Sem vložte zadání Vaší práce.

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

Bakalářská práce

Webová aplikace na odhad ceny vývoje software

Filip Staněk

Vedoucí práce: Ing. Jakub Hladík

11. května 2015

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Jakubovi Hladíkovi za jeho podporu při vytváření této práce. Děkuji mu především za čas, který mi věnoval a za cenné rady z praxe. Dále bych rád poděkoval své rodině, která mi byla oporou po celou dobu studia.

Prohlášení

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

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

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Staněk, Filip. *Webová aplikace na odhad ceny vývoje software*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Náplní této práce bylo navrhnout a vytvořit webovou aplikaci pro odhadování ceny vývoje softwarových projektů. V práci jsem se zaměřil na menší zakázkový software, jehož problematiku odhadování jsem analyzoval a na základě zjištěných technik a metod vhodných k jeho odhadu jsem navrhl funkcionalitu vlastní aplikace. Aplikaci jsem následně implementoval ve webovém frameworku Ruby on Rails a po vytvoření ji otestoval na reálném projektu, zhodnotil její přínosy a navrhl možnosti nasazení aplikace na trhu. Vytvořená aplikace umožňuje odhadovat softwarové projekty pomocí metody expertních odhadů a dekompozice. Pro výpočet celkového odhadu projektu je v aplikaci použita metoda PERT. Hlavním přínosem aplikace je zvýšená spolehlivost odhadnuté ceny vývoje, systematizace a správa softwarových odhadů.

Klíčová slova aplikace na odhadování softwaru, webová aplikace, Ruby on Rails, problematika odhadování softwaru, metody odhadu softwaru, PERT, projektové řízení

Abstract

The main goal of this thesis was to design and create web application for estimating the cost of software development. In this thesis I focused specially on smaller on-demand software and the issue of it's estimation. I analyzed methods and techniques suitable for estimating smaller software and I designed my own application based on them. The application was implemented with the Ruby on Rails framework and then tested on real project. Thesis also contains the summary of applications benefits and proposal for deploying the application on web. Created app supports expert based estimation and decomposition of estimates. PERT method is used in application to calculate total project cost. Application is primary intended for managing project estimates and to increase the reliability of estimation.

Keywords application for software estimation, web application, Ruby on Rails, issues in software estimation, software estimation methods, PERT, project management

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Odhadování nákladů na vývoje softwaru je důležitou součástí procesu tvorby komerčního softwaru. Obzvláště pro vývojáře zakázkového softwaru je odhadování ceny vývoje stěžejní pro vytváření cenových nabídek, popřípadě kalkulací. Pro zákazníka, který si software na zakázku objednává, je totiž právě cena často tím rozhodujícím faktorem, podle kterého si dodavatele softwaru vybírá. Na základě spolehlivého odhadu je možné zákazníkovi nabídnout cenu, která je pro něho dostatečně zajímavá a zároveň lze za ní software v požadované kvalitě realizovat a díky tomu zakázku získat a stále na ní vydělat. Mít k dispozici dostatečně přesný odhad pracnosti a ceny vývoje je tak prvním krokem k úspěšnému projektu a může znamenat nemalou konkurenční výhodu.

Z důvodu mnoha aspektů, které v době odhadu nejsou o softwaru a jeho způsobu implementace známy, je však odhadování ceny vývoje relativně náročná a nejistá záležitost. Proto, aby byla zajištěna přesnost odhadu, je třeba při jeho tvorbě aplikovat určité metody a postupy a mít v odhadech zavedený spolehlivý systém. Celkově tedy věnovat odhadům dostatečnou pozornost.

Odhady softwaru však bývají mnohdy zanedbávány, a to především u menších softwarových projektů, u kterých se při odhadu často vychází pouze ze zkušeností z předešlých zakázek bez toho, aby bylo hlouběji analyzováno jejich zadání nebo k odhadu použita nějaká sofistikovanější metoda, což může značně zvýšit riziko podhodnocení pracnosti projektu.

Jako vývojář mobilních aplikací se s nutností odhadovat cenu vývoje softwaru setkávám skoro u každé poptávky na vytvoření aplikace. Ocenil bych tedy nástroj, který by mi s vytvářením a správou odhadů pomáhal.

Přestože na tvorbu odhadů existují softwarové nástroje, jsou většinou určeny pro velké projekty a před použitím vyžadují zdlouhavé a náročné nastavování nebo historická data z předešlých projektů. Metody, které k odhadu používají, navíc nejsou pro menší software vhodné. Další možností pro tvorbu odhadů je použít některé nástroje pro projektové řízení, které umožňují odhadovat cenu softwaru, ty jsou však pro použití výhradně na odhadování zbytečně rozsáhlé.

V této práci se zabývám vytvořením vlastní specializované aplikace pro podporu odhadů sloužící především k odhadování menších softwarových projektů. Chci tak vývojářům a softwarovým firmám poskytnout jednoduchý nástroj, který budou moci využívat pro odhadování tohoto typu softwaru. Doufám, že aplikace vytvořená v rámci této práce pomůže odhadování softwaru zefektivnit a systematizovat a zajistí tak větší přesnost odhadnutých hodnot.

Před vytvořením samotné aplikace nejdříve analyzuji problematiku softwarových odhadů a metod, které se pro odhadování menšího softwaru používají. Na základě analýzy poté navrhnu funkce a vlastnosti, které bude aplikace podporovat. K realizaci svého řešení vyberu vhodné technologie, pomocí kterých budu aplikaci s navrhovanou funkcionalitou implementovat. Vytvořenou aplikaci otestuji na reálném projektu a zhodnotím její přínosy pro použíti v praxi a nakonec zmíním možnosti jejího nasazení na trhu.

Cíl práce Cílem práce je navrhnut a kompletně realizovat webovou aplikaci k tvorbě a správě softwarových odhadů, která umožní pomocí zadaných údajů odhadovat pracnost a cenu vývoje softwaru. K výpočtu odhadu bude aplikace používat metody identifikované v analýze problematiky odhadování menšího softwaru. Jádro aplikace bude implementováno ve webovém frameworku Ruby on Rails a na uživatelské rozhraní bude použit Twitter Bootstrap. Po implementování aplikace bude analyzována její přínosnost pro softwarový vývoj a bude diskutována možnost nasazení aplikace na trhu.

Kapitola 1

Analýza problematiky odhadu ceny softwaru

Odhadování ceny vývoje softwaru probíhá zpravidla ještě předtím, než se začne na softwaru jakkoliv pracovat, tedy v situaci, kdy je o způsobu implementace známo jen málo informací. Odhadování v této fazi může být velmi problematické a pokud se zanedbá, existuje velká šance, že bude zatíženo nemalou chybou. Proto, aby odhadnutá cena s co největší jistotou reflektovala reálné náklady na vývoj, je při odhadování vhodné dodržovat doporučená pravidla a postupy, které mohou přesnost odhadů výrazně zvýšit.

V této kapitole se zaměřuji na problematiku odhadu ceny vývoje zakázkového softwaru, konkrétně na odhady menších mobilních a webových aplikací. Zmiňuji zde, k čemu odhadování softwaru slouží, jaká úskalí představuje a jaké strategie a postupy se pro tvorbu odhadů používají. Dále se zde zaměřuji na nejpoužívanější metody pro výpočet očekávané pracnosti vývoje, které jsou vhodné právě pro odhady mobilních a webových aplikací. Postupy a metody popsané v této kapitole jsou poté stavebním kamenem při návrhu a realizaci vlastní webové aplikace pro podporu odhadů softwaru.

1.1 Odhad softwaru

Slovníková definice odhadu softwaru je: *předběžná kalkulace ceny projektu* nebo *prozatímní vyhodnocení nebo hrubá kalkulace*. Pro účely této práce je odhad softwaru chápán jako: předpověď, jak dlouho bude vývoj softwaru trvat nebo kolik bude stát. [2]

Odhad softwaru je často zaměňován s procesem plánování projektu nebo s tvorbou cenových nabídek. Přestože tyto činnosti spolu úzce souvisí a jejich součásti se prolínají, nelze je spolu jednoduše zaměňovat, neboť každá tato činnost má jiné cíle a jiné výstupy. Odhadování je nezaujatý, analytický proces jehož cílem je přesnost a objektivnost vzhledem k definici softwaru. [2]

V této práci uvažuji tedy pouze technické odhady softwaru, které nejsou zatížené obchodními hledisky. To znamená, že odhadnutá cena softwaru se vztahuje pouze na náklady na jeho vývoj. Odhad není ovlivněn situací na trhu, cílovou skupinou, požadovaným ziskem ani žádnými dalšími hledisky. Nezaujatost odhadů je důležitá zejména pro zpětnou analýzu, porovnávaní odhadů v budoucnosti a celkovou vypovídající hodnotu softwarových odhadů.

1.1.1 Odhad ceny vývoje

Odhad ceny vývoje softwaru se určuje na základě odhadnuté doby trvání (pracnosti) jednotlivých pracovních činností nutných k vytvoření softwaru a hodinových nákladů na jejich uskutečnění. Pokud je tedy k dispozici odhadnutá pracnost, stačí pro každou činnost určit hodinovou sazbu, za kterou je možné ji realizovat a odhad nákladů na vývoj tak jednoduše vypočítat. Hodinová sazba bude v případě softwarové firmy záviset na nákladech na zaměstnance, kteří budou na jednotlivých částech softwaru pracovat.

1.1.2 Použití odhadů

Odhadnutou dobu (pracnost) a cenu vývoje zakázkového softwaru lze použít pro:

- Tvorbu cenové nabídky
- Zjištění rozsahu projektu
- Studii proveditelnosti
- Vyjednávaní o rozsahu a ceně
- Zvážení funkcionalit softwaru a vyřazení nedůležitých požadavků
- Finální cenovou kalkulaci
- Řízení projektových nákladů
- Přidělovaní/nabíraní zaměstnanců na projekt
- Určení přibližného termínu dokončení

1.2 Odhadování menších mobilních a webových aplikací

Prvním důvodem, proč jsem se rozhodl svou práci zaměřit na odhadování menších mobilních a webových aplikací, byl zájem vyřešit vlastní problém s odhadováním tohoto typu softwaru, s jehož zakázkovým vývojem se sám zabývám. Druhým důvodem je, že vývojem menších mobilních a webových

aplikací na zakázku se věnuje značné množství vývojářů a softwarových firem, ať už se jedná o nezávislé vývojáře, studenty informatiky, začínající softwarové firmy nebo zavedená vývojářská studia. Nástroj vytvořený v této práci by jim tedy s odhadování tohoto typu softwaru mohl pomoci.

Než začnu popisovat samotnou metodiky odhadu ceny menšího zakázkového softwaru, nejdříve tuto oblast vývoje softwaru pro účely odhadů charakterizuji. Znalost postupů a požadavků kladených na vývoj menšího softwaru je důležitá k jejich správnému odhadování. Specifičnost vývoje menších aplikací hraje roli při výběru vhodných strategii a metod odhadu a ovlivňuje kdy, kdo a na základě jakých informací bude tyto odhady vytvářet. V této sekci vycházím zejména z vlastních zkušeností.

1.2.1 Vymezení menších mobilních a webových aplikací

Zde jsou uvedeny příklady typů aplikací, na které budu problematiku odhadů primárně vztahovat. Metodika odhadů bude fungovat i na jiné druhy softwaru, ale zde popisované metody a strategie jsou vhodné zejména pro tyto typy aplikací.

1.2.1.1 Menší webové aplikace

- Zakázkové internetové obchody
- Webové aplikace na bázi sociálních sítí
- Reklamní aplikace
- Firemní weby
- Informační portály
- Inzertní servery

1.2.1.2 Menší mobilní aplikace

- Reklamní aplikace
- Festivalové aplikace
- Doplňkové aplikace k webovým službám
- Geolokační aplikace
- Aplikace na bázi sociálních sítí
- Praktické aplikace

1.2.1.3 Malý projekt

U vývoje menších mobilních a webových aplikaci uvažuji, že se jedná o malý projekt. Projekt, na kterém pracuje cca 6 a méně technických pracovníků a trvá maximálně 6 měsíců. Počet osob, které na projektu pracují, se v průběhu realizace projektu nemění.

1.2.2 Obvyklý klient zakázkového vývoje

Potřeby a požadavky obvyklých klientů zakázkového vývoje menších mobilních a webových aplikací mohou podstatně ovlivnit způsob odhadování softwaru, v následujících odstavcích zmiňuji ty nejdůležitější, se kterými je třeba počítat při odhadu.

Častou charakteristikou obvyklého klienta je to, že neví, co přesně od objednávané aplikace požaduje. Má jen obecnou představu, zda se má jednat o firemní web, e-shop, propagační aplikaci, apod. To nemusí vždy vadit, umožňuje to určitou volnost při návrhu řešení, nicméně je nutné počítat s tím, že zákazník bude v průběhu vývoje měnit často názor.

Z tohoto důvodu je zásadní dohodnout se na pevné specifikaci požadavků, která bude součástí softwarové smlouvy a každé další požadavky na změnu si klient bude muset zaplatit zvlášť. Někdy může být problém dohodnout se na tom, co už je změna a co je například odlišné pochopení požadavků klienta. S tím je nutné počítat již při odhadu ceny aplikace a do odhadu započítat rezervu na drobné úpravy, které se nebudou klientovi počítat nad rámec základní smlouvy.

Další věcí, se kterou se lze u klienta často setkat je, že předběžnou cenu softwaru požaduje ještě předtím, než do aplikace investuje jakékoliv vlastní prostředky. To znamená, že tvorbu cenové nabídky zákazník nejen nezaplatí, ale ve finále si může k realizaci aplikace vybrat někoho jiného. Je proto důležité, aby odhad nejen co nejvíce odpovídal realistickým nákladům na vývoj, ale také aby jeho tvorba nezabrala zbytečně moc času.

Zákazník obvykle chce za aplikaci zaplatit co nejméně a zároveň od aplikace očekává mnoho funkcí. Odhad tak může být použit kromě podkladu pro tvorbu cenové nabídky i pro jednání o rozsahu aplikace v závislosti na její ceně. Samotná cenová nabídka by ale neměla být závaznou, ale pouze orientační představou výsledné ceny softwaru, která bude upřesněna při podrobnější analýze, specifikaci požadavků a po vytvoření finální kalkulace.

1.2.3 Fáze nejistoty odhadu

Míra jistoty, že odhadnutá cena vývoje bude odpovídat reálným nákladům, závisí také na procesu postupného upřesňování zadání, který probíhá během analýzy softwaru.

Průběh analýzy malého softwarového projektu lze rozdělit na jednotlivé milníky, které reprezentují kolik informací je o projektu v dané chvíli známo.

S každým dalším milníkem dochází k lepšímu pochopení zadání a k jeho detailnější specifikaci. To značně ovlivňuje přesnost odhadů, kdy v době, kdy existuje pouze počáteční koncept zadání aplikace, se bude pracnost odhadovat mnohem hůře a s větší možností chyby, než na konci analýzy, kdy se vychází z finální specifikace požadavků. [2]

Milníky nebudou většinou jednoznačně vymezeny a mohou se v každém projektu lišit. Průběh analýzy menší aplikace bude obvykle obsahovat tyto milníky [2]:

- Počáteční koncept – počáteční zadání
- Odsouhlasená definice aplikace
- Dokončená analýza požadavků
- Vytvořený prototyp – wireframe
- Sestavení finální specifikace

Odhady provedené ve fázi počátečního konceptu mohou být nepřesné až s koeficientem 4 nahoru i dolů, proto je vhodné se odhadům v této fázi vyvarovat nebo s nepřesností minimálně počítat [2]. Vhodné je vytvořit cenovou nabídku až po několika schůzkách se zákazníkem, po odsouhlasené definici aplikace a po konzultaci požadavků se zákazníkem. Znamená to však nutnou investici do analýzy aplikace ještě před tím, než zákazník podepíše smlouvu. Nicméně přesnost takového odhadu bude větší a cenová nabídka bude mnohem více odpovídat konečné ceně softwaru nebo finální kalkulaci.

1.2.3.1 Wireframe

Nejpřesnější odhad s únosnou časovou investicí do počáteční analýzy lze sestavit po vytvoření tzv. wireframe aplikace, jejímž obsahem jsou prototypy jednotlivých obrazovek aplikace a interakce mezi nimi. Jde o návrh definující základní funkce a vlastnosti, sloužící k celkově lepšímu pochopení aplikace. Vytvoření takového návrhu pomůže vyjasnit zadání aplikace, vymezit požadovanou funkcionalitu a identifikovat potencionální problémy, které by při vývoji aplikace mohly nastat.

Na základě wireframe lze projekt relativně dobře rozložit na jednotlivé úkoly, které budou pro vytvoření aplikace potřeba, a ty odhadnout. Wireframe je také dobrou pomůckou k tomu, aby se při odhadu na nic nezapomnělo. Odhad ceny vzniklý na základě wireframe lze použít nejen pro cenovou nabídku, ale také pro finální kalkulaci a může být i součástí finální specifikace a lze z něj vycházet i při vývoji samotné aplikace.

1.3 Základní strategie odhadu

Jedním ze zásadních rozhodnutí při tvorbě odhadu je úroveň abstrakce, na které se odhad bude provádět. V jednom extrému lze odhadovat pracnost projektu jako celku, v druhém pak odhadovat pracnost jednotlivých činností a aktivit, které se při vývoji softwaru provádějí. Jakou abstrakci si zvolit záleží jak na charakteru projektu, jeho velikosti, množství známých informací, tak i na zkušenostech z předešlých projektů. [3]

V následující sekci představím dvě nejpoužívanější strategie odhadu, a to přístup shora-dolů (top-down) a zdola-nahoru (bottom-up). U každé strategie jsou uvedeny její výhody a nevýhody a typy projektů, pro které je vhodná.

1.3.1 Přístup shora-dolů (top-down)

V případě přístupu shora-dolů se odhaduje pracnost projektu přímo jako celku, a to jako souhrn pracnosti veškerých činností potřebných k dodání produktu. Použitím této strategie lze získat celkový odhad založený na globálních vlastnostech odhadovaného softwaru. [3] Přístup shora-dolů je vhodný v případě rané fáze analýzy, kdy má zákazník pouze obecnou představu o tom, co má software dělat a zatím nelze přesně identifikovat jednotlivé součásti a pracovní činnosti, které budou pro realizaci softwaru potřeba.

Jedná se o přirozený a logický přístup, při kterém se postupuje od nejvyšší úrovně abstrakce k té nejnižší, kdy s přibývajícími informacemi je možné identifikovat čím dál tím menší části softwaru a tím postupně získat detailnější představu o odhadovaném softwaru. Tato strategie umožňuje mít v každém okamžiku analýzy reprezentativní odhad reflektující informace, které jsou v dané době dostupné. [4] Tedy i v době, kdy nelze odhad dekomponovat na jednotlivé pracovní činnosti.

1.3.1.1 Výhody přístupu shora-dolů

- Lze jej použít v rané fázi projektu, kdy je znám pouze počáteční koncept aplikace.
- Vytvoření odhadu není časově náročné.
- Poskytuje spolehlivý odhad v každé fázi analýzy, vzhledem k dostupným informacím.
- Je výhodný pro porovnávání s odhady minulých projektů.

1.3.1.2 Nevýhody přístupu shora-dolů

- Z důvodu vyšší úrovně abstrakce mohou být technické problémy, které jsou na detailní úrovni ihned viditelné, snadno přehlédnuty. [4]

- Je velká pravděpodobnost, že bude vynechána významná část práce na projektu a tím negativně ovlivněna přesnost odhadu. To je z důvodu, že u *pohledu shora* lze lehce zapomenout na některé pracovní činnosti, které pro vytvoření aplikace bude třeba udělat.

- Tato metoda není příliš vhodná pro menší projekty, které lze relativně dobře dekomponovat na jednotlivé aktivity. U menších projektů je výhodná pouze pro odhad na základě počátečního konceptu, kdy je známo jen velmi málo informací.

