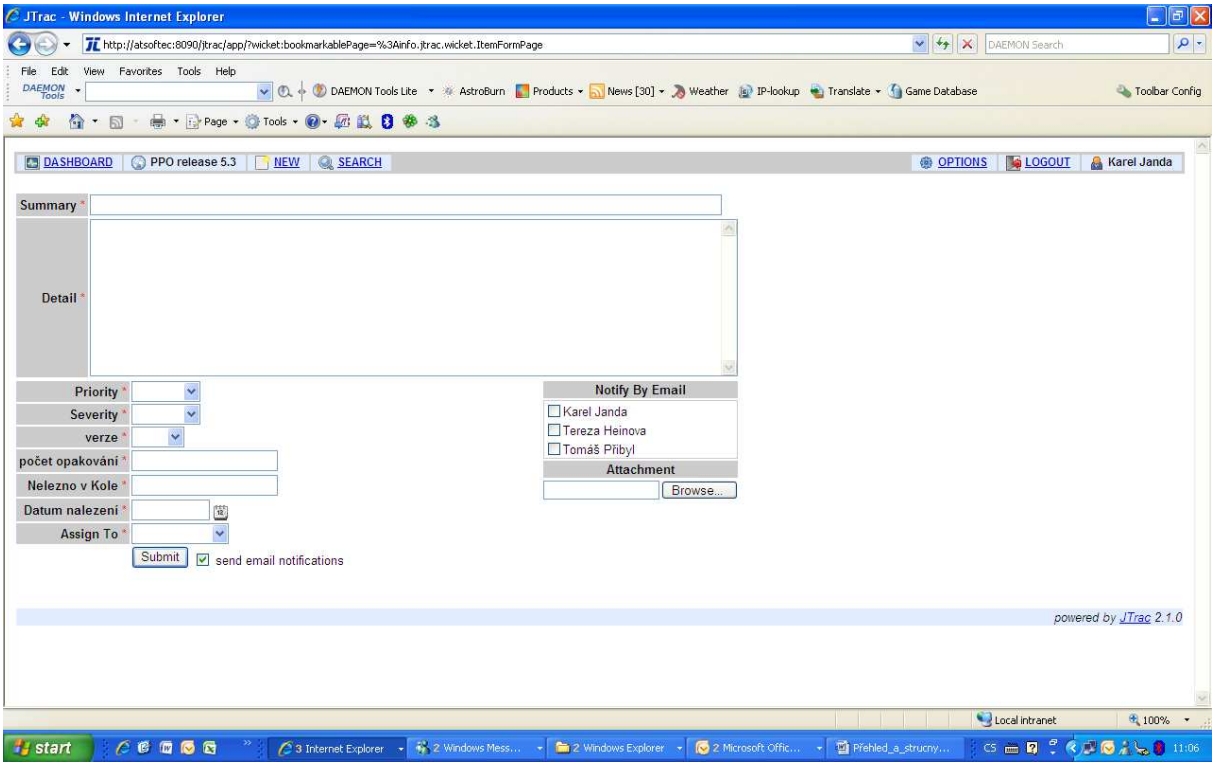
Po kliknutí na link ve sloupci ID se zobrazí detail defektu



Obrázek č. 16

Záložka ***New***

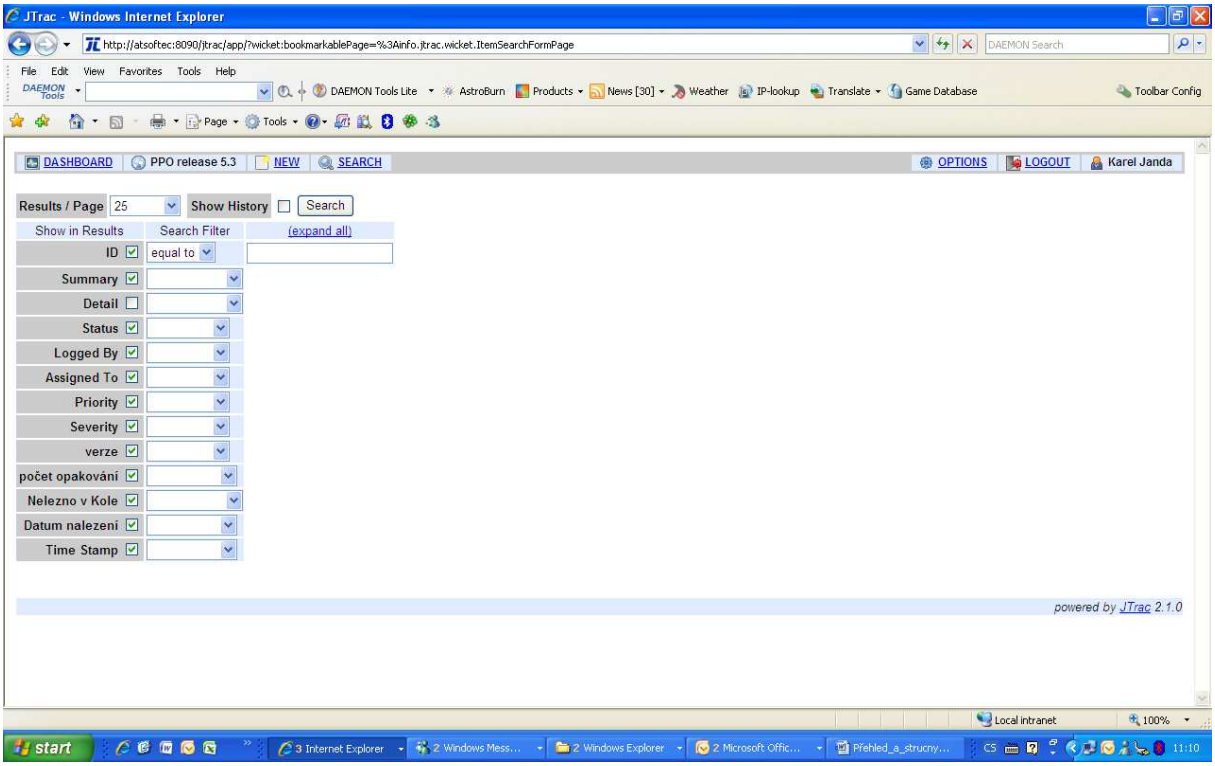
Používáme k zadávání nových chyb



Obrázek č. 17

Záložka ***Search***

Slouží k vyhledání defektů dle kritérií



Obrázek č. 18

hodnocení:

Administrátorské rozhraní – jednoduché, není potřeba zasahovat do databáze při změně nebo tvorbě nových práv, vše proběhne na front-endu aplikace

Grafická část – nástroj JTrac je uživatelský přívětivý (intuitivní), není třeba žádné odborné znalosti k využití tohoto nástroje

Funkční část – funkčnost nástroje JTrac byla bez chyb a to jak při zadávání defektů, změně stavu defektů, tak i při reportu defektů

Použitelnost – Program je použitelný jak pro malé projekty, kde se eviduje jeden projekt, tak i pro projekty velkého charakteru, kde je rozděleno více samostatných projektů.

Administrátorské rozhraní – 25 bodů

Grafická část – 25 bodů

Funkční část – 25 bodů

Použitelnost – 23 bodů

Celkové hodnocení programu je **98** bodů.

Z hlediska administrátorského rozhraní, grafické části, funkčnosti a použitelnosti nástroje můžu zhodnotit nástroj JTrac jako velice podařený freeware nástroj na zadávání, evidenci a report defektů. Jedinou nevýhodou je nemožnost vytvoření a ukládání skriptů, případná jejich editace. JTrac má řadu výhod jako je například velmi snadná istalace, přizpůsobitelný workflow, možnost nastavení E-mailových upozornění a fulltextové vyhledávání. Nástroj umožňuje také export reportů do excelu. JTrac je vhodný jak pro velké, tak malé projekty právě díky snadné instalaci a ovládání.

