VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPRÁVA PROJEKTŮ Z OBLASTI HUMAN-COMPUTER INTERACTION

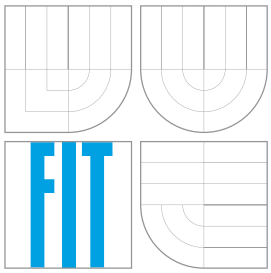
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER FARBIAK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPRÁVA PROJEKTŮ Z OBLASTI HUMAN-COMPUTER INTERACTION

HCI PROJECTS MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER FARBIAK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RUDOLF KAJAN

BRNO 2012

Abstrakt

Tato diplomová práce se zaobírá problematikou testování, především manuálním a beta testováním. Dále diskutuje oblast interakce mezi člověkem a počítačem a hledá možnosti efektivní správy projektů z této oblasti. Jako řešení problému je navržen projekt z kategorie project hostingu. Následovně je tento pojem vysvětlen a jsou představeny existující řešení. V praktické části je navržen a implementován systém pro správu projektů z oblasti HCI na základě předcházejícího výzkumu.

Abstract

This master's thesis deals with problems of testing, especially manual and beta testing. Further thesis discuss area of Human - Computer Interaction and looks for possibilities of effective managing of HCI projects. As solution project hosting area is proposed, explained and existing solutions are examined. In practical part of thesis system for managing HCI projects is designed and implemented based on previous research.

Klíčová slova

Web 2.0, RIA, Human-Computer Interaction, HCI, Project hosting, Testovanie, Beta testovanie, ASP .NET, Entity Framework.

Keywords

Web 2.0, RIA, Human-Computer Interaction, HCI, Project hosting, Testing, Beta testing, ASP .NET, Entity Framework

Citace

Peter Farbiak: Správa projektů z oblasti Human-Computer Interaction, diplomová práce, Brno, FIT VUT v Brně, 2012

Správa projektů z oblasti Human-Computer Interaction

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Rudolfa Kajana

.....
Peter Farbiak
17. mája 2012

Poděkování

Děkuji Ing. Rudolfovi Kajanovi za vedení a odbornou pomoc pri zpracování této diplomové práce.

Obsah

Kapitola 1

Úvod

Každoročne na Fakulte Informačních Technologií VUT v Brně, iných univerzitách a prácou nezávislých vývojárov vznikne veľké množstvo projektov. Mnohé z nich však skončia s veľkým množstvom chýb kvôli nesprávnemu otestovaniu alebo zostanú nepoznané verejnosťou pretože nie sú prezentované tak aby sa o nich verejnosť mohla jednoducho a rýchlo dozvedieť, získať a používať ich. Z týchto dôvodov je potrebná existencia nástroja, ktorá by podobné služby, teda zdieľanie projektov, možnosť testovania, diskusiu a podobne poskytoval.

Táto diplomová práca sa zaoberá práve vyššie zmienenými problémami a snaží sa nájsť a implementovať riešenie, ktoré ich je schopné eliminovať. Navyše sa práca nezaoberá všetkými typmi projektov. Namiesto toho sa špecializuje na oblasť Human - Computer Interaction. Cieľom tohto profilovania je dosiahnuť čo najefektívnejší systém pre konkrétnu oblasť záujmu a neutopiť tak projekty v zmesi všetkých možných. Existuje totiž množstvo vývojárov a užívateľov, ktorí vyhľadávajú aplikácie s vymedzenými hranicami záujmu. Aplikácia, ktorá vznikla pre potreby tejto práce (s názvom CrowdProof) sa snaží ponúknuť nástroj pre zdieľanie a objavovanie nových projektov práve ľuďom zaujímajúcim sa o interakciu človeka s počítačom. Myšlienkou projektu je vytvorenie komunity, ktorá bude skrz svoju tvorbu posúvať hranice HCI a dávať ľuďom nové možnosti pri používaní zariadení a aplikácií rozmanitých platforiem.

Ťažným bodom aplikácie CrowdProof je jednoduchosť. Snaží sa brať dobré a odstrániť zlé s podobne zameraných projektov a dať jej užívateľom možnosť zdieľať svoju tvorbu v čo najkratšom čase.

Pred samotnou implementáciou je samozrejme nutné vytvoriť teoretický základ preto v kapitole 2 sa čitateľ zoznámi s pojmami Web 2.0 a Rich Internet Application, ich prínosom z hľadiska zapojenia užívateľa pri vytváraní obsahu internetu a interaktivity, ktorú ponúkajú. Interaktivita s užívateľom je ďalším z ťažných bodov v aplikácii CrowdProof.

V kapitole 3 je predstavená základna koncepcia projektov z kategórie project hosting a následne pojednávané o troch z najznámejších projektov tejto kategórie. Na základe poznatkov získaných v tejto kapitole je neskôr navrhnutý dizajn a funkčnosť vlastného projektu.

Náväzne na túto kapitolu je stručne pohovorené o projektoch z oblasti Human-Computer Interaction a celkovo o tomto pojme, keďže práve pre projekty zamerané na HCI je určený vytvorený systém. 4

V samostatnej kapitole, konkrétne kapitole 5 je načrtnutá problematika a potreba správneho testovania aplikácií. Uvedené tu sú niektoré typy testovania a bližšie sa diskutuje o takzvanom beta testovaní, ktoré najbližšie definuje spôsob testovania, ktorý prebieha za

využitia aplikácie z oblasti project hostingu.

Šiesta kapitola 6 diplomovej práce potom za použitia všetkých zozbieraných informácií predstaví návrh aplikačnej časti. Predstavený bude diagram užitia tried so stručným popisom aktérov a základných systémov aplikácie a konečne návrh databázovej štruktúry nad ktorým je aplikácia vystavaná. Táto kapitola plynulo prechádza do ďalšej 7, v ktorej sú krátko popísané technológie použité pri implementácii. Po kapitole teoreticky rozoberajúcej nástroje použité pri programovaní prichádza kapitola praktická. Tá rozoberá akým spôsobom bola aplikácia implementovaná. Ukazuje zábery z hotovej aplikácie a demonštruje jej použitie.

Konečne kapitola 9 poukazuje na problémy pri tvorbe aplikácie a popisuje niektoré použité techniky testovania aby v kapitole 10 prešla k pojednaniu o možných rozšíreniach a návrhoch na zlepšenie.

Na konci dimplovej práce samozrejme nechýba záver, ktorý zhŕňa najdôležitejšie body, ktoré má práca za cieľ v jej čitateľovi a užívateľovi zanechať.

Kapitola 2

Web 2.0

Pod pojmom Web 2.0 je možné si predstaviť web, ktorého statický obsah bol nahradený dynamickým a vo vysokej miere je závislý a tvorený práve užívateľmi. Zo statickej internetovej stránky poskytujúcej informácie uverejnené autorom sa stáva samostatne sa vyvíjajúci informačný kanál, kde autor je postavený do roly akéhosi moderátora, ktorý koriguje sémantickú hodnotu webu.

Podľa vyššie uvedených kritérií sú internetové aplikácie najlepšie zapadajúce do kategórie Web 2.0 napríklad blog, sociálna sieť alebo wiki. Vďaka novým technológiám, orientovaným na užívateľa sa internet stal platformou poskytovania novej generácie služieb elektronickým spôsobom. Cez internet je dnes možné okrem prehliadania stránok tiež telefonovať, viesť videokonferenciu alebo sledovať televíziu.

Podľa [26] je možné webové aplikácie patriace pod Web 2.0 charakterizovať niekoľkými vlastnosťami. Dôležitým sa napríklad stáva **dizajn cielený na užívateľa**. Dizajn je tvorený takým spôsobom, aby plnil všetky potreby užívateľa a ponúkol mu širokú škálu možností ako prispôsobiť určité aspekty vrámci dizajnu. Jeden z najznámejších a najlepšie demonštrujúcich príkladov je prispôsobiteľná domovská stránka od spoločnosti Google iGoogle.

Aplikácie Web 2.0 zavádzajú nový pojem **crowd - sourcing**. Crowd - sourcing znamená, že príspevok každého užívateľa je dôležitý a tvorí vyššiu relevanciu webovej stránky. Príkladom sú blogy, ktoré vzhľadom k množstvu prispievateľov dokážu ponúknuť väčšie množstvo informácií ako tlačové médiá, z čoho vyplýva **spolupráca** užívateľov pri tvorbe webu. Umožňuje tak zbieranie veľkého množstva dát, ktoré pri určitej korekcii zodpovednou osobou vytvorí zmysluplný a bohatý zdroj informácií s **dynamickým obsahom**. Obsah nie je len v rukách tvorcu webu, naopak je dynamicky vytváraný činnosťou a príspevkami jeho užívateľov. Najlepším príkladom je známy web Wikipedia.

S crowd - sourcingom súvisí aj ďalšia vlastnosť aplikácií patriacich pod Web 2.0 a to je takzvaná **decentralizácia sily**. Tá spočíva v odtrhnutí webových služieb od nutnosti vedenia administrátorom a jeho implementáciou užívateľských požiadavkov. Užívateľ si môže vystavať stránku svojpomocne pomocou nástrojov, ktoré mu poskytne (súvisí s dizajnom cieleným na užívateľa).

Jednou z najdôležitejších charakteristík aplikácií Web 2.0 je skutočnosť, že definujú novú **platformu**, ktorá predstavuje nezávislosť na klientovi, možnosť pristupovať na internetové stránky z rôznych internetových prehliadačov bežiacich na rôznych operačných systémoch. Možnosti Webu 2.0 sa dokonca posúvajú ešte ďalej a zavádzajú **software ako službu**. Čo si predstaviť pod týmto pojmom budú najlepšie demonštrovať príklady ako napríklad Google Docs alebo pojem Cloud Computing. V jednoduchosti si teda môžeme predstaviť odpútanie sa od potreby inštalovať software na svoj počítač. Ten bude nainštalovaný na

výkonných serveroch, kde budú zároveň bežať všetky potrebné výpočty a osobný počítač tak len slúži ako nástroj na zobrazenie výstupu. Takýto prístup výrazne odľahčí lokálny počítač od výpočetných úloh a potreby výkonného hardwaru. Tento bod zasahuje do budúcnosti počítačových technológií a viac sa spája s nástupcom technológie Web 2.0 a to Web 3.0.

Posledným bodom v poradí, no nie dôležitosťou je **bohatý užívateľský zážitok**. Užívateľovi v dnešnej dobe nestačí len obsah, rovnako dôležitá je aj forma akou je podaný. Preto sa Web 2.0 snaží o atraktívnejší, rýchlejší a prirodzenejší užívateľský zážitok. Takéto správanie a vzhľad webu sa zjednotil pod názvom Rich Internet Application (RIA), čo v preklade znamená bohatá internetová aplikácia.

2.1 Rich Internet Application

Rich Internet Application (ďalej **RIA**) je webová aplikácia, ktorá sa snaží čo najvernejšie napodobniť užívateľský zážitok z desktopových aplikácií v prostredí internetu. K tomuto cieľu sa používajú rôzne pokročilé technológie a techniky. RIA aplikácie zavádzajú komponenty a koncepty podobné tým, ktoré je možné nájsť v klasických desktopových programoch. Príkladmi takýchto techník sú napríklad **MDI** koncept (**multiple document interface**), čo je možné predstaviť si ako hierarchiu okien, pričom okno potomka môže mať prístup k informáciám rodiča a byť im tak prispôsobené. Iným príkladom je komfort funkcií odpovedajúcim klasickému desktopovému riešeniu (použitie klávesových skratiek, kontextová nápoveda, *drag&drop* ovládanie) a ďalšie. [25]

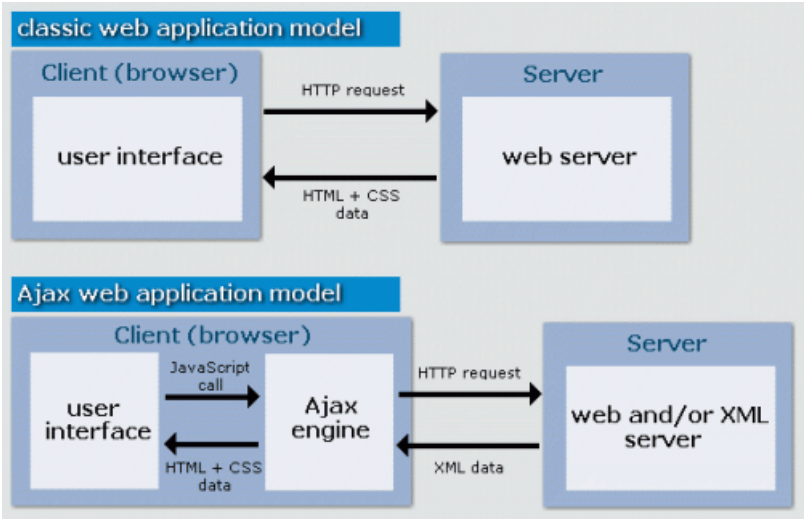
Od tradičných webových aplikácií sa teda RIA aplikácie odlišujú v niekoľkých bodoch: [12]

- **Priama interakcia** - Obsahujú širšie spektrum ovládacích prvkov oproti klasickým webom, dokonca v niektorých prípadoch môže užívateľ tieto prvky sám meniť.
- **Obnovenie časti stránky** - V štandardnom HTML spôsobí zmena stránky dotaz na server, ktorý túto zmenu spracuje a vráti späť celú aktualizovanú stránku. Za pomoci protokolu HTTP a jazyka HTML to je jediná cesta obnovy stránky. Pri tomto prístupe musí teda užívateľ čakať na spracovanie celej stránky aj pri nepatrnej zmene. Vrámci RIA sa tento problém rieši či už použitím cache mechanizmov, za pomoci virtuálnych strojov na strane klienta alebo najpopulárnejšie použitím vývojárskych metód ako **AJAX**.
- **Offline beh** - Aplikácie môžu ukladať svoj stav na strane klienta a byť tak použiteľné aj pri výpadku pripojenia k internetu.
- **Vplyv na výkon** - Vplyv na výkon aplikácie môže byť zvýšený tým, že dokáže spracovať požiadavky na strane klienta a nemusí sa tak dotazovať na server.

Medzi najpoužívanejšie technológie pri tvorbe RIA aplikácií patria **AJAX**, **Adobe Flash** a **Microsoft Silverlight**.

2.1.1 AJAX

Asynchronous JavaScript and XML, skrátene AJAX je koncept, navrhnutý pre RIA a ako taký pozostáva z viacerích webových technológií (HTML, JavaScript, DHTML, DOM). Princíp technológie AJAX najlepšie demonštruje obrázok 2.1.



Obrázok 2.1: Princíp práce modelu AJAX oproti klasickému Webu.

Základom AJAXu je objekt **XMLHttpRequest**, čo je v princípe objekt JavaScriptu a umožňuje asynchrónne volanie. Vďaka tomuto objektu môže AJAX vyvolať ľubovoľný počet nezávislých požiadavkov, ktoré sa spracujú na pozadí a ktoré ovplyvnia len patričné časti užívateľského rozhrania, bez nutnosti načítania celej stránky. [11]

2.1.2 Adobe Flash

Adobe Flash je multimediálna platforma používaná k pridaniiu animácií, videa a interaktivity webu. K svojmu výkonu potrebuje plugin FlashPlayer. Podľa spoločnosti Adobe až 98,9 % počítačov pripojených k internetu disponuje nainštalovaným nástrojom FlashPlayer [21]. Práve použitím FlashPlayeru sa stáva kompatibilným so všetkými internetovými prehliadačmi a operačnými systémami. Nad platformou Flash spoločnosť Adobe vybudovala viacero technológií, ktoré umožňujú vytvárať RIA weby. Medzi najvýraznejšie patrí SDK Adobe Flex, IDE Adobe Flash Builder vystavaný na základoch platformy Eclipse alebo aplikácia pre jednoduchú tvorbu webu Adobe Dreamweaver. [25], [21]

2.1.3 Microsoft Silveright

Microsoft Silveright je bezplatný plugin pre internetové prehliadače, ktorý ponúka interaktívny zážitok, bohaté biznis aplikácie a pohlcujúce mobilné aplikácie. (preklad oficiálneho textu spoločnosti Microsoft) [19]

Silveright je aplikačný rámec pre tvorbu a vývoj RIA a biznis aplikácií. Rovnako ako Flash potrebuje k svojmu behu nainštalovaný plugin a vďaka nemu je dostupný nezávisle na internetovom prehliadači alebo operačnom systéme. V začiatkoch sa zameriaval najmä na streamované médiá, súčasne je však rovnako cielený na grafiku a animácie. Užívateľské rozhranie v Silveright aplikáciách je deklarované za použitia **XAML** (eXtensible Application Markup Language) a programované za užitia podmnožiny **.NET** jazykov.

Okrem desktop aplikácií podporuje Silveright aj mobilné aplikácie a je jednou z dvoch vývojárskych platforiem mobilného operačného systému Windows Phone 7.

Veľkou devízou je zázemie spoločnosti Microsoft a integrácia s vývojovým prostredím Visual Studio. Vývoj nad platformou Silveright sa tak stáva veľmi pohodlným.

Aktuálna verzia je Silveright 5 vydaná v Decembri 2011.

Kapitola 3

Project Hosting projekty

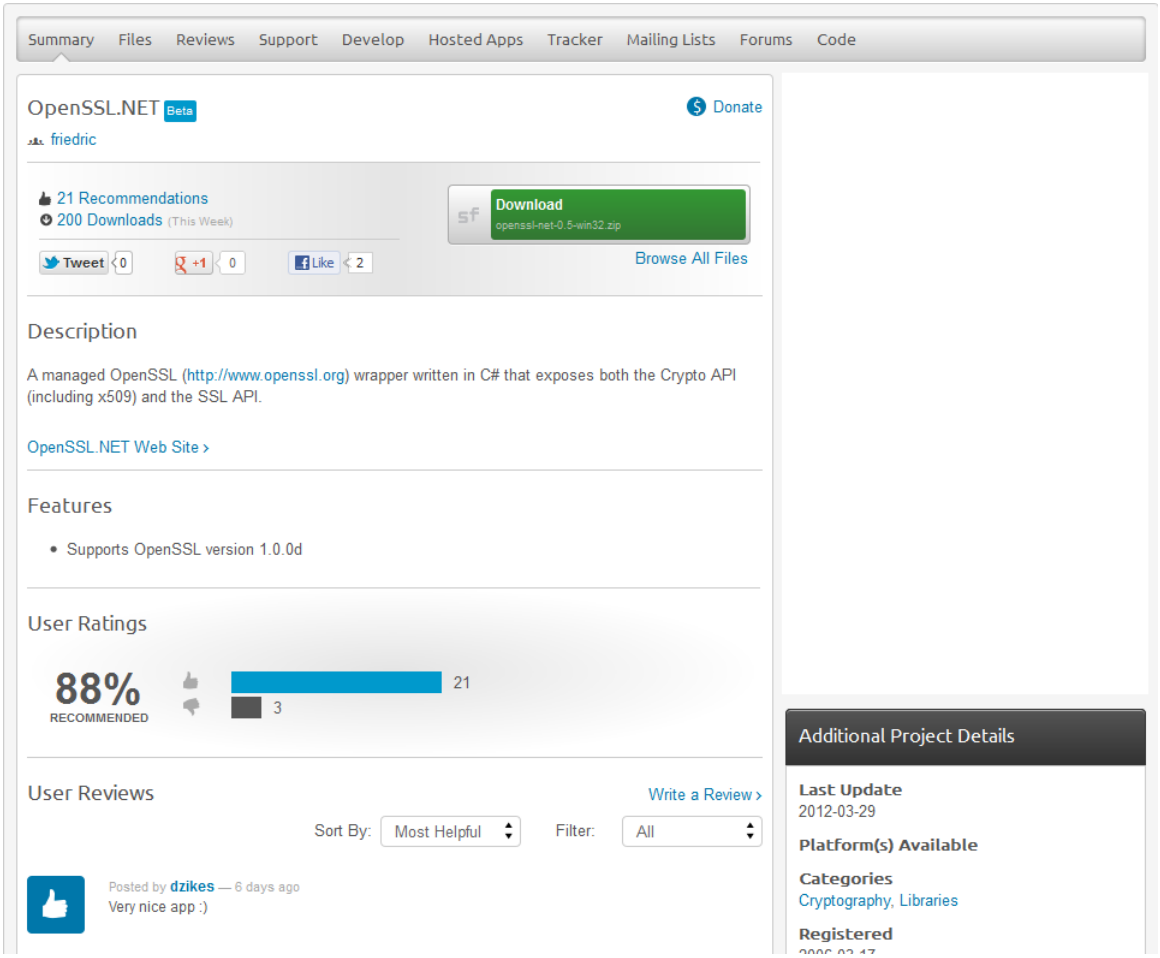
Project Hosting sú projekty, ktoré sa už podľa názvu zameriavajú na hosting projektov. Za pomoci takýchto nástrojov môžu vývojári jednoducho zdieľať svoje aplikácie so širšou verejnosťou a rovnako tak sťahovať a upravovať či rozširovať *open-source* projekty iných vývojárov. Pod názvom project hosting sa však neskrýva len jednoduché úložisko aplikácií. Väčšina z nich poskytuje tiež ďalšie možnosti využitia ako je *subversioning, bugtracking*, diskusné fóra alebo integráciu s nástrojmi pre konstrolu nad zdrojovými kódmi ako Mercurial alebo Team Foundation Server a ďalšie rozšírenia. Tým vzniká nástroj skrz, ktorý je možné aplikácie nielen jednoducho šíriť, ale tiež testovať, získavať nápady a postrehy od užívateľov, ktorý aplikáciu používajú a celkovo tak zvýšiť kvalitu vytvorených projektov. Veľkú popularitu zažívajú *project hosting* systémy na poli *open-source* vývoja. Viaceré z nich však umožňujú využitie ich služieb komerčným aplikáciám za poplatky. V nasledujúcich riadkoch uvediem pár najrozšírenejších a najznámejších projekov, ktoré sa project hostingu venujú a porovnám ich klady a zápory. Poznatky z jednotlivých projektov budú neskôr použité pri návrhu vlastnej aplikácie.