[3]

1.3.2 Přístup zdola-nahoru (bottom-up)

Oproti přístupu shora-dolů, přístup zdola-nahoru funguje přesně naopak. Neodhaduje se pracnost projektu jako celku nebo jeho částí, ale postupuje se odspodu tak, že se odhadují jednotlivé aktivity, které jsou pro realizaci projektu potřeba vykonat. Pro tento způsob odhadu je nutná detailní znalost pracovních činností, které jsou s vývojem aplikace spojeny, a také je zásadní mít již dobře definované zadání projektu, aby bylo možné jednotlivé pracovní aktivity správně identifikovat. Pokud se tyto aktivity povede určit správně a na žádné se nezapomene, lze od této strategie odhadu očekávat vysokou úroveň přesnosti. [3] [4]

V případě, že je již dostatečně specifikováno zadání aplikace, je tato strategie nejlepší volbou pro účely menších webových a mobilních aplikací [2]. Metodami odhadu, které na této strategii staví, se budu zabývat více v dalších sekcích. Z této strategie vychází metoda dekompozice odhadu (sekce 1.4) a na ní postavené expertní odhady (sekce 1.5) a dále metoda výpočtu očekávané pracnosti PERT (sekce 1.6).

1.3.2.1 Výhody přístupu zdola-nahoru

- Pokud jsou projektové aktivity správně identifikovány zajišťuje tato strategie vysokou míru přesnosti.

- Pracnost jednotlivých aktivit se lépe odhaduje, a to s menším rizikem nepřesnosti než u odhadování velkých částí softwaru.

- Proces dekompozice pomůže lépe pochopit zadání a identifikovat možné problémy.

1.3.2.2 Nevýhody přístupu zdola-nahoru

- Není vhodná pro ranou fázi analýzy, kdy není možné správně určit jednotlivé pracovní činnosti.

- Je časově náročnější než u přístupu *shora-dolů*, ale identifikované projektové aktivity mohou být použity i pro plánování projektu.

- Může se stát, že se do odhadu započítají některé činnosti několikrát, protože budou součástí různých aktivit.

[3]

1.4 Dekompozice odhadu

Dekompozice je metoda rozdělení odhadu pracnosti na mnoho malých částí, které se odhadují samostatně a následně proběhne zpětné poskládání těchto odhadů jednotlivých částí do souhrnného odhadu projektu. Tento přístup vychází z přístupu *zdola-nahoru* a jedná se o elementární metodu používanou v odhadech softwaru. Základní výhodou rozdělení projektu na menší části je, že jednotlivé části lze mnohem jednodušeji a přesněji odhadnout. Dekompozice odhadu také často rozkryje vlastnosti a úkoly, které by byly jinak opomenuty. Odhad vytvořený metodou dekompozice tedy bude zpravidla přesnější než celkový odhad. [2]

1.4.1 Zákon velkých čísel

Další vlastností, která napomáhá přesnosti metody dekompozice, je zákon velkých čísel. Jádro zákona spočívá v tom, že pokud se vytvoří jeden velký odhad, tak možná chyba v odhadu bude buď zcela na horní nebo zcela na dolní straně. Pokud je ale vytvořeno několik malých odhadů, tak některé chyby budou na horní straně a některé na dolní. Chyby se tak do jisté míry navzájem vyruší a výsledný odhad bude obsahovat chybu menší, než při celkovém odhadu.

Tento zákon by měl u odhadování fungovat nejen teoreticky, ale podle výzkumu (Lederer a Prasad 1992) funguje také v praxi. [2]

1.4.2 Použití dekompozice

Dekompozici projektu je možné pro účely odhadu použít pouze v situaci, kdy je o vyvíjené aplikaci známo dostatečné množství informací, to je nejlépe v pokročilé části analýzy, například při dokončené analýze požadavků nebo ještě lépe, když je vytvořen prototyp (wireframe) aplikace.

1.4.3 Dekompozice pomocí WBS

Jednou z možností jak provést dekompozici projektu na jednotlivé aktivity, je použít WBS (Work Breakdown Structure). Jedná se o jednoduchou analytickou techniku, jejímž cílem je identifikovat projektové aktivity. [5]

Rozložení na aktivity pomocí WBS může být podle složitosti a rozsahu projektu různě strukturované. V případě malého projektu bude v mnoha případech stačit dvoustupňová struktura, kde jsou jednotlivé aktivity zařazeny do kategorii nebo lze použít i pouze jednoduchý seznam aktivit. [5]

Hlavním požadavkem ale je identifikovat všechny tyto aktivity. To je také jedno z hlavních pravidel WBS, nazvané *pravidlo 100%*, které říká, že souhrn veškerých prací pro splnění aktivit v pracovním balíčku (kategorie) musí zahrnovat 100% práce, nutné k jeho dokončení. [3]

1.4.4 Vlastnosti aktivit

Jednou z možností jak zjistit, zda jednotlivé pracovní aktivity vniklé dekompozicí jsou validní a vhodné pro odhadování, je ověřit, zda splňují následující vlastnosti:

Definable (Definovatelnost) Aktivitu lze snadno a srozumitelně popsat a pro účastníky projektu je jednoduše pochopitelná.

Manageable (Kontrolovatelnost) Zodpovědnost za aktivitu může být přiřazena konkrétní osobě.

Estimable (Odhadnutelnost) Úsilí a čas potřebný k dokončení přidělené aktivity lze snadno a spolehlivě odhadnout.

Independent (Nezávislost) Každá aktivita je nezávislá (nebo jen minimálně) na ostatních aktivitách.

Measurable (Měřitelnost) Aktivita může být využitelná při měření postupu při realizaci projektu.

Adaptable (Přizpůsobitelnost) Aktivita je dobře přizpůsobitelná, lze u ní jednoduše manipulovat s rozsahem – přidávat a ubírat pracovní úkoly.

Accountable (Přiřaditelnost) Uskutečnění aktivity lze přiřadit jedné osobě.

Čerpáno z [3].

1.4.5 Pracnost aktivit

Pro odhadování pracnosti jednotlivých aktivit je třeba volit vhodné jednotky. Nejčastěji používané jednotky pro odhad pracnosti jsou MH a MD (člověkohodina a člověkoden). Tyto jednotky je třeba při odhadování aktivit správně zaokrouhlovat. Nesmí se například stát, že u aktivity odhadované na 2 hodiny, bude uvedeno 0,5 MD, to by odhad značně zkreslilo. [6]

1.4.6 Opomíjené aktivity

Jedna z nejčastějších chyb v odhadech je opomíjení nutných úkolů v projektu. Studie zjistila, že vývojáři mají tendenci odhadovat velmi přesně práci, na kterou si v odhadu vzpomenou, ale přehlíží 20% až 30% potřebných úkolů, což samozřejmě vede k chybě v odhadu. (van Genuchten 1991) [2]

V odhadu často nejsou zahrnuty aktivity, které se přímo netýkají vývoje a požadavků na aplikaci. Tedy například aktivity související s řízením projektu, komunikací se zákazníkem, vydáváním aplikace nebo požadavky mimo požadovanou funkcionalitu aplikace jako je například: bezpečnost, rozšiřovatelnost, spolehlivost, výkon atd. Je tedy důležité si uvědomit, co všechno bude k vytvoření aplikace potřeba a do odhadu to zahrnout. Jednou z dalších možností jak opomíjení aktivit předcházet, je použití standardizovaných metod dekompozice jako je například metoda WBS. [2]

Zde jsou uvedeny aktivity, které mohou být u menší webové a mobilní aplikace opomíjeny a je tedy dobré je do odhadu zahrnout:

- Analýza
  - Tvorba prototypu – wireframes
  - Sestavení specifikace požadavků
- Implementace
  - Návrh uživatelského rozhraní
  - Návrh databázového modelu
  - Neobvyklé požadavky na technologii
  - Rezerva na změny
- Testování
  - Tvorba automatizovaných testů
  - Distribuce testovací verze aplikace zákaznikovi
- Podpůrné činnosti
  - Řízení projektu
  - Schůzky se zákazníkem
  - Komunikace se zákazníkem
  - Komunikace v týmu
  - Tvorba projektové dokumentace
  - Tvorba uživatelské příručky
- Vydání
  - Nasazení aplikace na server
  - Vydaní mobilní aplikace

1.5 Expertní odhady

Expertní odhady jsou nejpoužívanější metodou pro tvorbu odhadů. Podle výzkumu jsou metody odhadu postavené na expertních odhadech používány u 72–86 % odhadů softwarových projektů. Základním principem je odhadování pracnosti jednotlivých projektových aktivit vzniklých dekompozicí projektu metodou zdola-nahoru. Pracnost projektových aktivit je pak určena na základě úsudku expertů. Tito experti by měli mít zkušenosti s vývojem obdobného softwaru a s technologiemi, které se na projektu budou používat. Velkou výhodou je také zkušenost s odhadováním a znalost základních metod a postupů teorie softwarových odhadů. [2]

1.5.1 Kdo tyto odhady vytváří?

U odhadů konkrétních pracovní aktivit – například implementace a testování určité funkcionality – vytvoří nejpřesnější odhad ti, kteří budou danou aktivitu nakonec sami provádět. Odhady vytvořené lidmi, kteří danou práci nebudou dělat, jsou přesné méně a je zde mnohem větší riziko, že odhadovanou pracnost podhodnotí. Pokud zatím nelze projekt dekomponovat na úroveň jednotlivých úkolu, například z nedostatku informací, měl by být odhad proveden nejlepším dostupným pracovníkem, který má s podobným typem projektu zkušenosti. Může se jednat o pracovníka z oboru vývoje nebo projektového manažera, který má s odhady větší zkušenost.

Přístup, kdy pracnost odhaduje přímo pracovník, který bude daný úkol vykonávat, je velmi vhodný u malých projektů, hodí se tedy na odhadování menších webových a mobilních aplikací. Pokud je odhad úkolu zadán pracovníkovi, který na něm později bude pracovat, máme zajištěnou dobrou znalost dané problematiky, a také motivaci dobře úkol odhadnout. [2]

1.5.2 Strukturovaný úsudek

Aby expertní odhady poskytovaly dostatečnou přesnost odhadu je vhodné, aby byly strukturované a nejednalo se pouze o neformální intuitivní úsudek, který má tendenci k nepřesnosti. [2]

1.5.2.1 Rozpad na menší celky

Jednou z nejlepších cest, jak přesnost expertních odhadů zlepšit, je správná dekompozice projektu na jednotlivé aktivity. Důležité je, aby jednotlivé aktivity nevyžadovaly déle jak jeden den práce. Projekt se rozdělí na menší úkoly, čímž se eliminuje prostor pro nečekanou, skrytou práci a zároveň velká konkrétnost úkolů výrazně ulehčí jejich odhad. [2] (více v sekci 1.4)

1.5.2.2 Použití intervalů

Použití pouze jedné hodnoty při odhadování aktivit je velmi omezenou možností. Určit dobu pracnosti zcela přesně je nereálné a tedy určovat při odhadu pouze jednu hodnotu může být velmi nevýhodné. Bodový odhad je příliš velké zjednodušení, protože nevyjadřuje přesnost a rizika odhadu. Pro některé aktivity může být takový odhad velmi přesný a případně se změní jen nepatrně, pro jiné se bude jednat spíše o nástřel, který se může odchýlit i několikanásobně.

Proto je výhodné používat intervalové odhady. Tedy pro každou aktivitu určit rozmezí, jakou dobu může trvat, což bude mít mnohem vetší vypovídající hodnotu o pracnosti dané aktivity. Například určit nejlepší a nejhorší scénář pracnosti rozšiřuje pohled na danou aktivitu a tím napomáhá hledat potencionální problémy, které mohou u aktivity nastat. [2]

„Přemýšlení o nejhorších variantách někdy vede k rozeznání dalších prací potřebných pro dokončení, což může navýšit nominální hodnotu.“ [2]

1.5.3 Přílišný optimismus

U expertních odhadů je potřeba si dát pozor na příliš optimistické odhady vývojářů. Ve studii 300 softwarových projektů se zjistilo, že odhady vývojářů jsou ovlivněny 20% až 30% faktorem optimismu [2].

Po dokončení odhadu je vhodné odhad na chvíli odložit stranou a dát ho poté vývojářům ještě jednou zrevidovat. Neboť po čase se na jednotlivé položky budou dívat kritičtěji a tím odhalí optimistické odhady a přepracují je. Další věcí, která může pomoci, je zpětná vazba vhledem k již realizovaným odhadům, tedy jak byl odhad daného vývojáře v minulosti přesný. [6]

1.6 Program Evaluation and Review Technique (PERT)

Technika nazývaná Metoda Vyhodnocování a Kontroly Programu (Program Evaluation and Review Technique, PERT) umožňuje vypočítat očekávané dokončení pracovní úlohy. Stutzke v roce 2005 navrhl použití této metody pro odhadování pracnosti softwaru. PERT je při odhadování softwaru použit pro výpočet celkové očekávané pracnosti, a to na základě agregace odhadů jednotlivých aktivit vytvořených expertními odhady. [3]

Tato technika je založena na metodě dekompozice a na expertních odhadech. Pro použití metody PERT je navíc třeba, aby každá aktivita byla odhadnuta pomocí tří hodnot: *nejlepší případ*, *nejhorší případ* a *nejpravděpodobnější případ*. Očekává se, že nejpravděpodobnější případ nebude přesně ve středu intervalu od nejlepšího a nejhoršího případu, ale bude objektivním úsudkem experta. [2]

1.6.1 Výpočet očekávané pracnosti

Z hlediska teorie pravděpodobnosti se předpokládá, že odhady jednotlivých aktivit jsou navzájem nezávislé. Na základě tohoto předpokladu je celková očekávaná pracnost projektu (střední hodnota) vypočtena jako součet očekávaných pracností jednotlivých aktivit. Celkový rozptyl tohoto PERT rozdělení je vypočítán jakou součet rozptylů jednotlivých aktivit a směrodatná odchylka pak jako druhá odmocnina součtu směrodatných odchylek aktivit. Směrodatná odchylka bude využita pro určení konfidenčního intervalu očekávané pracnosti. [3]

1.6.1.1 Značení ve vzorcích

$Expected$ = očekávaná pracnost
$Total$ = celková
Min = nejhorší případ
Max = nejlepší případ
ML = nejpravděpodobnější případ
σ = směrodatná odchylka
μ = střední hodnota

1.6.1.2 Výpočet očekávané pracnosti

[3]

$$Expected_{Total} = \sum_{i=1}^{n} Expected_i \tag{1.1}$$

kde

$$Expected_i = \frac{Min_i + 4 * ML_i + Max_i}{6} \tag{1.2}$$

1.6.1.3 Výpočet směrodatné odchylky

[3]

$$\sigma_{Total} = \sqrt{\sum_{i=1}^{n} \sigma_i^2} \tag{1.3}$$

kde

$$\sigma_i = \frac{Max_i - Min_i}{6} \tag{1.4}$$

Výpočet směrodatné odchylky podle tohoto vzorce se používá v případě, že se projekt skládá z 10 a více aktivit [2], což by i v případě menší mobilní a webové aplikace mělo být vždycky.

1.6.1.4 Modifikovaný dělitel pro výpočet standardní odchylky

U vzorce 1.4 pro výpočet standardní odchylky může nastat problém s tím, že se statisticky očekává, že do odhadnutého intervalu ohraničeného nejhorším a nejlepším případem, bude patřit 99,7% všech skutečných možností pracnosti. To, aby z 1000 odhadů byly pouze 3 mimo interval, je ale dost nepravděpodobné. [2]

V praxi se většinou setkáme s tím, že experti jsou schopni odhadnout interval pracnosti s přesností spíše okolo 70%, tedy místo hodnoty 6 by se měl dělitel pro výpočet standardní odchylky blížit hodnotě 2. Hodnoty dělitelů v závislosti na procentech jsou uvedeny v tabulce 1.1. [2]

Procento skutečných dokončení v intervalu odhadu	Hodnota dělitele
40 %	1
50 %	1,4
60 %	1,7
70 %	2,1
80 %	2,6
90 %	3,3
99,7 %	6

Tabulka 1.1: Modifikovaný dělitel pro výpočet standardní odchylky [2]

1.6.2 Konfidenční interval očekávané pracnosti

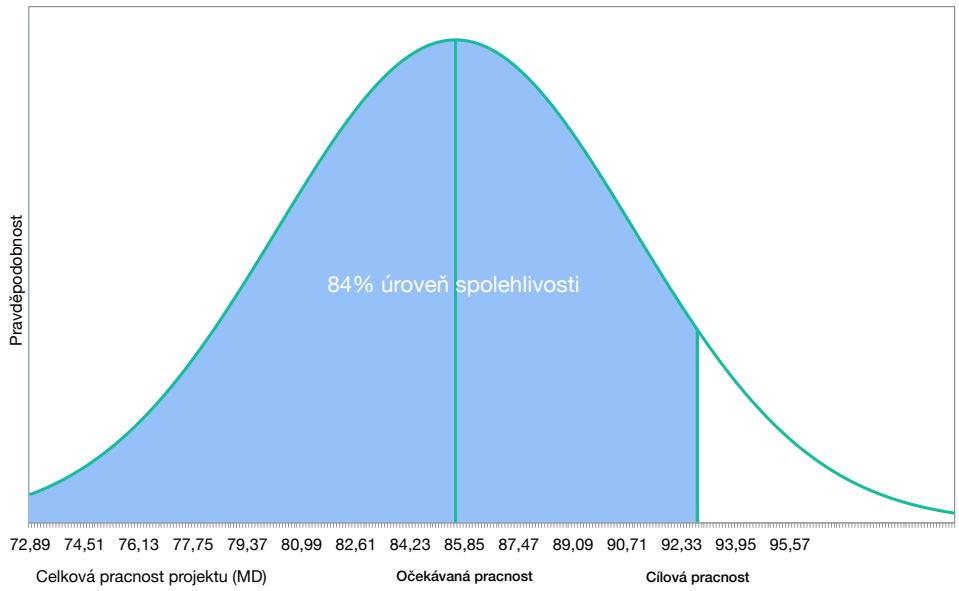
Pomocí metody PERT lze stanovit odhad pracnosti s pevně zvolenou mírou spolehlivosti. Je možné určit konfidenční interval, do kterého bude skutečná doba pracnosti s danou pravděpodobností spadat.

Pro tento účel je brána v úvahu očekávaná hodnota pracnosti (střední hodnota) a směrodatná odchylka. Při výpočtu konfidenčního intervalu se pak vychází z teorie pravděpodobnosti. Konkrétně se využívá centrální limitní věta, která říká, že součet mnoha nezávislých náhodných veličin konverguje s jejich zvyšujícím se počtem k normálnímu rozdělení. Pro účely celkového odhadu lze tedy usuzovat, bez dopuštění se podstatné chyby, že celkový odhad pracnosti je normálně rozdělen. [3]

Pro jednostranný konfidenční interval je na Obrázku 1.1 ukázána pravděpodobnost, že skutečná pracnost nepřekročí určitou cílovou pracnost.

Konfidenční interval pro očekávanou pracnost na základě normálního rozdělení a použití intervalových odhadů lze získat takto [3]:

$$Expected_{Total} \in (\mu - t * \sigma, \mu + t * \sigma) \tag{1.5}$$



Obrázek 1.1: Rozdělení pravděpodobnosti pro celkovou očekávanou pracnost se zvolenou mírou spolehlivosti (pro testovací projekt z prototypu výpočtu 2.4)

Kde t pro konfidenční interval s hladinou spolehlivosti p, $\alpha = 1 - p$ je kvantil $z_{\alpha/2}$ standardního normálního rozdělení $\mathcal{N}(1, 0)$. Tedy t lze vypočítat takto:

$$P(Z > z_\alpha) = \alpha \tag{1.6}$$

$$t = z_{\alpha/2} \tag{1.7}$$

Kapitola 2

Návrh vlastní aplikace pro podporu odhadů

Při návrhu vlastní aplikace pro podporu odhadů ceny zakázkového softwaru vycházím z poznatků získaných analýzou problematiky odhadů softwaru, provedenou v kapitole 1. Vytvářená aplikace se bude orientovat na odhady ceny menších mobilních a webových aplikací a její funkcionalita bude postavená na metodách a postupech odhadu, které jsou pro tento typ softwaru vhodné. Bude se jednat o specializovaný software, který umožní jednoduše vytvořit strukturovaný odhad, který bude poskytovat spolehlivý odhad ceny a pracnosti aplikace na základě odhadů jednotlivých projektových aktivit a následném výpočtu souhrnného odhadu.