5.2.3.3 iTracker

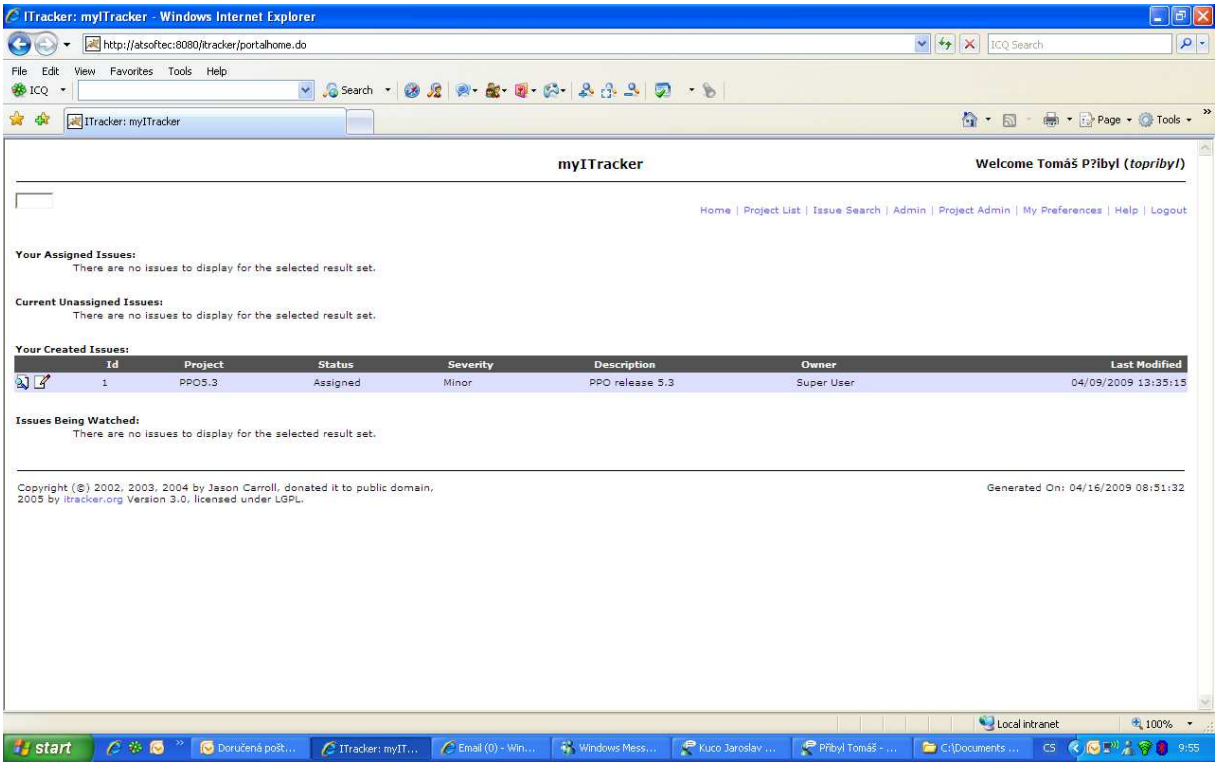
Je nástroj pro vytváření a evidenci chyb zjištěných při testování software, který je napojen na databázi. Také se dá ITracker použít na vytváření scriptů. Nástroj samotný působí velmi stroze. Jeho ovládání není intuitivní a u některých tlačítek a vstupních polí uživatel na první pohled neví k čemu je využít. Jedinou výhodou je snadná definice nových rolí. Nástroj je založený na Java, to znamená, že je omezený na možnosti distribucí Javy (v současné době pro Linux, Solaris a Win32). Na spuštění je potřeba webový server s Java web kontejnerem, je tedy možné zpřístupnit funkce i přes internet. Podporuje jen MySQL DB.

Uživatelům ITrackeru je možné dát tato oprávnění (jejich význam je poměrně přesný z jejich názvu a proto ho z prostorových důvodů blíže nespecifujeme):

Create Issues	Create Issues for Others
Edit All Issues	Edit User's Own Issues
Full Issue Edit	Close Issues
Can Be Assigned Any Issue	Assign Issues to Self
Unassign Issues from Self	Assign Issues to Others
View All Issues	View User's Own Issues
Project Admin	

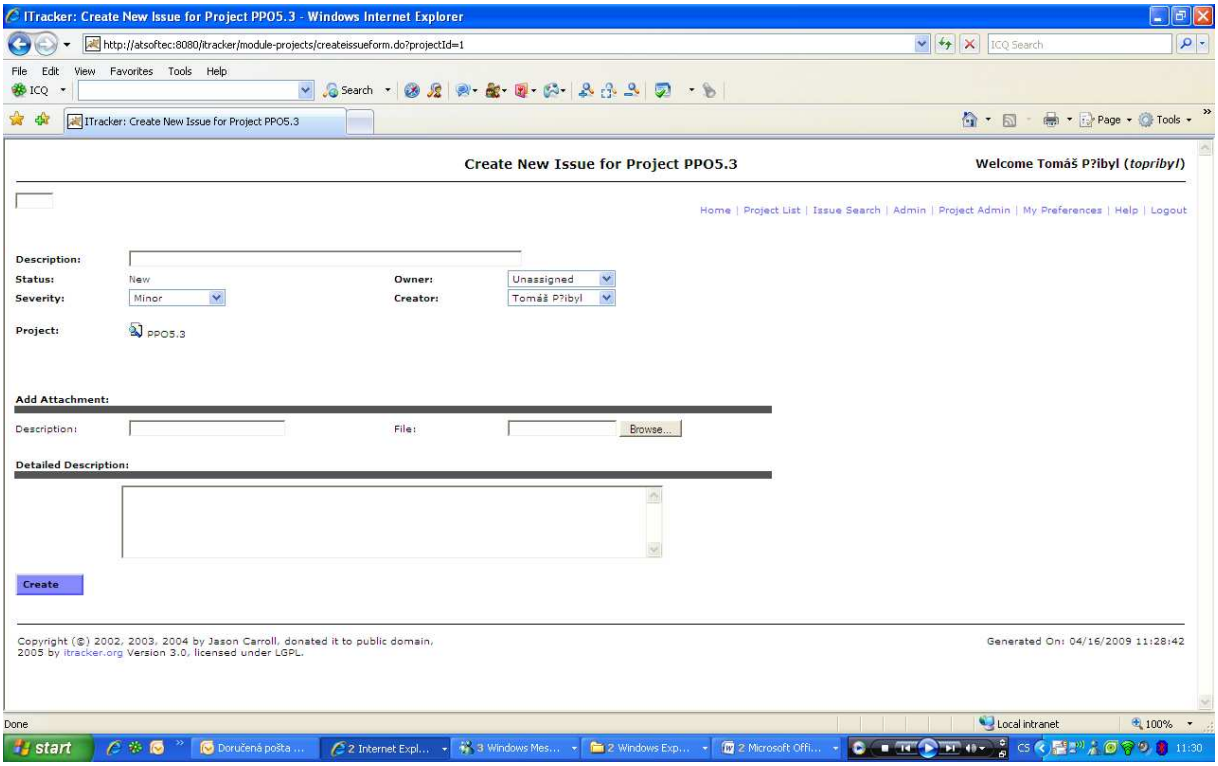
Jednoduchá ukázka použití iTrackeru :

Po přihlášení se uživateli zobrazí obrazovka Home. Zde vidí základní položky vázající se k jeho loginu, může je editovat nebo jen prohlížet.



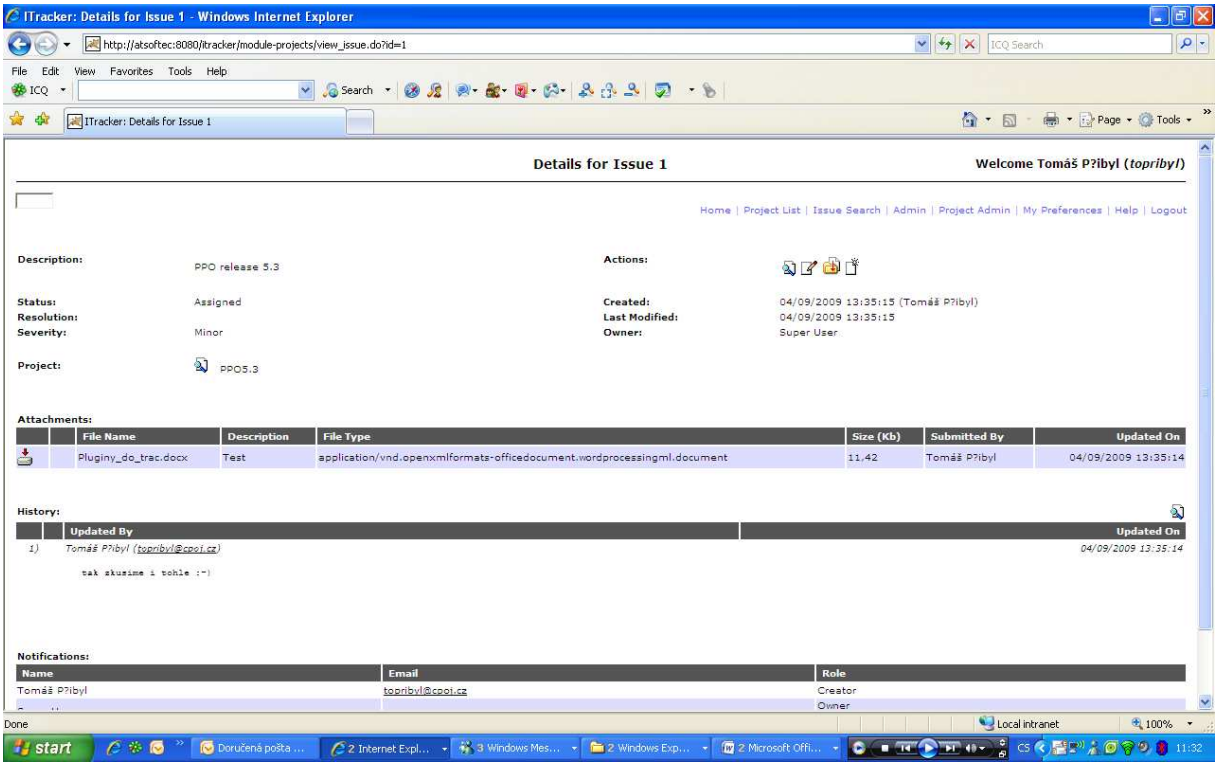
Obrázek č. 19

Po stisku ikonky **Create New Issue** for Projet PPO5.3 na obrazovce Home se uživatel dostane na obrazovku Create New Issue for Projet, kde může zadat nový defect.