3.1 SourceForge

Projekt s názvom **SourceForge** je pravdepodobne známy každému vývojárovi, ktorý sa aspoň v minimálnej miere zaoberá témou open-source projektov. SourceForge je prvý projekt s podobným zameraním, ktorý ponúkol možnosť zdieľať, manažovať a kontrolovať open source projekty zdarma. Bol spustený už v novembri 1999 a dnes má 3,4 milióna užívateľov (vývojárov), ktorý vytvorili viac ako 324 000 projektov.

Medzi najväčšie prednosti projektu SourceForge patrí veľká vybavenosť. SourceForge ponúka najširšie možnosti čo sa týka rôznych rozšírení a nástrojov. Podporuje mnoho version-control systémov, má Wiki, fóra, mailinglist, štatistiky alebo možnosť nastaviť web stránku pre projekt a veľa ďalších. V minulosti bolo nevýhodou pomerne komplikované a neprehľadné užívateľské rozhranie, po obmene dizajnu však SourceForge túto nedokonalosť odstránil. Výhodou a zároveň nevýhodou môže byť množstvo užívateľov a projektov a to najmä v prípade ak ste začínajúci vývojár a pridávate svoje prvé projekty. Tieto môžu zostať nepovšimnuté vďaka každodennému prílevu nových projektov.

Zaujímavosťou je, že SourceForge je blokovaný v niektorých krajinách ako Kuba, Irán, Severná Kórea, Sudán a Sýria. Dočasne SourceForge blokoval aj Čínu a v tejto dobe čelil výraznej kritike a mnohým hackerským útokom.



Obrázok 3.1: Ukážka služby SourceForge.

3.2 Google Project Hosting

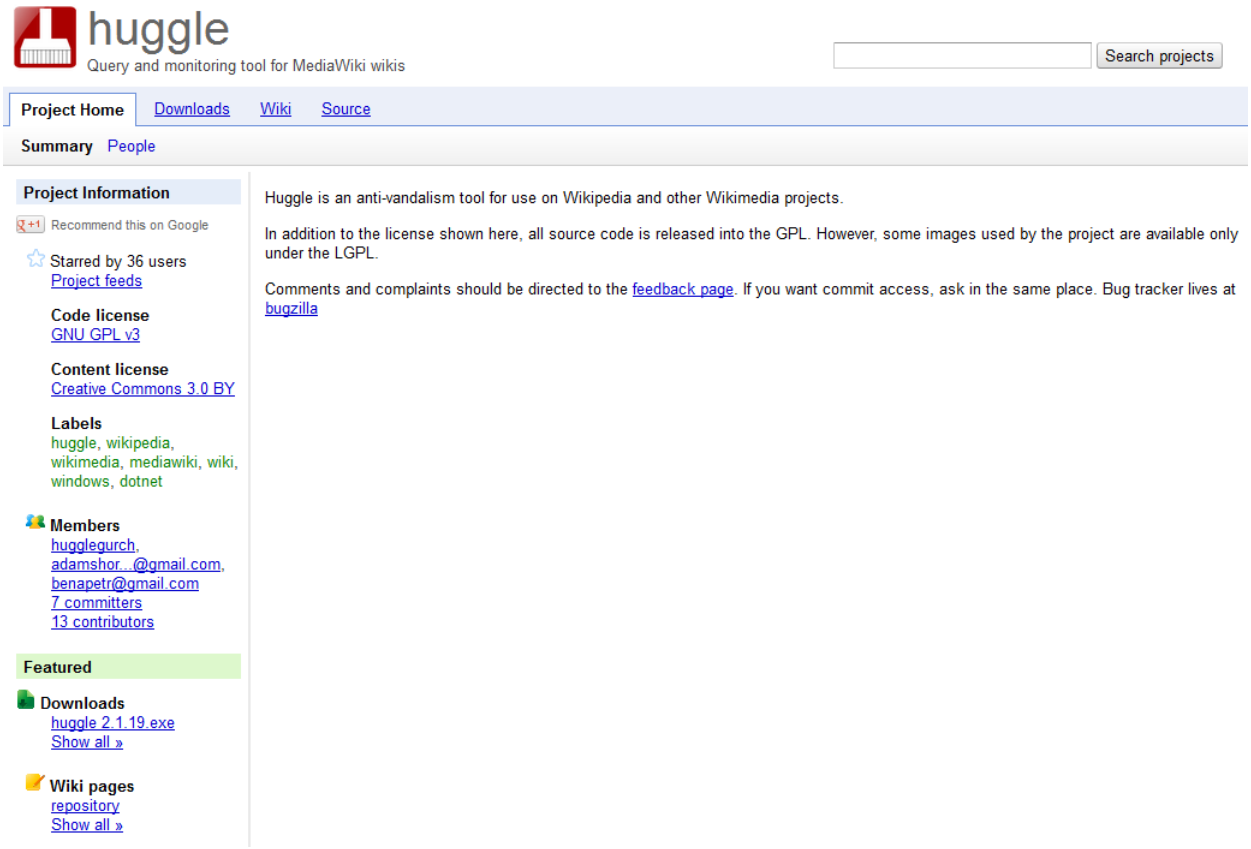
Google Project Hosting je produkt vytvorený gigantom Google a je súčasťou projektu Google Code, čo je web mierený na vývojárske nástroje, API a technické zdroje spoločnosti Google. Google Code ponúka napríklad API pre služby ako YouTube, Google Maps alebo Google Apps.

Výhodami Google Project Hosting je samozrejme jeho zázemie spoločnosti Google a tým pádom istota v podpore tohto projektu. Ďalšou veľkou výhodou je užívateľské prostredie. Zo všetkých tu zmienených projektov je práve Google Project Hosting najjednoduchšie ovládateľný, prehľadný a intuitívny. Jeho sila je v jednoduchosti a aj keď nepodporuje všetky nástroje čo SourceForge vývojár tu nájde všetko potrebné pre project hosting. Príkladom je podpora verzovacích systémov, bugtracking, fórum a podobne.

To čo môžeme na jednej strane považovať za kladnú vlastnosť Google Project Hosting a to príslušnosť ku korporácii Google je aj jeho najväčšou nevýhodou. Problémom sa totiž stáva fakt, že pri každej akcii musí byť užívateľmi prihlásený svojim Google účtom. A to aj vtedy ak chce len pridať nájdený bug do zoznamu chýb. Toto môže mnohých potenciálnych užívateľov odradiť.

Aj napriek tomuto faktu sa Google Project Hosting svojou jednocuchosťou, prehľadnosťou a vysokou mierou použiteľnosti stáva ideálnym nástrojom pre začiatočníkov, ale rovnako aj pokročilých užívateľov hľadajúcich mieto pre svoje projekty.

Zaujímavosťou je, že je blokovaný v rovnakých krajinách ako SourceForge. Ďalšou zaujímavosťou, že Google Code nezobrazoval na svojich stránkach žiadne reklamy. Tie sa nachádzali na stránkach SourceForge aj Codeplex.

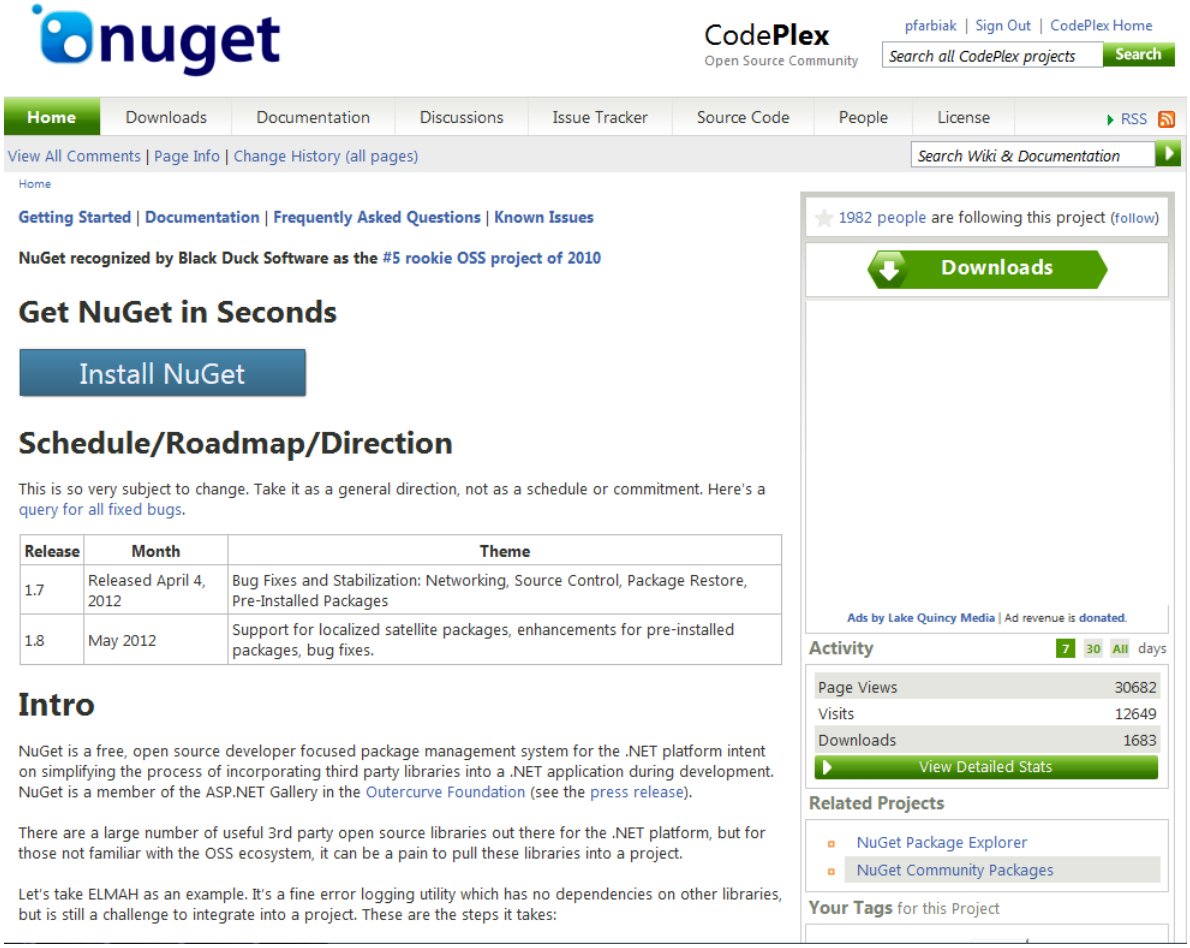


Obrázok 3.2: Ukážka služby Google Code.

3.3 Codeplex

CodePlex je projekt z dielne spoločnosti Microsoft,poskytuje podobnú funkčnosť ako Google Project Hosting. Oproti svojim konkurentom ponúka podporu Team Foundation Server. Nie je obmedzený na určité typy projektov no v drvivej väčšine sú projekty uložené na tomto webe úzko spojené s technológiami spoločnosti Microsoft, čo znamená, že projekty, ktoré tu môžete nájsť sa najčastejšie týkajú platformy .NET, programovacích jazykov podporovaných touto platformou, Silverightu a podobne.

Výhodami sú jednoduché a príjemné užívateľské rozhranie a rovnako dostačujúca funkcionalita. Naopak nevýhodou je pomerne úzke škálovanie projektov ani nie tak zo strany spoločnosti Microsoft ako vlastníka tejto služby, ale z pohľadu užívateľov.



Obrázok 3.3: Ukážka služby CodePlex.

3.4 Ďalšie

GitHub je pomerne rozšírený projekt. Jeho nevýhodou je spoplatnenie v prípade ak projekt má viac ako 5 vývojárov, jeho užívateľské rozhranie je menej intuitívne a nie je celkom vhodný pre začiatočníkov.

Zaujímavým a pomerne populárnym projektom je **Launchpad**. Pravdepodobne najzaujímavejším projektom, ktorý Launchpad zastrešuje je Ubuntu (voľne šíriteľný operačný systém založený na Linuxe).

V dnešnej dobe existuje viacero webových nástrojov, kde je možné svoj projekt umiestniť, sprístupniť ho verejnosti a zároveň získavať informácie od užívateľov. Všetky zo spomenutých projektov poskytujú základnú funkčnosť, ktorú vývojár očakáva. Výber nástroja, ktorý vývojár použije teda pravdepodobne zostane na osobných sympatiách, špecifických funkciách v ktorých sa môžu líšiť alebo zameraniu projektov (napríklad špecifikovanie CodePlex na projekty založené na technológiách spoločnosti Microsoft). Práve špecifikovaním na určitý typ projektov bude odlišná táto práca. Jej zameranie bude konkrétne a to na projekty z oblasti Human-Computer Interaction.

Kapitola 4

Human-Computer Interaction

Human-Computer Interaction (ďalej **HCI**) je možné preložiť ako interakcia človeka s počítačom. Je to obor skúmajúci vzťah človek - počítač ich, ich vzájomnú interakciu a komunikáciu. Aj keď sa HCI priamo viaže na mnoho disciplín, veľmi dôležitá je väzba s počítačovou vedou a systémovým dizajnom, pretože zahŕňa dizajn, implementáciu a hodnotenie interaktívnych počítačových systémov v kontexte činnosti užívateľov. [4] Z hľadiska počítačovej vedy ide špecificky o interakciu medzi jedným alebo viacerími ľuďmi s jedným alebo viacerími počítačovými systémami. [7] Prehľad disciplín spätých s problematikou HCI je možné vidieť na obrázku 4.1

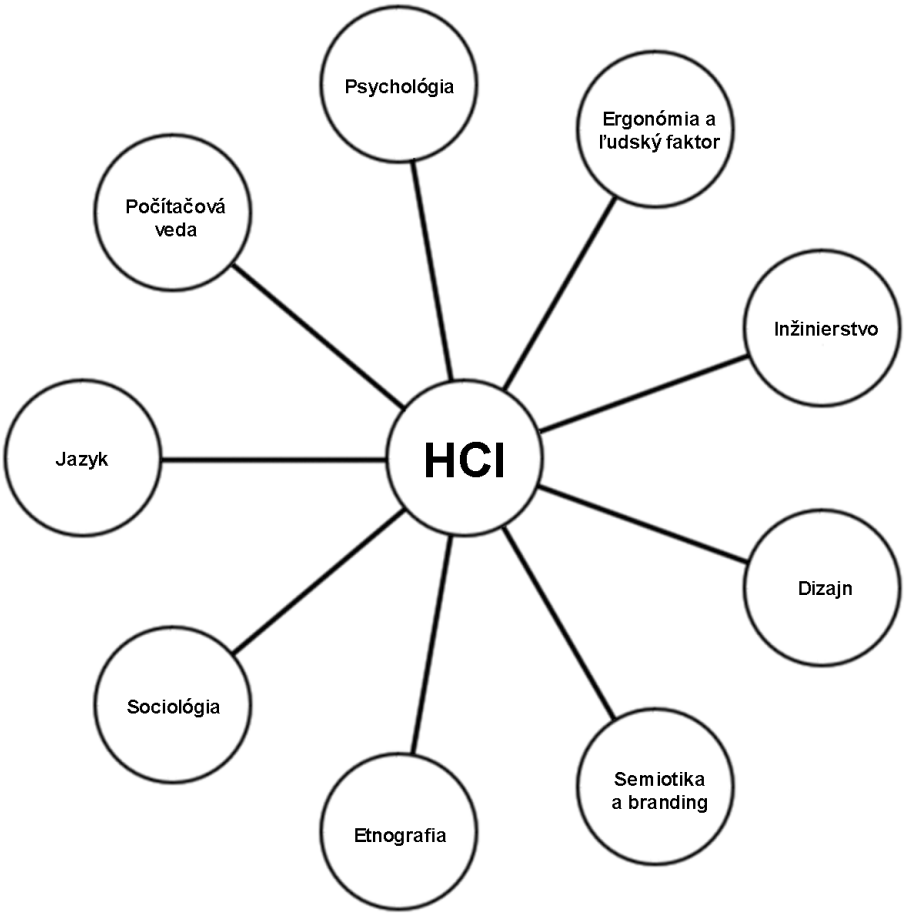
Interackia medzi človekom a počítačom sa deje na úrovni užívateľského rozhrania, čo zahŕňa ako hardware, tak software. Základným cieľom HCI je zdokonaliť tieto rozhrania tak aby pre užívateľa bola interakcia čo najjednoduchšia, zrozumiteľná a efektívna. Dlhodobým cieľom HCI je potom navrhnúť systém, ktorý minimalizuje bariéru medzi ľudským kognitívnym modelom toho čo človek chce dosiahnuť a počítačovým chápaním zadaných úloh užívateľom.

4.1 Užívateľské rozhrania

Užívateľské rozhranie je styčným bodom interakcie človek - počítač a záujmovým bodom oblasti HCI. Jeho cieľom je efektívne ovládanie stroja a jeho vhodnej spätnej väzby človeku. Užívateľské rozhranie zahŕňa ako hardware, tak software. V nasledujúcich kapitolách sú krátko diskutované tri, momentálne používané, typy užívateľských rozhraní. Vedci však stále pracujú na nových spôsoboch interakcie človeka so strojom. Nereálne už nie je ani takzvané „mozgovo-počítačové rozhranie“ (z angličtiny brain-computer interaction). V roku 2009 rakúska spoločnosť Guger Technologies OG vydala prvý komerčný systém pre hláskovanie za pomoci ovládania za pomoci BCI. V marci 2012 dokonca firma zverejnila videá, na ktorých demonštrujú za pomoci BCI ovládanie populárnej online hry World of Warcraft. (odkaz na video je možné nájsť v dodatku C)

4.1.1 Grafické užívateľské rozhranie

Je rozhranie, ktoré používa okná, ikony, menu a ďalšie grafické prvky pričom je ovládané vstupnými perifériami (typicky myš a klávesnica). Uľahčuje prácu s počítačom tým, že pre užívateľa oddeľuje logické vrstvy od prezentačnej. Z toho vyplývy, že jednoduchou interakciou človeka s počítačom skrz **grafické užívateľské rozhranie** (**GUI**) nie je možné urobiť



Obrázok 4.1: Disciplíny späté s oblasťou Human - Computer Interaction.

syntakticky nesprávnu operáciu, pretože o logiku danej akcie sa stará program v pozadí a akciu vykoná alebo upozorní na prípadnú chybu. Prvé grafické užívateľské rozhranie uviedla spoločnosť Xerox Alto v roku 1973 avšak populárne sa stalo až s nástupom spoločnosti Apple a ich osobným počítačom Macintosh.