2.1 Popis navrhované aplikace

V této sekci podrobně popisuji vlastnosti a součásti navrhované aplikace a způsob jejího fungování. Souhrnný seznam požadavků na aplikaci je uveden v sekci 2.2.

2.1.1 Webová aplikace

Navrhovaná aplikace bude implementována jako webová aplikace, dostupná z webového prohlížeče. Výhodou webové aplikace je možnost používat ji téměř na jakémkoliv operačním systému a na různých typech zařízení. Díky tomu nebude aplikace limitována pouze na desktop, pro který je primárně určena a bude možné ji používat i na zařízeních jako jsou tablety či chytré telefony. Také to umožní, aby na odhadování projektu mohlo jednoduše spolupracovat více osob, což je jeden z důležitých požadavků na funkčnost aplikace.

Dostupnost aplikace na webu eliminuje nutnost uživatele aplikaci jakkoliv instalovat, umožňuje úpravy a vylepšení aplikace za běhu, aniž by ji uživatel musel sám aktualizovat. Aplikace nebude výpočetně náročná, ani pro ní

nebude stěžejní rychlost odezvy, a proto se tento typ řešení jeví jako nejvýhodnější.

2.1.2 Základní funkcionalita

Hlavní funkcí navrhované aplikace bude možnost vytvářet strukturované expertní odhady softwaru. V aplikaci toho bude docíleno tak, že se každý odhadovaný projekt bude skládat z jednotlivých pracovních aktivit nutných pro jeho vytvoření.

Tyto aktivity budou určeny na základě dostupných informací o zadání projektu a do projektu je budou vkládat uživatelé (experti), kteří se na tvorbě odhadu projektu podílí. Při vkládání je budou členit do kategorii reprezentujících určitou část projektu, případně druh činnosti, aby se v nich bylo možné dobře orientovat a také odhad podle těchto kategorii strukturovat. Následně uživatelé pomocí tři různých hodnot určí pracnost jednotlivých aktivit. Aplikace pak na základě metody PERT vytvoří souhrnný odhad projektu a vypočte celkovou očekávanou pracnost a cenu.

2.1.3 Uživatel aplikace

Aby mohl uživatel aplikaci začít používat, bude nutné, aby se do aplikace nejdříve zaregistroval. Při registraci uživatel vyplní své celé jméno, přihlašovací email, heslo a v případě, že je uživatelem firma, uvede i název společnosti. Dále si uživatel bude moci zvolit profilový obrázek, podle kterého pak půjde uživatele například snadněji rozpoznat při vyhledávání.

Na základě registrace bude v aplikaci vytvořen uživatelský účet, na který se prostřednictvím přihlašovacího emailu a hesla uživatel přihlásí a teprve po přihlášení mu bude umožněno aplikaci používat – vytvářet a odhadovat vlastní projekty a spolupracovat na odhadování projektů ostatních uživatelů.

Všechny vytvořené odhady projektů budou pevně spjaté s uživatelským účtem a v případě, že se uživatel rozhodne účet zrušit, budou spolu s ním smazány. Kromě smazaní účtu, si uživatel svůj uživatelský profil bude moci kdykoliv zobrazit a libovolně ho editovat a to včetně změny přihlašovacího emailu a hesla. Uživatel si pro účely spolupráce s ostatními uživateli bude moci zobrazit i jejich uživatelské profily.

2.1.4 Správa projektů

Pro každý odhadovaný software si uživatel v aplikaci založí nový projekt. V aplikaci budou o projektu evidovány tyto informace:

- Jméno aplikace
- Krátký popis aplikace

- Specifikace zadání (zadání projektu) – formátovaný text specifikace s odkazy na další materiály jako je například wireframe aplikace
- Ikona aplikace – pro rychlejší orientaci mezi projekty
- Používaná jednotka pracnosti – MD/MH
- Výchozí sazba za hodinu práce

Ve svých projektech se uživatel bude orientovat pomocí jednoduchých filtrů. Bude si moci zobrazit: seznam nedávných projektů, na kterých naposledy pracoval, seznam aktivních projektů, jejichž odhad ještě není hotový a seznam úspěšných a neúspěšných projektů, podle toho, zda uživatel zakázku na odhadnutý projekt nakonec získal nebo nezískal.

Po otevření projektu bude moci uživatel začít projekt odhadovat. Postupně bude do projektu přidávat kategorie projektových aktivit a do nich jednotlivé aktivity, potřebné ke splnění dané kategorie, vyplňovat a následně i odhadovat. Takto bude uživatel při odhadu postupovat, dokud projektové aktivity nepokryjí všechny činnosti nutné ke splnění zadaní projektu. Další možností bude do projektu přidat odhadce – spolupracovníky, kteří budou na vyplňování kategorii a odhadování aktivit s uživatelem spolupracovat.

V průběhu odhadování bude možné kdykoliv zobrazit aktuální souhrnný odhad projektu s detailním rozpisem odhadu podle kategorii a aktivit. U projektu bude možné editovat zadané informace a upravovat specifikaci zadání, případně projekt i se všemi jeho kategoriemi a aktivitami smazat.

2.1.5 Aktivity

Každá aktivita bude mít svůj název, který by měl co nejlépe vystihovat předmět aktivity. Dále bude možné doplnit aktivitu ještě o popis, který bude detailněji popisovat jaké pracovní činnosti jsou její součástí. V neposlední řadě bude každá aktiva zařazena do jedné z předem vytvořených kategorií.

Pro odhadnutí aktivity je třeba stanovit tři různé hodnoty její pracnosti: nejlepší případ, nejhorší případ a nejpravděpodobnější případ (metoda PERT). Nejlepší případ určuje za jakou dobu, by šla aktivita splnit v ideálním případě, pokud by nenastaly žádné komplikace. Nejhorší případ by měl naopak počítat ze všemi možnými problémy, které by u realizace aktivity mohly nastat. Nejpravděpodobnější případ se určuje na základě zkušeností experta, jak dlouho obdobná aktivita většinou trvá. Doba pracnosti se bude uvádět v jednotce, která byla u odhadovaného projektu zvolena jako výchozí. Dokud nebudou všechny tři hodnoty vyplněny, nelze aktivitu považovat za odhadnutou a nebude tak zahrnována do souhrnného odhadu projektu.

2.1.6 Kategorie aktivit

Každá kategorie bude určená názvem a popisem. Popis kategorie by měl vycházet ze specifikace projektu a měl by kategorii specifikovat natolik, aby se podle něj dalo určit, jaké aktivity má obsahovat. Kategorie tedy bude reprezentovat typy projektových činností nebo funkční celky odhadované aplikace. Souhrn těchto kategorii by pak měl zahrnovat všechny činnosti nutné k realizaci projektu.

Pro každou kategorii bude možné nastavit hodinovou sazbu, která určuje hodinové náklady na realizaci aktivit, které jsou její součástí, podle toho aplikace vypočítá očekávaná cenu z očekávané pracnosti kategorie. Na základě sazeb všech projektových kategorii a jejich odhadnutých pracností bude poté aplikace určovat celkovou očekávanou cenu projektu.

Ke kategorii bude dále možné přiřadit jednoho z projektových odhadců, kterému tím bude umožněno kategorii odhadovat. Přiřazenému odhadci bude umožněno si danou kategorii včetně jejího popisu zobrazit a spravovat její aktivity – přidávat, odhadovat, upravovat a mazat.

2.1.7 Celkový odhad projektu

Přehled o souhrnném odhadu pracnosti a ceny projektu bude dostupný v celkovém odhadu projektu. Budou zde uvedeny všechny dostupné informace o odhadu projektu získané pomocí metody PERT z odhadů jednotlivých projektových aktivit. Celkový odhad se bude skládat z několika částí a bude v něm možné nastavovat parametry odhadu a manipulovat s tím, co do odhadu bude zahrnuto.

2.1.7.1 Odhadnutá pracnost a cena

Klíčovými informacemi, které zde budou uvedeny, bude celková odhadnutá pracnost a cena projektu. Kromě přímo určených hodnot zde budou vypočtena také rozmezí, do kterých by s určitou pravděpodobnosti měla skutečná pracnost a cena projektu spadat, tedy konfidenční intervaly pro očekávávanou cenu a pracnost projektu. Procento jistoty, kterým by měly skutečné hodnoty do intervalu spadat, bude možné v celkovém odhadu nastavit – čím vyšší procento jistoty bude zvoleno, tím větší rozsah intervalů bude a naopak.

U celkového odhadu bude možné nastavit parametr pokrytí odhadnutých aktivit. Tedy procento jakým skutečné pracnosti aktivit budou spadat do odhadnutých intervalů ohraničených nejlepším a nehorším případem. Tento parametr značně ovlivní celkové vypočtené rozmezí pracnosti a ceny projektu.

2.1.7.2 Rozpis kategorií a aktivit

Dále bude v celkovém odhadu k dispozici rozpis odhadů dílčích kategorií a bude možné zjistit, kolik která kategorie zabírá času z celkového odhadu prac-

nosti a jaká je její odhadovaná cena. Pro každou kategorii ještě bude uveden výpis aktivit, které do kategorie patří a jejich odhadnutá pracnost a cena. U kategorie bude dále dostupný i její popis a v případě, že je ke kategorii přiřazen projektový odhadce, bude u ní uvedeno i jeho jméno.

Rozpisy kategorii a aktivit budou také sloužit k manipulaci s obsahem celkového odhadu projektu. Bude možné z odhadu projektu zvolené kategorie a aktivity jednoduše vyřadit. Vyřazené položky pak nebudou započítávány do celkové pracnosti a ceny a ani nebudou zobrazovány v tisku celkového odhadu. Do odhadu je pak bude možné zase kdykoliv vrátit. Tento selektivní výběr se bude hodit zejména k vyjednávání se zákazníkem o ceně a rozsahu projektu.

Celkový odhad bude možné jednoduše vytisknout a výstup z tisku pak použít jako podklad pro tvorbu cenové nabídky nebo kalkulace.

2.1.8 Spolupráce na odhadech

Každý uživatel bude moci kromě odhadování svých projektů spolupracovat i na odhadech projektů ostatních uživatelů. Uživatel, který bude na odhadování projektu jiného uživatele přiřazen, najde tento projekt v seznamu odhadovaných projektů, který je od vlastních projektů uživatele oddělen. Uživatel si u takového projekt bude moci zobrazit specifikaci zadání a kategorie aktivit, na které byl vlastníkem projektu přiřazen.

Obsah těchto přiřazených kategorii bude moci libovolně spravovat, stejně jako u svých projektů. Vyplněním kategorií aktivitami, podle jejich popisu a specifikace zadání projektu, a po jejich následném odhadnutí se uživatel stane expertním odhadcem projektu. Další možností uskutečnění expertního odhadu bude, že vlastník projektu předvyplní aktivity do kategorie, ale neodhadne je a spolupracující uživatel pak pracnosti těchto aktivit odhadne.

Jinak, než přidáváním a odhadováním aktivit do přidělených kategorií, nebude moci uživatel do odhadovaného projektu zasahovat. Nebude ho moci jakkoliv editovat a ani si nebude moci zobrazit výsledný odhad projektu.

2.1.9 Notifikace

Notifikace budou uživatele informovat o tom, že byl přidán do projektu jiného uživatele jako odhadce, nebo že mu byla přidělena nová kategorie na odhadovaní. Rozkliknutím notifikace bude možné rovnou zobrazit přidělený projektu, případně kategorii. Stejně tak budou notifikace uživatele informovat o tom, že byl z projektu či kategorie odebrán.

2.2 Požadavky na aplikaci

Zde uvádím souhrnný seznam požadavků na navrhovanou aplikaci, který navazuje na popis aplikace v sekci 2.1. Shrnuji zde všechny důležité vlastnosti

a funkce, jaké bude navrhovaná aplikace podporovat. Požadavky kladené na aplikaci jsou standardně rozděleny na funkční a nefunkční požadavky.

2.2.1 Funkční požadavky

Uživatelé

FP1 Aplikace bude podporovat registraci a přihlášení uživatele.

FP2 Aplikace bude evidovat informace o uživatelích.

FP3 Aplikace umožní uživatelům zobrazovat a upravovat uživatelský profil.

Projekty

FP4 Aplikace bude evidovat odhadované projekty.

FP5 Aplikace uživateli umožní vytvářet, zobrazovat, editovat a mazat projekty.

FP6 Aplikace umožní uživateli zobrazit seznam vlastních a odhadovaných projektů a tyto seznamy filtrovat.

FP7 Aplikace umožní zadávat a upravovat specifikaci zadání projektu pomocí WYSIWYG editoru.

FP8 Aplikace bude umožňovat do projektu přidávat a odebírat uživatele – odhadce.

FP9 Aplikace umožní přidávání odhadců pomocí dynamického vyhledávacího pole.

Kategoriie aktivit

FP10 V projektu bude možné přidávat, upravovat a mazat kategorie aktivit.

FP11 U kategorie bude možné nastavit hodinovou sazbu pro její aktivity.

FP12 Ke kategorii půjde přiřadit odhadce projektu.

Aktivity

FP13 V projektu bude možné přidávat, upravovat a mazat aktivity.

FP14 Aplikace bude podporovat odhadování aktivit pomocí tří hodnot – metoda PERT.

FP15 Aktivity budou řazeny do kategorii.

Celkový odhad projektu

FP16 Aplikace vypočítá celkový odhad ceny a pracnosti projektu pomocí metody PERT.

FP17 Aplikace pro odhadnutou cenu a pracnost určí konfidenční interval.

FP18 Aplikace zobrazí rozpis kategorii a aktivit a jejich odhadnutou pracnost a cenu.

FP19 Aplikace bude podporovat selektivní výběr kategorii a aktivit do celkového odhadu projektu.

FP20 Aplikace umožní měnit procento spolehlivosti konfidenčního intervalu.

FP21 Aplikace bude umožňovat nastavení procenta pokrytí odhadu aktivit.

Spolupráce na odhadech

FP22 Aplikace bude umožňovat spolupráci více uživatelů na odhadu projektu.

FP23 Aplikace umožní uživateli odhadovat kategorii, ke které byl přidělen – přidávat, upravovat, mazat a odhadovat aktivity v této kategorii.

FP24 Aplikace umožní odhadci projektu zobrazovat projekt a jeho součásti jen v omezené míře – pouze jeho specifikaci, přiřazené kategorie a jejich aktivity.

FP25 Aplikace bude uživatele notifikacemi informovat o přiřazení nebo odebraní z odhadovaného projektu, případně z přiřazené kategorie.

2.2.2 Nefunkční požadavky

NP1 Webová aplikace postavená na třívrstvé architektuře.

NP2 Aplikace bude podporovat autentifikaci a autorizaci uživatele.

NP3 Aplikace na základě autentifikace uživatele umožní zobrazovat a spravovat pouze jemu přístupný obsah.

NP4 Aplikace bude podporovat rezponzivní design.

NP5 Aplikace bude umožňovat pohodlné ovládání na desktopu a tabletu.

NP6 Aplikace bude implementovaná pomocí frameworku Ruby on Rails a technologii JavaScript, HTML a CSS.

NP7 Aplikace bude používat k ukládání databázi PostgreSQL.

Důvod výběru požadovaných technologii je upřesněn v kapitole 3.

2.3 Případy užití

V této sekci uvádím diagramy případů užití. Případy užití slouží k zachycení základních interakcí uživatelů s navrhovanou aplikací. Pro lepší pochopení uvádím kromě obecného uživatele i případy užití uživatelských rolí a jejich interakcí vzhledem k vlastnictví či nevlastnictví projektu.

2.3.1 Aktéři

Uživatel Obecné interakce každého uživatele, který do aplikace přistupuje. (Obrázek 2.1)

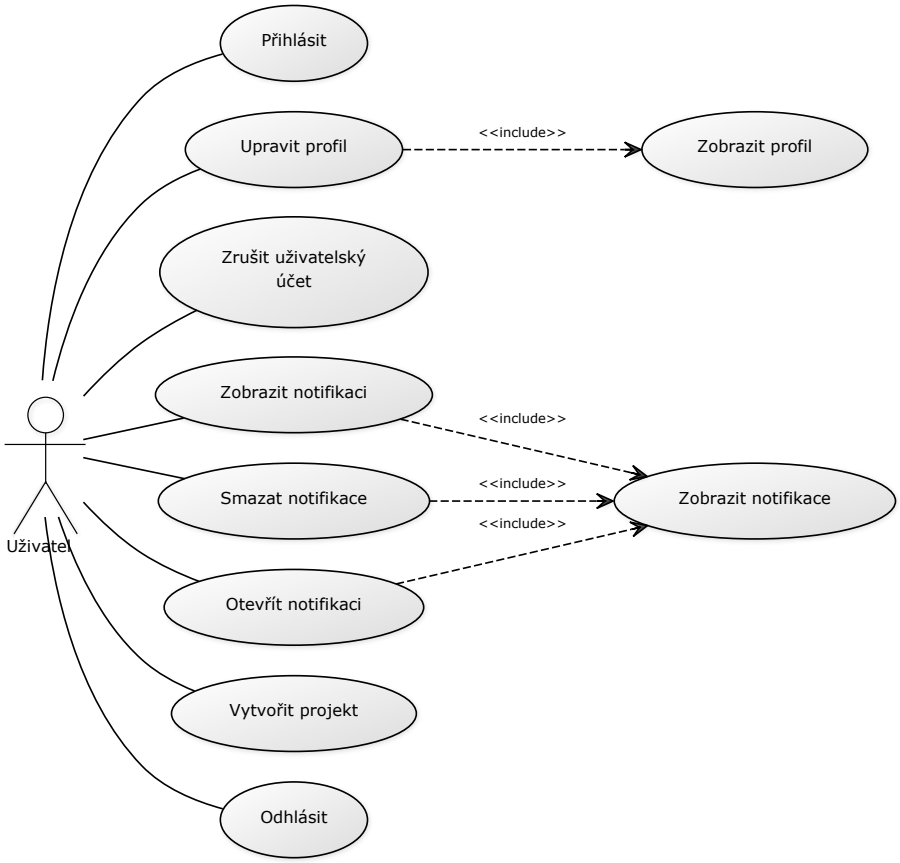
Vlastník projektu Uživatelská role, která ukazuje jaké interakce uživatel aplikace provádí se svým vlastním projektem. (Obrázek 2.2)

Odhadce projektu Role uživatele při spolupráci na odhadovaní projektu jiného uživatele. (Obrázek 2.2)