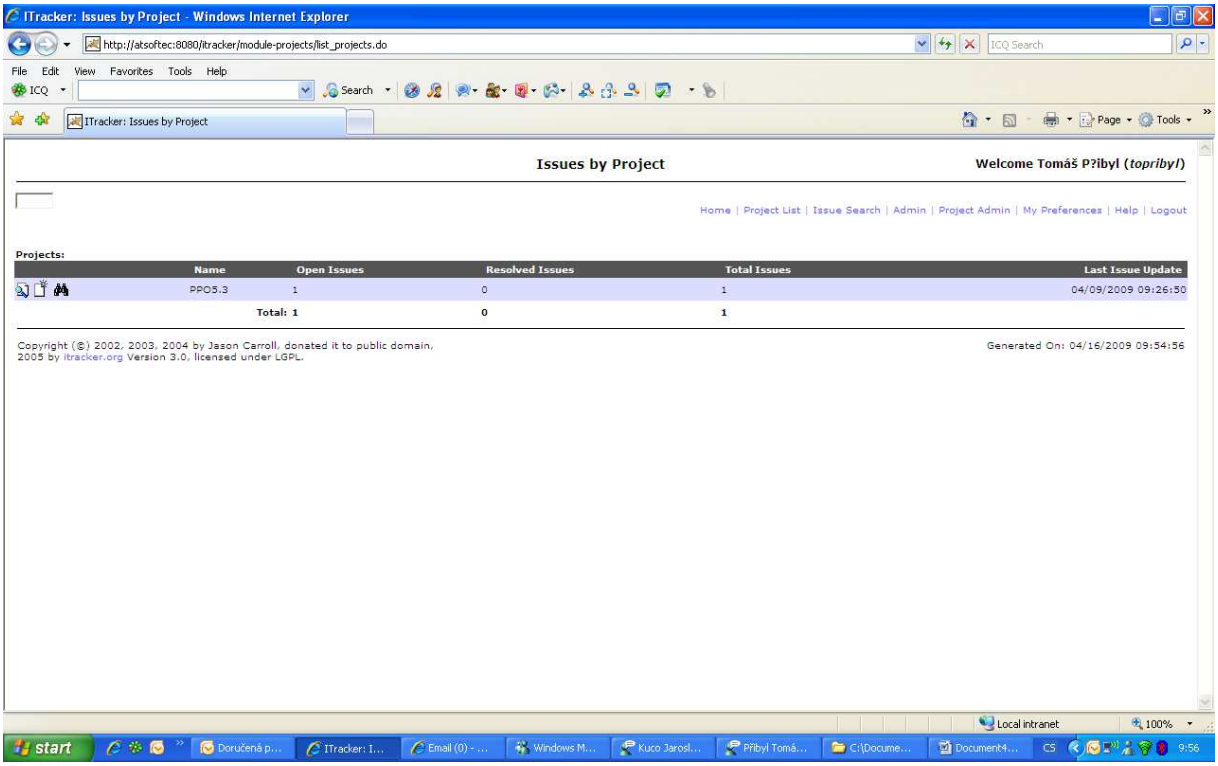
Obrázek č. 20

Stiskem tlačítka **View Issue** 1 uživatel vidí obrazovku s detailem defectu.



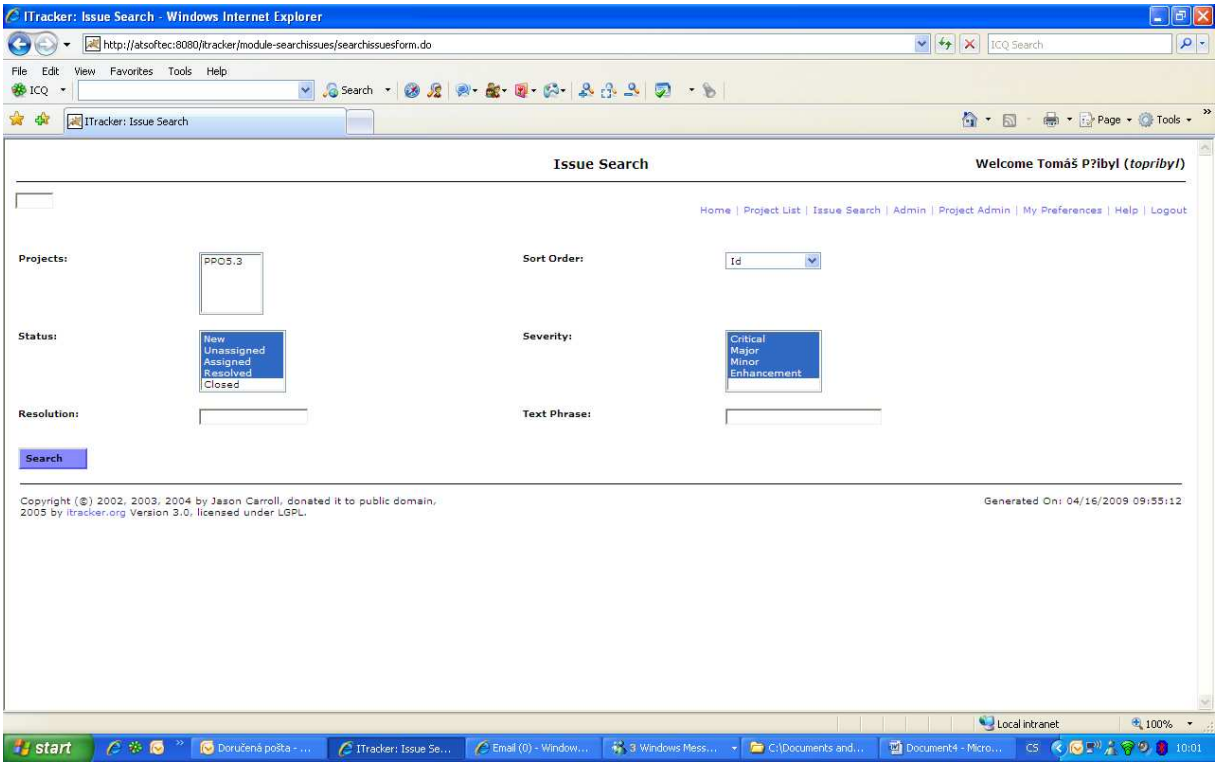
Obrázek č. 21

Všechny defekty je možno vidět na obrazovce **Project list**. Zde může uživatel prohlížet, editovat.



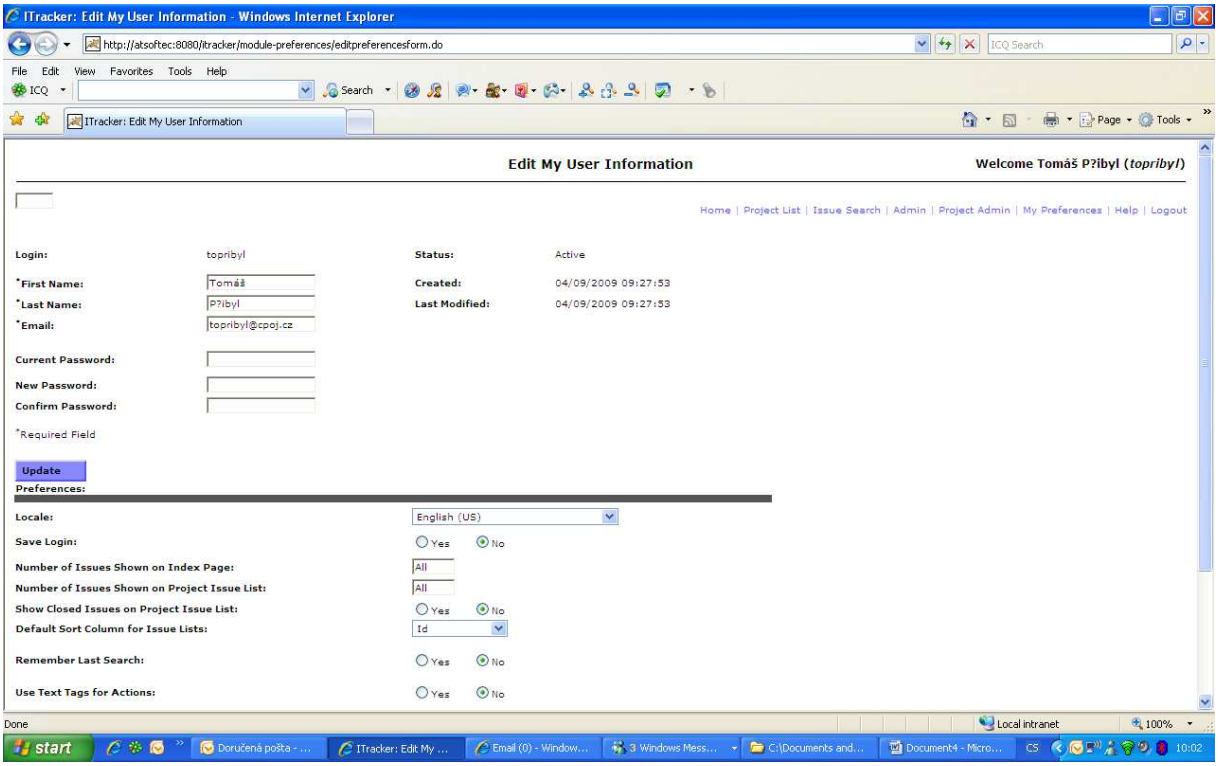
Obrázek č. 22

Obrazovka **Issue Search** slouží k hledání defektů.