4.1.2 Zvukové užívateľské rozhranie

Zvukové užívateľské rozhranie (**VUI**) umožňuje interakciu človeka s počítačom na základe zvuku/reči v automatizovaných službách. VUI má tri základné elementy výzvu, gramatiku a logiku dialógu. Výzvy sú všetky nahrávky prehrávané užívateľovi počas dialógu. Gramatika alebo gramatiky udávajú rozsah toho, čo užívateľ môže odpovedať na výzvu. Systém dokáže porozumieť len tým frázam, ktoré sú súčasťou gramatiky. Konečne logika dialógu definuje reakcie systému na odpovede užívateľa. [3]

4.1.3 Multimodálne užívateľské rozhranie

Multimodálny systém dokáže spracovať dva a viac vstupných kanálov ako reč, dotyk, obraz, vstup daný polohovacím zariadením a podobne. Tento typ rozhrania pravdepodobne

Obrázok 4.2: Osobný počítač Macintosh spoločnosti Apple.

určuje smer akým sa budú vyvíjať technológie v najbližších rokoch.[23] Skvelým príkladom je napríklad aplikácia Siri spoločnosti Apple, ktorá je schopná reagovať na hlas, textový vstup ale aj dotykovú obrazovku. Rovnako tak výstupy aplikácie sú zvukové a zároveň textové. Aplikácia dokáže simulovať rozhovor a reagovať na požiadavky užívateľa jednoduchými odpoveďami, ale aj interakciou s inými aplikáciami ako je znázornené na obrázku 4.3

4.2 Problémy HCI

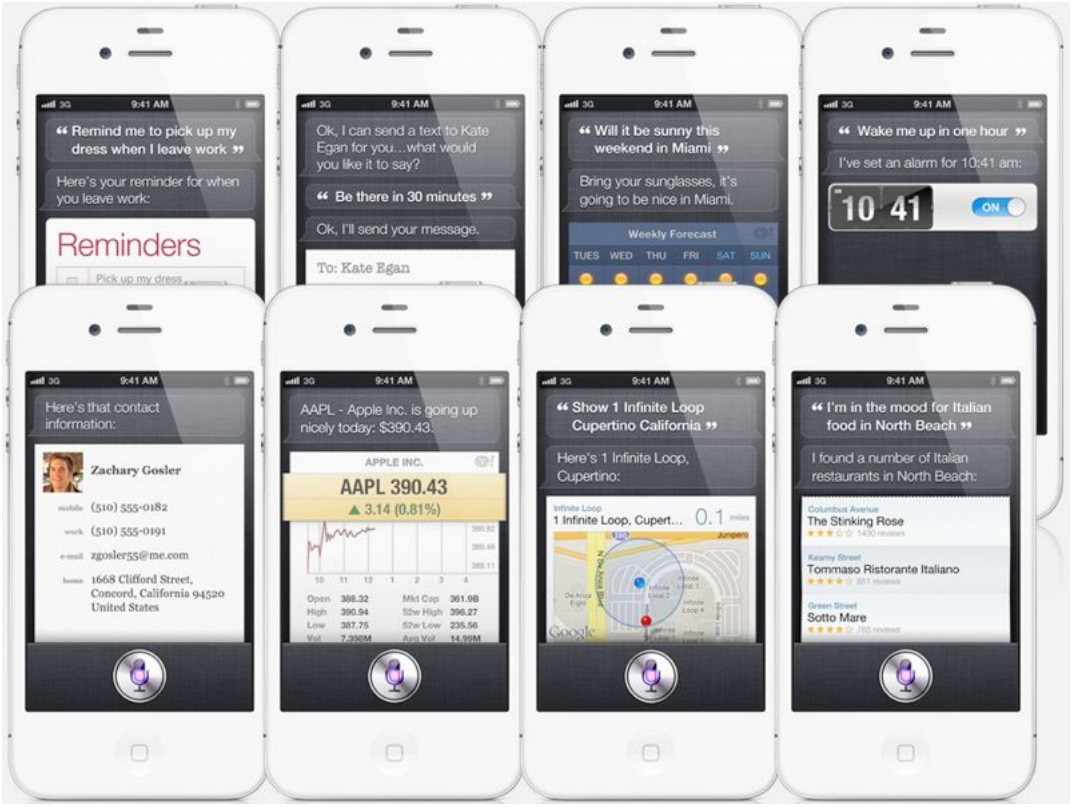
Vývoj v oblasti Human - Computer Interaction už zo svojej podstaty, interakcie medzi dvomi rôznymi entiatmi, čelí množstvu problémov, na ktoré je nutné brať ohľad pri návrhu rozhraní. Niekoľko závažných je uvedených v nasledujúcej kapitole.

4.2.1 Dostupnosť

V populácii je veľké percento ľudí so špeciálnymi potrebami. Podľa [18] tvorí populáciu vo Veľkej Británii 20 % ľudí do 16 rokov, 15 % ľudí nad 65 rokov a 10 % ľudí s postihnutím. Podobné štatistiky sa dajú očakávať aj v iných krajinách západného sveta. Úlohou užívateľského rozhrania je sprístupniť komfortné používanie aplikácií čo najväčšej škále užívateľov vrátane tých, ktorý potrebujú špeciálne technológie. Problémom zostáva fakt, že vrámci skupiny s potrebou špeciálneho prístupu k užívateľskému rozhraniu existuje vysoká variabilita disfunkcií (zraková, sluchová, rečová, chýbajúce končatiny a podobne). Z tohto dôvodu je otázka dostupnosti v oblasti HCI veľmi náročnou. Odpoveďou by mohol byť vývoj a nasadzovanie nových multimodálnych užívateľských rozhraní v čo najširšej miere.

4.2.2 Kultúrny kontext

Dôležitým aspektom pri návrhu užívateľského rozhrania je variabilita kultúr. Základným bodom kultúr je jazyk a písmo. Vďaka štandardizovaným znakovým sadám je možné dobre



Obrázok 4.3: Ukážka možností aplikácie Siri spoločnosti Apple. Jednotlivé telefóny zobrazujú kooperáciu Siri s rôznymi aplikáciami.

zvládať túto rôznorodosť no je treba brať ohľad aj na ďalšie faktory kultúrnych rozdielov. Musia tak byť tiež zachytené faktory ako čítanie zľava doprava verzus čítanie zprava doľava, rozloženie textu a ostatných užívateľských komponent a podobne. Napríklad podľa [18] sú preferencie vzhľadu internetových stránok v jednotlivých krajinách odlišné a to nasledovne:

- Užívatelia zo Španielska, Anglicka a Francúzska majú radi stránky s výrazným množstvom grafiky
- Nemecký užívatelia na druhej strane nemajú radi grafiku ani príliš farebné stránky
- Užívatelia z Číny preferejú čo najviac farieb a ich efektivita pri práci so stránkou sa zvýšila v prípade, že informácie boli prezentované obrazovou formou
- Americký užívatelia sú na rozdiel od čínskych efektívnejší pri použití alfanumerických znakov

4.2.3 Použiteľnosť

Použiteľnosť je jednou zo základných podmienok akceptácie systému ako celku užívateľmi. Podľa [22] je použiteľnosť definovaná piatimi bodmi

1. **Naučiteľnosť** - Užívateľ by sa mal dokázať systém naučiť jednoducho a byť schopný dosiahnuť vysokej efektivity pri práci v čo najkratšom čase.

2. **Účinnosť** - Systém by mal byť efektívny, takže užívateľ, ktorý sa naučí systém ovládať by mal dosahovať vysokú produktivitu.

3. **Zapamätateľnosť** - Systém by mal byť ľahko zapamätateľný, takže užívateľ by nemal mať problém vrátiť sa k používaniu systému aj po dlhšej dobe.

4. **Chyby** - Systém by mal mať nízke percento chýb a z chýb, ktoré sa vyskytnú sa musí jednoducho zotaviť. Zároveň sa nemôžu vyskytovať žiadne katastrofické chyby.

5. **Spokojnosť** - Systém by mal byť príjemný na používanie, takže užívatelia sú spokojný pri jeho používaní.

Výskum v oblasti HCI je cielený na vývoj nových metodológií v dizajne, experimentovanie s hardwarovými zariadeniami, vytváranie prototypov softwarových systémov a skúmanie nových paradigmát pre interakciu s počítačom a počítačovým systémom.

Kapitola 5

Testovanie

Mimo samotného zverejnenia projektu a jednoduchého sprístupnenia verejnosti je najdôležitejšou funkciou project hosting systémov možnosť otestovať aplikáciu na reálnej vzorke užívateľov, jednoducho zistiť chyby a nedostatky programu a rovnako jednoducho získať spätnú väzbu a návrhy na zlepšenie od užívateľov. Táto kapitola sa bude venovať práve problematike testovania a to najmä manuálneho testovania užívateľom za reálneho behu aplikácie.

5.1 Motivácia

Každoročne viacero školských projektov na Fakulte Informačních Technologií v Brně, iných univerzitách alebo nezávislých projektov napriek vydanému úsiliu vývojárov zlyhá. V mnohých prípadoch má tento fakt spoločného menovateľa a tým je nedostatočné otestovanie aplikácie. Z mnohých dôvodov sa táto skutočnosť objavuje nielen na univerzitnej alebo neziskovej scéne, ale rovnako na scéne komerčnej.

Jedným z najčastejších dôvodov býva nedostatok času respektíve nesprávna analýza projektu z hľadiska času potrebného na jeho vytvorenie alebo nevhodné rozvrhnutie inak správne odhadnutého času pri procese vývoja aplikácie.

Ďalšou možnosťou neúspechu je neotestovanie aplikácie ako celku. Zložitejšie aplikácie sa skladajú z niekoľkých modulov, projektov, tried, jednoducho prvkov, ktoré musia spolupracovať bezchybne v jednom celku. Tieto prvky väčšinou nevyvíja jedna a tá istá osoba, naopak každý ucelený prvok môže vytvárať jeden alebo viac ľucí. Preto sa môže stať a často sa stáva, že všetky prvky samostatne fungujú bez problémov, no po ich prepojení v jeden celok aplikácia zlyháva.

Ďalším z dôvodov je to, že veľa vývojárov berie tento krok za takmer bezvýznamný a pred jeho započatím je aplikácia v podstate dokončená. Toto je však veľký omyl, ktorý sa jednoducho vyvracia zlyhaním aplikácie počas jej behu. V niektorých spoločnostiach venujú testovaniu aplikácie až 30-40 % z celkového času stráveného na projekte a napríklad pri vývoji komponent do lietadiel sa venuje testovaniu a dokumentácii 90-95 % z celkového času stráveného na projekte. Toto je však extrémna situácia, kde jedna chyba aplikácie môže mať nedozierne následky na ľudských životoch.

Dôležitosť testovania v dnešnej dobe, keď pomocou počítačových, webových, mobilných aplikácií platíme účty, kupujeme si lístky na autobus, koncert alebo kontrolujeme či je naše dieťa v škole je testovanie jedným z najdôležitejších a pravdepodobne najdôležitejším krokom pri vývoji softwaru.

Typy testovaní je možné rozdeliť do viacerích kategórií na základe úrovne testovania, vlastnosti aplikácie na ktorú sa test zameriava alebo prístupu k testovaniu. Čo je testovanie je, čo je chyba a koľko stojí, niektoré základné kategórie a aspekty diskutujú nasledujúce kapitoly.

5.2 Čo je testovanie a jeho ciele

Testovanie softwaru je proces alebo séria procesov navrhnutá k uisteniu sa, že zdrojové kódy robia to, k čomu boli navrhnuté a nerobia nič, čo nebolo zámerom. Software by mal byť predvídateľný, konzistentný a neprodukovať žiadne prekvapenie pre užívateľa. [5]

Cieľom testovania je otestovať kód tam, kde je vysoká pravdepodobnosť odhalenia chyby a demonštrovať, že software korešponduje s požiadavkami uvedenými v produktovej špecifikácii vzhľadom k funkcionalite, vlastnostiam a výkonu. [1]

Ciele testovania sa dajú rozdeliť do dvoch kategórií a to na priame a nepriame. Priame ciele sú odhaliť a identifikovať čo najviac chýb a priviesť systém na akceptovateľnú úroveň kvality v určenom finančnom a časovom limite. Nepriamym cieľom je napríklad vytvoriť záznam o softwarových chybách používaný k prevencii chýb. [20]

Podľa [1] by mal dobre navrhnutý test spĺňať nasledujúce charakteristiky:

- Softwarový tým by mal predstaviť formálne technické správy.
- Testovanie začína na úrovni komponent a pokračuje k integrácii celého systému.
- V rôznych bodoch v čase je potrebné použiť rôzne techniky testovania.
- Testovanie je vykonané vývojármi softwaru (pri veľkých projektoch) a nezávislým testovacím týmom.
- Ladenie (debugging) musí byť zahrnuté v každej stratégii testovania.

5.3 Čo je chyba a koľko stojí

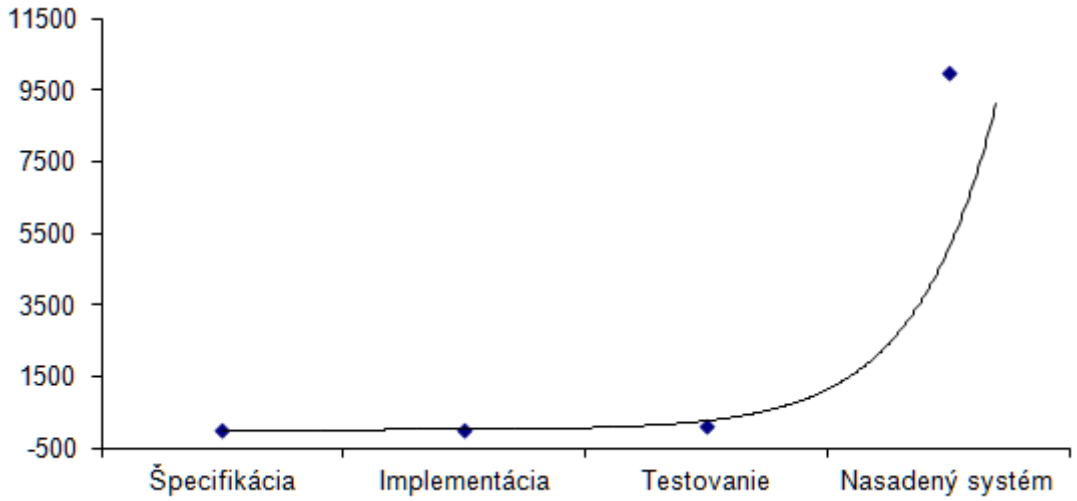
Podľa [24] sa stáva chyba chybou vtedy keď jedna alebo niekoľko z nasledujúcich podmienok je pravdivých :

1. Software nerobí niečo, čo by podľa produktovej špecifikácie mal.
2. Software robí niečo, čo by podľa produktovej špecifikácie nemal.
3. Software robí niečo, čo produktová špecifikácia nepopisuje.
4. Software nerobí niečo, čo produktová špecifikácia nepopisuje, ale mala by.
5. Software je nezrozumiteľný, ťažko používateľný, pomalý alebo podľa testera s ním koncový užívateľ nebude spokojný.

Z predchádzajúcich riadkov je možné jednoducho usúdiť, že prvým krokom pri testovaní sa stáva už tvorba špecifikácie produktu. A práve v tejto fáze je možné chyby najjednoduchšie nájsť a najmä odtrániť. Obecne platí, že čím neskôr v procese vývoja softwaru sa chyba nájde tým náročnejšie a nákladnejšie je jej odstránenie.

Ron Patton v [24] hovorí, že chyba odstránená vo fáze špecifikácie stojí jednu jednotku peňazí. Odstránenie tej istej chyby pri vývoji softwaru by stálo desať jednotiek, odstránenie

vo fáze testovania sto jednotiek a odstránenie tejto chyby po nasadení softwaru môže stáť tisíce až milióny jednotiek peňanzí v závisloti od rozšírenia softwaru a typu chyby. Túto závislosť zobrazuje graf 5.1. Z grafu je jasne vidieť závislosť ceny odstránenia chyby k fáze



Obrázok 5.1: Graf zobrazujúci nárast ceny opravy chyby v jedntolivých fázach vývoja preložený exponenciálnou funkciou.

projektu a zároveň času stráveného od začiatku projektu k danému bodu v čase jeho vývoja. (Vývojom sa v tomto prípade myslí celý životný cyklus projektu, nie len písanie kódu aplikácie.) Rovnako je možné pozorovať obrovský nárast práve medzi fázou testovania a fázou keď je systém plne nasadený. Graf je preložený exponenciálnou funkciou, ktorá najviac vystihuje zmenu nárastu ceny v priebehu času.

5.4 Úrovne testovania

Základné úrovne testovania sú definované ako súbor testov, ktorými je software testovaný na danom stupni podrobnosti. Podľa toho v akej fáze a s akým časovým odstupom od spísania kódu sa testovanie vykonáva ho delíme do piatich základných úrovní.

5.4.1 Testovanie programátorom

Je testovanie bezprostredne po vytvorení programového kódu. Ide o najzákladnejšie testovanie aplikácie, kde progamátor zisťuje, či je aplikáciu možné preložiť a následne spustiť. Zároveň testuje korektné, prípadne niektoré nekorektné vstupy. Ďalší krok vrámci tejto fázy testovania je odovzdanie ucelenej časti programového kódu inému programátorvi, ktorý pripraví na základe obdržanej dokumentácie vlastné testy. [6]

5.4.2 Testovanie jednotiek

Anglicky *unit testing*. Testovanie jednotiek verifikuje funkcionalitu izolovaných častí softwaru. V závislosti na použitom programovacom paradigmate, technológii, konkrétnom kontexte a podobne môže ísť o väčšie či menšie časti aplikácie. Napríklad v objektovo orientovanom

prostredí na úrovní tried alebo projektov. Testovnie jednotiek je presne definované v IEEE štandarde pre softwarové testovanie jednotiek (IEEE1008-87). Testovanie jednotiek typicky prebieha na základe prístupu testovania bielej skrinky, za pomoci ladiacich nástrojov a môže zahŕňať účasť programátora. [8]

Existuje viacero dôvodov prečo testovanie jednotiek používať. Najdôležitejší je fakt, že veľkosť jedného modulu je niekoľko násobne menšia ako veľkosť celého systému. Toto umožňuje testerovi lokalizovať chybu oveľa rýchlejšie a efektívnejšie. Zároveň sa eliminujú chyby, ktoré vznikajú interakciou viacerích modulov. [1]

Problémom testovania jednotiek je, že vstupy a výstupy dokáže často prijímať alebo spracovávať len hlavný program, ktorý je mimo daného modulu. K vyriešeniu tohto problému za používa takzvaný „stub“. Stub vytvára rozhranie pre modul a väčšinou je veľmi jednoduchý. V prípadoch kedy jednoduchý stub nemôže nahradiť funkcionalitu hlavného programu sa testovanie modulu zväčša vynecháva a modul je testovaný až v ďalších fázach vrámci čiastočného alebo celého systému. [24]

Testovanie jednotiek má za úlohu odhaliť chyby ako porovnávanie rôznych dátových typov, nekorektné logické operácie, nekorektné porovnávanie premenných, nesprávne ukončovanie cyklov a podobne.

5.4.3 Integračné testovanie

Integračné testovanie je systematické konštruovanie programu z jednotlivých modulov, pričom sú zároveň vykonávané testy odhaľujúce chyby spojené s rozhraniami medzi jednotlivými modulami.

Primárnym cieľom integračného testovania je otestovať rozhrania a uistiť sa, že sa nevyskytujú chyby v predávaní parametrov v momente keď jeden modul vyvoláva druhý. Po každom integračnom kroku je otestovaný čiastočný systém. Výsledkom integračného testovania je stav, keď sú všetky moduly integrované do jedného systému. [1]

Existuje viac prístupov k integračnému testovaniu líšiacich sa v spôsoboch integrácie jednotlivých modulov a testovaním čiastkových systémov. Vhodné je spomenúť aspoň dva prístupy a to inkrementálny a regresný.

Inkrementálny prístup predstavuje postupné spájanie modulov, pričom v každom kroku je k čiastočnému systému pripojený ďalší model. Tento systém je otestovaný a sú odstránené prípadné chyby. [1]

Regresné testovanie je špeciálnym typom. Používa sa totiž nielen pri testovaní vrámci integračného testovania, ale zároveň aj na testovanie po pridaní funkčných zmien alebo opravách programu. Jeho význam je určiť, či tieto zmeny neovplyvnili iné aspekty programu. Zvyčajne je vykonávaný opätovným spúšťaním testov, ktoré boli použité pri prvotnom testovaní pred uvedením produktu. Regresné testovanie je dôležité najmä z dôvodu, že opravy a zmeny majú tendenciu byť náchylné k chybám oveľa viac ako originálny programový kód. [5]

5.4.4 Systémové testovanie

Aplikácia je testovaná ako jeden funkčný a kompletný systém. Systémové testovanie sa vykonáva v neskorších fázach vývoja, kde väčšina chýb by mala byť odhalená už počas testovania jednotiek a integračného testovania. V tejto fáze sa zameriava najmä na požiadavky mimo funkčnosť kódu teda na vlastnosti ako splenie požiadaviek definovaných špecifikáciou, bezpečnosť, výkon, spoľahlivosť, spoluprácu s externými rozhraniami (aplikáciami, hardwarom, operačným systémom a podobne). [8]

K systémovému testovaniu sú potrebné formálne spísané ciele pre produkt a snahou testera je nájsť miesta, kde produkt týmto cieľom nevyhovuje alebo je ich uspokojenie nejasné. [5]

5.4.5 Akceptačné testovanie

V prípade, že predchádzajúce fázy prebehli úspešne je aplikácia poskytnutá zákazníkovi k takzvanému akceptačnému testovaniu. Ten pripraví svoje vlastné testy a svoje výsledky, prípadne nezrovnalosti medzi špecifikáciou a aplikáciou sú oznámené vývojárskemu tímu, ktorý ich následne analyzuje a opraví, prípadne odmietne. Akceptačné testovanie prebieha vo viacerích iteráciách a výsledkom je aplikácia, ktorú zákazník akceptuje ako výsledný produkt. [6]

5.5 Beta testovanie

V prípade, že software nie je vyvíjaný pre konkrétneho zákazníka, ale mienený ako produkt pre verejnosť nastáva problém vo fáze akceptačného testovanie. Kto vyhodnotí produkt ako akceptovateľný? Odpoveďou na túto otázku je **beta testovanie**.