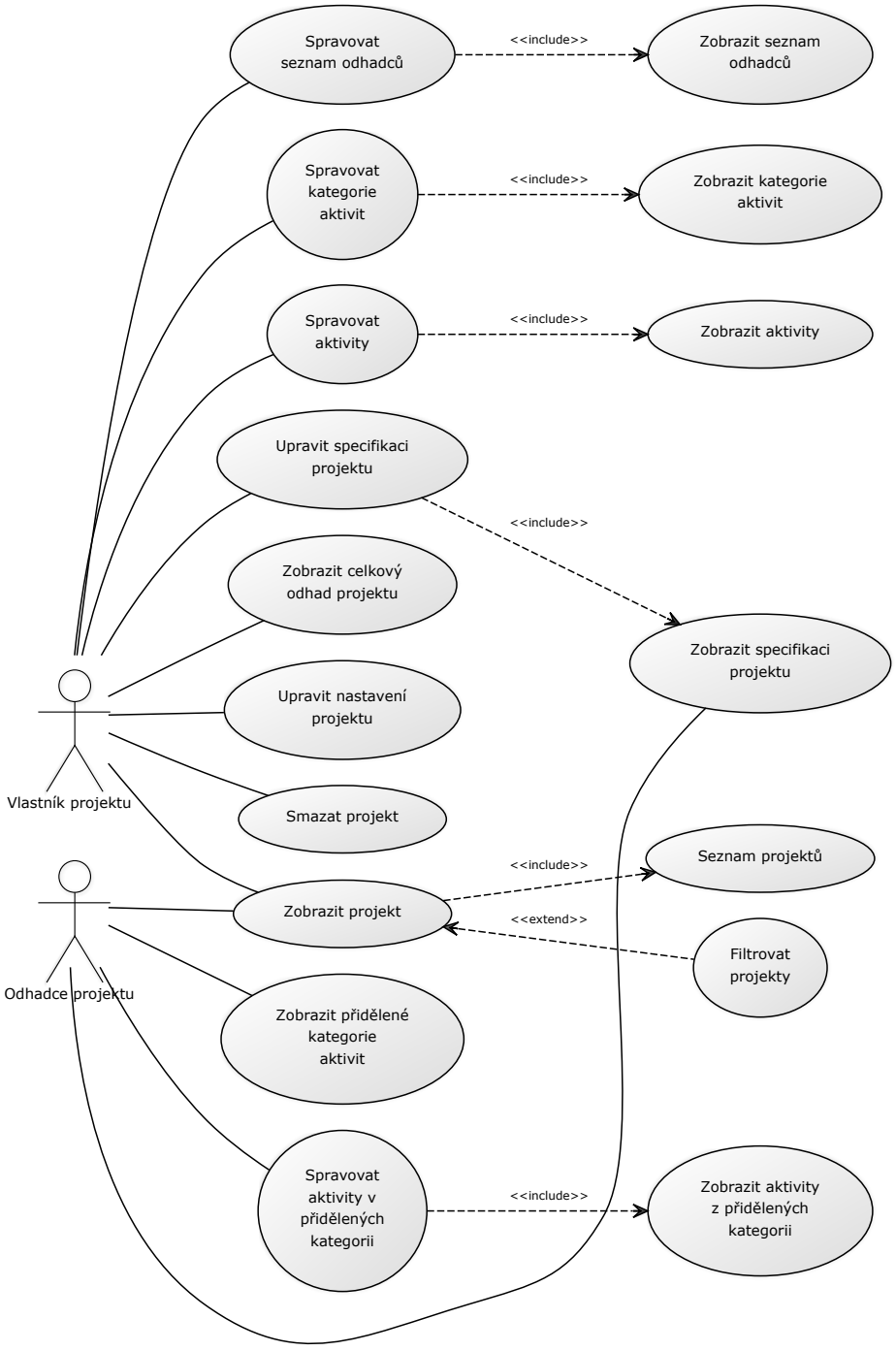
2.4 Prototyp výpočtu ceny

Poslední částí návrhu aplikace je vytvoření funkčního prototypu výpočtu celkové ceny a pracnosti projektu pomocí metody PERT. Tento prototyp bude sloužit jako referenční řešení, podle kterého budu výpočet v aplikaci implementovat. Prototyp výpočtu jsem vytvořil v tabulkovém procesoru MS Excel a je dostupný v příloze práce. Výstup z prototypu je uveden na obrázku 3.1.

Odhad testovacího projektu, který je v prototypu vytvořen, se stejně jako v navrhované aplikaci skládá z kategorií s nastavenou hodinovou sazbou, které jsou vyplněny aktivitami. Jednotlivé aktivity jsou pak odhadnuty třemi hodnotami pracnosti. U jednotlivých aktivit a kategorií prototyp vypočítává očekávanou cenu, pracnost a směrodatnou odchylku. Nakonec prototyp vypočte souhrnnou očekávanou cenu a pracnost projektu a to včetně jejich konfidenčních intervalů s nastavitelným procentem jistoty.



Obrázek 2.1: Diagram případu užití aplikace pro obecného uživatele



Obrázek 2.2: Diagram případu užití správy projektu pro vlastníka a odhadce projektu

Implementace					Hodinová sazba:	500,00 Kč
Název aktivity	**Nejlepší případ (MH)**	**Nejpravděpodobnější případ (MH)**	**Nejhorší případ (MH)**	**Směrodatná odchylka**	**Očekávaná pracnost (MH)**	**Očekávaná cena**
Dashboard	5,00	6,50	9,00	0,67	6,67	3 333 Kč
Přihlášení do aplikace	2,00	6,00	10,00	1,33	6,00	3 000 Kč
Implementace statistik	8,00	13,00	20,00	2,00	13,33	6 667 Kč
Sledování pohybu	15,00	20,00	24,00	1,50	19,83	9 917 Kč
Celkově	**30,00**	**45,50**	**63,00**	**2,91**	**45,83**	**22 917 Kč**

Testování					Hodinová sazba:	250,00 Kč
Název aktivity	**Nejlepší případ (MH)**	**Nejpravděpodobnější případ (MH)**	**Nejhorší případ (MH)**	**Směrodatná odchylka**	**Očekávaná pracnost (MH)**	**Očekávaná cena**
Testování hlavních funkcí	9,00	18,00	33,00	4,00	19,00	4 750 Kč
Testování mezních případů	6,00	8,00	10,00	0,67	8,00	2 000 Kč
Testování zátěže	1,50	2,00	4,00	0,42	2,25	563 Kč
Testy uživatelského rohraní	6,00	10,00	17,00	1,83	10,50	2 625 Kč
Celkově	**22,50**	**38,00**	**64,00**	**4,47**	**39,75**	**9 938 Kč**

Projekt celkem	
Spolehlivost odhadu	98%
Odhadnutá pracnost	73,17 MH – 97,99 MH
Odhadnutá cena	26868,93 Kč – 38839,41 Kč
Směrodatná odhylka	5,33
Očekávaná pracnost (MH)	85,58
Očekávaná cena	32 854 Kč

Obrázek 2.3: Výstup z prototypu výpočtu souhrnné pracnosti a ceny projektu metodou PERT, vytvořeného v tabulkovém procesoru MS Excel

Kapitola 3

Použité technologie

V této kapitole představím technologie a nástroje, které jsem při realizaci vlastní webové aplikace pro podporu softwarových odhadů použil. Uvádím zde, co mě k výběru daných technologii vedlo a v čem jsou tyto technologie vhodné právě pro moje řešení.

3.1 Ruby on rails

Pro implementaci samotné aplikace jsem si vybral webový framework Ruby on Rails, který je nejpoužívanějším webovým frameworkem postaveným na programovacím jazyce Ruby [7]. V následujících sekcích představím jak samotné Ruby on Rails, tak i použitý jazyk Ruby a další technologie, které jsou s Ruby on Rails spjaté a v implementaci aplikace jsem je použil. Z popisu by také mělo vyplynout, proč jsem si právě tyto technologie vybral.

3.1.1 Ruby

Ruby je moderní interpretovaný všestranně použitelný programovací jazyk, ve kterém se vyvíjí webové, desktopové i mobilní aplikace a vytvářejí skripty. Vytvořil ho v roce 1995 Yukihiro „Matz“ Matsumota, jehož cílem bylo vyvinout programovací jazyk, který bude nejen snadno použitelný, ale bude v něm i zábavné programovat a programátorům umožní soustředit se na kreativní stránku programování a nepřidělává jim zbytečné starosti [8].

Ruby je ryze objektově orientovaný, dynamicky typovaný programovací jazyk, který vychází z jazyků jako je Perl, Python nebo Smalltalk [9]. Jeho syntaxe je velmi přehledná a dobře naučitelná a umožňuje psát úsporný, lehce pochopitelný kód. Ruby je velmi produktivní jazyk, který umí skvěle pracovat s řetězci a kolekcemi a jde v něm jednoduše používat regulární výrazy nebo closures.

Přestože mám předchozí zkušenosti s vývojem webových aplikaci v jazyku PHP a v jeho frameworku Symfony, který by bylo možné pro vytvoření apli-

kace použít, rozhodl jsem se pro jazyk Ruby. Zde uvádím hned několik důvodů proč Ruby oproti PHP upřednostňuji.

Prvním důvodem výběru jazyka Ruby je jeho podpora čistě objektového programování, se kterým jsem se na úrovni jakou Ruby umožňuje, kromě jazyka Smalltalk, ještě nesetkal. Jako vývojář mobilních aplikací pro platformu iOS jsme sice zvyklý programovat v jazyku Objectiv-C, který stejně jako Ruby vychází ze Smalltalku a využívá některé sofistikovanější objektově orientované techniky, ale v Ruby jde OOP ještě o něco dál a má mnohem přehlednější a jednodušší syntaxi.

V Ruby je například všechno objekt, a to včetně primitivních datových typů, což je ukázáno na následujícím kódu:

```
5.times { print 7 + 11 }
# => 1818181818
```

Na objekt `5` třídy `Fixnum` je v příkazu volána metoda `times` s argumentem bloku kódu, který je rovněž objekt. V bloku je pak na objekt `7` volána metoda `+` s argumentem `11`, ekvivalentně by šla metoda sčítaní zapsat i takto:

```
7.+(11)
```

V Ruby může každý objekt přijímat zprávy, volání metody na objektu je vlastně poslání zprávy se jménem metody a volitelnými parametry. Metoda je jako všechno v Ruby také objekt a je možné ji uložit do proměnné.

Demonstrace výše zmíněných vlastností jazyka Ruby je ukázána na následujících třech ekvivalentních zápisech zaslání zprávy `:abs` objektu `-0.5` třídy `Fixnum`.

```
value = -0.5
print value.abs
# => 0.5

value = -0.5
print value.send(:abs)
# => 0.5

value = -0.5
method_object = value.method(:abs)
print method_object.call
# => 0.5
```

Druhým důvodem výběru jazyka Ruby je expresivita jeho syntaxe, která umožňuje snadno a rychle vyjádřit myšlenku a výsledný kód, který vznikne, je přehledný a dobře čitelný. Oproti jazyku PHP je čitelnost kódu opravdu výrazně lepší, kód v Ruby může mít až „literární povahu“, často může vypadat jako jednoduchá anglická věta:

```
print self.message unless self.message.blank?
```

Předchozí příkaz odpovídá větě v přirozeném jazyce: „Vypiš zprávu pokud není prázdná“. V následující ukázce je zase demonstrována expresivita Ruby. I bez znalosti jazyka lze jednoduše poznat, co bude výsledkem příkazu.

```
print ['cat', 'dog', 'bird'].sort.last.capitalize
# => Dog
```

Ale opravdu tím hlavním důvodem výběru jazyka Ruby je právě existence výborného webového frameworku Ruby on Rails, který spolu s vymoženostmi a funkcionalitami jazyka Ruby a s jeho celkovou filosofií a jednoduchostí zaručuje, že je radost v Ruby respektive v Ruby on Rails webové aplikace vyvíjet.

3.1.2 Rails

Ruby on Rails, někdy také pouze Rails, je framework pro vývoj webových aplikací napsaný v jazyce Ruby. Vytvořil jej v roce 2003 dánský programátor David Heinemeier Hansson při práci na projektu Basecamp[1] a v roce 2004 jeho zdrojové kódy uvolnil pod open-source licencí [10].

Ruby on Rails je navrženo tak, aby usnadňovalo vývoj webových aplikací a jeho používaní bylo co nejednodušší a nejproduktivnější. Rails toho dosahuje tím, že vychází z obecných předpokladů o tom, co každý vývojář pro implementaci webové aplikace potřebuje. To umožňuje psát méně kódu a zároveň dosáhnout více než v mnoha jiných jazycích a platformách. Stejně jako jazyk Ruby i Rails se řídí filosofii, že programování v něm by mělo být nejen efektivní a jednoduché, ale programátoru by mělo dělat i radost. [11]

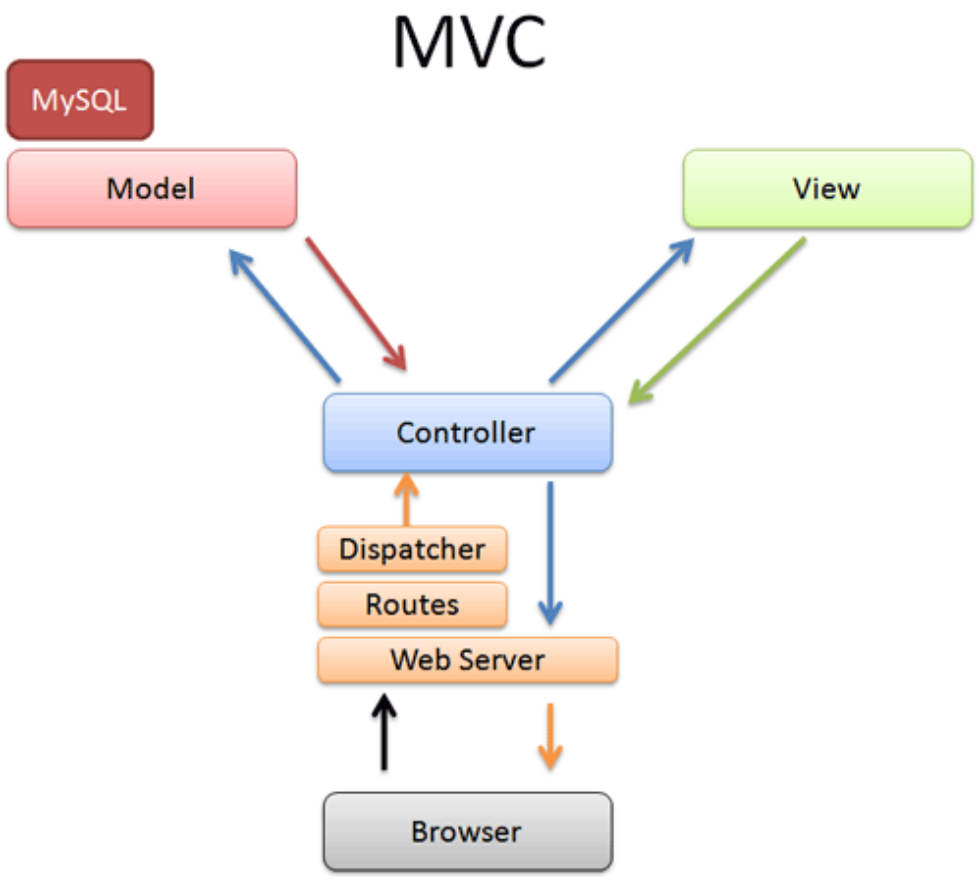
Ruby on Rails obsahuje několik principů a vzorů, které jsou pro tento framework klíčové, proto je blíže přiblížím v následujících odstavcích.

3.1.3 Model View Controller

Jako většina moderních webových frameworků i Ruby on Rails staví na osvědčené architektuře Model, View, Controller, obvykle označovanou jako MVC. MVC primárně slouží k oddělení datového modelu a aplikační logiky od uživatelského rozhraní a řídící logiky. Díky architektuře MVC má aplikace přehlednou struktura kódu, která usnadňuje údržbu, a také podporuje znovupoužitelnost kódu. [11]

Model Model reprezentuje data v aplikaci a pravidla pro práci s nimi a měl by zahrnovat veškerou aplikační logiku. V případě Rails je primární funkcí modelů interakce s příslušnou tabulkou v databázi. Ve většině případů by měla jedna tabulka v databázi odpovídat jednomu modelu v aplikaci.

[1] Webová aplikace pro řízení, organizaci a správu projektů. `http://basecamp.com`



Obrázek 3.1: Ruby on Rails – Architektura MVC (převzato z [1])

View Pohledy reprezentují uživatelské rozhraní aplikace a je to jediná vrstva, která je skrze webový prohlížeč nebo jiný nástroj aplikace pro uživatele přístupná. V Rails jsou pohledy obvykle HTML šablony s vloženými částmi Ruby kódu, které vykonávají pouze úkony spojené s prezentací dat.

Controller Controllery jsou vrstva, která propojuje datový model a views. Slouží ke zpracování požadavků, které přicházejí z webového prohlížeče od uživatele, na jejichž základě pak controller získává data z modelů, která pak odesílá do views, kde jsou zobrazeny. Controllery by měli obsahovat co nejméně aplikační logiky a tedy většina funkčního kódu aplikace by měla být v modelu.

3.1.4 Convention over configuration

Convention over configuration – konvence má přednost před konfigurací je jeden z hlavních principů fungovaní frameworku Ruby on Rails. Konvence Railsů předpokládá, co chce programátor udělat a jak to chcete udělat a místo toho, aby byl programátor nucen specifikovat každou drobnost v nekonečném množství konfiguračních souborů, stačí, aby se držel těchto konvencí a zabýval se pouze neobvyklými aspekty aplikace, které nejsou v konvenci zahrnuty. [12] To v praxi například znamená, že pokud se vytváří třída modelu s názvem `Project`, tak se nemusí pro ni nastavovat v databázi název tabulky, protože bude automaticky pojmenována `projects`. Nebo například není potřeba pro každou action metodu v controlleru specifikovat view, který se má zobrazit, ale stačí ho pojmenovat stejně jako metodu.

3.1.5 Don't Repeat Yourself (DRY)

„Don't repeat yourself" aneb „neopakujte se" je dalším důležitým principem, který Ruby on Rails prosazuje. Základní myšlenkou je psát znovupoužitelný kód a vyvarovat se opakování stejného kódu na mnoha místech aplikace. Snížení duplicit vede k čistému a dobře udržovatelnému kódu, který obsahuje méně chyb. [12] Rails tomuto principu napomáhá nejen svou architekturou, ale i různými funkcemi, které poskytuje. Pro příklad, častou věcí, kterou je potřeba v controlleru aplikace dělat, je u každé action metody ověřit, zda je uživatel, který k ní přistupuje přihlášený. Toho lze jednoduše docílit vestavěnou metodou `before_action` jejíž parametrem je jméno metody, která se má před každou akcí controlleru zavolat.

3.1.6 Active Record

Jednou z důležitých vlastností Railsů je podpora objektově-relačního mapování databáze, které umožňuje pracovat s databází objektově. Tato vlastnost je v Railsu docílena pomocí návrhového vzoru Active Record.

Active Record je základem všech modelů aplikace (modely od něj dědí) a jeho použití je nezávislé na jazyku SQL, či na konkrétním typu databáze. Active Record poskytuje jak základní operace pro čtení, vytváření, editaci a mazání záznamů (tzv. CRUD, Create Read Update Delete), tak i pokročilé metody pro získávaní dat z databáze. Práce s ním je jednoduchá a intuitivní. V neposlední řadě jde pomocí Active Record definovat vztahy mezi jednotlivými modely a také model pomocí mnoha vestavěných, případně vlastních metod validovat. [13]

3.1.7 Rapid development

Kromě výše zmíněných vlastností a principů, které mě k výběru Ruby on Rails vedly, existuje ještě další důvod, proč jsem se rozhodl právě pro framework

Ruby on Rails. Tím důvodem je možnost rapidního vývoje a postupného agilního vývoje aplikací v Ruby on Rails.

Při vývoji webové aplikace v Ruby on Rails je možné postupovat agilním způsobem. Místo toho, aby se na začátku nadefinoval celý model aplikace, včetně atributů všech entit a vztahů mezi nimi, vytvoří se třeba jen jeden model s několika málo atributy. K němu se pak postupně dodávají další atributy a do aplikace se následně přidají další modely a definují se vztahy mezi nimi. Tento postup je v Railsu umožněn díky možnosti vytvořit migraci databáze, podle které Active Record aktualizuje modelové schéma.