Obrázek č. 23

Na obrazovce **Edit My User Information** si uživatel může upravovat svůj profil.



Obrázek č. 24

<u>Hodnocení:</u>

Slovní:

Administrátorské rozhraní - vyžaduje časté zalogování, sám uživatele odhlašuje

Grafická část - velmi nepřehledná

Funkční část - neintuitivní a uživatelsky nepohodlná

Použitelnost - dá se použít i pro více uživatelů

Bodové:

Administrátorské rozhraní – 5 bodů

Grafická část – 10 bodů

Funkční část – 10 bodů

Použitelnost – 15 bodů

Celkové hodnocení programu je **40** bodů.

iTracker neshledávám jako dobrou volbu. Práce s tímto nástrojem není uživatelsky příjemná. Aplikace obsahuje pole a tlačítka, o kterých uživatel na první pohled neví, k čemu slouží.
Na každé obrazovce je vstupní pole, u kterého není jasný jeho účel. Toto tlačítko je prakticky nepoužitelné pro jakoukoli funkci. Funkčnost aplikace je poněkud zmatečná. iTracker také vyžaduje časté zalogování uživatele (po pěti minutách nečinnosti automaticky odloguje uživatele). Nástroj nelze doporučit pro malé ani velké projekty. Práce s ním je velmi časově náročná a z uživatelského hlediska se zdá být velmi nespolehlivým nástrojem.

5.2.3.4 Fixx

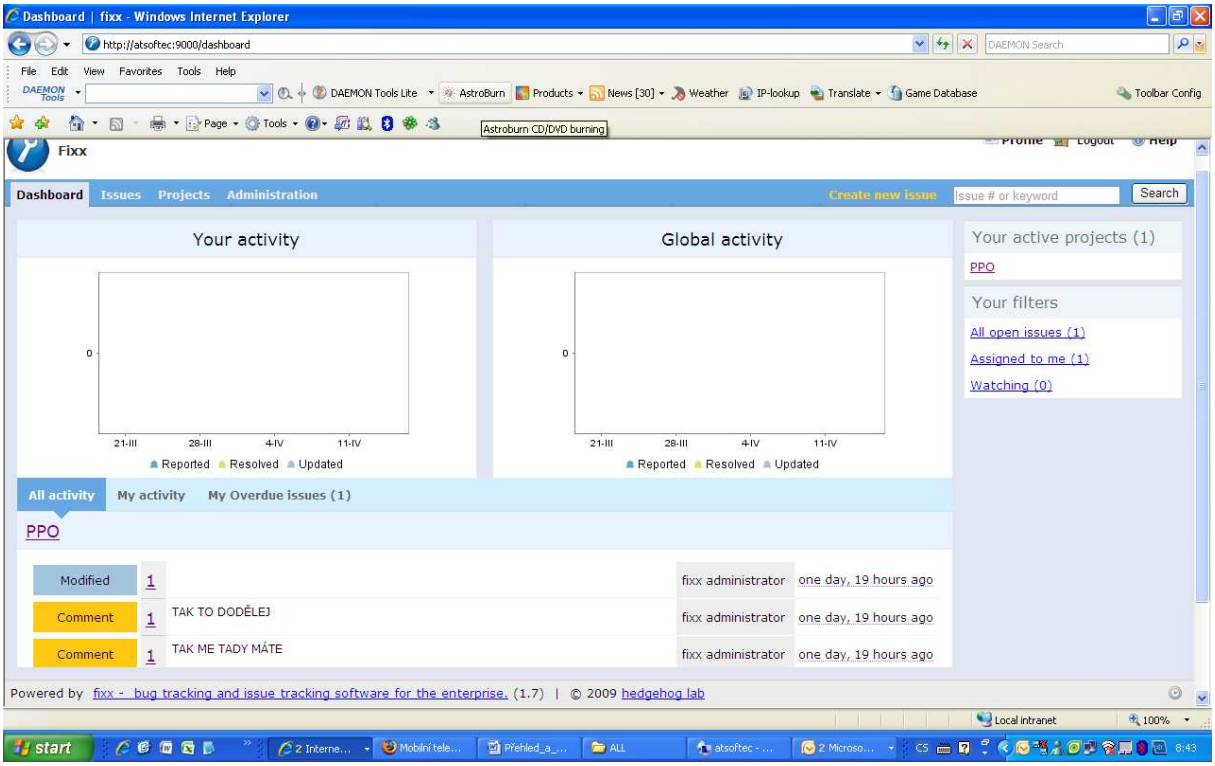
Jedním z dalších nástrojů pro zadávání defektů, správu defektů a report defektů je webová aplikace Fixx, která je napojena na vlastní databázi. Nástroj Fixx má oproti ostatním nástrojům výhodu mnoha možností nastavení. Jeho velkou nevýhodou je, že není freeware pro skupinu uživatelů. Freeware je pouze tehdy, bude-li nástroj používat pouze jeden uživatel.

Založený na Java, to znamená, že je omezený na možnosti distribucí Javy (v současné době pro Linux, Solaris a Win32). Na spuštění je potřeba webový server s Java web kontejnerem, je tedy možné zpřístupnit funkce i přes internet. Nástroj podporuje MySQL a MS SQL DB. Podporuje Internet Explorer a FireFox.

Jednoduchá ukázka prostředí

Záložka ***Dashboard***

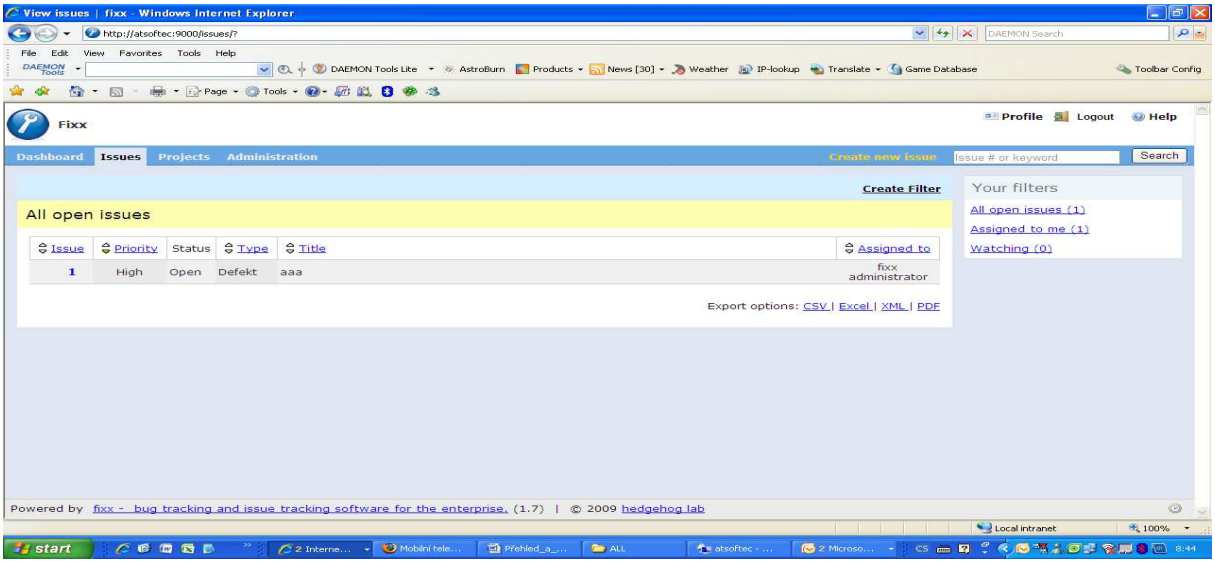
V záložce Dashboard jsou graficky znázorněny aktivity, které byly vykonány za určitý čas. Také jsou zobrazeny všechny změny, komentáře a změny stavů v defektu, které byly provedeny.