Beta testovanie je postup, kde takzvaná **beta verzia** aplikácie (v tomto momente aktuálna a finálna verzia) je zverejnená určitému okruhu ľudí, ktorí aplikáciu používajú v reálnom prostredí a následne oznamujú chyby vývojárskemu tímu.

Beta testovanie je teda svojim spôsobom akceptačné testovanie, kde skupina potenciálnych zákazníkov je zapúzdrená do jedného akceptora. Prebieha na princípe čiernej skrinky, tester má prístup len k spustiteľnej verzii produktu a testuje globálne všetky vlastnosti aplikácie.

V dnešnej dobe je beta testovanie veľmi populárne aj u najväčších firiem a organizácií. Napríklad spoločnosť Mirosoft takto testuje operačný systém Windows. Najväčšej popularite sa však beta testovanie teší v oblasti počítačových hier. Spoločnosti ako Blizzard, EA, BioWare alebo LucasArts poskytujú svoje výtvory na obmedzené časové obdobie nadšencom, ktorí im na oplátku radi oznámia všetky chyby, nedostatky a návrhy zmien, ktoré odhalia. Mimo samotného otestovania aplikácie má beta testovanie aj ďalší významný rozmer. O voľné licencie a kľúče k softwaru, hrám a iným projektom sa uskutočňujú rôzne súťaže, akcie a vytvárajú sa dlhé internetové diskusie. Týmto vzniká široká marketingová základňa zasahujúca cieľovú skupinu. Napríklad YouTube kanál jedného z testerov hry Starcraft II spoločnosti Blizzard dosiahol cez 611 000 odoberateľov a celkový počet zobrazení videí na tomto kanály bolo cez 260 miliónov.

Práve k beta testovaniu je možné prirovnať testovanie pomocou project hostingových riešení, kde užívateľ poskytuje vývojárom spätnú väzbu v podobe nájdených chýb a ich zaznamenaní v *bugtracker* nástroji, prípadne návrhom zmien a diskusiou.

5.6 White-box testovanie, Black-box testovanie

V predchádzajúcom texte boli spomenuté dva doposiaľ nevysvetlené pojmy a to testovanie bielej a testovanie čiernej skrinky. Testovanie bielej skrinky (white-box testing) a testovanie čiernej skrinky (black-box testing) sú dva rôzne prístupy k testovaniu softwaru, kde rozdiel v týchto prístupoch hrá najmä testerova znalosť o vnútornej štruktúre produktu. Základný popis a fakty o týchto typoch testovaní sú vysvetlené v nasledujúcich kapitolách.

5.6.1 White-box testovanie

Testovanie bielej skrinky alebo tiež štrukturálne testovanie je prístup, kde testovací tím má kompletnú znalosť o internej štruktúre softwaru, dokonca testy sú od tejto štruktúry odvodené. Testovanie bielej skrinky býva väčšinou vykonávané nad menšími časťami produktu ako moduly alebo subrutiny. Tester môže analyzovať kód a odvodiť z týchto vedomostí testovacie dáta. Výhodou možnosti analýzi kódu je skutočnosť, že tester môže odvodiť počet testovacích prípadov potrebných k spusteniu každého príkazu v programe aspoň jeden krát.

Štrukturálne testovanie umožňuje softwarovým inžinierom testovať prípady s tým, že: [1]

- Je garantované, že každá cesta v programe bola odskúšaná aspoň raz
- Sú otestované všetky logické rozhodnutia na True a False.
- Sú otestované všetky cykly s ich hraničnými hodnotami.
- Sú otestované vnútorné dátové štruktúry a ich validita.

Výhodami testovania bielej skrinky je odhalenie chýb v skrytom kóde, odhalenie mŕtveho kódu, donútenie vývojára opatrne rozmýšľať o implementácii a iné.

5.6.2 Black-box testovanie

Oproti testovaniu bielej skrinky nie je v testovaní čiernej skrinky alebo funkcionálnom testovaní testerovi známa vnútorná štruktúra softwaru. Testovacie prípady sú tak odvodené z požiadavok a špecifikácie programu alebo jeho modulov. Funkcionálne testovanie zahŕňa iba sledovanie výstupov programu k daným vstupom bez analýzi kódu. Tento prístup k testovaniu sa špecializuje na splenie požiadaviek a umožňuje testerovi určiť množstvo testovacích prípadov k tomu aby boli otestované všetky zadané požiadavky. Pre testera sú známe len vstupy a im odpovedajúce výstupy. Dôležité je uvedomiť si, že testovanie bielej a čiernej skrinky nepôsobia ako alternatívy, ale naopak ako dva rozdielne prístupy na odhalenie iných typov chýb. Testovanie čiernej skrinky odhaľuje nasledujúce typy chýb: [1]

- Nekorektné alebo chýbajúce funkcie.
- Chýbajúce alebo zlyhávajúce rozhranie.
- Chyby v dátovom modeli.
- Chyby v prístupe k dátovému zdroju.

Medzi výhody testovania čiernej skrinky patria skutočnosti, že test je objektívny, keďže dizajnér a tester sú od seba nezávislý, tester nepotrebuje ovládať programovací jazyk, test je vykonávaný z pohľadu užívateľa, nie dizajnéra, testy môžu byť vytvorené hneď po zhotovení špecifikácie a iné.

5.7 Modely životného cyklu softwaru

Úlohou modelu životného cyklu softwaru je viesť vývoj produktu systematickou, dobre definovanou a cenovo efektívnou cestou. Model životného cycklu by mal napomáhať k pochopeniu celého procesu tvorby produktu, posilňovať štrukturovaný prístup k vývoju, predčasnému plánovaniu zdrojov, dovoliť manažmentu sledovať progres tvorby systému a podobne. [1]

V tejto kapitole budú predstavené niektoré modely životného cyklu, pričom text bude zameraný predovšetkým na spôsoby a princípy testovania v týchto modeloch. Zároveň predstavené modely boli vybrané tak aby demonštrovali čo najrôznorodejšie prístupy k testovaniu.

Základným prvkom modelu životného cyklu je fáza. Fáza by nemala začať skôr ako sú splnené jej vstupné kritéria a analogicky by nemala skončiť skôr ako sú splenené jej výstupné kritériá. Takto môže efektívne a jasne oddeľovať kroky vrámci vývoja produktu. Typicky majú modely osem fáz. Tie sú : [1]

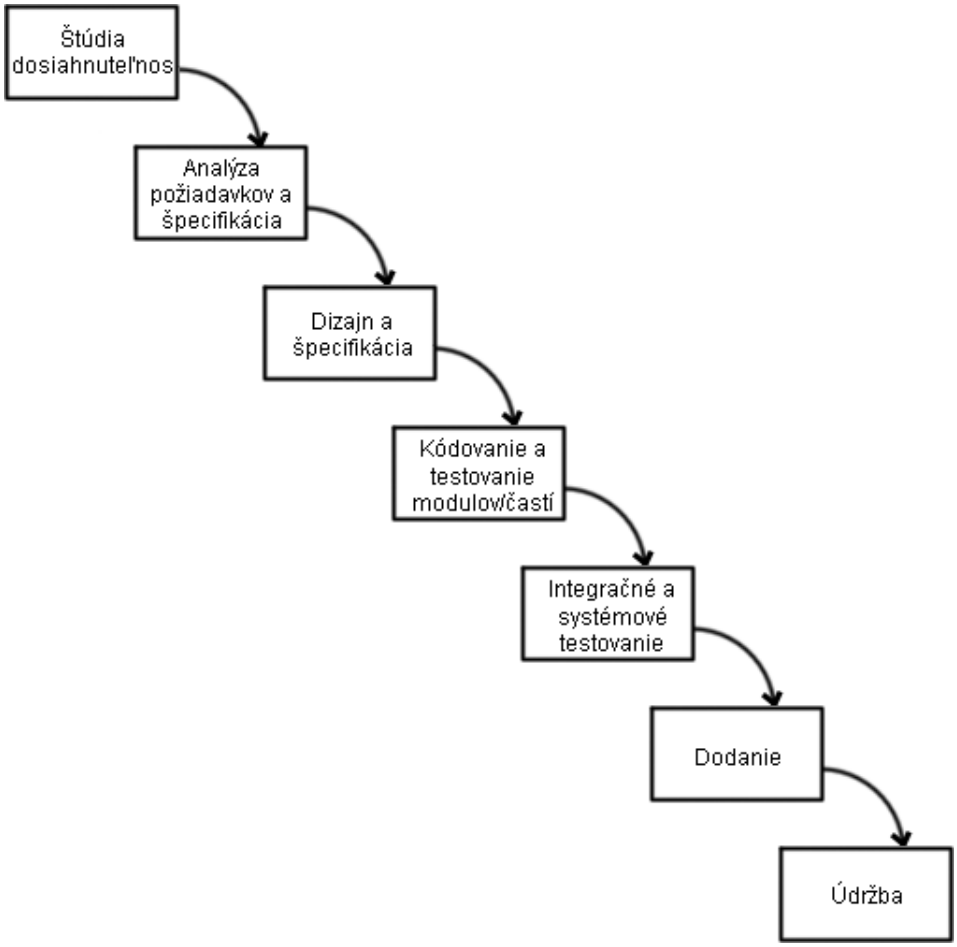
- **Iniciácia a plánovanie projektu** - Prvá fáza v životnom cykle projektu. Jej úlohou je definovať problém a určiť základy požadovaného systému.
- **Identifikácia a selekcia (štúdia dosiahnuteľnosti)** - Štúdia dosiahnuteľnosti má niekoľko podfáz. Ide konkrétne o organizačnú, ekonomickú, technickú a operačnú, z ktorých každá má zistiť možnosti v danej oblasti. Výsledkom identifikácie a selekcie sú poznatky o tom ako dobre systém zapadá do stratégie firmy, aké budú náklady na jeho tvorbu, aký bude profit, potreby softwaru, hardwaru, ľudských a iných zdrojov, očakávanú akceptáciu užívateľmi a následnú podporu, prípadne ďalšie faktory.
- **Projektová analýza** - Je detailná štúdia rôznych operácií systému a ich vzťahov vnútri a mimo systém. Výsledkom tejto fázy je poskytnutie riešení problémov definovaných v predchádzajúcich fázach. Projektová analýza je iteratívny proces, ktorý musí brať v úvahu vyskytovanie sa a vyhodnocovanie nových informácií až kým nie je dosiahnuté požadované riešenie.
- **Systémový dizajn** - Je najkreatívnejšia fáza životného cyklu. Systémový dizajn popisuje finálny systém a proces jeho vývoja vrátane potrebných technických aktivít k jeho vytvoreniu.
- **Kódovanie** - Samotná implementácia produktu.
- **Testovanie** - Testovanie je hlavný nástroj kontroly kvality použitom vrámci vývoja softwaru. Jeho základnou úlohou je hľadať chyby a to všetky typy od funkčných, cez nedodržanie špecifikácie až po chyby v dizajne. Testovanie je kritická a časovo náročná aktivita, vyžadujúca správne plánovanie. Výstupy testovania sú testovacia správa a správa o chybách. Testovanie nemôže ukázať neprítomnosť chýb, môže ukázať len ich prítomnosť.
- **Implementácia** - Implementáciou sa v tejto fáze myslí nasadzovanie systému. To je spojené s prípravou serverov, inštaláciou produktu, tréningom užívateľov a podobne. Zahŕňa tiež finálne testovanie a zabezpečenie ochrany systému.
- **Údržba / podpora** - Údržba a podpora sú dôležité fázy v životnom cykle softwaru. Zahŕňajú opravy chýb, ktoré sa dostali do produkčného prostredia, podporu užívateľov alebo zmeny systému. Opravy chýb vo fáze údržby a podpory sú 50 % až 80 % nákladnejšie ako opravy vo fáze vývoja. Preto je odhaľovanie chýb v predchádzajúcich fázach životného cyklu softwaru kritickou úlohou.

5.7.1 Vodopádový model

Vodopádový model je často používaný model životného cyklu. Jeho meno, vodopádový, vychádza z faktu, že prechádza kaskádovito z jednej fázy do druhej, pričom výstup jednej fázy je vstupom nadchádzajúcej. Tento princíp je znázornený na obrázku 5.2.

Všimnime si tri základné fakty vodopádového modelu [24]

- Veľký dôraz sa kladie na špecifikáciu produktu. Samotné kódovanie je iba jeden blok predchádzaný viacerími blokmi presne určujúcimi očakávaný výstup.
- Jednotlivé fázy sa neprekrývajú a sú striktne oddelené.
- Nie je tu cesta späť. Z aktuálneho kroku nie je možné dostať sa do kroku predchádzajúceho.



Obrázok 5.2: Princíp vodopádového modelu

Testovanie sa za použitia vodopádového modelu sa vykonáva v dvoch fázach. Prvý krát spoločne s kódovaním. Testovanie modulov je často špecifikované štandardom spoločnosti. Obsahuje presné definície testovacích plánov, kritérií a manažment testovacích prípadov. Významná súčasť testovania v tejto fáze je ladenie aplikácie. Vo fáze kódovanie tiež môžu prebiehať inšpekcie štandardov kódovania, kvality softwaru a funkčnej správnosti.

V samostatnej fáze prebieha potom integračné a systémové testovanie vrámci, ktorého sú moduly integrované do jedného celku. Integrácia prebieha štandardne vo viacerích iteráciách, pričom v každej je čiastočne integrovaný systém otestovaný. Po integrácii všetkých častí prichádza na rad systémové testovanie. Úlohou systémového testovania je zistiť, či výsledný produkt spĺňa požiadavky definované v produktovej špecifikácii. Prebieha v troch fázach : alfa, beta a akceptačnom testovaní.[1]

Ťažnými vlastnoťami vodopádového modelu sú jeho linearita, segmentácia, systematičnosť a jednoduchosť. Z týchto vlastností vychádzajú niektoré výhody. Produkt vytváraný na základe vodopádového modelu je jednoducho manažovateľný pretože sa jednotlivé fázy neopakujú, kroky pri tvorbe produktu sú jednoducho rozdeliteľné do týmov a rovnako jednoduché je definovanie zodpovedných osôb za danú fázu.

Na druhej strane má množstvo nevýhod. Medzi najvýraznejšie je možné uviesť náročnosť pri definovaní všetkých požiadaviek na začiatku projektu, neschopnosť vyrovnávať sa so zmenami, fakt, že až do vydania živej verzie neexistuje žiadna medziverzia a iné. Z týchto dôvodov vodopádový model nie je vhodný pre veľké projekty alebo projekty, pri ktorých nie je možné na ich začiatku určiť presné požiadavky na produkt.

5.7.2 Špirálový model

Špirálový model dokáže pokryť takmer každý iný model životného cyklu softwaru a určiť, ktorá kombinácia modelov najlepšie sedí pre daný software. [2]

Jeho obecná myšlienka spočíva v tom, že nie každý detail je definovaný hneď na začiatku. Namiesto toho vývoj produktu začína po malých častiach. V prvom rade sa nadefinujú dôležité vlastnosti, tie sa vyskúšajú, získa sa spätná väzba od zákazníka a potom sa posunie na ďalší stupeň. Tento proces sa potom opakuje až kým nie je hotový finálny produkt. [24]

Obrázok 5.3 znázorňuje prechody jednotlivými fázami špirálového modelu. Špirála je rozdelená na kvadranty. Prvý kvadrant identifikuje ciele fázy . Druhý kvadrant vyhodnocuje rôzne alternatívy na základe cieľov a obmedzení, pričom sa zohľadňuje risk analýza projektu. Tretí kvadrant je ďalším krokom zdôrazňujjúcim vývoj stratégií, ktoré riešia nezrovnalosti a riziká. Môže zahŕňať aktivity ako simulácie a tvorbu prototypov. V poslednom kvadrante sú definované ciele, ktoré by mali byť naplnené v ďalšom cykle. [1]

Testovanie vrámci tohto modelu prebieha vo viacerích cykloch, čo vyplýva z podstaty modelu. Testovaný je každý prototyp a na jeho základe vzniká diskusia so zákazníkom, ktorá je vstupom do ďalších fází projektu. V konečnej fáze cyklu prichádza na rad testovanie jednotiek, integračné a akceptačné testovanie.

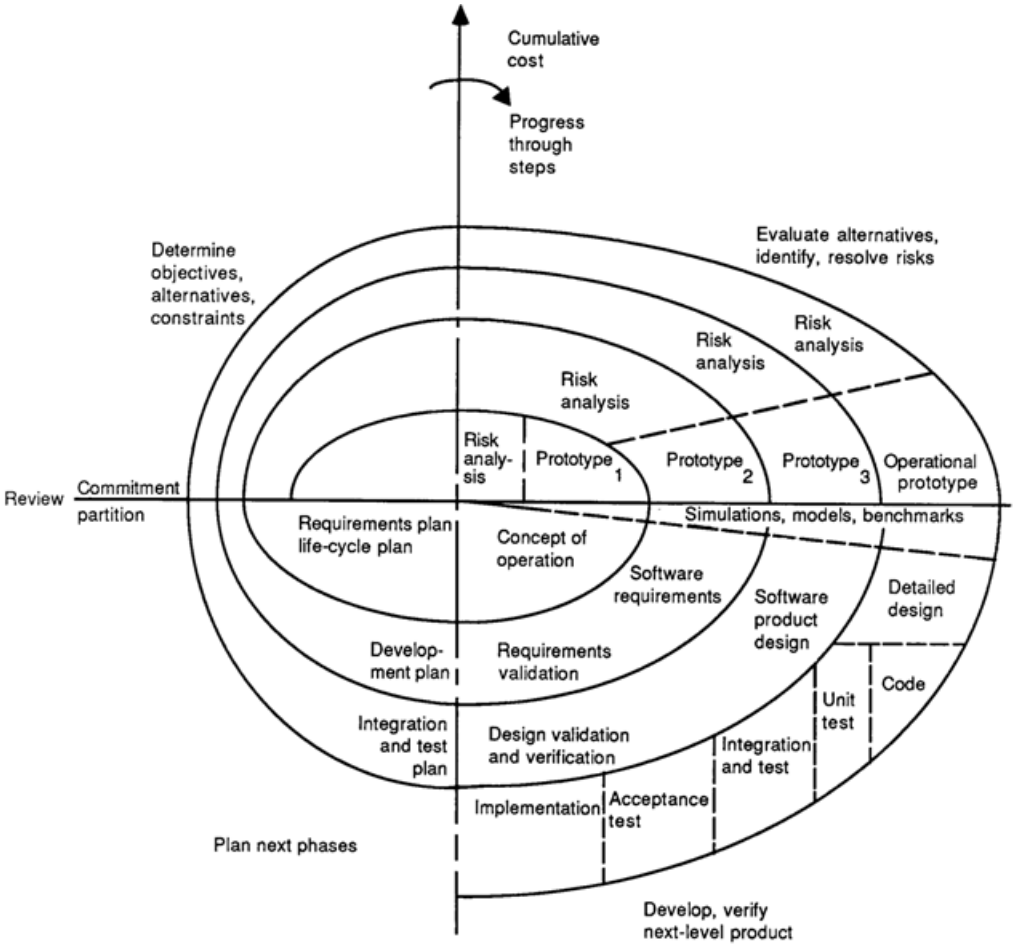
Výhodami špirálového modelu sú jeho flexibilita, fakt, že je to model riadený rizikami a používa prototypy, na ktorých základe je možná priebežná kontrola a diskusia s klientom.

5.7.3 RAD model

RAD alebo Rapid Application Development model je používaný v prípadoch keď požiadavky a ich riešenia dovoľujú pozdeliť produkt do viacerích nezávislých systémov alebo komponent tak, že môžu byť vyvíjané viacerími nezávislými tímami. Po dokončení vývoja týchto systémov sú integrované do jedného celku.

Vo fáze testovania sú použité už otestované programy vytvorené jednotlivými tímami. čo znamená, že testovanie zaberá kratšiu dobu. Otestované musia byť len nové programy a komponenty.

Výhodou prístupu RAD je to, že sa stáva rýchlym vďaka využitiu viacerích vývojárskych týmov. Nevýhodami sú potreba správneho rozvrhnutia týmov, nutnosť možnosti rozdeliť



Obrázok 5.3: Princíp špirálového modelu

systém do nezávislých častí a ďalšie.