Framework Ruby on Rails je charakteristický rapidním vývojem aplikací. Ten podporují základní principy Railsů jako je Convention over configuration, samotný jazyk Ruby nebo pricip DRY, ale tou hlavní vlastností Railsu, která je právě pro rapidní vývoj klíčová, je možnost používat generátory zdrojového kódu. Pomocí generátoru zdrojových kódu lze velmi rychle vytvořit základní kostru aplikace. Lze jednoduše generovat soubory s definicemi tříd modelu, controllery, view nebo třeba jednotkové testy a spoustu dalších věcí. [12]

Následující příkaz vygeneruje třídu modelu Project a zároveň vytvoří migraci databáze, která do databáze přidá tabulku projects s atributy name a specifikacation, nakonec tento příkaz vygeneruje i jednotkové testy modelu.

```
$ rails generate model Project name:string specification:text
```

3.1.7.1 Scaffolding

Kromě možnosti generovat zvlášť modely, controllery a view, existuje v Rails ještě mocnější nástroj pro generování zdrojového kódu, takzvaný scaffolding (v překladu lešení). Ten umožňuje vygenerovat vše najednou a jeho výstupem je univerzální kód, na jehož základě je možné vytvořit celou funkční kostru aplikace.

Následující příkaz vygeneruje vše potřebné pro přidaní entity `Project` do aplikace. Vygeneruje model, migrace, controller, routy, pohledy, jednotkové testy a další pomocné soubory. Vygenerovaný kód umožní v aplikaci přidávat, zobrazovat, upravovat a mazat Projekty.

```
$ rails generate scaffold Project name:string specification:text
```

Přestože vygenerovaný kód bude ve většině případů nutné podle vlastních potřeb upravit, je dobrým základem, na kterém je možné vlastní aplikaci postavit a podstatně tím urychlit její vývoj.

3.1.8 Šablonovací systém

Pro pohledy ve formátu HTML se v Rails v základu používá šablonovací systém ERB, který umožňuje rozšiřovat HTML o Ruby kód. Rails navíc nabízí i spoustu pomocných metod, které lze v šablonách používat ke generovaní

HTML. V následující ukázce je vidět použití podmínky `if` a pomocné metody ke generování odkazu v šablonovací systém ERB.

```
<% if can? :edit, @project %>
  <%= link_to edit_project_path(@project) %>
<% end %>
```

Rails umožňuje do jednotlivých view vnořovat takzvané parciální šablony, které dávají možnost view efektivně rozdělit na jednotlivé součásti – partials, které mohou být použity na více místech (princip DRY). Je možné do nich předávat libovolné parametry a podle toho upravovat jejich chování. Partials mohou být použity ke generování tabulek, formulářů, nebo třeba navigačních prvků.

3.1.8.1 Haml

Kromě základního ERB existují pro Rails i alternativní šablonovací systémy. Já jsem se rozhodl ve své aplikaci použít systém Haml (HTML abstraction markup language). Přestože ERP je poměrně dobře použitelné a zpočátku implementace aplikace jsem ho využíval, přešel jsem nakonec na Haml kvůli jeho přehlednosti a čitelnosti. Haml také ušetří spoustu kódu, protože není potřeba uzavírat HTML tagy, podmínky ani bloky, to je řešeno pomocí odsazování kódu. Porovnání stejného kódu v ERB a Haml:

```
<%# ERB %>
<section class="container">
  <h1><%= post.title %></h1>
  <h2><%= post.subtitle %></h2>
  <div class="content">
    <%= post.content %>
  </div>
</section>

# Haml
%section.container
  %h1= post.title
  %h2= post.subtitle
  .content
    = post.content
```

[14]

3.1.9 Další vlastnosi

Framework Ruby on Rails disponuje spoustou dalších vlastností, které jsou typické pro mnoho webových frameworků a ještě jsem je nezmínil. Napří-

klad převod požadavků na vnitřní řídící prvky aplikace takzvaným routováním nebo možnost použití různých typů databází, AJAX, validací a generování formulářů. [12]

Poslední věcí, která stojí za zmínku je možnost využívat balíčkovací systém jazyka Ruby (RubyGems) a pomocí něho do Railsů přidávat různé volně dostupné knihovny. Ty je možné použit k řešení běžných požadavků na aplikaci jako je autentizace, autorizace, a to bez nutnosti vlastní implementace. Existuje nepřeberné množství těchto knihoven a jejich používaní značně urychlí vývoj.

3.2 PostgreSQL

Ruby on Rails umožňuje použití velkého množství různých databází: klasické relační databáze (SQLite3, MySQL), takzvané NoSQL databáze jako třeba grafové (Neo4J) nebo dokumentové databáze (MongoDB).

Já jsem se ve své aplikaci rozhodl použít osvědčenou objektově-relační SQL databázi PostgreSQL. Jedná se o kvalitní výkonnou a bezpečnou databázi, kterou mnoho vývojářů aplikací v Ruby on Rails preferuje. PostgreSQL je kompletně open-source, má velkou škálu funkcí a speciálních datových typů (například pole) a je pravidelně rozšiřován a vylepšován. [15]

3.3 Twitter Bootstrap

Twitter Bootstrap je front-end framework pro vytváření webového uživatelského rozhraní. Umožňuje rychle a jednoduše vytvořit dobře vypadající webové rozhraní, které se přizpůsobí různým typům zařízení. Framework je postavený na technologiích jako je CSS, Javascript (konkrétně jQuery) a HTML. [16]

Bootstrap byl původně vyvinut firmou Twitter jako framework pro podporu konzistence interních nástrojů. V roce 2011 byl uvolněn jako open-source projekt na serveru GitHubu. Bootstrap má velkou základnu vývojářů, kteří se podílejí na vylepšovaní frameworku, což zajišťuje jeho aktuálnost. [17]

3.3.1 Vlastnosti

Bootstrap nabízí velké množství předpřipravených funkcionalit a prvků uživatelského rozhraní, které je možné rovnou použit. Tyto prvky jsou kompletně nastavené a graficky zpracované elementy, patří mezi ně například: tlačítka, navigační menu, tabbary, oznámení, panely, ukazatele průběhu, modální okna a mnoho dalších. Bootstrap je také použit ke stylování textu, tabulek nebo formulářů. Součástí Bootstrapu jsou i tzv. Glyphicons – grafické ikony, které mohou být použity ke zpřehlednění uživatelského rozhraní.

Důležitou vlastností Bootstrapu je jeho responzivnost, díky které lze webové rozhraní vytvořené v tomto frameworku pohodlně používat na různých typech

zařízení, tedy kromě desktopu i na zařízeních jako jsou tablety a mobilní telefony. Boostrap používá k rozložení jednotlivých prvků na obrazovce tzv. Grid (mřížka) systém, který rozděluje elementy uživatelského rozhraní do řádků a sloupců. Podle této mřížky se pak rozhraní na různých zařízeních přizpůsobuje. [17]

Následuje jednoduchá ukázka použití některých prvků frameworku Twitter Bootstrap.

```
<div class="panel-body">
  <div class="row">
    <div class="col-md-10">
      <h3>Project Specification</h3>
    </div>
    <div class="col-md-2">
      <a class="btn btn-primary btn-sm" href="/projects/31/edit">
        <span class="glyphicon glyphicon-edit"></span>
        Edit specification
      </a>
    </div>
  </div>
</div>
```

Pro použití frameworku Bootstrap v mé aplikaci jsem se rozhodl z důvodu toho, že značně urychlí tvorbu uživatelského rozhraní, je jednoduchý na používaní a zajišťuje responzivnost rozhraní. Aplikace je pak uživatelsky přívětivá a celkově dobře vypadá.

Kapitola 4

Implementace

V této kapitole se věnuji již samotné implementaci webové aplikace. Cílem implementace bylo realizovat kompletní aplikaci pro tvorbu odhadů ceny softwaru s funkcemi navrhovanými v kapitole 2. K implementaci aplikace jsem využíval technologii a nástrojů popsaných v kapitole 3.

V této části je popsán postup implementace aplikace, její struktura a úlohy jednotlivých komponent aplikace, a to včetně ukázek zdrojových kódu přímo z aplikace. Uvedené zdrojové kódy mohou být oproti kódu aplikace lehce poupraveny a pro účely ukázky je z nich vždy vybrána pouze určitá část.

4.1 Postup při implementaci

Při realizaci aplikace jsem jednotlivé části aplikace implementoval postupně. Nejdříve jsem implementoval základní funkcionalitu nutnou k tvorbě odhadů projektů a k ní jsem postupně přidával další navrhované funkce.

Na začátku jsem implementoval funkce týkající se správy projektů, registrace a přihlášení uživatelů do aplikace. K projektům jsem poté přiřadil uživatele – vlastníky, které jsem doplnil o všechny potřebné údaje k jejich identifikaci. Následně jsem implementoval přidávání aktivit do projektu, odhadování aktivit a zařazení aktivit do kategorii. V seznamech projektů jsem zavedl filtrování a možnost nastavovat parametry odhadu u projektu.

Následovala implementace celkového odhadu a funkce přidávání odhadců do projektu a s tím spojená spolupráce ostatních uživatelů na odhadování projektů. Nakonec byly implementovány notifikace uživatelů a možnost manipulovat s rozsahem celkového odhadu.

4.2 Struktura aplikace

Základní kostru k jednotlivým částem aplikace jsem vytvořil pomocí generátorů zdrojových kódů – Scaffoldingem, který pro každou generovanou entitu

vytvořil základní strukturu, na které pak bylo možné aplikaci stavět. Přestože generovaný kód bylo nutné pro potřeby aplikace do značné míry upravit, získal jsem tím pro každou entitu kompletně vytvořenou MVC strukturu, migrace databáze a další soubory potřebné k implementaci a testování.

Takto vygenerované soubory jsem podle požadavků na aplikaci upravil. Modifikoval jsem výchozí chovaní controllerů, rozšířil jsem je o nové metody a kompletně jsem předělal routování. K modelu jsem přidal validace a metody zajišťující aplikační logiku a to včetně metod, potřebných k výpočtu odhadu pracnosti a ceny. Většina vygenerovaných pohledů byla, z důvodu požadavků na uživatelské rozhraní, buď zcela předělána nebo odstraněna a následně podle action metod controllerů vytvořeny pohledy nové. Všechny nevyužité vygenerované části byly z aplikace smazány.

Členění aplikace podle architektury MVC a úloha těchto částí v aplikaci je podrobněji popsána v následujících sekcích.

4.3 Model

Velká část modelu aplikace přímo vychází z datového modelu. Jednotlivé modely reprezentují databázové entity a jsou určeny k interakci s příslušnou tabulkou v databázi. Kromě přístupu k atributům přes objektově-relační mapování je možné přes model přistupovat i k vztahově závislým entitám. Vztahy k ostatním entitám modelu je možné nadefinovat přímo v těle třídy modelu a jejich definice umožní na modelu volat metody, pomocí kterých je možné k závislým entitám přímo přistupovat.

V následující kódu je ukázaná definice vztahů pro model `Project`, která umožní například volání metody `estimating_users`. Tato metoda z databáze získá všechny uživatele, kteří spolupracují na odhadu projektu.

```
class Project < ActiveRecord::Base
  has_many :activities, dependent: :delete_all
  has_many :activity_categories, dependent: :delete_all
  has_many :estimators
  has_many :estimating_users, through: :estimators, source: :user
  belongs_to :user
end
```

Databázové entity jsou v modelu navíc rozšířeny o metody, které řeší většinu logiky celé aplikace. Součástí modelu jsou dále ještě dvě třídy, které z datového modelu nevycházejí. Třída `Estimation` je určena k zapouzdření elementárních vzorců metody PERT, čímž je umožněno tyto vzorce bez zásahu do ostatních modelů měnit. Třída `Ability` potom definuje oprávnění uživatelů přistupovat k určitým částem aplikace.

4.3.1 Databázové entity

Databázové entity, ze kterých model aplikace vychází, jsou znázorněny na vztahovém diagramu entit 4.1. V diagramu jsou uvedeny atributy entit a jejich vzájemné asociace.

Kromě vztahů přímo viditelných z diagramu, bych chtěl zmínit, že entita `Notification` má ještě navíc odesílatele (`User`) a předmět, ke kterému se vztahuje. Předmětem notifikace může být jakákoliv entita.

Entita `Estimator` reprezentuje dekompozici vztahu M:N uživatele a projektů. Pomocí této entity se zaznamenává, že uživatel spolupracuje na odhadu projektu, tedy je jeho odhadcem. Uživatel může spolupracovat na libovolném množství projektů, stejně tak projekt může mít libovolné množství odhadců.

Definice databázového schématu s entitami a jejich atributy vznikla pomocí souborů s migracemi. Na základně migrací databáze se pak k definovaným entitám vytvoří v databázi příslušné tabulky.

4.3.2 Aplikační logika

Většina logiky je v aplikaci obsažena v podobě metod modelu, které lze k dosažení potřebných funkcí volat. Tyto metody je možné používat v controllerech a pohledech a jejich implementace je v modelu aplikace zapouzdřená.

Velkou částí aplikační logiky tvoří metody určené k odhadování aktivit a k výpočtu souhrnného odhadu ceny a pracnosti jednotlivých kategorií a celkového projektu. Popis metod potřebných k odhadování je uveden v následující části.

4.3.2.1 Aktivity

Aktivity disponují čtyřmi metodami pro výpočet odhadu, které jsou pak základem k výpočtu celkového odhadu projektu. Tyto metody jsou:

`estimated?` – Určí, zda jsou u aktivity odhadnuty všechny tři hodnoty pracnosti.

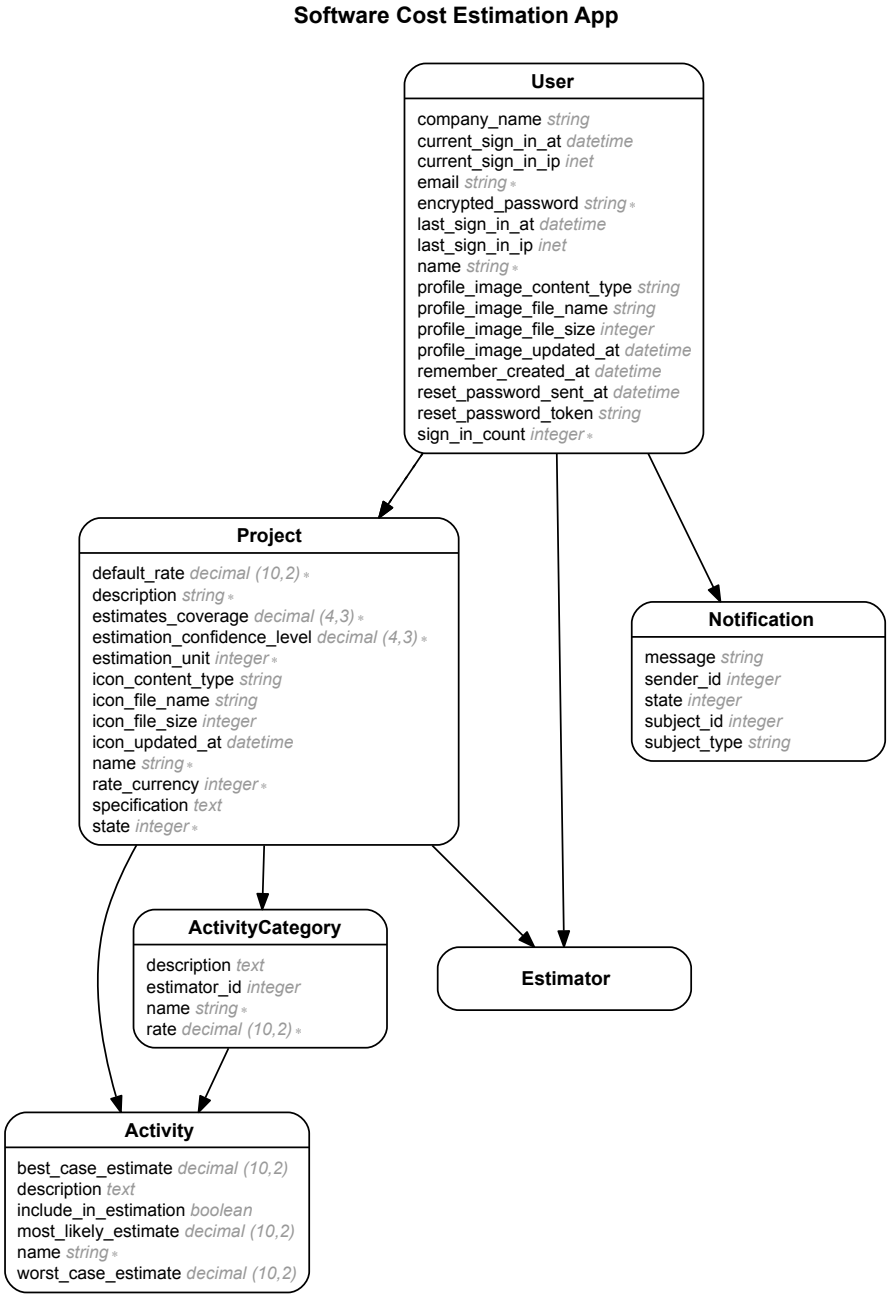
`expected_duration` – Vypočítá očekávanou dobu trvání aktivity na základě vzorce z metody PERT.

`estimated_cost` – Vypočte odhadnutou cenu aktivity na základě očekávané doby trvání a hodinové sazby kategorie, do které aktivita patří.

`standard_deviation` – Určí pro aktivity hodnotu standardní odchylky pomocí metody PERT.

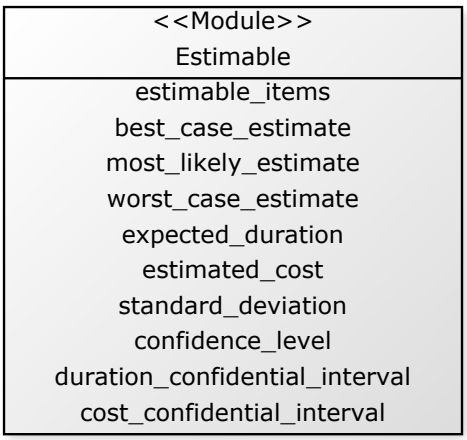
4.3.2.2 ActivityCategory a Project

Model `ActivityCategory` a `Project` disponují stejnými metodami k výpočtu souhrnných odhadů. Rozhraní těchto metod je součástí modulu `Estimable`



Obrázek 4.1: Entity relationship diagram

(Obrázek 4.2) a je vlastně rozšířením metod a atributů modelu `Activity`, s tou změnou, že vypočtené hodnoty jsou souhrnným odhadem všech aktivit obsažených v kategorii, případně v celém projektu. Podobnost rozhraní umožňuje s modely při zjišťování odhadnuté pracnosti a ceny dobře pracovat, díky jejich polymorfnímu chování.



Obrázek 4.2: Modul Estimable

Z důvodu stejného rozhraní metod a velmi podobné implementaci některých těchto metod, jsem vytvořil modul `Estimable`, který oba dva modely podporují. Modul definuje všechny metody, které mají modely podporovat a zároveň několik těchto metod přímo implementuje. V modelech pak díky implementaci jednoduché metody `estimable_items` není nutné podstatnou část metod zvlášť implementovat, protože díky polymorfnímu rozhraní může být jejich implementace obsažena již přímo v modulu.

Následuje ukázka zdrojového kódu metody přímo implementované v modulu `Estimable`. V kódu je ukázána implementace metody k výpočtu konfidenčního intervalu pro očekávanou pracnost.

```
module Estimable
  # Asymptotic normality, Normal-distribution
  def duration_confidential_interval
    mean = expected_duration
    return [0, 0] if mean == 0
    p_value = Distribution::Normal.p_value((1 + confidence_level) / 2)
    interval_difference = p_value * standard_deviation
    [mean - interval_difference, mean + interval_difference]
  end
end
```

4.3.3 Autorizace

Součástí modelu je také třída `Ability`, v jejímž konstruktoru jsou definována oprávnění přístupu uživatelů k projektům, kategoriím, aktivitám a dalším zdrojům v aplikaci. Tato třída je definičním souborem použité autorizační knihovny `CanCanCan`. Na základě této knihovny a definice oprávnění je pak danému uživateli buď povolen nebo zakázán přístup k určitému zdroji.