Obrázek č. 25

Záložka ***Issues***

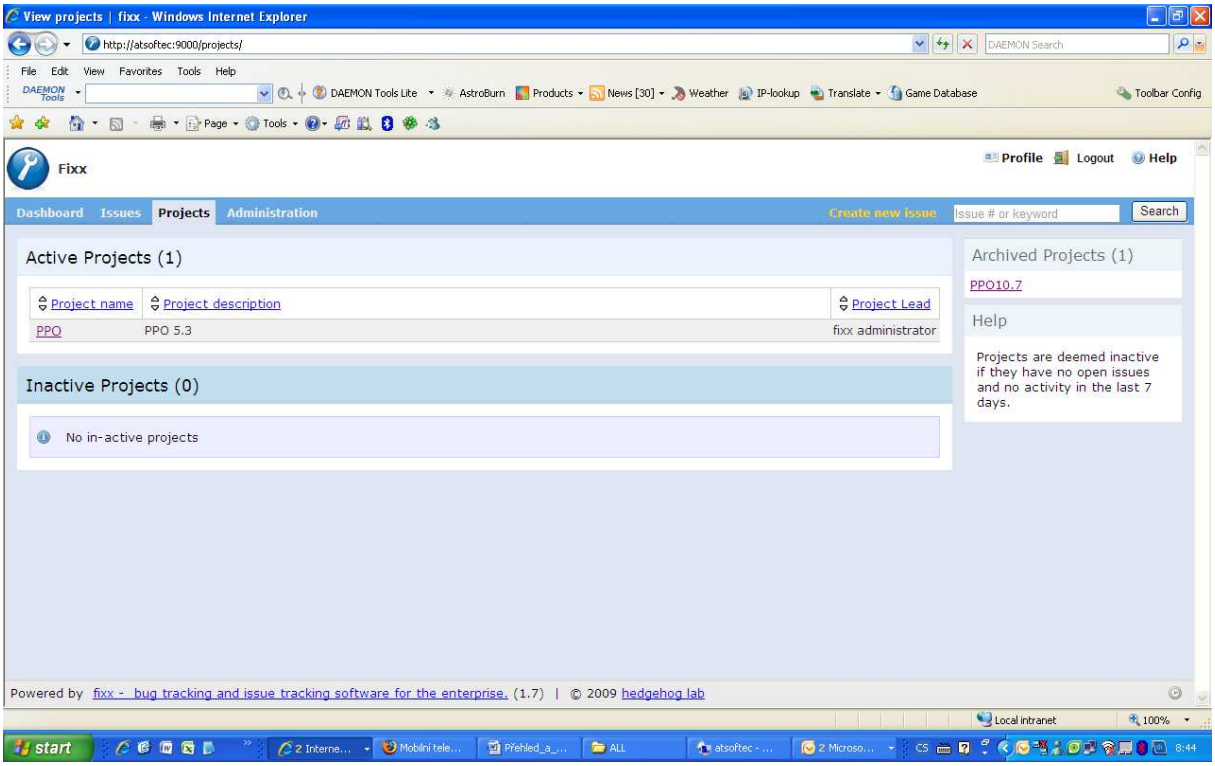
Po kliknutí na záložku Issues budou zobrazeny všechny otevřené defekty.



Obrázek č. 26

Záložka ***Projects***

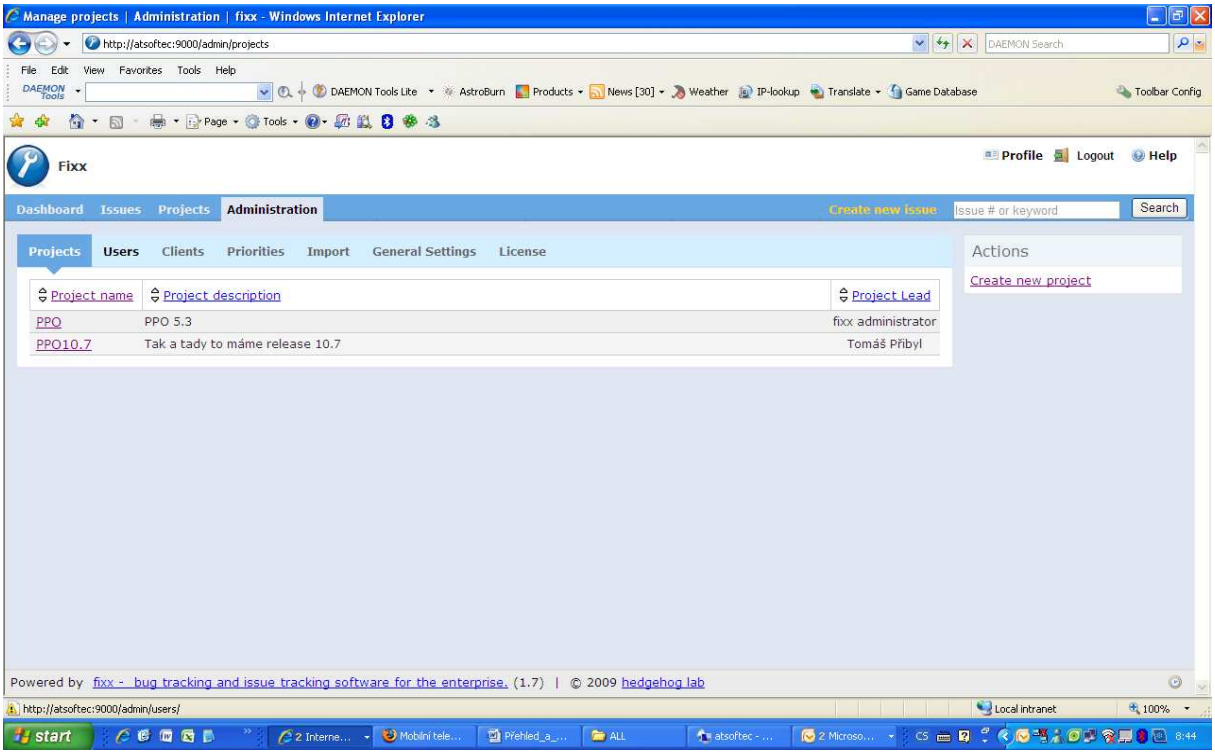
Po kliknutí na záložku Projects jsou zobrazeny všechny projekty které, byly vytvořeny a jsou rozděleny na aktivní a již uzavřené.



Obrázek č. 27

Záložka ***Administration***

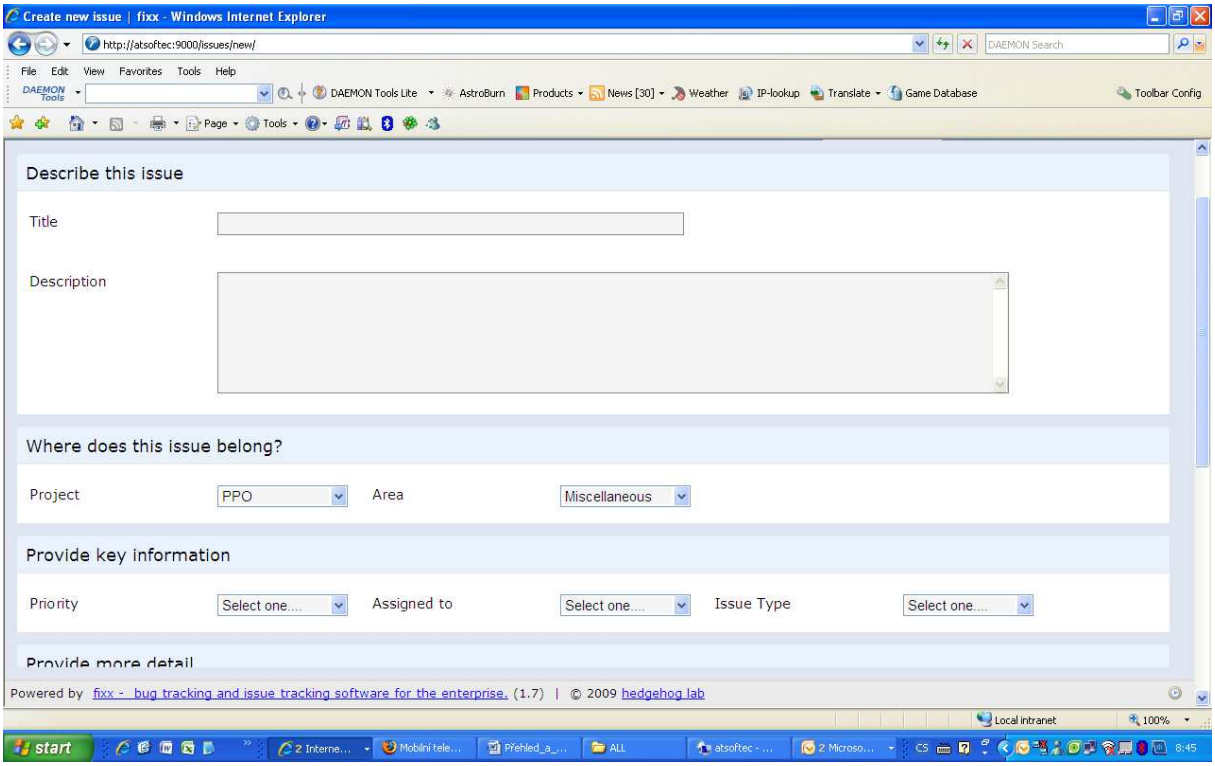
Po kliknutí na záložku Administration je zobrazeno submenu s názvy Projects, Users, Clients, Priorities, Import, General settings a Licence.