RAD model sa tak používa len pri projektoch, ktoré je možné rozdeliť na jednotlivé podsystémy s dobou vývoja do 90 dní. [1]

5.7.4 Big Bang model

Big Bang model je od všetkých predošlých rozdielny. Jeho princíp je jednoduchý. Snaží sa čo najskôr začať s vývojom a minimalizovať čas strávený plánovaním alebo formálnymi procesmi. Najčastejšie sa používa v prípadoch keď produktové požiadavky nie sú dobre definované alebo je možné ich počas vývoja pozmeniť a dátum zverejnenia produktu je flexibilný.

Formálne testovanie sa v tomto modeli takmer nenachádza a v prípade, že áno je to pred vydaním produktu. Testovanie produktu vytvoreného na základe Big Bang modelu je jednoduché aj náročné zároveň. Tester má totiž k dispozícii kompletný a hotový program, problémom však zostáva náročnosť opravy chýb v už kompletnom produkte.[24]

Kapitola 6

Analýza a návrh

Cieľom tejto diplomovej práce bude vytvoriť plnohodnotný a fungujúci projekt zameraný na project hosting. Pri návrhu aplikácie budem vychádzať zo skúseností nadobudnutých pri analýze projektov popísaných v kapitole 3.

Predtým ako bude možné vrhnúť sa na riešenie konkrétnych problémov je nutné vyšpecifikovať požiadavky, ktoré aplikácia musí spĺňať, definovať jej funkcionalitu a základnú štruktúru.

6.1 Neformálna špecifikácia

Základnou požadovanou funkcionalitou projektu je samozrejme možnosť umiestniť projekt na úložisko dát a sprístupniť ho verejnosti. Štruktúra aplikácie by mala umožniť priradiť projekt do určitej kategórie, napríklad podľa platformy. Keďže projekty prechádzajú určitým vývojom a väčšina z nich vychádza viackrát v rôznych verziách, bude samozrejmosťou možnosť uloženia viacerích verzií, pričom vždy jedna bude označená ako stabilná. Každý projekt by mal patriť a byť udržiavaný skupinou užívateľov, takzvanou projektovou skupinou. Táto by mala mať svojich administrátorov a užívateľov, ktorí budú pôsobiť ako vývojári daného projektu.

Následne po vložení projektu autorom aplikácie, jedným z členov projektovej skupiny, sa projekt uloží do databázy a dostane sa do stavu *Sandboxed*, teda stavu keď čaká na schválenie administrátorom aplikácie, takzvaným ultimate užívateľom. Ten má možnosť odobriť alebo zamietnuť projekt na špeciálnej stránke prístupnej len tomuto užívateľovi. Po odobrení projektu ultimate užívateľom sa projekt sprítupní pre všetkých užívateľov. Pre všetkých užívateľov bez rozdielu musí byť videľná stránka so základnými informáciami o projekte. Tá by mala obsahovať meno a projektovú skupinu, popis aplikácie, rozmedzie v ktorom bude prebiehať jej testovanie, poznámky a návod k inštalácii a informácie o potrebnom hardwarovom a softwarom vybavení pre beh aplikácie. Zároveň musí stránka poskytovať možnosť prihlásiť sa k testovaniu tohto projektu. Pre administrátorov projektovej skupiny, ktorej projekt patrí musí byť z tejto stránky dostupná editačná stránka projektu, kde môže administrátor meniť a dopĺňať vyššie spomenuté základné informácie projektu.

Po prihlásení sa k projektu sa užívateľ stáva jeho testerom. Tester musí mať možnosť jednoducho stiahnuť aplikáciu a intuitívnym a prehľadným spôsobom vkladať chyby, ktoré v aplikácii našiel. Preto musí aplikácia obsahovať stránku so zoznamom chýb, tzv. buglist. Tá by mala vyobrazovať všetky nájdené chyby projektu v tabuľke. Každý riadok v tabuľke bude predstavovať jednu nájdenú chybu, pričom položky tabuľky bude možné zoradiť podľa

ich vlastností. Vlastnosti chyby zadáva tester pri jej vytváraní, ktoré je dostupné rovnako z tejto stránky. Po uložení chyby ju môže meniť a editovať administrátor projektovej skupiny, nemôže však zmeniť informácie o tom kto a kedy chybu pôvodne vložil. Každá chyba obsahuje meno, krátky popis a niekoľko vlastní. Ide o závažnosť chyby (nízka, stredná, vysoká, kritická), jej typ (funkčná chyba, návrh na zmenu, nedefinovaná), interval výskytu (vždy, akciou, nedefinované) a stav (otvorená, čakjúca na schválenie, zatvorená). Chyba bude mať aj vlastnú stránku, na ktorej budú zobrazené bližšie informácie, jednoduchá diskusia o chybe a návrh riešenia, ten má však možnosť meniť len člen projektovej skupiny. Z tejto stránky má administrátor možnosť dostať sa na editačnú stránku chyby, kde okrem informácii o nej môže priradiť riešiteľa tejto chyby z množiny členov projektovej skupiny.

Ďalej by mal projekt obsahovať diskusnú stránku. Tá by mala byť jednoduchá a obsahovať dve úrovne, témy a príspevky. Príspevky by mali byť zoradené analogicky a stránka by mala podporovať možnosť stránkovania, teda zobrazovania príspevkov po častiach, nie všetkých na jednej stránke. Celkovo by mala byť aplikácia jednoduchá na používanie, s čím by mal korešpondovať aj jednoduchý a funkčný dizajn. Aplikácia musí zasielať informácie o akciách nad projektom (napríklad zmena stavu projektu) členom projektovej skupiny a jej testerom.

6.2 Analýza požiadavkov

Nasledujúca kapitola predstaví entity vystupujúce v aplikácii, diagram užitia a na základe týchto poznatkov potom návrh databázy prezentovaný prehľadným ER diagramom.

6.2.1 Aktéri

Z neformálnej špecifikácie vyplýva hneď niekoľko typov aktérov, v ktorých môže užívateľ vystupovať:

- **Užívateľ (user)** - Má len základné práva, teda môže prezerať prehľad uložených projektov, prípadne si vytvoriť vlastnú projektovú skupinu.
- **Tester** - Užívateľ, ktorý sa prihlásil k odberu určitého projektu sa stáva jeho testerom. Môže vidieť bližšie informácie k tomuto projektu, pridávať chyby, ktoré našiel a diskutovať o projekte v diskusnom fóre.
- **Člen projektovej skupiny** - Člen vytvorenej skupiny. Môže pridať nový projekt vrámci tejto skupiny, môže pridávať chyby a môže byť pridaný ako osoba zodpovedná za odstránenie oznámenej chyby.
- **Administrátor skupiny** - Člen skupiny, ktorý je jej zakladateľom alebo bol pridaný ako adminstrátor. Má práva ako člen projektovej skupiny, navyše môže zatvárať chyby v zozname chýb, pridávať úlohu ich analýzy a opravy jednotlivým členom skupiny. Má právo manažovať projektovú skupinu, pridávať a odoberať jej členov, prípadne pridávať a odoberať administrátorov skupiny.
- **Zakladateľ projektu** - Člen skupiny, ktorý projekt založil (aj keď nie je administrátorom skupiny) má vrámci tohto projektu práva ako adminstrátor. Môže teda pridávať, odoberať položky v zozname chýb a priradiť člena skupiny, ktorý je zodpovedný za analýzu a opravu chyby.

- **Ultimate užívateľ** - Je užívateľ, ktorý má neobmedzené práva nad celou aplikáciou. Stať sa takýmto aktérom nie je možné nijakým spôsobom skrz užívateľské rozhranie a je možné nastaviť ho len priamo v databáze. Tento užívateľ môže ako jediný presúvať projekty zo stavu Sandboxed do aktívneho stavu projektu a uviesť ich tak do stavu viditeľného pre verejnosť.

6.2.2 Prípady užitia

Z vyššie uvedenej množiny aktérov je možné vytvoriť prípady užitia vrámci projektu. Aplikácia bude obsahovať 3 základné systémy - projekt,užívateľ a projektová skupina (skupina užívateľov spolupracujúcich na jednom alebo viacerích projektoch). Prípady užitia zobrazuje diagram prípadov užitia na obrázku 6.1. Nasledujúce riadky ho stručne popisujú.

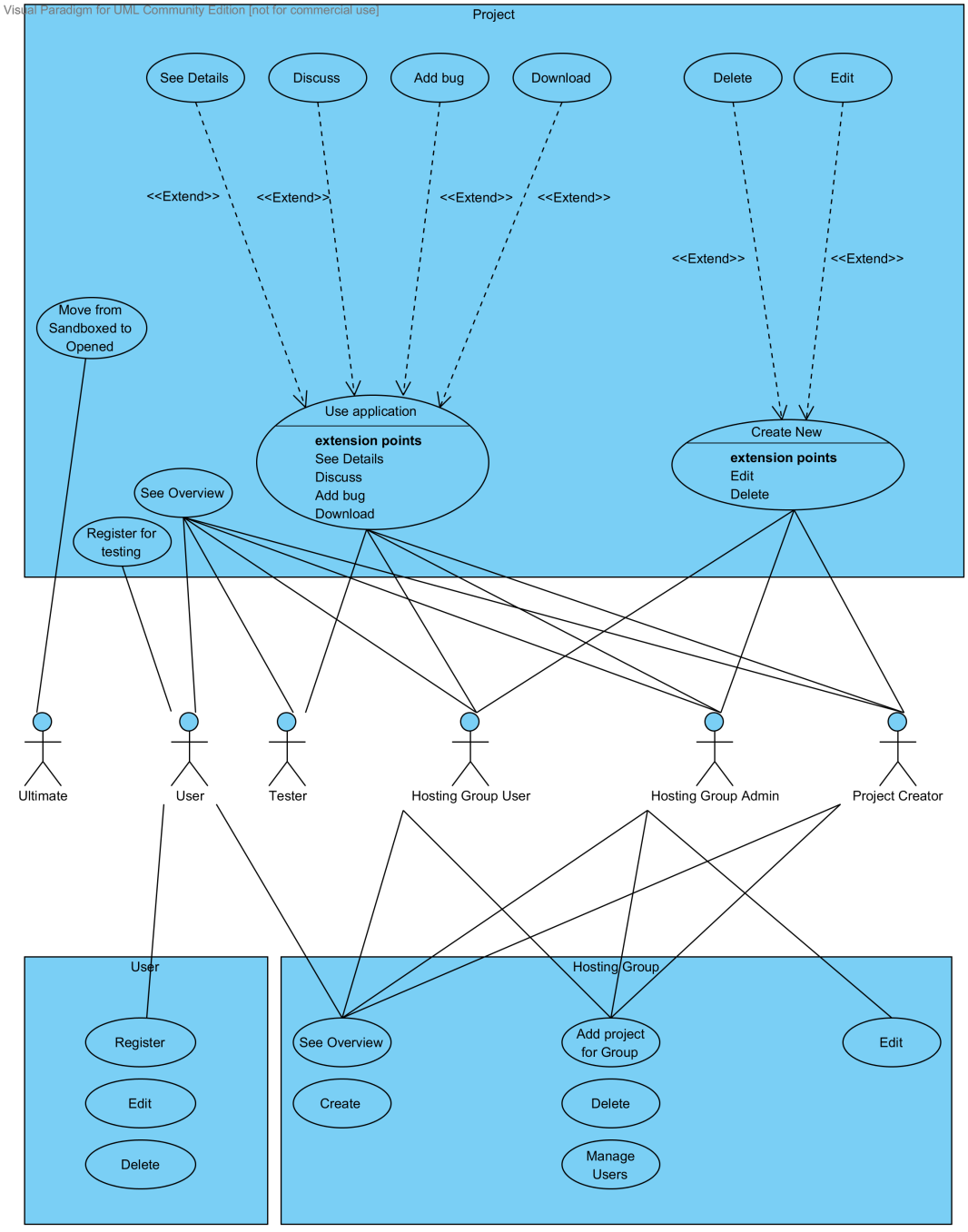
- **Projekt** - Tvorí ho užívateľ, ktorý ho musí priradiť pod existujúcu projektovú skupinu, ktorej je členom alebo vytvoriť novú skupinu. Po jeho vytvorení sa nezávislý užívateľ môže prihlásiť k odberu tohto projektu, stiahnuť si ho a následne prispievať svojimi zisteniami do diskusie a zoznamu chýb.

- **Užívateľ** - Každý užívateľ má svoje jedinečné meno a prístupové heslo. Je tak v systéme jednoznačne identifikovaný. Vrámci systému môže byť jeho úloha odlišná na základe príslušnosti k projektovej skupine, prípadne nasledovania projektu. Užívateľ môže meniť svoje základné informácie ako popis, email a podobne.

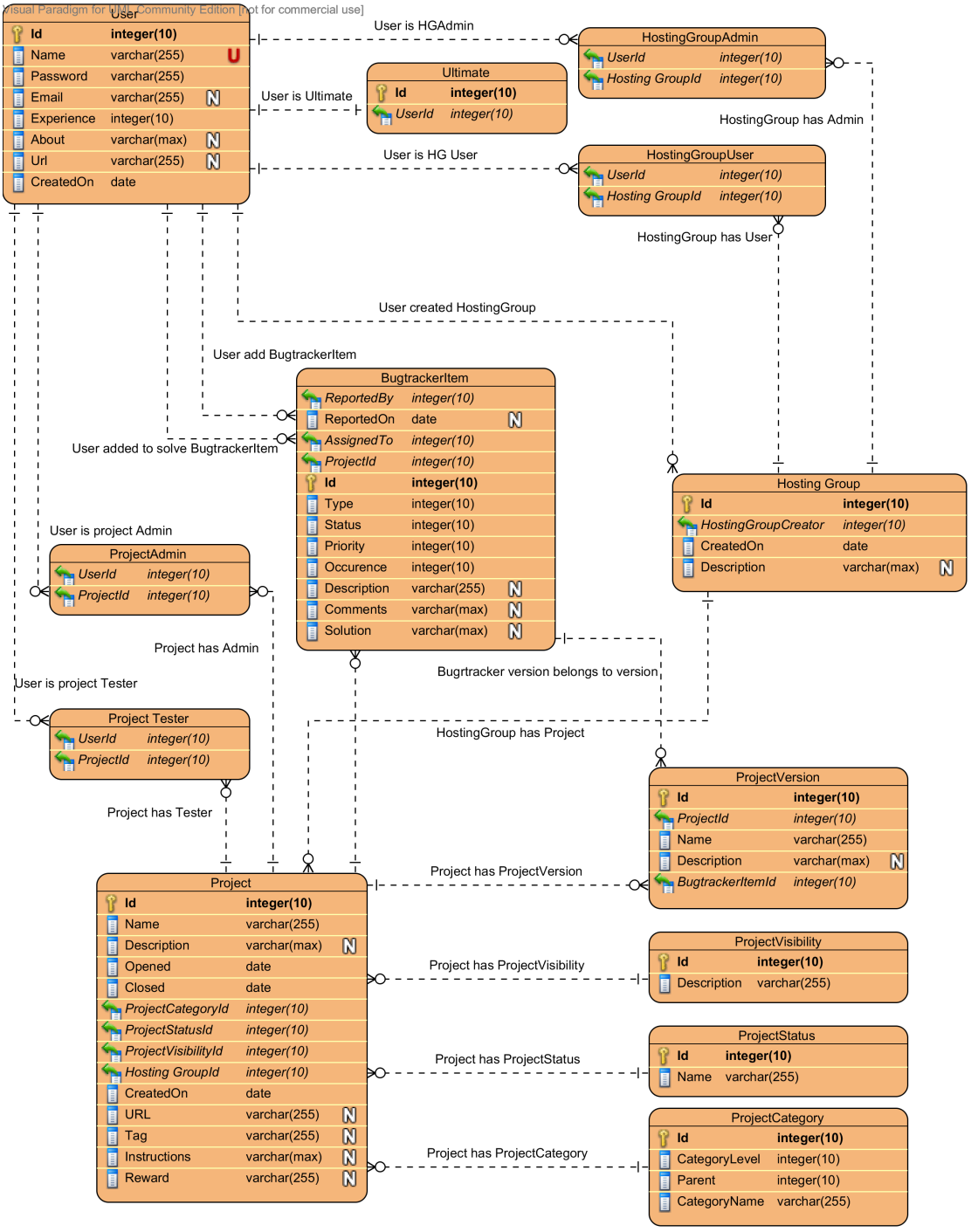
- **Projektová skupina** - Je skupina užívateľov, ktorý sa spoločne podieľajú na vývoji jedného alebo viacerích projektov. Skupina obsahuje dva typy aktérov a to užívateľa a administrátora. Užívateľ môže k skupine pridať projekt a môže byť administrátorom pridaný ako riešiteľ chyby v zozname chýb. Administrátor môže skupinu neobmedzene meniť, manažovať jej užívateľov, ukončovať projekty alebo odsúhlasovať úspešnosť opráv chýb.

6.3 Návrh databázy

Databázovú štruktúru vytvorenú na základe analýzy a prípadov užitia demonštruje ER diagram na obrázku 6.2. Existujú tu 4 základné entity, korešpondujúce so systémami znázornenými v diagrame prípadov užitia a teda Užívateľ, Projekt, Hostingová skupina. Naviac sa k ním pridáva entita Bugtracker Item, do ktorej budú pridávané chyby nájdené užívateľmi.



Obrázok 6.1: Diagram prípadov užitia navrhnutých pre aplikáciu CrowdProof



Obrázok 6.2: ER diagram databázy navrhnutej pre aplikáciu CrowdProof

Kapitola 7

Použité technológie

V tejto kapitole budú v krátkosti predstavené technológie použité k implementácii aplikácie CrowdProof, teda aplikácie vytvorenej pre účely tejto diplomovej práce. Všetky technológie pochádzajú z dielne spoločnosti Microsoft. Vďaka jednoduchej a bezproblémovej integrácii týchto technológií bol vývoj, čo sa týka pripravenia vývojárskeho prostredia veľmi pohodlný.

7.1 .NET a C#

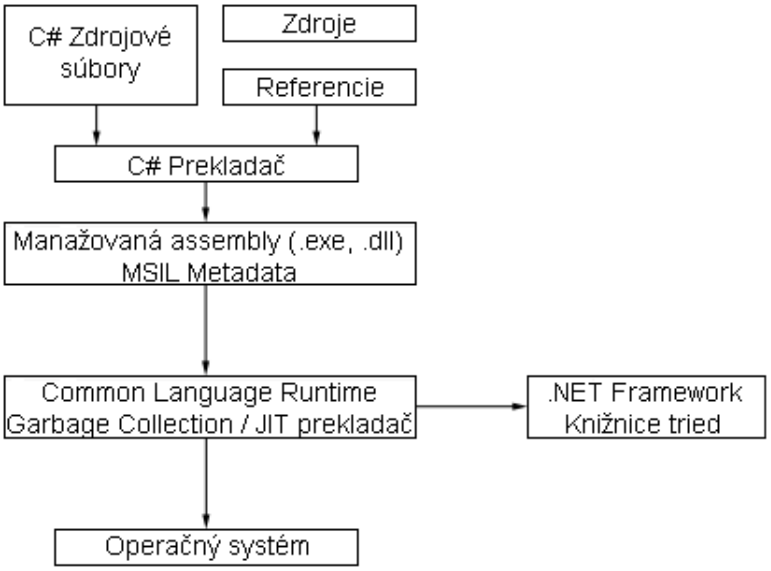
Platformu .NET vyvinula spoločnosť Microsoft. Vznikla ako reakcia na potrebu komponenty, ktorá podporuje budovanie a beh aplikácií a služieb novej generácie. Jej hlavné myšlienky je možné zachytiť v nasledujúcich bodoch [17]

- Poskytnúť konzistentné objektovo-orientované prostredie, vrámci ktorého objektový kód je uložený a vykonávaný lokálne, ale distribuovaný po internete alebo vykonávaný vzdialene.
- Poskytnúť prostredie, ktoré minimalizuje konflikty pri nasadzovaní a verzovaní softwaru.
- Poskytnúť prostredie, ktoré podporuje bezbečné vykonávanie kódu, vrátane kódu vytvoreného neznámou treťou stranou.
- Poskytnúť prostredie, ktoré eliminuje problémy výkonu skriptových a interpretovaných prostredí.
- Poskytnúť vývojárom konzistenciu skrz širokú škálu aplikácií a prostredí (ako Windows alebo web).
- Budovať celú komunikáciu na priemyselných štandardoch a tým sa uistiť, že kód založený na .NET frameworku môže byť integrovaný s každým iným kódom.

Základom platformy .NET sú dve komponenty Common Language Runtime (CLR) a .NET Framework class library (knižnica tried).