To znamená, že na základě pravidel ve třídě `Ability` je kontrolováno, zda daný uživatel má oprávnění požadovat zobrazení, provádět úpravu či mazat daný zdroj, a to je zajišťováno kontrolou oprávnění v každé action metodě controllerů. Zároveň uživateli, který má ke zdroji omezený přístup, například jen pro jeho čtení, mohou být zobrazeny jen některé informace o zdroji nebo celkově přizpůsobeno uživatelské rozhraní.

```
class Ability
  include CanCan::Ability
  def initialize(user)
    user ||= User.new
    # Project
    can :manage, Project, user: user
    can :read, Project do |project|
      project.estimating_users.where(id: user.id).any?
    end
  end
end
```

V předchozí ukázce je uvedena definice oprávnění pro přístup k projektům. Autorizace přístupu uživatele k projektu závisí na jeho vztahu k němu. Spravovat projekt může pouze vlastník projektu a zobrazovat ho mohou uživatelé, kteří spolupracují na jeho odhadu, ostatní uživatelé mají přístup k projektu zakázán.

Oprávnění založené na vztahu k entitě je obdobně jako pro projekt definováno pro všechny ostatní entity aplikace. Takto implementovaná autorizace umožňuje bezpečnou spolupráci na odhadech projektů. Odhadce projektu si může projekt zobrazit společně s jeho specifikací a kategoriemi, které mu byly k odhadu přiděleny a muže spravovat v nich obsažené aktivity. Nemůže ale zasahovat do nastavení projektu nebo do kategorii a aktivit, na které přidělen nebyl. Kromě odhadců projektu může ke zdrojům v aplikaci přistupovat a spravovat je pouze jejich vlastník.

4.3.4 Validace modelu

Každá entita modelu je validována pomocí několika validačních pravidel, podle kterých je definováno, jak mají atributy platného modelu vypadat. Díky va-

lidaci modelu se pak zabrání tomu, aby se do databáze uložila entita, která obsahuje neplatné údaje.

Prostřednictvím validace modelu jsou validovány i formuláře na vytvoření a úpravu daných entit. Pokud jsou údaje vyplněné ve formuláři vzhledem k modelu neplatné, potom se entita vytvořená nebo upravená na jejich základě nejen neuloží, ale pro každý neplatný atribut je vygenerována i chybová zpráva, kterou pak aplikace může zobrazit uživateli.

Rails obsahuje spoustu předpřipravených validačních metod, které stačí jen použít. Tyto metody validují přítomnost atributů, jejich jedinečnost nebo například validitu řetězců a numerických hodnot. Tyto validační metody se vkládají přímo do těla třídy modelu.

V následujícím kódu je ukázána validace modelu `ActivityCategory`, u kterého je validována přítomnost atributů `name`, `rate` a `project_id`, a také zda je atribut `rate` vetší nebo roven 0.

```
class ActivityCategory < ActiveRecord::Base
  validates :name, :project_id, :rate, presence: true
  validates :rate, numericality:  { greater_than_or_equal_to: 0 }
end
```

Pokud je potřeba komplexnější validace, je možné si vytvořit vlastní validační metodu. Toho využívám v modelu `Estimator`, k validaci, zda přidávaný odhadce projektu náhodou již projekt neodhaduje, nebo zda uživatel s daným id vůbec existuje. Validace modelu `Estimator` je uvedena v další ukázce.

```
class Estimator < ActiveRecord::Base
  validates :user, :project, presence: true
  validate  :team_member_email_validation
  validates :user, uniqueness: { scope: :project }

  private

  def team_member_email_validation
    if user_id
      project = Project.find(project_id)
      existing_member = project.estimators.where(user_id: user_id).first
      if existing_member
        errors.add(:email, "User with email: #{@email} is already ...")
      end
    else
      errors.add(:email, "User with email: #{@email} doesn't exists.")
    end
  end
end
```

4.4 Controllers

Controllery v aplikaci slouží doslova k jejímu „ovládání“. Pomocí action metod, které controllery implementují se zpracovávají požadavky uživatele na jejichž základně pak controllery manipulují s jednotlivými entitami – získávají entity z databáze a informace o nich, přidávají nové entity, upravují ty stávající nebo entity případně mažou. Po obsloužení požadavku controller získaná data z entit předá pohledu a ten pak uživateli zobrazí.

Chování controllerů je v aplikaci do značné míry ovlivněno hierarchií modelových entit, a také architekturou REST, na jejímž základě framework Rails pracuje. REST architektura aplikace určuje pro jednotlivé entity („zdroje“) jednoznačné identifikátory URL, podle kterých se ke zdrojům v aplikaci prostřednictvím action metod controllerů přistupuje.

Každé platné URL je mapováno na konkrétní akci v controlleru, která na jeho základě musí požadavek uživatele zpracovat. Z URL tedy musí být jasné, kterých entit se požadavek týká, jaký je jejich vztah s ostatními zdroji aplikace a jaká akce se má provést. To znamená například, pokud si uživatel bude chtít zobrazit seznam aktivit, musí být v URL identifikován i projekt, případně kategorie, ze kterých se mají tyto aktivity zobrazit.

4.4.1 Routy

Struktura URL identifikátorů a jejich mapování na akce controllerů je v aplikaci definována v souboru `routes.rb`. Způsob mapování, tedy definice route zachycuje všechny potřebné vztahy mezi entitami, toho je dosaženo pomocí tzv. vnořených prostředků (vnořených route).

V následující ukázce je definováno routování pro projekt a pro entity na projektu závislé.

```
resources :projects do
  collection do
    get 'filter/*filter' => 'projects#index', as: :filter
  end
  member do
    get :settings
    get :estimation
  end
  resources :activities
  resources :activity_categories do
    resources :activities
  end
  resources :estimators
end
```

V předchozím kódu je vidět právě vnořovaní route do sebe. Do projektu jsou zde vnořeny aktivity a kategorie a do kategorií jsou pak znovu vnořeny aktivity. Takto definované routy jsou potřeba pro správné rozpoznání vztahů mezi entitami a pro celkové korektní fungování controllerů. Příklady URL odkazů, které jsou pomocí předchozího kódu nadefinovány:

- `/projects/1/activity_categories/1/activities`
- `/projects/1/activities`
- `/projects/1/estimation`
- `/projects/1/activity_categories`

4.4.2 Autentifikace

V souvislosti s controllery bych ještě chtěl zmínit autentifikaci uživatelů, neboť v každé akci contrelleru probíhá kontrola přihlášení uživatele. Autentifikace uživatelů je v aplikaci řešena pomocí gemu `Devise`, což je nejpoužívanější autentifikační knihovna pro framework Ruby on Rails [18]. Knihovna `Devise` zajišťuje komplexní řešení autentifikace uživatelů, včetně přihlašování a registrace uživatelů nebo například možnost obnovení zapomenutého hesla.

`Devise` nabízí několik předpřipravených metod ke kontrole, zda je uživatel přihlášen. Jednou z nich je metoda `authenticate_user!`, která se používá v akcích controlleru. Pokud uživatel není přihlášen, tak mu tato metoda danou akci znepřístupní a automaticky ho přesměruje na přihlašovací obrazovku, kde se může do aplikace buď přihlásit nebo zaregistrovat. Tato metoda je v aplikaci volána před každou akcí controlleru, což je zajištěno pomocí následujícího kódu umístěného v globálním controlleru aplikace:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!
end
```

Další velmi důležitou funkcí knihovny `Devise` je možnost získat aktuálně přihlášené uživatele, a to pomocí metody `current_user`, kterou je možné zavolat v jakémkoliv controlleru nebo i v pohledu. Metoda vrací přímo objekt přihlášeného uživatele a pokud uživatel není přihlášen vrací `nil`.

V následujícím kódu je ukázáno, jak použít objekt přihlášeného uživatele v action metodě controlleru. V tomto případě slouží k přiřazení vlastnictví projektu do metody přistupujícímu uživateli.

```
class ProjectsController < ApplicationController
  # POST /projects
  def create
    @project = Project.create(project_params)
```

```
    @project.user = current_user
    if @project.save
      redirect_to @project, notice: 'Project was successfully created.'
    else
      render :new
    end
  end
end
```

4.5 Views

Pohledy tvoří uživatelské rozhraní aplikace. Prezentují data uživatelům a umožňují aplikaci ovládat. Součástí přílohy práce jsou snímky obrazovek z výsledné aplikace ukazující celé uživatelské rozhraní (Příloha A).

V této sekci se tedy budu zabývat v souvislosti s pohledy pouze některými částmi uživatelského rozhraní, protože vše ostatní je možné vidět a pochopit přímo ze snímků obrazovek aplikace.

4.5.1 Šablony

V aplikaci jsou pohledy reprezentovány převážně HTML šablonami napsanými v jazyce Haml, které aplikace na základě požadavků zobrazuje uživatelům. Uvnitř šablon je definováno rozložení stránky spolu s prvky uživatelského rozhraní a je zde určen také způsob zobrazení dat. Data, která se mají v pohledu zobrazit, dostane pohled od controlleru a pomocí Ruby kódu, který je v šabloně vložen z nich vygeneruje informace, které se pak uživateli zobrazí. Ruby kód vložený v šablonách většinou řeší zobrazení entit a jejich kolekcí a generování odkazů na další obrazovky – požadavky na controllery, které na základě těchto odkazů zobrazí další pohled.

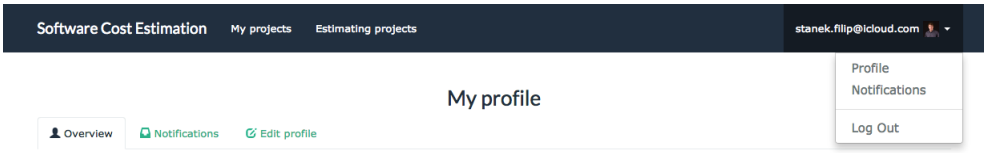
Každá obrazovka, která se uživateli zobrazuje, se skládá z několika do sebe vnořených pohledů, které dělí uživatelské rozhraní na více částí. Většinou jde o navigační část a případně pro více obrazovek společné prvky, do kterých jsou pak vloženy konkrétní informace pro danou obrazovku.

Šablony jsou z velké části ovlivněny frontendovým frameworkem Twitter Bootsrap. Uživatelské rozhraní definované v šablonách používá komponenty a způsob rozložení vycházející z frameworku Bootstrap.

4.5.2 Základní navigační prvky uživatelského rozhraní

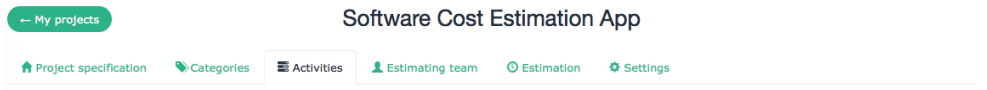
Součástí každé obrazovky je navigační lišta (Obrázek 4.3) umístěna v horní části obrazovky, která slouží k základní navigaci v aplikaci. S její pomocí se může uživatel přepínat mezi vlastními projekty a projekty na kterých spolupracuje. Dále na liště uživatel vidí svůj přihlašovací email, spolu s profilovým obrázkem a informací s počtem nepřečtených notifikaci. Po rozkliknutí emailu

se uživatel může dostat k zobrazení svého uživatelského profilu a k jeho editaci, může si zobrazit notifikace nebo se z aplikace odhlásit. Šablona navigační lišty je umístěna v základním layoutu obrazovek, díky čemuž je součástí každé obrazovky aplikace.



Obrázek 4.3: Navigační lišta spolu s lištou záložek pro navigaci v uživatelském profilu

Dalším důležitým navigačním prvkem aplikace jsou tzv. nav-tabs (lišta záložek). Tento prvek slouží k navigaci v detailu projektu (obrázek 4.4) a v detailu uživatele. V detailu projektu slouží lišta záložek k procházení mezi specifikací projektu, kategoriemi a aktivitami projektu, dále pak umožňuje zobrazit odhadce projektu, jeho nastavení a celkový odhad projektu. Tato navigace je umístěna ve zvláštní šabloně, do které jsou pak vkládány jednotlivé pohledy pro zobrazení součástí projektu.



Obrázek 4.4: Navigace v detailu projektu pomocí lišty záložek

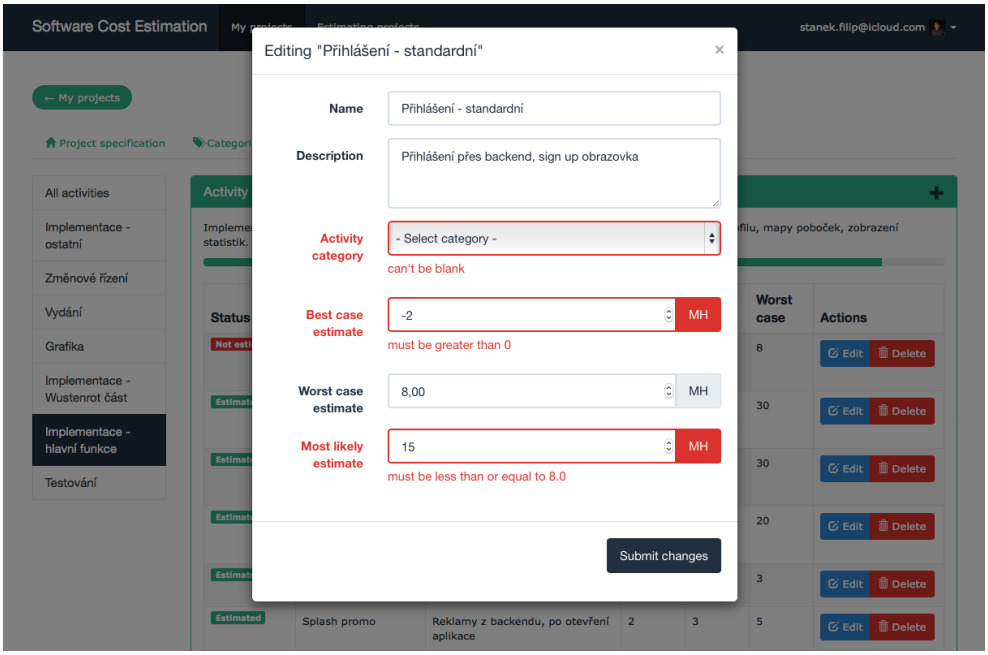
4.5.3 Formuláře

Formuláře jsou důležitou částí uživatelského rozhraní, která slouží k vytváření a úpravě položek v aplikaci. Všechny formuláře v aplikaci jsou nastylované pomocí Boostrapu. Pro jejich generování používám Gem `bootstrap_form`, který generuje formuláře přímo s Bootstrap styly. Tato metoda je použitelná v jakékoliv šabloně pohledu, její použití pro generování formuláře pro `ActivityCategory` je ukázáno v následujícím kódu:

```
= bootstrap_form_for([@project, @activity_category]) do |f|
  = f.text_field :name
  = f.text_area :description, rows: 3
  = f.number_field :rate, append: @project.rate_currency
  = f.select :estimator_id, estimating_team_for(@project, @activity_category)
  = f.primary 'Submit changes'
```

Validace formulářů probíhá na straně serveru a pokud jsou v odeslaném formuláři neplatné údaje, server zobrazí formulář znovu spolu s červeně vyznačenými neplatnými položkami a textovým popisem chyb. Takto neplatný formulář je ukázán na obrázku 4.5.

4.5.4 Modální formulář



Obrázek 4.5: Formulář pro vytváření a úpravu aktivit umístěný v modálním okně, včetně chybně vyplněných polí

Kromě pohledů ve formátu HTML používám v aplikaci i několik javascriptových. Javasriptové pohledy jsou v aplikaci potřeba, aby bylo možné přidávat a upravovat projektové aktivity prostřednictvím modálního okna. Jelikož přidávání a úprava aktivit je nejčastější úkon, který uživatel odhadující projekt provádí, rozhodl jsem pro použití modálního formuláře. Uživatel tak nemusí být pokaždé přesměrováván na obrazovku s formulářem a celá obrazovka se seznamem aktivit se nemusí přenačítat. Modální formulář aktivit tak celkově zjednoduší a zefektivní proces přidávání a úpravy aktivit.

Javascriptové pohledy jsou použity k zobrazení formuláře a k obnově seznamu aktivit po jeho odeslání. Pomocí javascriptu se do určitých částí stránky vykresluje HTML kód, který je vygenerován již klasickou HTML šablonou, konkrétně je to šablona pro tabulku se seznamem aktivit a formulář pro ak-

tivitu. Odesílání modálního formuláře a získávání částí stránky ze serveru funguje pomocí AJAX.

4.5.5 Helper metody

V souvislosti s pohledy bych chtěl ještě zmínit tzv. helper metody. Tyto metody jsou v pohledech používány například ke generování HTML a jejich používáním se lze vyvarovat psaní složitější logiky pro zobrazování přímo do pohledů. Následuje ukázka helper metody, která vytvoří možnosti pro výběr uživatelů v selectu formuláře.