Obrázek č. 28

Záložka ***Create new issue***

Záložka Create new issue slouží k zadání nových defektů



Obrázek č. 29

Hodnocení:

Slovní:

Administrátorské rozhraní - jednoduché

Grafická část – nástroj Fixx je uživatelsky přívětivý, není třeba odborné znalosti k využití tohoto nástroje

Funkční část – funkčnost nástroje Fixx byly bez chyb

Použitelnost - program je použitelný jak pro malé projekty, tak pro větší projekty

Bodové:

Administrátorské rozhraní – 25 bodů

Grafická část – 18 bodů

Funkční část – 15 bodů

Použitelnost – 17 bodů

Celkem hodnocení programu je **75** bodů

Protože hlavním kritériem při zadání RP bylo najít vhodný freeware nástroj na zadávání, správu a report defektů byl nástroj Fixx vyřazen z výběru vhodných nástrojů. Tento nástroj není určen pro velké projekty a jeho použitelnost pro malé projekty je také značně rozporuplná. Lze ho využívat pouze soukromě pro jednoho uživatele, přičemž náročnost na odborné znalosti z oblasti IT k jeho instalaci a provozu jsou velké. Není tedy vhodný pro malé projekty s ne IT uživateli.

5.2.3.5 Bugbase

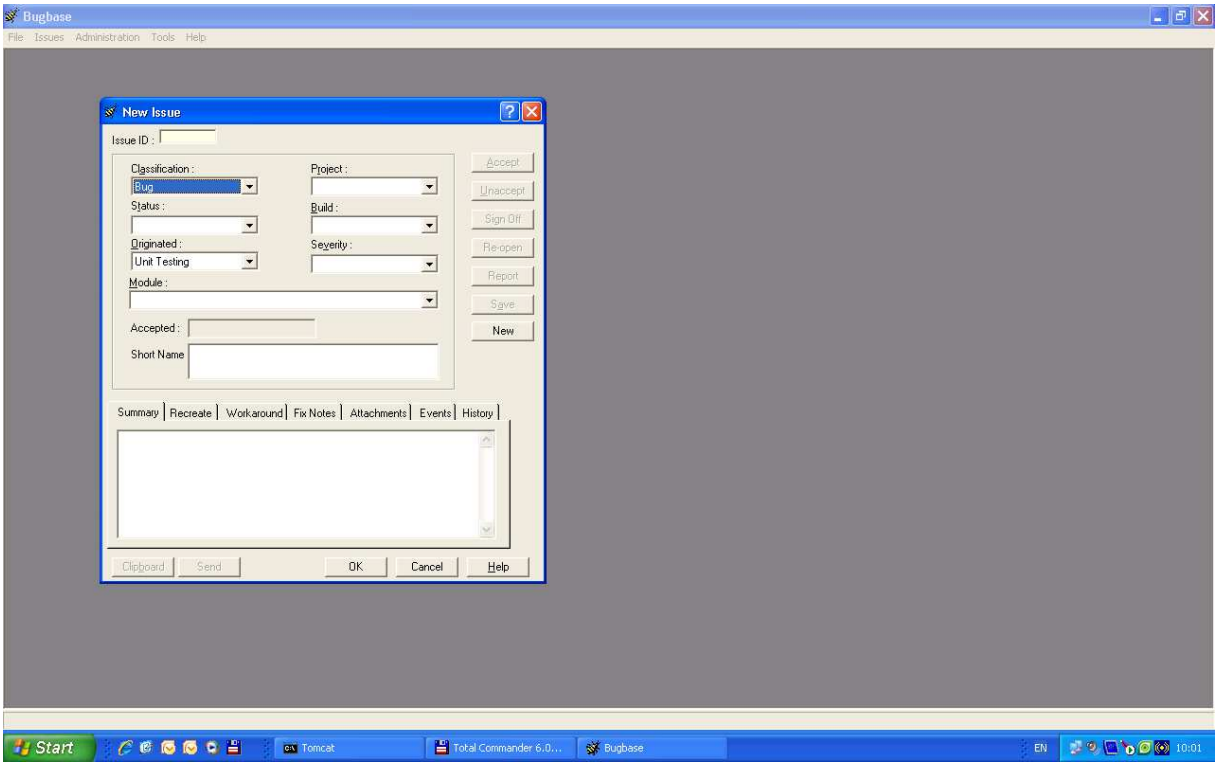
Dalším z nástrojů, který byl vybrán k bližšímu prozkoumání je nástroj Bugbase. Výhodou tohoto nástroje je snadná instalace a prvotní nastavení. V nástroji Bugbase je možno zadávat defekty, vytvářet reporty defektů a také upravovat defekty. Mezi klady tohoto nástroje bezesporu patří jednoduché ovládání. V nástroji Bugbase nelze vytvořit samostatný nový projekt z důvodu registrace, proto byl Bugbase vyřazen z výběru na freeware nástroj na správu chyb.

Určeno pro Microsoft Windows XP, s vlastní správou dat (nepoužívá externí DB). Nástroj není skriptovací jazyk, ale je to standardní WIN32 aplikace. Testováno na Internet Explorer 7.