CLR je prostredie zodpovedné za vykonávanie .NET aplikácií, manažuje pamäť, spúšťanie vlákien, kompiláciu a ďalšie systémové služby. CLR uľahčuje návrh aplikácií a komponent, ktorých objekty interagujú medzi programovacími jazykmi. Tie spolu môžu navzájom komunikovať. Napríklad je možné definovať triedu a potom použiť iný jazyk a odvodiť inú triedu z pôvodnej alebo na ňu volať metódy. Takáto medziazyková interakcia je možná vďaka

faktu, že prekladače využívajú spoločný typový systém definovaný CLR vrámci Common Language Specification (CLS). Zdrojový kód napísaný v niektorom z .NET jazykov je prekladaný do Intermediate Language (IL), jazyka, ktorý vyhovuje CLS. Kód v IL spoločne s metadátami je potom uložený vo forme spustiteľného súboru alebo knižnice a CLR naň použije *just-in-time* kompiláciu, čím je IL kód preložený do natívneho strojového kódu.



Obrázok 7.1: Princíp kompilácie .NET aplikácií

Knižnica tried je kolekcia **znovupoužiteľných tried**, rozhraní, typov a hodnôt, ktoré urýchľujú a optimalizujú proces vývoja. Obsah tejto knižnice siaha od práce s reťazcami cez prístup a prácu s databázou až po špecifické vývojárske scenáre.

Z rodiny .NET jazykov bol pre potreby tejto aplikácie vybraný **jazyk C#**. Jazyk C# je objektovo orientovaný jazyk, ktorý je svojou syntaxou veľmi blízky jazykom C, C++ alebo Java a v nemálo prípadoch je v tomto smere oproti spomínaným jazykom ešte zjednodušený. Zároveň však poskytuje mocné nástroje ako enumeráciu, delegátov, lambda výrazy alebo priamy prístup do pamäte. Je silne typovaný a jeho cieľom je maximalizovať produktivitu vývojára.

7.2 ASP .NET

ASP .NET je nová generácia technológií spoločnosti Microsoft pre vytváranie webových aplikácií bežiacich na strane serveru postavená na .NET frameworku.

Základné piliere ASP .NET sa dajú zhrnúť do siedmych bodov [10]

1. **Integrácia s .NET framework** - To znamená, že vývojár môže využiť funkcionalitu, ktorú ponúka class library definovaná .NET frameworkom. Obrovskou výhodou potom je, že aj napriek tomu, že existujú triedy špecifické pre Windows a pre webové aplikácie drvivá väčšina z nich je použiteľná pre akýkoľvek typ aplikácie.

2. **Neiterpretuje, ale kompiluje sa** - Tak ako všetky .NET aplikácie sa kompiluje a to rovnakým spôsobom ako je uvedené na obrázku 7.1. Je vhodné poznamenať, že JIT

kompilácia neprebieha pri každom požiadavku na webovú stránku. IL kód sa vytvorí iba raz a generuje sa len v prípade modifikácie zdroja. Rovnako sa chovajú aj súbory s natívnym strojovým kódom.

3. **Je viacjazyčné** - Táto vlastnosť vyplýva z faktu, že všetky jazyky musia byť v súlade s CLS a kompilujú sa do IL.

4. **Pracuje vrámci CLR**

5. **Je objektovo orientované** - Nielen, že kód má úplný prístup k objektom .NET frameworku, je rovnako možné využiť všetkých výhod objektovo orientovaného prítupu ako vytvárať znovupoužiteľné triedy, rozširovať triedy dedením, používať rozhrania a podobne. Napríklad veľavravnou ukážkou je zapúzdrenie serverových ovládacích prvkov s ktorými vývojár môže manipulovať z kódu. ASP .NET teda ponúka serverové ovládacie prvky ako abstrackiu pre nízkoúrovňové detaily pre programovanie HTML a HTTP.

6. **Podporuje všetky prehliadače** - ASP .NET sa s týmto problémom vysporiadalo bez nutnosti zásahu programátora (aj keď umožňuje získať informácie o prehliadači klienta). Serverové ovládacie prvky ASP .NET realizujú svoj kód adaptívne, teda zohľadňujú schopnosti klienta.

7. **Jednoducho sa konfiguruje a nasadzuje** - Vďaka použitiu Internet Information Server (IIS) je nasadenie webovej aplikácie otázka minút. V princípe spočíva v nakopírovaní súborov do virtuálneho adresára bez potreby ďalších registračných krokov. ASP .NET rovnako jednoducho rieši aj konfiguráciu aplikácie tým, že minimalizuje závislosť na nastavení IIS. Väčšina nastavení sa ukladá v XML súbore *Web.config*, ktorý je umiestnený v rovnakom adresári ako samostatné webové stránky. K jeho editácii postačí jednoduchý textový editor a v prípade modifikácie nastavení aplikácie ASP .NET automaticky zemnu zaznamená a reštartuje aplikáciu v novej doméne, pričom existujúcu doménu ponechá tak dlho, kým sa neukončí spracovanie zatiaľ nevyriešených požiadavkov.

7.3 ADO .NET Entity Framework

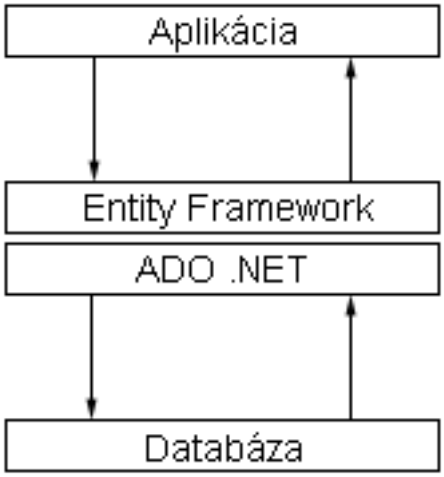
ADO .NET Entity Framework sa stáva základnou platformu spoločnosti Microsoft pre prístup k dátam pri tvorbe .NET aplikácií. Bol vydaný vo júli 2008 ako súčasť Visual Studio 2008 Service Pack 1 a .NET 3.5 Service Pack. Vývojármi nebol Entity Framework privítaný s veľkým nadšením a pristupovali k nemu pomerne skepticky, avšak so zverejnením Visual Studia 2010 a .NET 4 v apríli 2010 vyšiel rovnako aj vylepšený Entity Framework. V tom momente zaujal veľkú skupinu vývojárov, ktorá si ho rýchlo osvojila.

Entity Framework je ďalším krokom v budovaní technológií pre prístup k dátam spoločnosti Microsoft a je rozšírením ADO .NET. ktorý vývojárom ponúka ďalší mechanizmus pre prístup k dátam a prácu s nimi. Entity Framework ponúka nástroje integrované do Visual Studia, ktoré umožňujú vizuálne vytvoriť model alebo vygenerovať model z existujúcej databázy.[13], [14] Výhodou toho, že Entity Framework pracuje ako nadstava ADO .NET je okrem faktu, že prináša nové možnosti práce s databázou aj to, že zachováva pôvodne výhody ADO .NET. Dokáže tak napríkald pracovať s rôznymi databázovými serverami. Jediné, čo musí vývojár urobiť je správne nastaviť connection string. Veľkou výhodou je

schopnosť použiť technológiu LINQ. LINQ fungujúci v jazykoch C# a Visual Basic je rozšírený dotazovací jazyk, navyše realizuje kontrolu dotazov už v čase kompilácie a pomocou Intellisense dokonca primo pri písaní kódu.

Najväčšou výhodou Entity Frameworku je, že oslobudzuje vývojara od starostí so štruktúrou databázy. Celý prístup a ukladanie dát je vykonávaný proti konceptuálnemu modelu dát, ktorý odráža biznis objekty. S Entity Frameworkom tak nie je nutné dotazovať sa na schému databázy, ale na schému vytvorenú na základe biznis modelu. Vývojár tak nemusí konvertovať riadky a stĺpce databázy na objekty, pretože sú ako objekty vrátené vo výsledku dotazu.

Entity Framework používa Enity Data Mode (EDM). EDM je dátový model na klientskej strane a je jadrom Entity Frameworku. EDM v aplikácii popisuje štruktúru biznis objektov. [9]



Obrázok 7.2: Princíp práce Entity Framework, ktorý pracuje ako nadstavba nad ADO .NET

7.4 MS SQL 2008

Je relačný databázový serverový systém spoločnosti Microsoft. Je vysoko škálovateľný, flexibilný, bezpečný a momentálne najpoužívanejší databázový systém v podnikovej sfére. S určitými obmedzeniami je možné získať ho zadarmo vo verzii Express, pre účely tejto práce však bola použitá verzia Standard poskytnutá vrámci licencie MSDNAA.

Použitie databázového servra MS SQL 2008 pre účely tejto aplikácie bolo dané najmä jednoduchou integráciou s ostatnými technológiami použitými, ktoré sú výlučne z dielne spoločnosti Microsoft.

7.5 Visual Studio 2010

Visual Studio je profesionálne vývojárske prostredie od spoločnosti Microsoft, ktoré podporuje komplexnú sústavu rôznych dizajnérskych nástrojov, nástrojov podporujúcich ladenie aplikácie alebo takzvaný code-behind (oddelenie .NET kódu od webového pri tvorbe ASP .NET aplikácie). Visual Studio má dokonca zabudovaný vlastný webový server, ktorý je

určený pre rýchle testovanie aplikácie bez potreby konfigurácie. Vývojárovi dáva možnosť rýchlejšieho a jednoduchšieho písania kódu pomocou technológie IntelliSense, ktorá pri písaní kódu zachytáva chyby a ponúka rôzne doporučenia. Samozrejmosťou je bezproblémová integrácia so všetkými doteraz uvedenými technológiami na vysokej úrovni. Pre vývoj na platforme .NET prakticky neexistuje žiadna rovnocenná náhrada Visual Studia a práve preto bol vybraný aj pre tvorbu aplikačnej časti tejto diplomovej práce.

Kapitola 8

Vlastná implementácia

V kapitole budú bližšie popísané entity prítomné v aplikácii tak ako boli navrhnuté v kapitole 6 a tak ako v konečnom výsledku vystupujú v jej finálnej podobe. Ďalej budú popísané typy zdrojových súborov a ich úloha v projekte. V závere kapitoly je krátko hovorené o dizajne aplikácie a obsahuje tiež niekoľko ukážok jej finálnej podoby.

8.1 Základné systémy

V tejto kapitole budú v širšom zábere popísané systémy navrhnuté v kapitole 6.2.2. Pri ich implementácii bolo v čo najväčšej miere vychádzané z analýzy a návrhu aplikácie avšak pri vývoji došlo k niekoľkým novým ideám a rozšíreniam.

8.1.1 Projekt

Úlohou a zmyslom aplikácie CrowdProof je testovanie projektov, preto je tento systém základným kameňom programu. Každý projekt patrí pod projektovú skupinu a je spravovaný jej administrátormi a členmi. Každý užívateľ môže prejaviť záujem o testovanie projektu a v prípade odobrenia jedným z administrátorov skupiny dostane možnosť si aplikáciu stiahnuť a prispievať do zoznamu chýb a diskusie.

Po vytvorení je projekt automaticky v stave *Sandboxed*. Tento stav značí, že projekt bol uložený v databáze a čaká na schválenie, po ktorom sa dostane do aktívneho stavu *Active* a bude tak prístupný užívateľom. Po skončení testovania je projekt ukončený a presunutý do stavu *Closed*.

Projekty v aplikácii sa delia do 3 základných kategórií, ktoré môžu obsahovať ďalšie podkategórie. Sú to webové aplikácie,ktoré sa ďalej nedelia, mobilné aplikácie, ktoré sú ďalej rozdelené podľa operačných systémov (Android, iOS, Windows Phone, Iné) a desktop aplikácie rozdelené rovnako podľa operačného systému (Windows, Linux, Mac OS, Iné).

Samozrejmosťou je, že každý projekt môže mať viac verzií, preto aj aplikácia podporuje verzovanie, pričom vždy jedna z verzií je označená ako stabilná.

Najdôležitejšou časťou projektu je zoznam chýb. Ten vytvárajú testeri tým, že do neho pridávajú chyby a návrhy zmien, ktoré používaním aplikácie našli. Spolu so záznamom každej chyby, je okrem jej názvu a popisu uložená aj informácia o tom, kto a kedy chybu uložil, jej priorita, spôsob výskytu chyby (vždy, náhodne, spustením akcie). V detailnom pohľadne môžu užívatelia chybu komentovať a pridávať tak vlastné skúsenosti.

Poslednou funkciou pre užívateľov je možnosť diskutovať o projekte v sekcii diskusie. Tá funguje ako jednoduché fórum s dvomi úrovňami. Prvou sú témy, druhou komentáre

v danej téme.

8.1.2 Užívateľ

Užívateľom sa myslí každý jedinec zaregistrovaný v aplikácii pod jednoznačným menom. Každý užívateľ môže mať viacero funkcií vzťahujúcich sa k rôznym projektom alebo projektovým skupinám. Tým aké práva užívateľ pre daný projekt má sú mu ponúknuté možnosti interakcie na stránke tohto projektu. Užívateľ môže v aplikácii vystupovať ako niekoľko typov aktérov ako je to uvedené v kapitole 6.2.1

8.1.3 Projektová skupina

Funkčnosť projektovej skupiny sa počas implementácie nezmenila a pracuje tak ako v návrhu. Je to teda skupina užívateľov spravujúca spoločne projekty, obsahujúca 2 typy aktérov - užívateľov a administrátorov.

8.2 Databáza

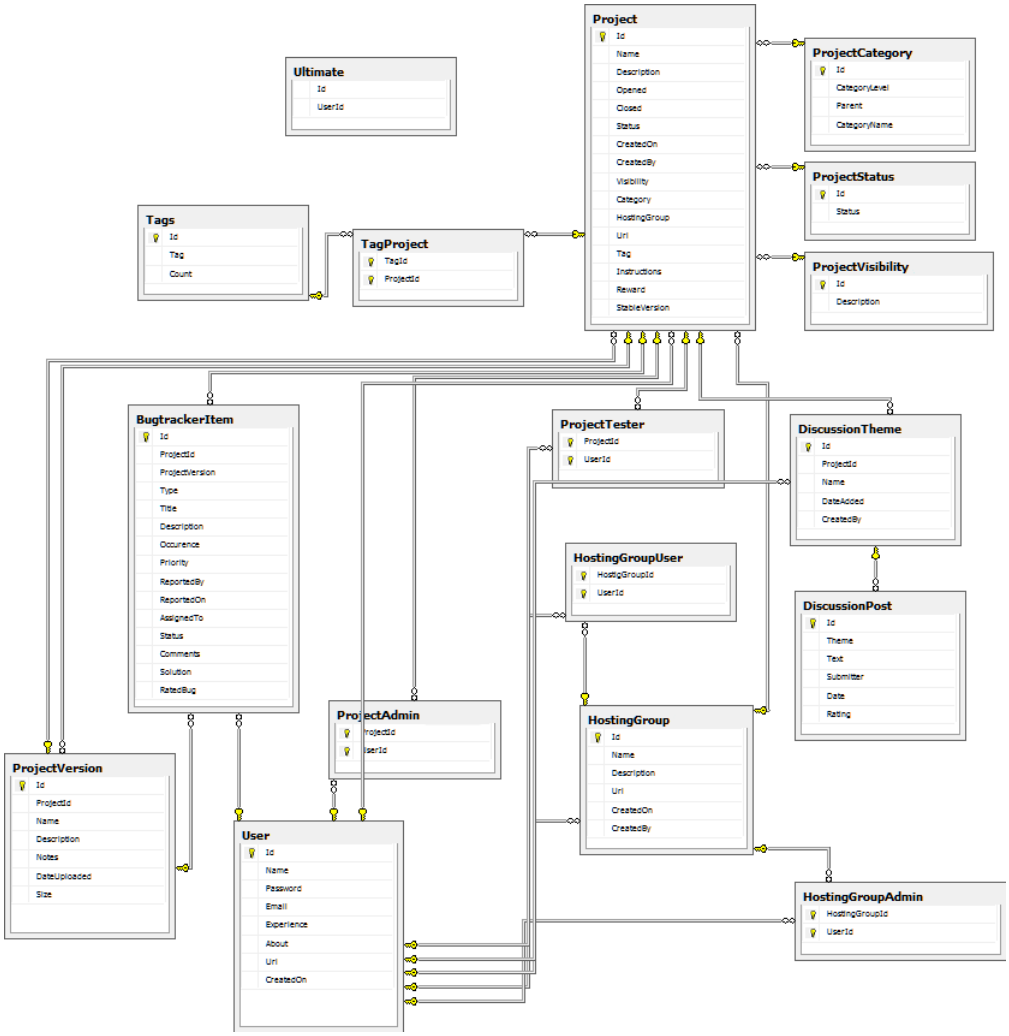
Architektúra databázy sa vo veľkej časti podobá tej, ktorá bola navrhnutá v kapitole 6.3. Pošas vývoja aplikácie sa však vyskytli nové myšlienky a nápady, prípadne iné spôsoby implementácie ako boli pri návrhu mienené. Preto sa konečný ER diagram databázových tabuliek mierne líší od pôvodne navrhnutého. Finálny vzhľad databázy zobrazuje obrázok 8.1.

Ku komunikácii medzi databázovým serverom a aplikáciou bolo využitých možností ADO .NET Entity Frameworku. Z databázy bol automaticky vygenerovaný model, kde tabuľky sú reprezentované ako entity, ktoré je možné instanciovať a používať ako objekty.

Vzhľadom k faktu, že programátorovi pri použití Entity Frameworku odpadá nutnosť implementovať komunikáciu aplikácia - databázový server nie je pre dátovú vrstvu nutné vytvárať nový projekt. Namiesto toho je v aplikácii prítomná pomocná trieda pre prácu s databázou. V nej sú za pomoci technológie LINQ vykonávané dotazy nad objektami modelu.

Nasledujúca ukážka demonštruje príklad získania objektu HostingGroup na základe unikátneho identifikátoru za pomoci technológie LINQ a Entity Frameworku.

```
public static HostingGroup getHostingGroupById(int id)
{
    using (DatabaseEntities database = new DatabaseEntities())
    {
        HostingGroup hGroup = (from hg in database.HostingGroup
            where hg.Id == id select hg).FirstOrDefault();
        database.Detach(hGroup);
        return hGroup;
    }
}
```



Obrázok 8.1: Skutočný vzhľad databázy exportovaný z Microsoft SQL Server Management Studio.

8.3 Popis stránok

Ako bolo uvedené v predchádzajúcej kapitole aplikácia bola vytvorená za pomoci technológie ASP .NET. Konkrétne bola použitá technika formulárov, kde každý formulár, až na niektoré nižšie uvedené výnimky predstavuje jednu webovú stránku. Každá takáto stránka sa skladá z troch súborov.

Súbor s príponou *.aspx* udáva vzhľad formuláru a je tvorený webovými prvkami a HTML tagmi. Súbor s príponou *.cs* je zdrojový súbor v jazyku C# a obsahuje deklarácie tried. Súbor s príponou *.designer.cs* je automaticky generovaný súbor, ktorý udržiava informácie o formulári vytvorenom pomocou Visual Studia.

Okrem samotných formulárov je vzhľad stránky daný skrývaním a zobrazovaním panelov obsahujúcich rôznu funkcionalitu, ktoré sa na stránke nachádzajú.

8.3.1 Špeciálne súbory

Master Page
Formulár Site nemá príponu *.aspx*, ale *.master*. Takéto stránky sa volajú MasterPage a fungujú ako vzory alebo šablóny stránok. Definujú fixný obsah a dekladujú časť stránky, kam je možné vkladať vlastný obsah.[10] Takýmto spôsobom je možné jednoducho vytvoriť základný vzhľad webu. Výhodou tohto prístupu je taktiež to, že pri zmene kódu v Master Page sa zmeny automaticky prejavia na všetkých stránkach, ktoré ju využívajú ako vzor.