Nyní jednoduchá ukázka prostředí

Záložka **Issue New**

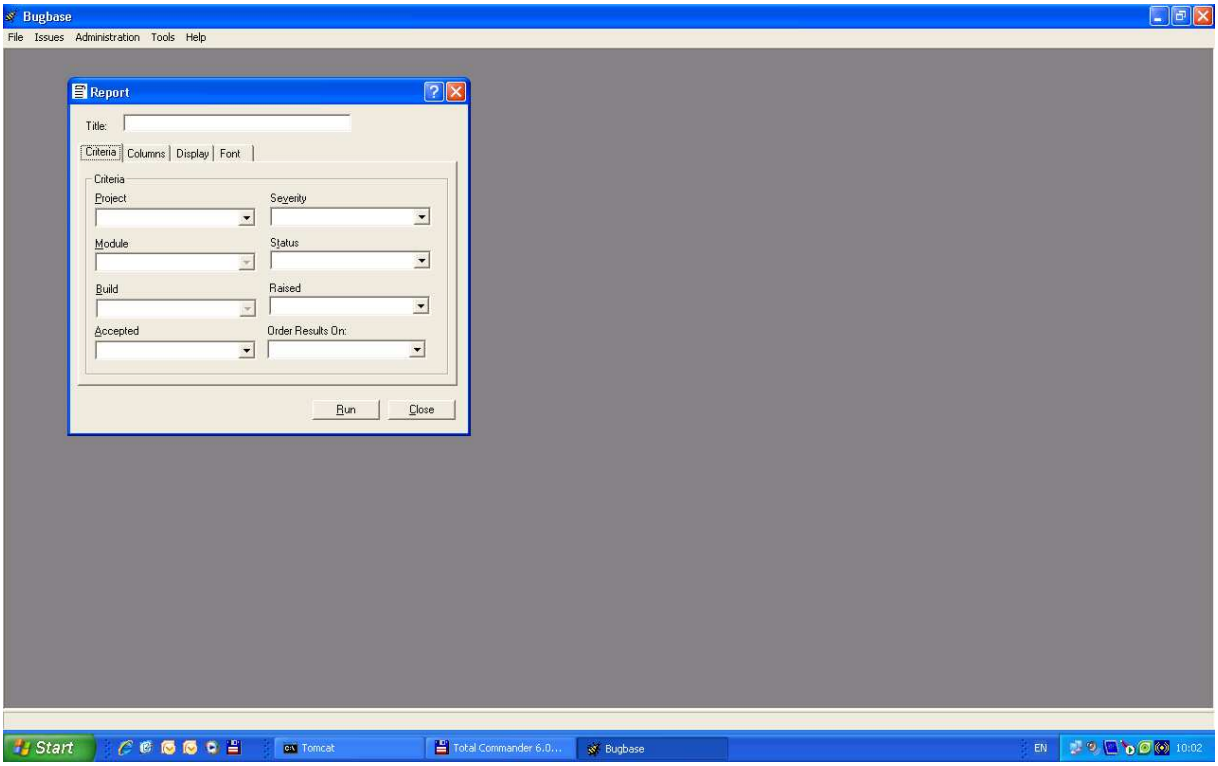
Slouží k vytvoření nového defektu



Obrázek č. 30

Záložka **Issue – Reports**

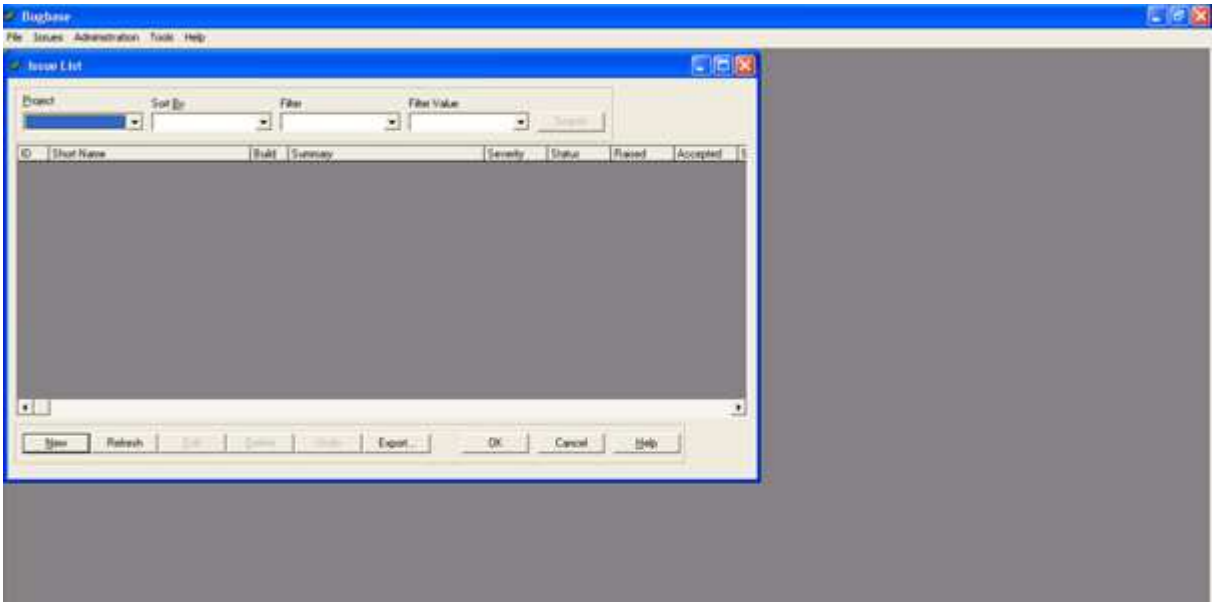
Slouží k vytvoření reportu o chybách



Obrázek č. 31

Záložka **Issue – View**

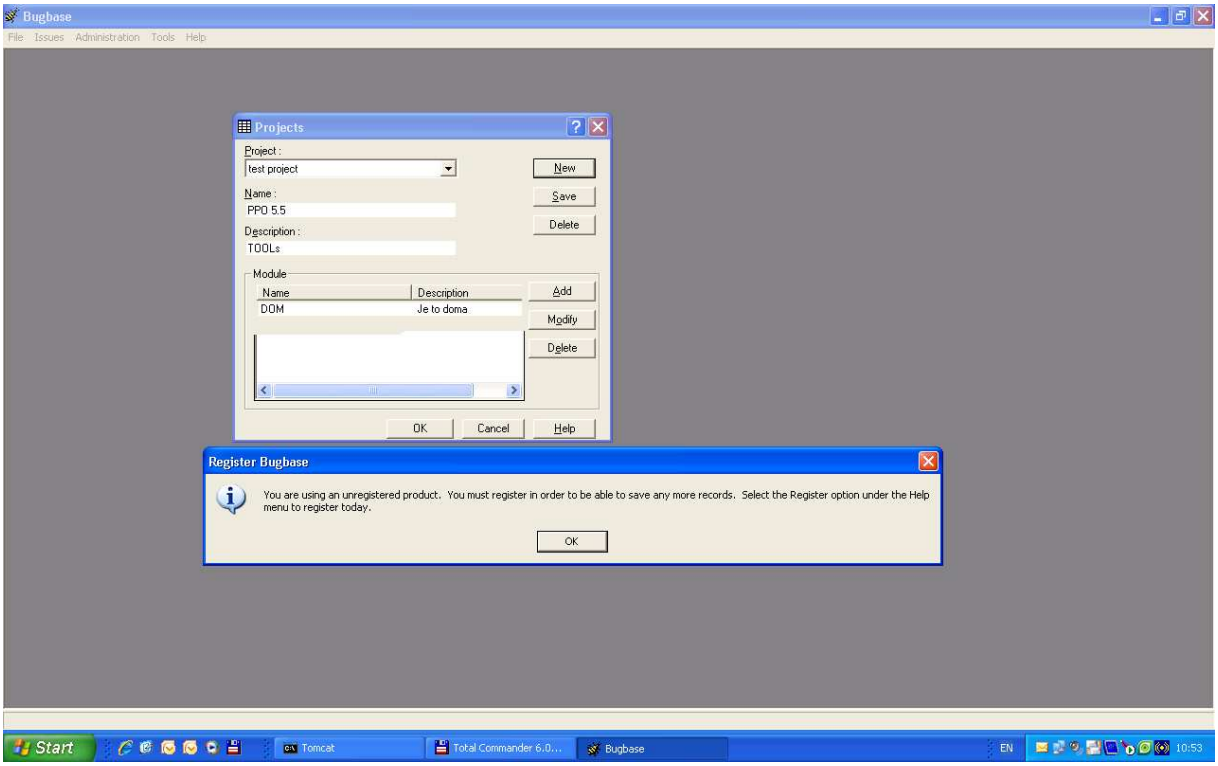
Slouží k zobrazení všech zadaných defektů



Obrázek č. 32

Záložka **Administration – Projects**

Slouží k vytvoření nového projektu – jak je uvedeno výše k vytvoření nového projektu je potřebná registrace nástroje Bugbase



Obrázek č. 33

Hodnocení:

Administrátorské rozhraní – jednoduché, nejsou třeba žádné odborné znalosti

Grafická část – nástroj Bugbase je uživatelsky přívětivý

Funkční část – funkčnost nástroj Bugbase je bez chyb, až na vytvoření nového projektu, kde při zakládání nového projektu nástroj požaduje registraci produktu.

Použitelnost - program je použitelný jak pro malé projekty, tak větší projekty

Administrátorské rozhraní – 13 bodů

Grafická část – 12 bodů

Funkční část – 4 bodů

Použitelnost – 16 bodů

Celkové hodnocení programu je **45** bodů.

Nástroj Bughase je nenáročný na instalaci a provoz a lze ho doporučit pro malé projekty. Pro velké projekty je značně nevhodný, zejména je to dáno nemožností zakládat v něm další projekt. Pokud by tedy uživatel nechtěl investovat finance.

Hodnocení

Souhrnné hodnocení jednotlivých nástrojů:

Název nástroje	**Počet bodů za Admin. rozhraní**	**Počet bodů za Grafickou část**	**Počet bodů za Funkční část**	**Počet bodů za použitelnost**	**Celkový počet bodů**
Trac	12	13	25	15	65 bodů
JTrac	25	25	25	23	98 bodů
iTracker	5	10	10	15	40 bodů
Fixx	25	18	15	17	75 bodů
Bugbase	13	12	4	16	45 bodů

Sumarizace (jaké funkčnosti vlastní hodnocené nástroje)

Nástroj	**Počet bodů**	**Slovní hodnocení**
Trac	65 bodů	+ velmi dobře nastavitelný a rozšiřitelný - uživatelsky komplikovaný a nepohodlný - nutnost technické podpory
JTrac	98 bodů	+ velice podařený -nemožnost vytvoření a ukládání skriptů
iTracker	40 bodů	-není uživatelsky příjemná -není jasný účel některých polí -časté zalogování
Fixx	75 bodů	+ uživatelsky přívětivý -použitelný jak pro malé projekty, tak pro větší projekty -pouze pro jednoho uživatele
Bugbase	45 bodů	+jednoduchý, uživatelsky přívětivý -zakládání nového projektu vyžaduje registraci

Závěrem lze doporučit nástroj JTrac. Po porovnání s ostatními nástroji na trhu, je zřejmé, že JTrac je vhodný nejen po stránce administrátorské a funkční, ale i jeho grafická část je uživatelky příjemná, srozumitelná a vhodná i pro nezkušené, začínající testery. Pro svou freeware dostupnost může být zajímavý pro malé firmy, které nemají prostředky na nákup drahých softwarů. Celý software je také velmi jednoduše rozšiřitelný.

6 Závěr

V jednotlivých kapitolách jsem popsala testování a jeho místo v životním cyklu vývoje software. Testování je důležitou součástí vývoje software, může být jak manuální tak automatické. Začíná před a končí dlouho po vykonání testů. K testování patří i aktivity jako je plánování, řízení, výběr testovacích podmínek, navržení testovacích případů, kontrola a vyhodnocování výsledků a samozřejmě reportování výsledků. Cílem testování je nalezení defektů nebo předcházení defektům. Na kvalitě produktu mají velkou zásluhu testeři. Testeři musí být vybaveni řadou znalostí a hlavně zkušeností.

Důležitou částí testování jsou jeho přípravy. Ty začínají již na počátku projektu. Při plánování testů se definují požadavky testů, požadavky na zdroje- jak na lidské tak na technické. Také se v této fázi definují kritéria pro akceptaci. Po fázi plánování vzniká funkční design, který je základem pro přípravu skriptů. Testovací skripty neboli scénáře jsou nepostradatelnou pomůckou testera. Je jim věnován čas a počítá se s jejich vytvořením v harmonogramu projektu. Podle testovacích skriptů se testuje ve všech kolech testování. Testovací skript by měl být připraven tak důkladně, aby podle něj mohl testovat kdokoli, i ten kdo o testovaném software nic neví. Do testovacího skriptu tester zanese vše podstatné z funkčního designu, aby se dobral co nejlepšího otestování software. Testovací skripty se uspořádají do testovacích sad a může se začít se samotným testováním. Testování by mělo začít u programátorů, kteří provádějí Unit testy. Jedná se o kontrolu kódu. Následují kola systémových testů, která jsou obvykle čtyři. Každé kolo trvá jeden týden. Vyhodnocování testů probíhá jak v jednotlivých kolech, tak na závěr celého testování a předání aplikace zákazníkovi. Při hodnocení v jednotlivých kolech se hodnotí jednotlivé skripty a jednotlivé kroky ve skriptech. Ve chvíli kdy tester nalezne v jednom kroku chybu, zadá defekt. Defect projde procesem zvaným „ bug lifecycle“. Po skončení systémových testů následují testy akceptační. V tomto období dělají testeři podporu a dá se říct i prostředníka mezi zákazníkem a programátorem. Po skončení akceptačních testů se aplikace migruje z testovacího prostředí do ostrého provozu. Migrace se provádí v noci nebo o víkendu, kdy je zaručeno, že nebude využívaná klienty.