Web.config *Web.config* je súbor nastavení a konfigurácii pre ASP .NET aplikácie. Je špecifikovaný XML tagmi a atribútmi. Všetky informácie sú zadané vnútri koreňového tagu `<configuration> </configuration>`. Dôležitou vlastnosťou tohto spôsobu konfigurácie ASP .NET aplikácie je možnosť meniť určité nastavenia a premenné bez nutnosti nasadenia jej novej verzie. [26]

Nasledujúce zdrojové kódy zobrazujú príklady nastavení niektorých vlastností aplikácie a jej premenných pomocou súboru Web.config.
Nastavenie Connection String pre pripojenie k databáze pri použití modelu Entity Frameworku

```
<connectionStrings>
    <addname="DatabaseEntities"connectionString=
        "metadata=res://*/Model.csdl|res://*/Model.ssdl|res://*/Model.msl;
        provider=System.Data.SqlClient;provider connection string=
        "data source=69.175.122.58,8888; initial catalog=Database_test;
        persist security info=True; user id=xfarbi00; password=123456789;
        multipleactiveresultsets=True;
        App=EntityFramework"providerName="System.Data.EntityClient"/>
</connectionStrings>
```

Nastavenie emailovej adresy vrátane SMTP serveru pre rozosielanie emailov aplikáciou Crowdproof.

```
<system.net>
    <mailSettings>
        <smtp deliveryMethod="Network" >
            <network host="eva.fit.vutbr.cz" userName="xfarbi00"
                password="12345678"port="587"/>
        </smtp>
    </mailSettings>
</system.net>
```

Nastavenie autentikácie aplikácie za pomoci formulárovej metódy s metódami implementovanými na stránke Login.aspx.

```
<authentication mode="Forms" >
    <forms loginUrl="~/Login.aspx"timeout="2880"/>
</authentication>
```

8.3.2 Životný cyklus stránky

Ako už bolo povedané samotné formulárové stránky udávajú vzhľad a logiku aplikácie za pomoci HTML tagov a konštrukcií jazyka C#. V tomto bode je vhodné popísať ako stránky

vlastne fungujú a predstaviť ich životný cyklus a akcie, ktoré sú vykonávané od požiavadku na stránku až po jej zrušenie. [16]

- **Požiadavka na stránku (Page Request)** - Požiadavka sa vykystne už pred tým ako sa samotný životný cyklus aplikácie začne. Po požiadavku na stránku ASP .NET rozhodne, či stránka potrebuje byť parsovaná a skompilovaná alebo je vrátená stránka uložená v pamäti cache.
- **Štart (Start)** - V kroku štart sa nastavujú vlastnosti (properties) stránky. Ide napríklad o Request, Response alebo IsPostback.
- **Inicializácia (Initzialization)** - V priebehu inicializácie sú načítané jednotlivé ovládacie prvky a je im priradené unikátne ID (UniqueId).
- **Načítavanie (Load)** - V prípade, že aktuálna požiadavka na stránku je postback sú načítané vlastnosti ovládacích prvkov spolu s informáciami získanými z ViewState.
- **Spracovanie udalosti postback (Postback event handling)** - Ak požiadavka na stránku je postback je vyvolaná udalosť ovládacieho prvku. Potom je volaná funkcia Valid na každý validátor a je nastavený príznak IsValid.
- **Rendrovanie (Rendering)** - Pred rendrovaním je uložený ViewState pre všetky ovládacie prvky.
- **Zrušenie (Unload)** - Stránka je kompletne vyrendrovaná, zaslaná klientovi a je pripravená k zahodeniu.

Zaujímavejšie pre praktickú implementáciu však nie sú samotné stavy stránky, ale udalosti, ktoré môže programátor odchytiť a reagovať na ne. Týmito udalosťami sú

- **PreInit** - Je vyvolaný po tom, čo je ukončená fáza štart. Je možné ho použiť k opätovnému vytvoreniu dynamických ovládacích prvkov alebo dynamickému nastaveniu Master Page.
- **Init** - Je vyvolaný po tom ako sú načítané všetky ovládacie prvky. Udalosť Init jednotlivých ovládacích prvkov sa vykytuje pred udalosťou Init samotnej stránky.
- **InitComplete** - Je vyvolaný po ukončení inicializačnej fázy a používa sa k zmenám ViewState, ktoré majú byť zachované po ďalšom postbacku.
- **PreLoad** - Udalosť PreLoad je vyvolaná po tom, čo je načítaný ViewState
- **Load** - Stránka volá udalosť OnLoad na objekt stránky a rekurzívne na všetky ovládacie prvky stránky. Udalosť Load ovládacích prvkov sa vyskytuje až po udalosti Load samotnej stránky. OnLoad udalosť sa používa k nastaveniu vlastností ovládacích prvkov a ustanoveniu pripojeniu k databáze.
- **LoadComplete** - Udalosť vyvolaná na konci fázy spracovávania udalostí. Je používaná k úlohám, ktoré vyžadujú načítané všetky ovládacie prvky stránky.
- **PreRender** - PreRender je vyvolaná po tom, čo objekt stránky vytvoril všetky ovládacie prvky potrebné k rendrovaniu stránky. Objekt stránky vyvolá udalosť PreRender na stránku a potom rekurzívne na všetky ovládacie prvky. Používa sa k vykonaniu posledných zmien obsahu stránky.

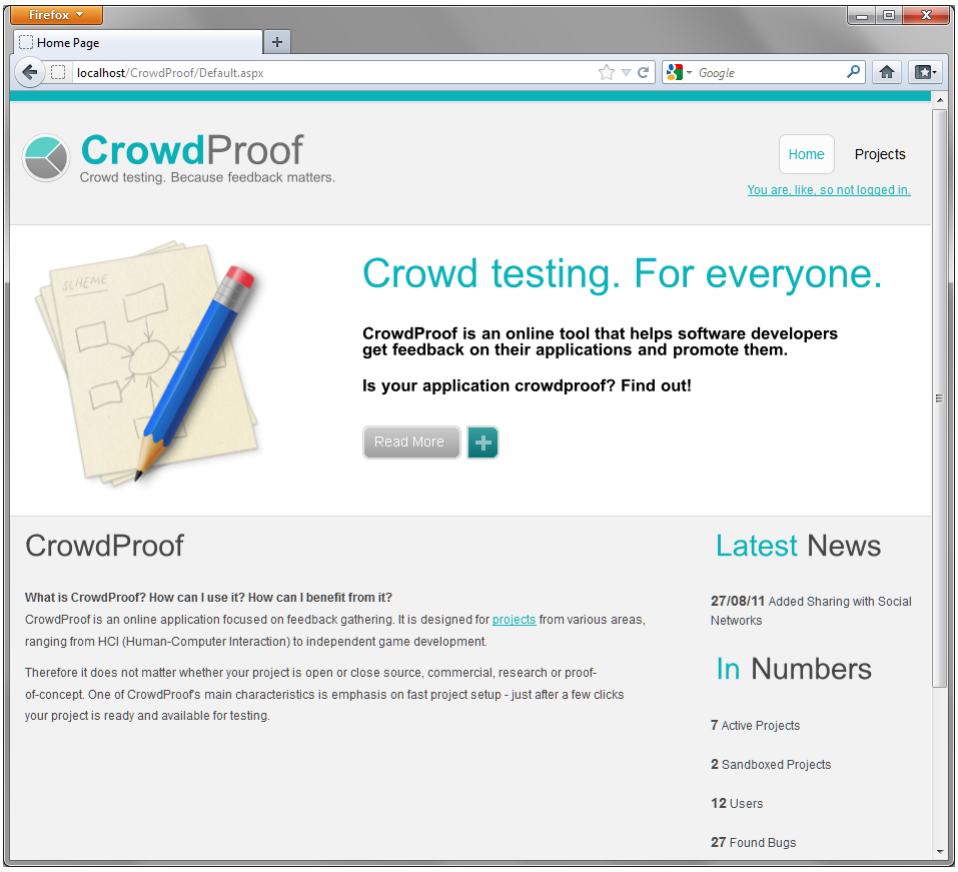
- **PreRenderComplete** - Je vyvolaný po tom, čo každý ovládací prvkov, ktorého DataSourceId je nastavené zavolá metódu DataBind.
- **SaveStateComplete** - Udalosť je vyvolaná po uložení ViewState.
- **Render** - Render nie je udalosť, namiesto toho objekt stránky v tomto bode volá metódu Render na každý ovládací prvok.
- **Unload** - Rovnako ako Render nie je udalosť, ale metóda. Volá sa na každý ovládací prvok a potom na stránku. Slúži na uzatvorenie otvorených databázových spojení, súborov alebo iných úloh.

8.3.3 Dizajn

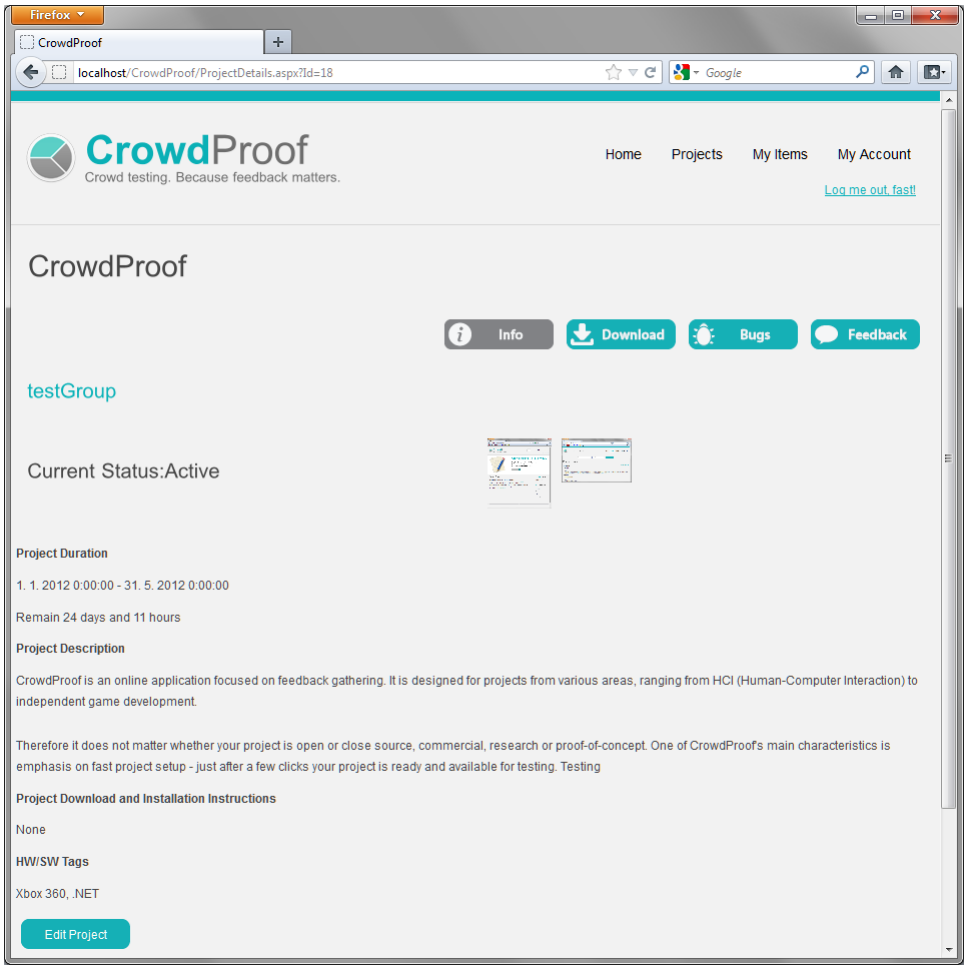
V dizajne vytvorenom pre aplikáciu dominuje najmä jednoduchosť. Tá by mala podčiarkovať rovnako jednoduché používanie aplikácie a udržať ju prehľadnú a zároveň funkčnú. Základná grafická kostra aplikácie nie je vytvorená autorom tejto diplomovej práce. Použitá bola voľne stiahnuteľná šablóna dostupná z internetovej stránky `www.hotwebsitetemplates.net`. Druhou použitou komponentou, ktorá nebola vytvorená priamo autorom je skript pre zobrazovanie obrázkov Lightbox. Mimo kostry dizajnu a komponenty Lightbox bol zvyšok grafickej stránky práce vytvorený jej autorom. To zahŕňa zvyšné štýly CSS, ikony, úpravy originálnej šablóny a ďalšie. Grafický dizajn sa v detailoch môže líši v rôznych internetových prehliadačoch, no celkový obraz aplikácie je vo všetkých veľmi podobný a jednoznačný.

8.3.4 Ukážky aplikácie

Nasleduje niekoľko ukážok aplikácie. Ide konkrétne o ukážku domovskej stránky aplikácie a stránky so základnými informáciami o projekte. Ďalšie ukážky aplikácie sú uvedené v dodatku A.



Obrázok 8.2: Stránka Default je prvá, stránka aplikácie. Zobrazuje základné informácie o aplikácii a niektoré štatistiky.



Obrázok 8.3: Stránka Project zobrazuje základné informácie daného projektu, navigáciu k ďalším nástrojom, možnosť prihlásiť sa k testovaniu alebo editovať projekt.

Kapitola 9

Testovanie aplikácie

Testovanie aplikácie vytvorenej pre účely tejto diplomovej práce prebehla použitím aplikácie samotnej. V aplikácii CrowdProof bol vytvorený nový projekt, ktorý cez URL v informáciách o projekte ukazoval sám na seba. V sekcii Bugspotting boli pridávané chyby a návrhy zmien aplikácie. Tie boli opravované a aplikácia tak bola jednoducho otestovaná.

9.1 Zaujímavé problémy zistené pri testovaní

Ako najväčší problém pri testovaní sa prejavila rôznorodosť internetových prehliadačov. Celý vývoj aplikácie a testovanie programátorom počas vývoja bolo vykonávané za použitia internetového prehliadača Mozilla Firefox 11.0. Finálne testovanie aplikácie však prebehlo na štyroch rôznych prehliadačoch. Konkrétne Mozilla Firefox vo verzii 11.0, Google Chrome vo verzii 18.0.1025.151, Opera vo verzii 11.60 a Internet Explorer vo verzii 9.0.5. V nasledujúich riadkoch budú uvedené najzaujímavejšie zistenia a chyby vrámci jednotlivých prehliadačov.

9.1.1 Cesty k obrázkom

Ktoré prehliadače fungovali správne : Mozilla Firefox, Opera
Ktoré prehliadače nefungovali : Google Chrome, Internet Explorer
Popis problému : Prehliadače spracúvajú rôznym spôsobom relatívne cesty k obrazovým súborom uloženým v adresári na serveri pri použití HTML komponenty img a použití adresovania v tvare `src="/adresár/súbor.jpg"`. Cesta k súboru je potom nesprávna a obrázok nie je zobrazený.
Riešenie : Namiesto relatívnych ciest vyššie zmieneného tvaru používa aplikácia cesty k obrázkom za využitia špeciálneho znaku tilda „~“. Znak tilda sa v ASP .NET používa k získaniu relatívnej cesty koreňového URL. Tak je jednoznačne definovaná absolútna cesta k obrázku. Toto riešenie funguje vo všetkých prehliadačoch bez obmedzení.

9.1.2 Nezobrazené nadpisy

Ktoré prehliadače fungovali správne : Mozilla Firefox, Opera, Google Chrome
Ktoré prehliadače nefungovali : Internet Explorer
Popis problému : Prehlidač Internet Explorer nezobrazuje texty s CSS triedami h1, h2 a h3 definovanými nasledujúcim predpisom.

```
h1
{
  margin: 0;
  padding: 40px 0 40px 60px;
  color: #6e6e6e;   font: normal 40px/1.2em Arial, Helvetica, sans-serif;
}

.body h2
{
  font: normal 30px Arial, Helvetica, sans-serif;
  color: #464646;
  padding: 10px 5px;
  margin: 5px 0 10px 0;
}

.body h3
{
  font: normal 20px Arial, Helvetica, sans-serif;
  color: #464646;
  padding: 10px 5px;
  margin: 5px 0 10px 0;
}
```

Výsledok tejto chyby je možné pozorovať na obrázku 9.1, ktorý demonštruje zobrazenie rovnakej stránky v prehliadačoch Mozilla Firefox a Internet Explorer vo vyššie uvedených verziách.
Riešenie : Programové riešenie nebolo nájdené. Zobrazenie textov s vyššie uvedenými triedami CSS je však možné v Internet Explorer Compatibility View.

9.1.3 Neobslúžená udalosť vyvolaná RadioButtonList komponentou

Ktoré prehliadače fungovali správne : Mozilla Firefox, Google Chrome, Internet Explorer
Ktoré prehliadače nefungovali : Opera
Popis problému : Na stránke NewProject.aspx sa nachádza komponenta RadioButtonList, ktorá po vybraní jednej z možností zobrazí panel prislúchajúci výberu, ktorý umožní ďalšie pokračovanie pri tvorbe nového projektu. V prehliadači Opera je však po kliknutí na jednu z položiek komponenty RadioButtonList vyvolaný postback no obsluha vyvolanej udalosti nie je spustená a v komponente RadioButtonList nie je ani jedna z položiek označená ako vybratá.
Riešenie : Riešenie problému spočívalo vo výmene použitej komponenty RadioButtonList za zloženie dvoch komponent RadioButton, ktorým bola nastavená spoločná vlastnosť GroupName. Vďaka nastaveniu tejto vlastnosti môže byť označená vždy len jedna z komponent RadioButton a na základe zmeny ich aktivity sa zobrazuje súvisiaci panel. Prehliadač Opera nemá problém zachytávať udalosti komponenty RadioButton, takže toto riešenie funguje bez akýchkoľvek problémov.

9.2 Rýchlosť aplikácie

Rýchlosť webovej aplikácie je po jej spoľahlivosti druhou najdôležitejšou vlastnoťou. Pri užívateľskom testovaní sa aplikácia v niektorých momentoch javila ako pomalá. Preto bola aplikácia preskúmaná za pomoci techniky trasovania (anglicky tracing).

Trasovanie technológie ASP .NET umožňuje zobrazenie diagnostickej informácie o jednom požiadavku na stránku, sledovanie cesty spustenej stránky, zobrazenie diagnostickej informácie za behu a ladenie aplikácie. [15]

Dôležitou vlastnosťou tejto technológie je možnosť vloženia vlastných trasovaích správ do zdrojového kódu. Touto technikou je možné jednoznačne označit miesto v kóde a zároveň čas, ktorý trvalo aplikácii prejsť od spustenia stránky do tohto miesta. Tak je možné v aplikácii označiť dva body a odčítaním ich časov získať informáciu o tom ako dlho stránka strávila v úseku kódu medzi týmito dvoma bodmi.

Pre povolenie trasovania musí konfiguračný súbor Web.config obsahovať nasledovný element

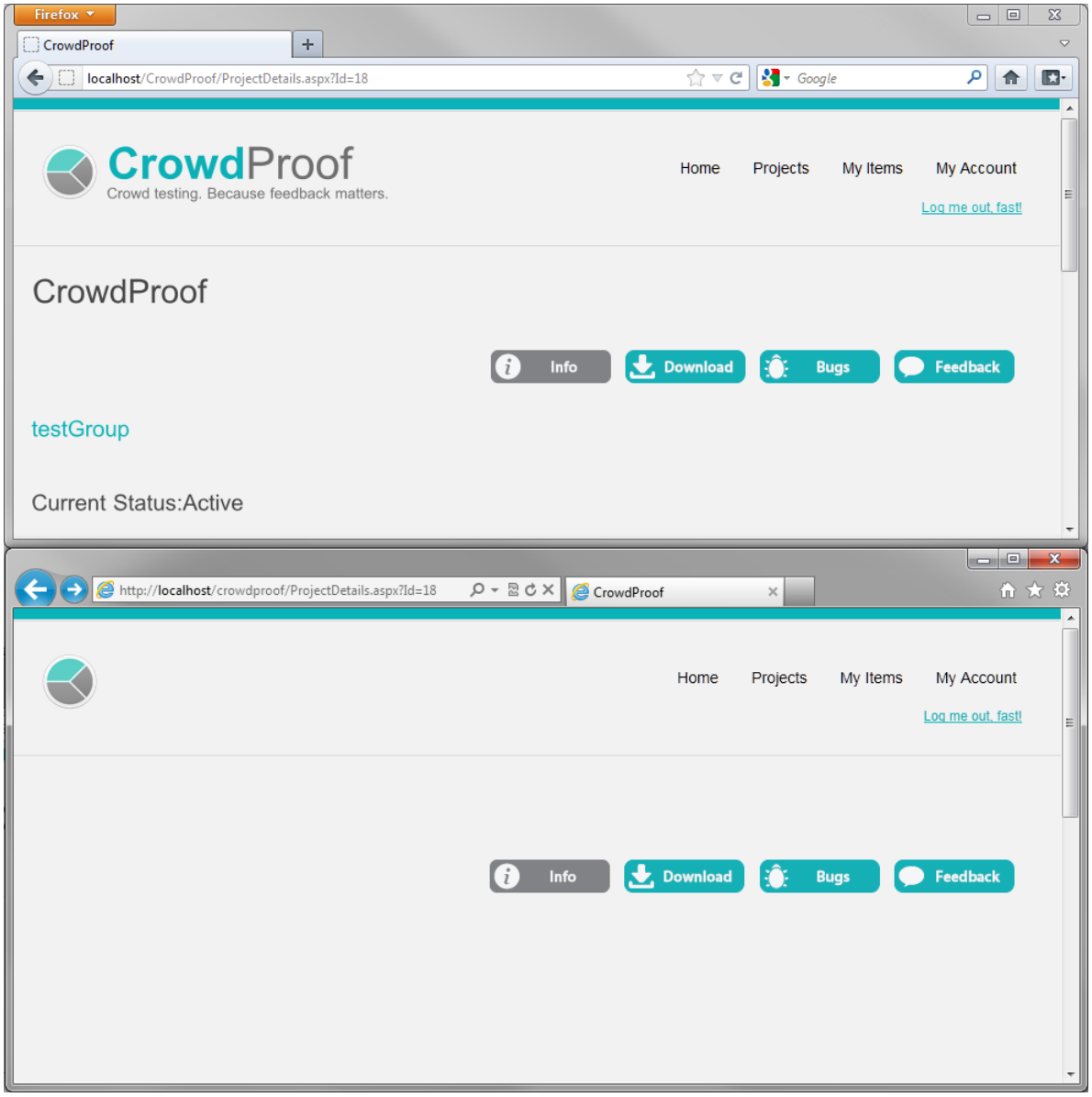
```
<configuration>
    <system.web>
        <trace enabled="true"/>
    </system.web>
</configuration>
```

Vloženie vlastnej trasovacej správy v zdrojovm kóde sa vykoná nasledovným zápisom
`Trace.Write("Moja trasovacia správa");`
Výsledky trasovanie je možné prezerať priamo v internetovom prehliadači a to zadaním URL s koreňovým adresárom, nasledovaným stránkou `"/trace.axd"`, teda napríklad
`www.crowd-proof.cz/trace.axd`
(v mojom prípade pri testovaní `"localhost/crowdproof/trace.axd"`).

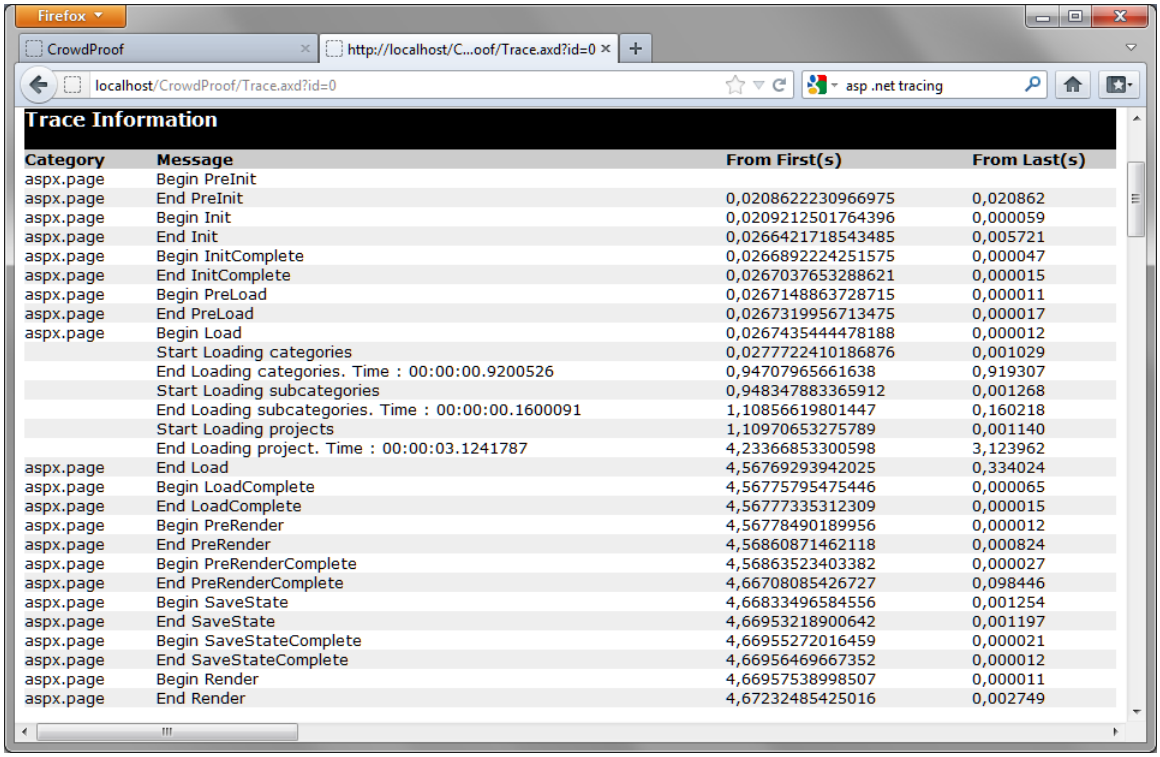
Po označení niektorých bodov v aplikácii bolo zistené, že aplikácia trávi najviac času získavaním dát zo vzdialenej databázy.

Demonštráciu tohto faktu je možné vidieť na obrázku 9.2, ktorý zobrazuje záznam trasovania stránky Search.aspx. Na tejto stránke siaha aplikácia do databázy 3 krát. Pri načítaní kategórií projektov, pri načítaní subkategórií a pri načítaní desiatich najnovších projektov. Trasovacie výpisi boli na tejto stránke nastavené v zdrojovom kóde vždy pred a za riadok, ktorý pristupoval k databáze. Z obrázku je vidieť, že stránke trvalo načítanie pri zaokrúhlení na tisíciny sekundy 4,672 sekundy. Z toho 0,92 sekundy trvalo načítanie kategórií, 0,16 sekundy načítanie subkategórií a až 3,124 sekundy načítanie projektov. Z celkového času načítania stránky 4,672 sekundy tak činil prístup k databáze 4,204 sekundy, čo je 89,98 % času z celkového času načítavania stránky.

Pomocou technológie trasovania tak bolo jednoznačne zistené, že problémom rýchlosti aplikácie je pomalý prístup do databázy. Z časti je tento problém riešený obmedzením prístupu do databázy, využívaním pamäte cache a techniky ViewState. Pre jednoznačné zlepšenie výkonu aplikácie je však potrebné zabezpečiť rýchlejšiu linku k databázovému serveru.



Obrázok 9.1: Ukážka rozdielu na stránke v prehliadačoch Mozilla Firefox a Internet Explorer



Obrázok 9.2: Ukážka stránky trace.axd zobrazujúcej trasovacie informácie

Kapitola 10

Možnosti rozšírenia

V projektoch podobného zamerania sa vždy nájde mnoho oblastí pre zlepšenie. Asi najvýraznejšou zmenou by bola tvorba profesionálneho dizajnu. Dizajn webovej aplikácie sa v poslednom období stal jedným z jej najdôležitejších bodov. Jeho tvorba si však vyžaduje skúseného dizajnéra orientujúceho sa v pokročilých grafických nástrojoch a dizajnových trendoch.

Ďalším bodom v ktorom je možné aplikáciu vylepšiť v širokej miere je funkcionalita. Súčasný stav umožňuje pohodlné testovanie vytvorených projektov, zoznam chýb, diskusie a ďalšie možnosti. Avšak existuje mnoho funkcií, ktoré by aplikáciu vhodne rozšírili. Ako príklady takýchto funkcií je možné uviezť možnosť dynamicky vytvoriť HTML stránku pre informačný panel projektu, možnosť vytvorenia dotazníku pri uzatvorení projektu, ktorý môžu vyplniť užívatelia testujúci projekt. Na základe takto vyplneného dotazníku by potom bolo možné exportovať výsledky do XML alebo XLS dokumentov, prípadne generovať grafy vhodné do dokumentácie.

Najvýraznejším bodom kde by aplikácia oproti testovanej konfigurácii potrebovala zlepšiť je rýchlosť dátovej linky medzi aplikačným a databázovým serverom. Práve táto sa počas testovania ukázala ako výrazým problém, ktorý je viac než vhodné odstrániť.

Vývoj podobných webových aplikácií je v dnešnej dobe živý cyklus, ktorý je možné neustále vylepšovať, rozširovať a upravovať. Preto aj aplikácia Crowdproof má možnosť ďalšieho vývoja vrámci funkcionality, ale napríklad aj nových trendov v oblasti HCI, ktoré môžu ešte viac priblížiť a zjednodušiť interakciu ľudí s počítačom.

Kapitola 11

Záver

Cieľom diplomovej práce bolo vytoriť systém, ktorý poskytne verejnosti zaoberajúcej sa problematikou Human - Computer Interaction nástroj skrz ktorý môžu jednoducho a rýchlo zdieľať svoju tvorbu, nechať ju testovať iným vývojárom a nadšencom, vytvoriť komunitu ľudí s vášňou pre oblasť interakcie človeka s počítačom a posúvať tak hranice v sfére HCI stále napred.

Súčasne s textom práce vznikala aplikácia CrowdProof, ktorá je praktickou odpoveďou na zmienené otázky. Jej cieľom je byť efektívna, zrozumiteľná a nápomocná pre jej užívateľov.

V prvých piatich kapitolách bola práca teoretickou prípravou pre neskoršiu implementáciu. Diskutovala dnešné trendy v tvorbe webu a projekty podobné tomu, ktorý je jej súčasťou. V samostatnej kapitole vysvetlila čo znamená HCI, kam až zasahuje táto oblasť a že nie je vecou technologických nadšencov, ale každodennou realitou nás všetkých. Práve uvedomenie tohto faktu podčiarkuje význam projektov, ktoré CrowdProof zastrešuje. Práca ďalej v rýchlosti popísala ako sa dnešné aplikácie testujú a priblížila fenomén zvaný beta testovanie, ktorý je stále viac a viac používaný a získava silné postavenie z viacerích hľadísk. Okrem základnej funkcie, testovania, totiž ponúka vytvorenie pevného stavebného kameňa pre marketing danej aplikácie. A práva marketing je dnes chceme, či nechceme to, čo užívateľov láka k produktu.

Práca týmito piatimi kapitolami dala kvalitný základ a podklady pre návrh aplikácie CrowdProof v kapitole 6, kde predstavuje základné systémi aplikácie a v hrubom nadhľade jej hlavné rysy. Za návrhom nasleduje rýchly popis technológií, ktoré umožnili aplikácii CrowdProof vzniknúť a ďalej v kapitole 8 jej samotnú implementáciu. Táto popísala výsledok práce a za pomoci screenshotov predstavila jej vzhľad a dizajn. Po nej nasledovala krátka diskusia o testovaní a problémoch vzniknutých pri tvorbe aplikačnej časti diplomovej práce a konečne návrhy a postrehy pre jej ďalšie rozšírenie.

Najdôležitejšími dvomi bodmi mojej diplomovej práce sú zanechať v jej percipientovi vedomie o existencii oblasti Human - Computer Interaction a miery do akej zasahuje do našich životov a dôležitosti správneho testovania aplikácií. Náväzne na to potrebu projektov podobných projektu CrowdProof, ktoré umožňujú podporovať tieto dva body.

Literatúra

[1] Agarwal, B. B.; Tayal, S. P.; Gupta, M.: *Software Engineering & Testing*. Jones and Bartlett Publishers, 2010, iSBN 978-1-934015-55-1.

[2] Boehm, B. W.: A Spiral Model of Software Development and Enhancement. 1988. URL http://weblog.erenkrantz.com/~jerenk/phase-ii/Boe88.pdf

[3] Cohen, M. H.; Giangola, J. P.; Balogh, J.: *Voice User Interface Design*. Addison-Wesley, 2004, iSBN 0-321-18576-5.

[4] Drobíková, B.: Zpráva z konference Human-computer interaction a informační služby. 2008, iSSN 1212-5075.

[5] Glenford, J. M.: *The Art of Software Testing, Second Edition*. Wiley & Sons, Inc., 2004, iSBN 0-471-46912-2.

[6] Hlava, T.: Úrovně provádění testů. 2011. URL http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu

[7] Kolektív autorov: ACM SIGCHI Curricula for Human-Computer Interaction. 1996, iSBN 0-89791-474-0. URL http://old.sigchi.org/cdg/index.html

[8] Kolektív autorov: Guide to the Software Engineering Body of knowledge. 2004, iSBN 0-7695-2330-7.

[9] Lerman, J.: *Programming Entity Framework*. O'Reilly Media, Inc., 2012, iSBN 978-0-596-80726-9.

[10] MacDonald, M.; Freeman, A.; Szpuszta, M.: *ASP.NET 4 a C# 2010*. ZONER software a.s., 2011, iSBN 978-80-7413-131-8.

[11] McLaughlin, B.: Mastering Ajax, Part 1: Introduction to Ajax. 2005. URL http://www.ibm.com/developerworks/web/library/wa-ajaxintro1/index.html

[12] Moravec, Z.: RIA - Rich Internet Applications. 2009. URL http://programujte.com/clanek/2009041200-ria-rich-internet-applications/

[13] MSDN: ADO.NET Entity Framework At-a-Glance. URL http://msdn.microsoft.com/en-us/data/aa937709

[14] MSDN: The ADO.NET Entity Framework Overview.
URL http://msdn.microsoft.com/en-us/library/aa697427%28v=vs.80%29.aspx

[15] MSDN: ASP.NET Tracing Overview.
URL http://msdn.microsoft.com/en-gb/library/bb386420.aspx

[16] MSDN: ASP.NET Page Life Cycle Overview. 2011.
URL http://msdn.microsoft.com/en-us/library/ms178472.aspx

[17] MSDN: .NET Framework Conceptual Overview. 2011.
URL http://msdn.microsoft.com/library/zw4w595w.aspx

[18] Nanni, P.: Human-Computer Interaction: Principles of Interface Design. 2004.
URL http://www.vhml.org/theses/nannip/HCI_final.htm#_Toc87596271

[19] Neznámy autor: About Silveright.
URL http://www.microsoft.com/silverlight/what-is-silverlight/

[20] Neznámy autor: Software Testing strategies.
URL http://www.cs.nott.ac.uk/~cah/G53QAT/G53QAT08pdf6up.pdf

[21] Neznámy autor: Adobe Flash Player PC Penetration. 2008.
URL http://www.adobe.com/products/player_census/flashplayer/PC.html

[22] Nielsen, J.: *Usability Engineering*. Academic Press, 1993, iSBN 0-12-518406-9.

[23] Oviatt, S.: Multimodal Interfaces. 2002.
URL http:
//www.cogsci.msu.edu/DSS/2004-2005/Oviatt/Multimodel%20Interfaces.pdf

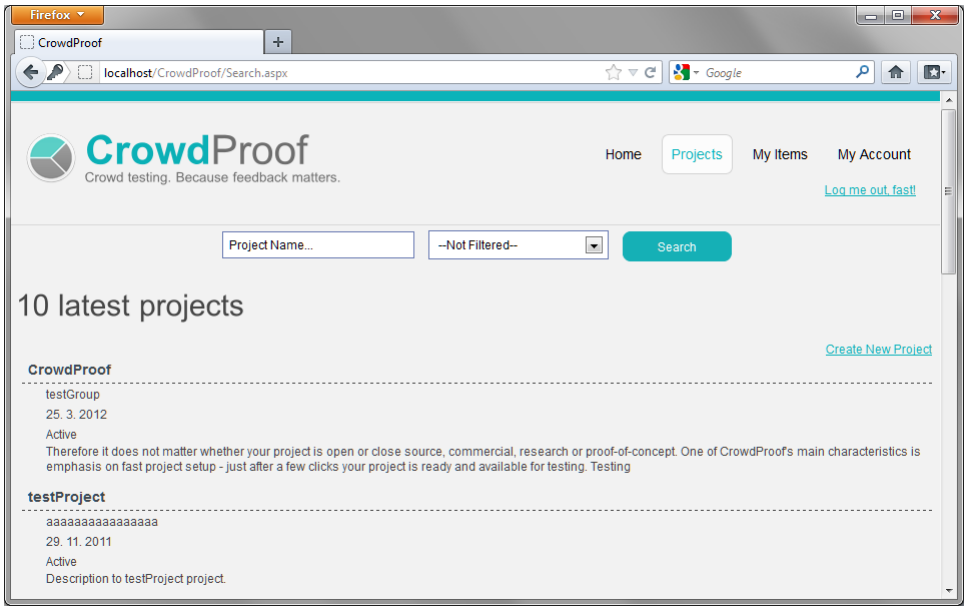
[24] Patton, J.: *Software Testing*. Sams Publishing, 2005, iSBN 0-672-32798-8.

[25] Pichlík, R.: Rich Internet Application. 2005.
URL http://interval.cz/clanky/rich-internet-application

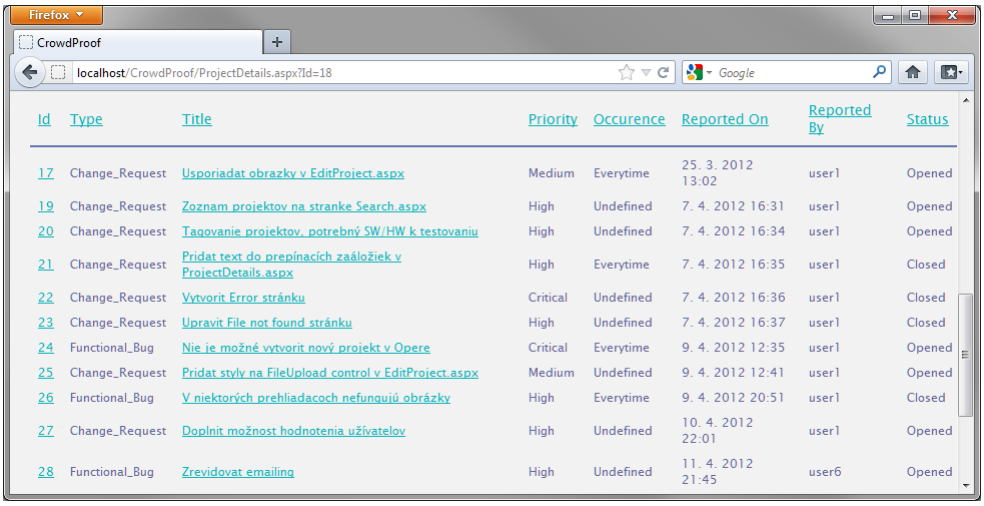
[26] Sharma, P.: Core characteristict of Web 2.0 Services.
URL http://www.techpluto.com/web-20-services

Dodatok A

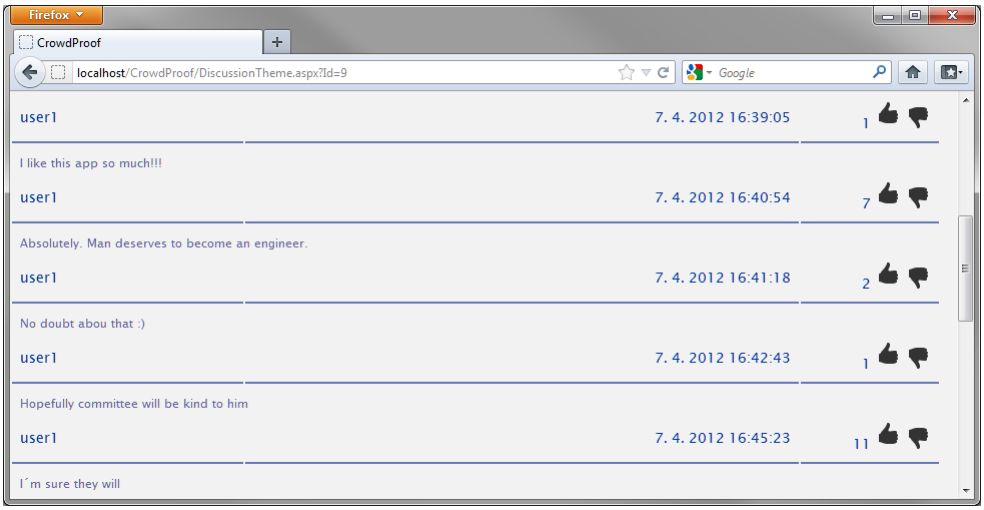
Ukážky aplikácie



Obrázok A.1: Stránka Search zobrazuje zoznam projektov prítomných v aplikácii a poskytuje možnosť vyhľadávania projektov na základe mena a kategórií.



Obrázok A.2: Stránka Project s aktívnym panelom Bugtracker, ktorý zobrazuje zoznam nájdených chýb.



Obrázok A.3: Druhý krát stránka Project s aktívnym panelom Feedback, ktorý poskytuje možnosť užívateľskej diskusie.

Dodatok B

Prevzaté obrázky

Obrázok 2.1 prevzatý z http://interval.cz/clanky/rich-internet-application/
Obrázok 3.1 prevzatý z http://sourceforge.net
Obrázok 3.2 prevzatý z http://code.google.com
Obrázok 3.3 prevzatý z http://codeplex.com
Obrázok 4.1 prekreslený podľa [4]
Obrázok 4.2 prevzatý z http://oldcomputers.net/pics/macintosh.jpg
Obrázok 4.3 prevzatý z http://macsite.sk/wp-content/uploads/siri.100411.001.jpg
Obrázok 5.2 prekreslený podľa [1]
Obrázok 5.3 prekreslený podľa [2]

Dodatok C

Odkazy

Odkaz na video g.tec intendiX-SOCI http://www.youtube.com/watch?v=en_RbBRGnmE

Dodatok D

Obsah CD

- CrowdProof - adresár so zdrojovými kódmi
- xfarbi00.pdf - text diplomovej práce vo formáte PDF
- xfarbi00_latex - zdrojové kódy k textu diplomovej práce
- plakat.pdf - plagát vytvorený k diplomovej práci vo formáte PDF