Testovací nástroje jsou důležitým pomocníkem při celém procesu testování. Používají je test koordinátoři, test analytici, testeři, analytici i vývojáři. V dobrém testovacím nástroji nalezneme několik modulů. Jedním z nejvyužívanějších je modul pro zadávání a správu defectů. Oblíbeným modulem je i modul na správu a tvorbu testovacích scénářů

a v neposlední řadě modul na samotnou realizaci testů. Nástroje jsou dostupné placené i free. Aplikace Mercury TestDirector for Quality Center je asi nejznámější a nejvyužívanější placený nástroj pro testování. Nástroj je určen pro správu a řízení všech fází procesu testování (správa požadavků, plánování testů, provádění testů, sledování řešení chyb, sledování pokrytí požadavků na systém). Podporuje současnou správu ručních i automatizovaných testů. Je přístupný přes webový prohlížeč a lze jej propojit s nástroji pro automatizované testování. V práci jsem se věnovala vlastnímu výzkumu free testovacích nástrojů, které bych doporučila běžným uživatelům i profesionálním testerům. Na počátku jsem stanovila kritéria pro výběr vhodného free testovacího nástroje. Hlavním kritériem byla free dostupnost. Nedefinovala jsme si však žádná kritéria týkající se služeb, které má nástroj obsahovat. Bylo důležité najít vyhovující nástroj s jakoukoli funkcí, která je potřebná při testování software. Dle tohoto kritéria jsem provedla rešerši, po jejímž vypracování jsem získala přehled free testovacích nástrojů. Jako nejlepší free testovací nástroj je JTrac. Po porovnání s ostatními nástroji na trhu, je zřejmé, že JTrac je vhodný nejen po stránce administrátorské a funkční, ale i jeho grafická část je uživatelky příjemná, srozumitelná a vhodná i pro nezkušené, začínající testery. JTrac může být zajímavý pro malé firmy, které nemají prostředky na nákup drahých softwarů. Celý software je také velmi jednoduše rozšiřitelný.

Testování je poměrně mladá disciplína, která se neustále rozvíjí a zdokonaluje. Při mé současné práci se testování věnuji jen okrajově, přesto ji vnímám jako jeden z nejdůležitějších článků v procesu vývoje software. Doufám, že mnoho mladých lidí, kteří se rozhodují o svém budoucím zaměstnání, moje práce zaujme a poskytne jim malou představu o této rozmanité profesi. Testování a hlavně testerům bych přála, aby bylo testování a testeři vnímáni jako plnohodnotní členové projektových týmů. Kvalitně otestovaný software je vizitkou každého dodavatele.

Literatura:

- BOROVCOVÁ, A. *Testování webových aplikací : základy testování* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW: <http://www.poeta.cz/Zaklady_testovani.pdf>.
- BUCHALCEVOVÁ, A; KUČERA, J. Hodnocení metodik vývoje informačních systémů z pohledu testování. *Systémová integrace*. 2008, roč. 15, č. 2, s. 42-54. ISSN 1210-9479.
- CRAIG, R.; JASKIEL, S. *Systematic software testing*. 1.ed. Boston : Artech House, 2002. 536 s. ISBN 1-58053-508-9.
- COPELAND, L. *A practitioner's guide to software test desing.* 1ed. Boston : Artech House, 2004. 294 s. ISBN 1-58053-791-X.
- DUSTIN, E.; PAUL, J.; RASHKA, J. *Automated software testing : introduction, management, and performance*. 9.ed. Reading : Addison-Wesley, 1999. 575 s. ISBN 0-201-43287-0.
- HEINOVÁ, Tereza. 2010. *Testování software : bibliografický soupis.* Praha, 2010-01-30. 17 s.

  Seminární práce k předmětu Bibliografické rešeršní služby.
- ELFRIEDE, D. *Effective software testing : 50 specific ways to improve your testing*. 1ed. Boston : Addison-Wesley, 2002. 271 s. ISBN 0-201-79429-2.
- EVERETT, G.; MCLEOD, R. *Software testing testing across the entire software development life cycle*. 1.ed. Hoboken : N.J Wiley-Interscience, 2007. 261 s. ISBN 978-0-471-79371-7.
- KHAN, R.; MUSTAFA, K. *Software testing concepts and practices.* Oxford : Alpha Science International, 2007. 293 s. ISBN 1-842-65367-9.
- LOVELAND, S.; at. al. *Software testing techniques : finding the defects that matter*. 1.ed. Hingham : Charles River Media, 2004. 362 s. ISBN 1-58450-346-7.
- MCGREGOR, J.; SYKES, D. *A practical guide to testing object-oriented software*. 1.ed. Boston : Addison-Wesley, 2001. 393 s. ISBN 0-201-32564-0.

- NAGY, R. Automatizované generovanie testovacích scenárov na základe špecifikácie v jazyku UML. *Automatizace*. 2005, roč. 49, č. 3, s. 194-197. ISSN 0005-125X
- PATTON, R. *Testování softwaru*. 1.vyd. Praha : Computer Press, 2002. 313 s. ISBN 80-7226-636-5.
- PERRY, W. *Effective methods for software testing*. 3.ed. Indianapolis : Wiley, 2006. 973 s. ISBN 978-0-764-59837-1.
- PYROCHTA, T. *Navrhování testovacích prostředků a metod pro aplikace informačních systémů v prostředí klient - server = Design of testing methods for information systems in client - server environment*. Brno : Vysoké učení technické, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2002. 29 s. ISBN 80-214-2231-9.
- RANKIN, C. The Software Testing Automation Framework. *IBM systems journal*. 2002, vol. 41, no. 1, s. 126. ISSN 0018-8670.
- ROBBINS, J. *Ladění a testování aplikací pro .NET a Windows*. 1.vyd. Praha : Grada, 2004. 646 s. ISBN 80-247-0774-8.
- RŮŽIČKA, D. FURPS –dimenze kvality software [online] 2009. [cit. 2011-20-6]. Dostupný z WWW: < http://www.testqa.cz/furps--dimenze-kvality-software-1-dil.html.>
- RUNESON, P. Software Testing - A Survey of Unit Testing Practices. *IEEE software*. 2006, vol. 23, no. 4, s. 22. ISSN 0740-7459.
- *Saenik.com : Co je to chyba.* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW: <http://www.saenik.com/modules.php?name=News&file=article&sid=5>.
- *Saenik.com : Druhy testování.* [online]. 2007 [cit. 2010-01-15]. Dostupný z WWW: <http://www.saenik.com/modules.php?name=News&file=article&sid=4>.
- *Saenik.com : Případ, scénář, skript aneb testovací dokumentace* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW:

  <http://www.saenik.com/modules.php?name=News&file=article&sid=6 >.
- SÝKORA, J. Výkonostní testy podnikových aplikací. *IT Systems*. 2008, roč. 10, č. 12/2008, s. 58-60. ISNN 1802-002X.

- *TestQA.cz : vše o testování software a quality assurance* [online]. Integra-IT, 2009 , 2009 [cit. 2010-01-15]. Dostupný také z WWW: <http://testqa.cz/uvodni-stranka.html>.
- *Testování softwaru* [online]. 2009 [cit. 2010-01-15]. Dostupný také z WWW: <http://testovanisoftwaru.blogspot.com/>.
- *Test Republic : ecosystem for Software Testing Professionals* [online]. Edista, 2010 [cit. 2010-01-15]. Dostupný také z WWW: <http://www.testrepublic.com/>.
- THOMAS, D.; HUNT, A. *Pragmatic unit testing in Java with JUnit*. 1.ed. Raleigh : Pragmatic Bookshelf, 2004. 159 s. ISBN 0-9745140-1-2
- VANĚK, Dušan. *Testing: typické rozdělení scope testování a spolupráce týmů testování (nad různými profily sad testů)* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW: < http://dusanvanek.webgarden.cz/testing/typicke-rozdeleni-scope-4.html>.
- VANĚK, Dušan. *Testing : deset klíčových principů procesu testování* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW: <http://dusanvanek.webgarden.cz/testing/deset-klicovych-principu-procesu-2.html>.
- VANĚK, Dušan. *Testing : Přizpůsobení sady Test Cases potřebám příslušného cyklu testování (pomocí Test Profiles)* [online]. 2007 [cit. 2010-01-15]. Dostupný také z WWW: <http://dusanvanek.webgarden.cz/testing/prizpusobeni-sady-test-cases-4.html>.
- VORÁČEK, K. *Výkonnostní testování webových aplikací .NET* 1.vyd. Praha : Grada, 2004. 251 s. ISBN 80-247-0822-1.

Evidence výpůjček

Prohlášení:

Dávám svolení k půjčování této bakalářské práce. Uživatel potvrzuje svým podpisem, že bude tuto práci řádně citovat v seznamu použité literatury.

V Praze, 5. srpna 2011

Tereza Heinová

Jméno	Katedra / Pracoviště	Datum	Podpis