```
def estimators_for_select(users)
  return unless users
  mapped_users = users.order(:email).map { |user| [user.to_s, user.id] }
  options_for_select(mapped_users)
end
```

4.5.6 WYSIWYG editor

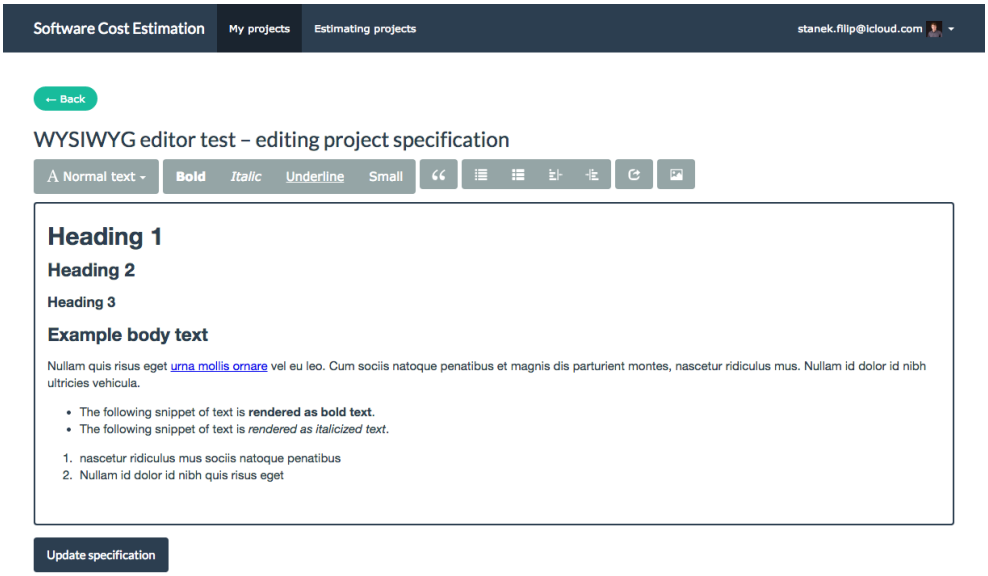
K editaci projektové specifikace je použit WYSIWYG editor, který umožňuje základní editaci textu. Možnost strukturovat text specifikace je důležité proto, aby byl přehledný. Do WYSIWYG editoru může být také jednoduše zkopírován text z textového editoru, emailu nebo webové stránky, aniž by přišel o své formátování. Pro implementaci WYSIWYG editoru jsem použil gem `bootstrap-wysihtml5-rails`. Screenshot WYSIWYG editoru je dostupný na Obrázku 4.6.

4.5.7 Typeahead vyhledávání

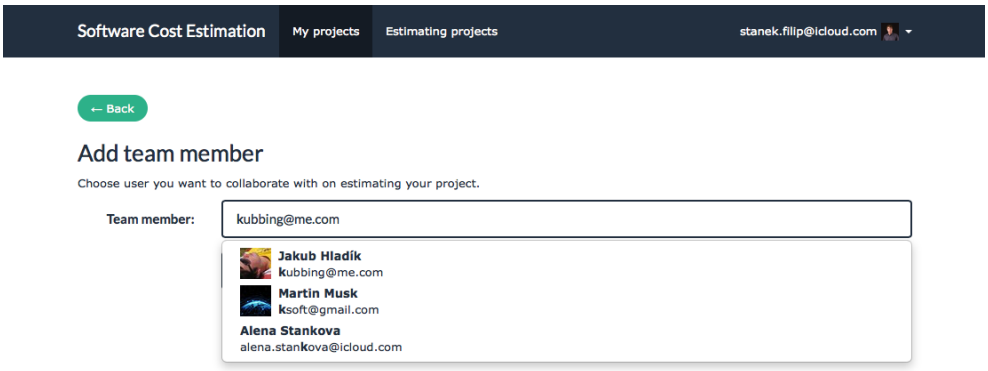
Pro přidání spolupracovníka na odhad projektu je nejdříve nutné tohoto uživatele vyhledat. K vyhledávání odhadců v aplikaci slouží dynamické pole implementované pomocí javascriptové knihovny `typeahead.js`, které vyhledává uživatele podle registračního emailu. Do pole stačí zadat jen část emailu a pole nabídne uživatele, jejichž email tuto část zahrnuje. U nabídnutých uživatelů jsou kromě emailu uvedeny jejich jména, názvy společnosti a případně i profilové obrázky. Vyhledávání je ukázáno na Obrázku 4.7.

4.6 Automatické testy

Součástí aplikace jsou také automatické testy, podle kterých je možné ověřit základní funkčnost aplikace. Testy jsou implementovány pomocí knihovny `RSpec` a testují funkčnost výpočtu odhadu tak, aby bylo možné v případě změny kódu rychle a jednoduše ověřit, že odhadování funguje jak má. V testech jsou důkladně otestovány jednotlivé modely a jejich metody pro výpočet odhadu.



Obrázek 4.6: Úprava specifikace projektu pomocí WYSIWYG editoru



Obrázek 4.7: Dynamické vyhledávání uživatelů pro spolupráci na projektech

Kapitola 5

Testování aplikace na reálném projektu

Po vytvoření aplikace jsem se rozhodl otestovat ji na reálném projektu, abych zjistil, jak přesný odhad lze použitím aplikace dosáhnout.

5.1 Odhad mobilní aplikace ProAuto

Pro otestování aplikace jsem si zvolil již realizovaný projekt, u kterého mohu zjistit, jaké byly jeho reálné náklady. Nebylo však žádoucí, abych aplikaci testoval na projektu, na kterém jsem sám pracoval, proto jsem k testování zvolil projekt mobilní aplikace ProAuto, kterou vytvořila nejmenovaná společnost. Jedná se o aplikaci pro platformu iOS, pro kterou sám aplikace vyvíjím a mám tedy s implementací obdobných projektů zkušenost. To by mělo zajistit dostatečnou přesnost dílčích odhadů aktivit.

Odhad projektu jsem tvořil na základě dokumentu se specifikací požadavků a prototypu obrazovek aplikace. Do odhadu jsem kromě samotné implementace zahrnul i analýzu, testování, změnové řízení a podpůrné činnosti. Jedinou část projektu, kterou jsem do aplikace nezahrnoval je tvorba grafiky, neboť jsem nezískal data o nákladech na ni a ani bych nebyl schopen časovou náročnost vytvoření grafiky sám odhadnout.

5.1.1 Popis aplikace

Pro představu o tom, co je funkcí aplikace, uvádím její popis v Apple AppStore:

„S aplikací ProAuto CZ budete mít dokonalý přehled o nákladech vydaných za vaše vozidlo, spotřebě paliva a ujetých kilometrech. Náklady jsou rozděleny do typických kategorií a zobrazeny v přehledných grafech nebo mapách. Pokud se navíc rozhodnete vytvořit si v rámci aplikace uživatelský účet, získáte navíc funkci upomínky, která vám pohlídá zaplacení pojištění, výměnu

oleje, či vypršení dálniční známky. Všechna data se vám budou zálohovat na server pro případ pozdější obnovy s novým přístrojem.“ [19]

5.2 Výsledek testu

V této sekci porovnávám výsledky celkového odhadu vytvořeného v aplikaci s reálnými náklady na realizaci projektu. Kompletně vytvořený odhad je k dispozici v příloze práce ve formě tisknutelného výstupu z celkového odhadu projektu (Příloha B), kde je možné si odhad prohlédnout podrobně, včetně rozpisu jednotlivých odhadnutých kategorii a aktivit. Odhad projektu ProAuto je také součástí testovacích dat, takže je možného si odhad projektu zobrazit přímo v aplikaci.

Součásti nákladů na projekt	Cena
Náklady na analýzu	7 800 Kč
Náklady na implementaci	84 000 Kč
Náklady na podpůrné činnosti	12 000 Kč
Náklady na změny	12 250 Kč
Náklady na testování	10 000 Kč
Náklady celkem	**126 050 Kč**

Tabulka 5.1: Tabulka reálných nákladů na vytvoření aplikace ProAuto

Součásti odhadu projektu	Odhadnuté rozmezí ceny
Analýza	7 569,43 Kč – 9 630,57 Kč
Implementace	69 158,81 Kč – 85 057,85 Kč
Podpůrné činnosti	8 666,55 Kč – 11 883,45 Kč
Rezervy na změny	7 396,01 Kč – 11 037,3 Kč
Testování	8 949,61 Kč – 13 467,06 Kč
Náklady celkem	**101 740,41 Kč – 131 076,26 Kč**

Tabulka 5.2: Tabulka odhadnutých rozmezí ceny pro jednotlivé části aplikace ProAuto

V předchozích dvou tabulkách je vidět porovnání reálných nákladů a ceny odhadnuté pomocí aplikace. Odhadnutá rozmezí jsou stanovena pro konfidenční interval s 98% spolehlivostí a 99% pokrytím odhadů. Z tabulek je vidět, že většina reálných nákladů za jednotlivé části aplikace spadá do odhadnutých intervalů.

Pokud bych snížil pokrytí odhadů aktivit na 90%, což je mnohem více realistická hodnota, tak by do odhadnutých intervalů náležely všechny dílčí náklady. Vliv nastavení pokrytí odhadu na celkové rozmezí ceny je ukázáno v tabulce 5.3.

Procento pokrytí	Celková odhadnutá cena (96% spolehlivost)
99%	103 459,18 Kč – 129 357,49 Kč
90%	92 864,42 Kč – 139 952,25 Kč
80%	86 525,67 Kč – 146 290,99 Kč
70%	79 410,75 Kč – 153 405,91 Kč

Tabulka 5.3: Tabulka odhadnuté ceny aplikace ProAuto v závislosti na zvoleném procentu pokrytí odhadu

Z testu aplikace na reálném projektu jsem zjistil, že celkové náklady na projekt (126 050 Kč) jsou úspěšně obsaženy v odhadnutém rozmezí (101 740 Kč – 131 076 Kč). Náklady jsou sice v horní částí intervalu a od očekávané odhadnuté ceny 116 408 Kč (střed intervalu) se liší o 9 642 Kč. Nicméně díky tomu, že výstupem aplikace je primárně rozmezí ceny místo jedné hodnoty, byl test úspěšný a náklady na vytvoření aplikace tedy odhadnuty správně.

Kapitola 6

Zhodnocení přínosů aplikace

V této kapitole uvádím stručný souhrn možných přínosů aplikace pro podnikovou praxi, tedy jaké výhody používaní aplikace přinese softwarovým firmám a vývojářům, kteří se zabývají vývojem mobilních a webových aplikací.

6.1 Systematizace odhadů

Vytvořená aplikace je specializovaným nástrojem k tvorbě a správě odhadů ceny vývoje menších softwarových projektů, jejíž hlavním cílem je poskytnout jednoduché řešení odhadů všem, kteří pro tento typ softwaru nemají zavedený způsob odhadování.

Používaní aplikace zavede do odhadování softwarů systém, který pevně definuje strukturu odhadů, způsob jejich tvorby a volbu metod použitých k výpočtu odhadu. Z odhadů se tak eliminuje neformálnost, která může negativně ovlivnit přesnost, a díky tomu budou jednotlivé odhady projektů navzájem konzistentní a půjdou spolu dobře porovnávat nebo zpětně analyzovat.

Systematizaci odhadů tedy aplikace do odhadování softwaru přinese následují:

- Evidenci odhadů a jejich správu
- Jednotnou strukturu a formu odhadů
- Konzistentní metody výpočtu odhadů
- Přehledný a strukturovaný odhad
- Odhad projektu rozdělený podle kategorii a aktivit
- Snadnou porovnatelnost odhadů
- Konsolidaci odhadů na jednom místě

6.2 Zvýšení spolehlivosti odhadů

- Metody k výpočtu odhadu ceny a pracnosti, které jsou v aplikaci použité, zajišťují spolehlivost a přesnost celkového odhadu projektu.
- Dekompozice projektu na aktivity vede k lepšímu pochopení odhadovaného softwaru a ke zpřesnění odhadu, prostřednictvím odhadování jeho dílčích celků.
- Zahrnutí různých případů do odhadu jednotlivých aktivit umožní do výsledného odhadu započítat možné problémy, které by při realizaci projektu mohly nastat.
- Pomocí metody PERT vypočtený rozsah pracnosti a ceny vývoje reflektuje možná projektová rizika.
- V případě správné dekompozice aktivit a stanovení možných případů jejich pracnosti aplikace vypočte interval, který se zvolenou pravděpodobností zahrnuje skutečnou pracnost a náklady na vývoj softwarového projektu.

6.3 Spolupráce na odhadech

- Aplikace umožní na odhadech jednoduše spolupracovat s více odhadci.
- Odhadování částí projektu experty na danou problematiku nebo přímo pracovníky, kteří budou danou část softwaru realizovat, pomůže výrazně zvýšit přesnost odhadů.
- Aplikace umožní odhad projektu mezi spolupracovníky jednoduše rozdělit a přidělit jim odpovědnost za jednotlivé části odhadu.

6.4 Další přínosy aplikace

- Jednoduchá manipulace s rozsahem odhadu při vyjednávání o ceně.
- Odhady je možné vytvářet a spravovat z různých typů zařízení a operačních systémů.
- Odhady projektů jsou uloženy v cloudu a jsou dostupné kdykoliv a odkudkoliv.

Kapitola 7

Možnosti nasazní aplikace na trhu

V této kapitole se zabývám možnostmi zpřístupnění aplikace potencionálním uživatelům. Popisuji, jakým způsobem bude aplikace provozována a jaká platforma bude pro běh aplikace zvolena. Uvádím zde také obchodní model, který by v případě, že aplikace bude úspěšná ve zkušebním provozu, mohl být použit k pokrytí nákladů na provoz aplikace i k jejímu zpeněžení.

7.1 Heroku

Heroku je specializovaná cloudová platforma pro běh webových aplikací v Ruby, Node.js, Python, Java, nebo PHP. Jedná se o tzv. řešení Platform as a Service (PaaS), které umožňuje vývoj, nasazení a správu webových aplikací, aniž by bylo nutné spravovat a udržovat server nebo aplikaci složitě na server nasazovat. Heroku se totiž o všechny tyto úkony postará. Hlavní filosofií platformy Heroku je, že vývojáři by se měli soustředit na vývoj aplikace a nezdržovat se jejím nasazením. [20]

Abych svou aplikaci na podporu odhadů zpřístupnil potencionálním uživatelům, rozhodl jsem pro nasazení aplikace právě prostřednictvím platformy Heroku. Prvním důvodem výběru platformy je jednoduchá integrace s frameworkem Ruby on Rails, kdy pro nasazení aplikace na Heroku stačí mít zdrojové kódy aplikace pod systémem správy verzí (Git) a nainstalovaný pro Heroku specifický RubyGem. Další výhodou je, že Heroku kromě samotného běhového prostředí poskytuje i PostgreSQL databázi pro ukládání dat.

V neposlední řadě je pak možné Heroku používat v základní verzi zdarma. Přestože je verze zdarma výkonnostně limitována, lze ji bez problému použít pro testování aplikace a v případě, že aplikaci nepoužívá najednou větší množství uživatelů, poskytuje verze zdarma dostatečný výkon pro plnohodnotné používaní aplikace.

7.2 Postup při nasazení aplikace

Proto, abych mohl svou aplikaci na Heroku nasadit, bylo potřeba provést několik jednoduchých úkonů (při popisu postupu nasazení vycházím z [21]). Začal jsem tím, že jsem si zdarma založil účet na `http://heroku.com` a následně po přihlášení do aplikace na správu Heroku, jsem vytvořil novou aplikaci. Specifikoval jsem u aplikace unikátní jméno, které je poté použito v doméně třetího řádu – s příponou `herokuapp.com`, přes kterou se k webové aplikaci přistupuje. Dále jsem zvolil preferované umístění serveru na Evropu pro co nejlepší odezvu na testování. Poté jsem již přímo do aplikace do souboru GemFile přidal Heroku RubyGem:

```
gem 'rails_12factor', group: :production
```

Po nainstalování tohoto gemu, je potřeba se do Heroku přihlásit z konzole pomocí následujícího příkazu:

```
$ heroku login
```

Heroku spoléhá na systém správy verzí Git pro nasazování aplikace, je tedy potřeba mít aplikaci pod jeho správou. Jelikož jsem při vývoji Git používal, stačilo v adresáři aplikace pouze spustit následující příkaz, který automaticky do Gitu přidá vzdálený repozitář Heroku:

```
$ heroku create
```

Tím je základní konfigurace hotová a aplikaci lze na Heroku nasadit. Nasazení aplikace probíhá příkazem:

```
$ git push heroku
```

Tento příkaz je vše, co je nutné k nasazení aktuální verze aplikace. Pomocí příkazu se na Heroku přenesou zdrojové kódy a automaticky se na serveru nainstaluje a nastaví vše potřebné k běhu aplikace. Při prvním nasazení nebo v případě, že do aplikace byla přidána migrace databáze, je ještě potřeba tuto migraci na Heroku provést a to příkazem:

```
$ heroku run rake db:migrate
```

Ve své aplikaci umožňuji uživatelům přidávat ikony k projektům a uživatelské profily. Tyto soubory ale nelze na Heroku přímo ukládat, proto jsem pro jejich ukládání musel použít externí úložiště. Pro ukládání obrázků jsem zvolil cloudovou službu Amazon Web Services, konkrétně Amazon Simple Storage Service (S3). Tato služba je v případě nepřekročení 20 tisíc požadavků za měsíc na první rok zdarma a umožňuje uložení až 5GB dat, což je pro potřeby aplikace dostačující. Úložiště S3 využívám také pro přenos testovacích dat z lokální databáze do aplikace běžící na Heroku.

7.3 Zkušební provoz

V první fázi nasazení aplikace poběží aplikace ve zkušebním provozu. Zkušební provoz bude sloužit ke zjištění, zda vývojáři softwaru mají o používání aplikace vůbec zájem, a tedy zda má cenu aplikaci nasazovat do ostrého provozu. V průběhu zkušebního provozu budu od testovacích uživatelů sbírat připomínky na aplikaci, které budu do aplikace za běhu zapracovávat. Mezi testovací uživatele plánuji zahrnout dvě malé softwarové firmy a několik vývojářů a také budu aplikaci sám používat při vývoji mobilních aplikací na zakázku.

Aplikace bude ve zkušebním provozu kompletně zdarma, proto je pro mě důležité, aby na provozování aplikace nebyly žádné náklady. Toho docílím, díky platformě Heroku, která poskytuje základní program pro běh aplikace zdarma. V tomto základním programu je zahrnut jeden tzv. `Dynos`, což je jednotka výpočetní síly vyhrazené pro běh aplikace. Jeden `Dynos` poskytuje jeden sdílený procesor a 512 MB RAM [22]. Nemusí se to zdát mnoho, ale mám vyzkoušeno, že pro běh aplikace je to dostatečný výkon, který zajišťuje svižnou odezvu aplikace. Problém s nedostatkem výkonu by nastal pouze v případě, kdy by aplikaci používalo větší množství uživatelů najednou, což při zkušebním provozu ale není pravděpodobné.

Dále Heroku poskytuje zdarma 10 tisíc řádků v PostgreSQL databázi [22], což by mělo stačit pro cca 40 testovacích uživatelů, z nichž počítám, že každý bude mít cca 4 vlastní projekty a každý projekt 60 záznamů, dále počítám u každého uživatele s místem na záznam uživatelského profilu a notifikací.

Jedinou nevýhodou použití Heroku v základním programu je, že aplikace se po hodině neaktivity uvolní z operační paměti serveru. A při prvním načtení aplikace po tomto období se musí čekat, než se znovu do operační paměti nahraje. To znamená, že první požadavek bude trvat až cca 10 sekund.

Aplikace v době psaní této práce již běží v testovacím provozu a je možné ji vyzkoušet na `https://software-cost-estimation.herokuapp.com`.

7.4 Obchodní model

V případě, že zkušební provoz aplikace bude úspěšný a tedy zjistím, že testovací uživatelé aplikaci reálně používají pro tvorbu odhadů svých softwarových projektů a jsou s ní spokojeni, tak aplikaci nasadím do ostrého provozu. Ostrý provoz se bude lišit zejména výkonem běhového prostředí a větším prostorem pro uživatelská data tak, aby mohlo aplikaci používat více uživatelů. S tím již budou spojené náklady na provoz aplikace, a proto jsem pro ostrý provoz navrhl obchodní model, díky kterému bych mohl tyto náklady pokrýt a potencionálně i na aplikaci něco vydělat.

Pro poskytování aplikace navrhuji obchodní model měsíčního předplatného. Konkrétně tři různé programy předplatného, které se liší zejména li-

mitem počtu odhadovaných projektů a jejich maximální velikostí. Uživatelé aplikace si podle svých potřeb budou moci vybrat program, který jim v danou chvíli nejvíce vyhovuje.

Předplatné je u obdobných webových aplikací hojně používáno. Výhodou tohoto obchodního modelu je, že v závislosti na počtu předplatitelů mohu úměrně zvyšovat a snižovat výkon serveru a velikost úložiště tak, abych prokryl měsíční náklady na provozování aplikace. Tento obchodní model je výhodný také pro uživatele aplikace, protože vyžaduje jen minimální počáteční investici a uživateli umožňuje v případě nespokojenosti s aplikací své předplatné kdykoliv zrušit.

Následuje popis navrhovaných programů předplatného spolu s jejich měsíční cenou. Jelikož aplikace je určená pro globální trh, cena je uvedena v dolarech.

7.4.1 Basic estimator

- 3 vlastní projekty
- 30 aktivit a 8 kategorii na projekt
- Maximálně 2 spolupracovníci na odhadu projektu
- Spolupráce na neomezeném množství projektů jiných uživatelů
- Cena – zdarma

Basic Estimator je program, který umožní aplikaci zdarma plnohodnotně vyzkoušet na třech menších projektech. Zejména je však tento program určen uživatelům, kteří se primárně účastní odhadů projektů ostatních uživatelů. To znamená, že pokud softwarová firma používá některý z placených programů, tak už nemá žádné další výdaje za pracovníky, kteří se na odhadech projektů podílejí.

7.4.2 Small business

- 30 vlastních projektů
- 150 aktivit a 30 kategorii na projekt
- Maximálně 6 spolupracovníků na odhadu projektu
- Spolupráce na neomezeném množství projektů jiných uživatelů
- Cena – $2.99/měsíc

Small business program je určen pro menší softwarové firmy a živnostníky, kteří se zabývají vývojem softwaru. Za nízkou cenu tento program poskytuje možnost odhadovat dostatečné množství vlastních projektů a mít až 6 spolupracovníků na projektu, což by mělo být vzhledem k běžné velikosti týmu při vývoji menších webových nebo mobilních aplikací dostačující.

7.4.3 Premium unlimited

- Neomezený počet projektů
- Neomezeně aktivit a kategorii na projektu
- Neomezený počet spolupracovníků na odhadu projektu
- Spolupráce na neomezeném množství projektů jiných uživatelů
- Cena – $7.99/měsíc

Program Premium unlimited umožňuje neomezené používaní aplikace a je vhodný pro toho, komu nestačí program Small business.

7.5 Ostré nasazení aplikace

Před nasazením aplikace do ostrého provozu bude nejdříve potřeba aplikaci rozšířit o několik součástí. Například do aplikace je nutné přidat informační část, tedy webovou prezentaci aplikace. Ta bude sloužit k propagování aplikace a budou v ní uvedeny veškeré potřebné informace o aplikaci, včetně informací o jejích funkcích, o ceně jednotlivých programů a jejich omezení a o obchodních podmínkách. Také bude obsahovat ukázky použití aplikace na testovacím projektu (use case), který bude rovněž sloužit jako jednoduchý návod na používání aplikace.

Další věcí, kterou bude do aplikace potřeba dodělat je modul zajišťující fungování předplatného. Tento modul bude řešit nákupy předplatného, přiřazení předplatného k uživatelskému účtu a také bude řešit limitace použití aplikace podle zvoleného programu.

Poslední věcí, kterou bych chtěl před nasazením do ostrého provozu udělat, je registrace vhodné domény druhého řádu, přes kterou se bude k aplikaci přistupovat místo základní domény, kterou poskytuje Heroku.

Aplikace i v ostrém provozu, tedy minimálně ze začátku poběží na platformě Heroku, ale oproti testovacímu provozu bude zvýšen výkon a prostor na data. Konkretně bude navýšen výpočetní výkon na dva `Dynos`, což pro běh aplikace poskytne dva sdílené procesory a 1024 MB RAM a také eliminuje uvolňování aplikace z paměti serveru. V době psaní této práce dva `Dynos` vyjdou na $34.50/měsíc, pokud bych ale chtěl na Heroku zvýšit výpočetní výkon

ještě o něco víc, cena by byla $538. [22] Takto vysoká cena by již za jednoduchost nasazování aplikace nestála, v takovém případě bych přešel na jinou službu.

Prostor v databázi na Heroku plánuji rozšířit na 10 milionů řádku za $9/měsíc [22], což by teoreticky mělo vystačit na 1785 uživatelů s programem Small business (cca 5600 záznamů na uživatele), ale samozřejmě se musí počítat i s uživateli s programem zdarma nebo neomezeným. Nicméně pro začátek by to měl být dostatečný velký prostor a v případě potřeby lze prostor databáze rozšířit například za $50/měsíc na 64 GB [22].

Celkově by mě provoz aplikace vyšel na $43.50 za měsíc a tyto náklady by se mi vrátily například v případě, že aplikaci použije alespoň 15 uživatelů s předplatným Small business.

Závěr

Všechny požadavky ze zadání této bakalářské práce byly splněny. V rámci práce byla vytvořena webová aplikace pro odhadování ceny vývoje softwarových projektů, která umožňuje tvorbu a správu odhadů. Aplikace vypočítá celkovou pracnost a cenu vývoje softwaru a odhadne rozmezí, které se zvolenou pravděpodobností reflektuje skutečné náklady na vývoj projektu.

V práci jsem se nejdříve zabýval problematikou odhadování softwaru, kde jsem zjišťoval, jak se má software správně odhadovat a jaké metody a techniky k zajištění spolehlivosti odhadů použít. Poté jsem se věnoval přímo metodám výpočtu odhadu, které bych mohl ve své aplikaci pro odhadování menších softwarových projektů aplikovat.

Na základě poznatků z analýzy jsem navrhl funkce a vlastnosti aplikace a zvolil jsem techniku odhadování PERT, jakožto základní metodu k výpočtům odhadů, na které se celá aplikace zakládá.

Následně jsem se zabýval volbou vhodných technologii pro realizaci aplikace. Jako základní stavební kámen jsem se rozhodl použít webový framework Ruby on Rails, což se ukázalo jako dobrá volba i přesto, že jsem tento framework používal poprvé a musel jsem tak v průběhu implementace zjišťovat jak funguje. K tvorbě uživatelského rozhraní jsem použil Twitter Bootstrap, díky kterému jsem u aplikace docílil jednoduchého a dobře vypadajícího webového rozhraní, které se navíc přizpůsobuje různým typům zařízením.

V práci jsem se věnoval i samotnému způsobu realizace aplikace, kde aplikaci dělím podle architektury MVC a popisuji, jak jsem jednotlivé části implementoval. Zabývám se tím, jak přesně aplikace funguje a popisuji zde například strukturu databázových entit, způsob výpočtu celkového odhadu nebo způsob fungování spolupráce více uživatelů na odhadech.

Po zrealizování aplikace jsem se věnoval jejímu testování na reálném projektu, kde jsem zjistil, že použité metody k výpočtu jsou dostatečně přesné a aplikace produkuje spolehlivé hodnoty odhadu ceny vývoje. Nakonec jsem zhodnotil přínosnost aplikace pro použití v podnikové praxi a analyzoval jsem možnosti jejího nasazení na trhu.

Literatura

[1] Azad, K.: *Intermediate Rails: Understanding Models, Views and Controllers* [online] [cit. 10. dubna 2015]. Dostupné z: http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

[2] McConnell, S.: *Odhadování softwarových projektů: jak správně určit rozpočet, termín a zdroje.* Brno: Computer Press, 2006, ISBN 80-251-1240-3, 311 s.

[3] Trendowicz, A.; Jeffery, R.: *Software Project Effort Estimation.* Springer International Publishing, 2014, ISBN 978-3-319-03629-8, 469 s.

[4] Surňák, P.: *Odhadování softwarových projektů.* Diplomová práce, Vysoká škola ekonomická v Praze, Praha, 2010.

[5] *WBS (Work Breakdown Structure)* [online]. Managementmania [cit. 10. dubna 2015]. Dostupné z: https://managementmania.com/cs/work-breakdown-structure

[6] *Doporučení pro tvorbu odhadů* [online]. Profinit, s.r.o., 2012 [cit. 10. dubna 2015]. Dostupné z: http://www.profinit.eu/fileadmin/Content/profinit.eu/Academy/sweng-materialy/odhady/Profinit_metodika_tvorby_odhadu.pdf

[7] *Web App Frameworks* [online]. The Ruby Toolbox [cit. 10. dubna 2015]. Dostupné z: https://www.ruby-toolbox.com/categories/web_app_frameworks

[8] *Ruby on Rails a revoluce ve vývoji pro web. Část první: Ruby* [online] [cit. 10. dubna 2015]. Dostupné z: http://blog.karmi.cz/2007/6/16/co-je-ruby-on-rails-cast-1.html

[9] *About Ruby* [online] [cit. 10. dubna 2015]. Dostupné z: https://www.ruby-lang.org/en/about/

[10] *Ruby on Rails Study Guide: The History of Rails* [online] [cit. 10. dubna 2015]. Dostupné z: http://code.tutsplus.com/articles/ruby-on-rails-study-guide-the-history-of-rails--net-29439

[11] *Začínáme s Rails* [online]. Ruby on Rails Guides [cit. 10. dubna 2015]. Dostupné z: http://guides.rubyonrails.cz

[12] *Getting Started with Rails* [online]. Ruby on Rails Guides [cit. 10. dubna 2015]. Dostupné z: http://guides.rubyonrails.org/getting_started.html

[13] *Active Record Basics* [online]. Ruby on Rails Guides [cit. 10. dubna 2015]. Dostupné z: http://guides.rubyonrails.org/active_record_basics.html

[14] *Beautiful, DRY, well-indented, clear markup: templating haiku* [online]. Haml. [cit. 11. dubna 2015]. Dostupné z: http://haml.info

[15] Stěhule, P.: *PostgreSQL* [online] [cit. 11. dubna 2015]. Dostupné z: http://postgres.cz/wiki/PostgreSQL

[16] Michálek, M.: *K čemu je dobrý Bootstrap a frontend frameworky?* [online]. 2013 [cit. 11. dubna 2015]. Dostupné z: http://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky/

[17] *Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web* [online]. Bootstrap [cit. 11. dubna 2015]. Dostupné z: http://getbootstrap.com/

[18] *Rails Authentication* [online]. The Ruby Toolbox [cit. 10. dubna 2015]. Dostupné z: https://www.ruby-toolbox.com/categories/rails_authentication

[19] *iTunes Preview - ProAuto CZ* [online]. Apple Inc [cit. 23. dubna 2015]. Dostupné z: https://itunes.apple.com/cz/app/proauto-cz/id918123890?mt=8

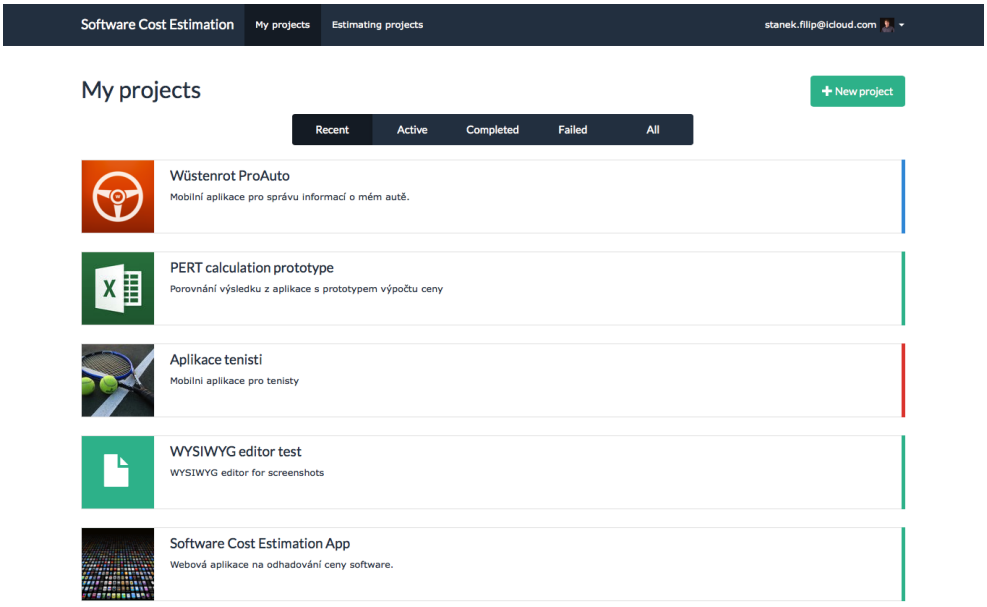
[20] *About Heroku* [online]. Heroku, Inc. [cit. 5. května 2015]. Dostupné z: https://www.heroku.com/about

[21] *Getting Started with Rails 4.x on Heroku* [online]. Heroku, Inc., 2015 [cit. 5. května 2015]. Dostupné z: https://devcenter.heroku.com/articles/getting-started-with-rails4

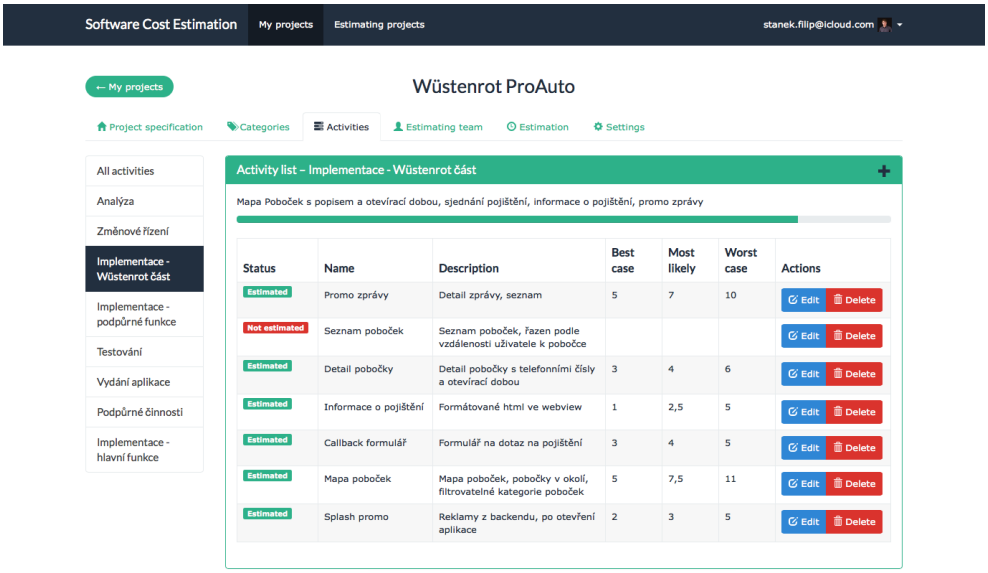
[22] *Heroku pricing* [online]. Heroku, Inc. [cit. 5. května 2015]. Dostupné z: https://www.heroku.com/pricing

Příloha A

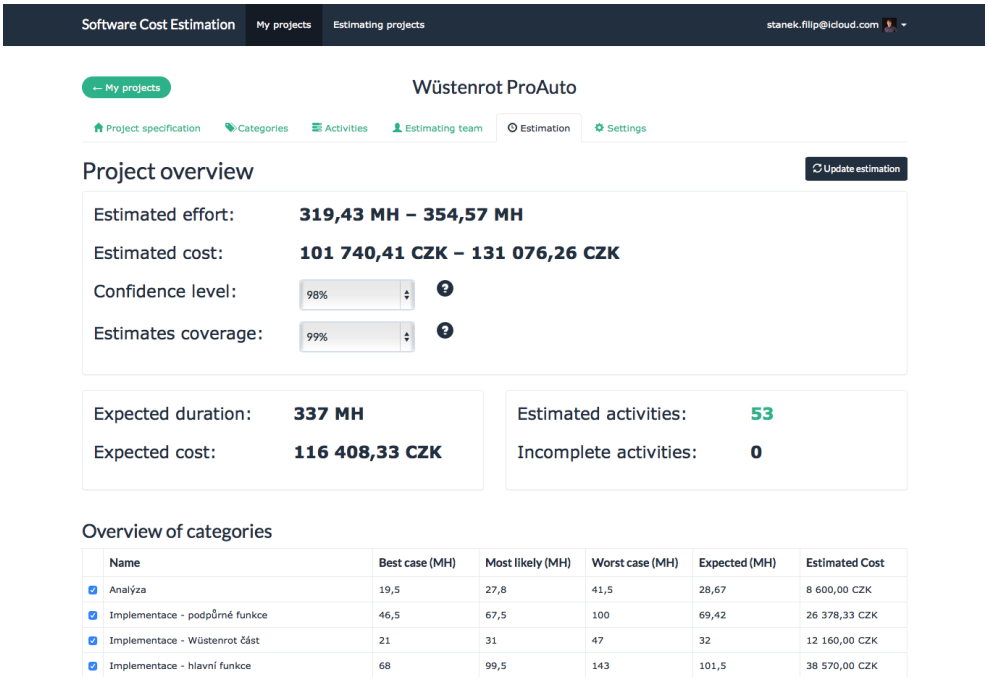
Screenshoty z aplikace



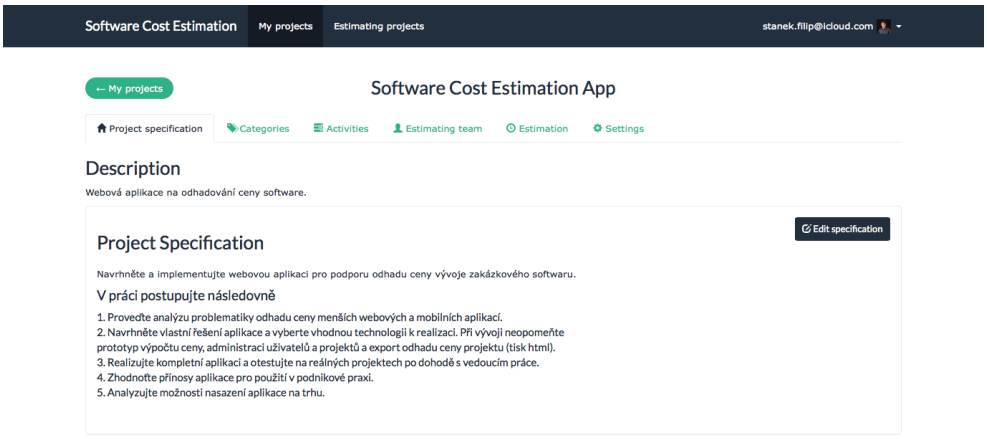
Obrázek A.1: Seznam vlastních projektů s filtrací



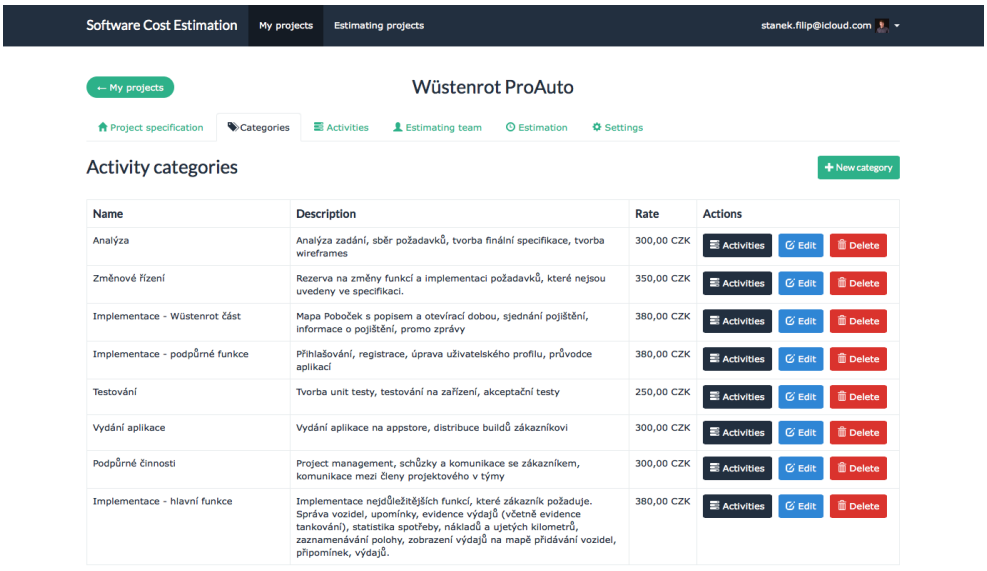
Obrázek A.2: Seznam aktivit pro vybranou kategorii



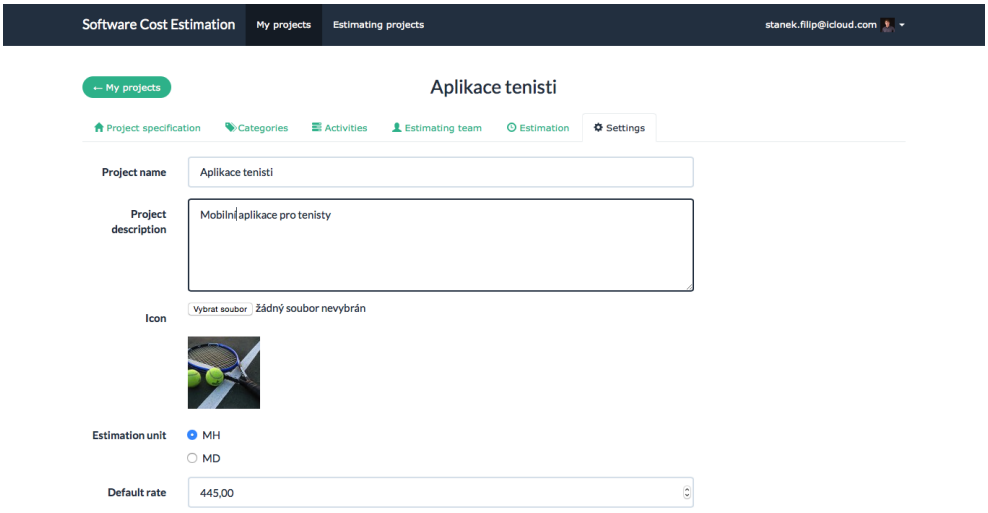
Obrázek A.3: Celkový odhad projektu ProAuto



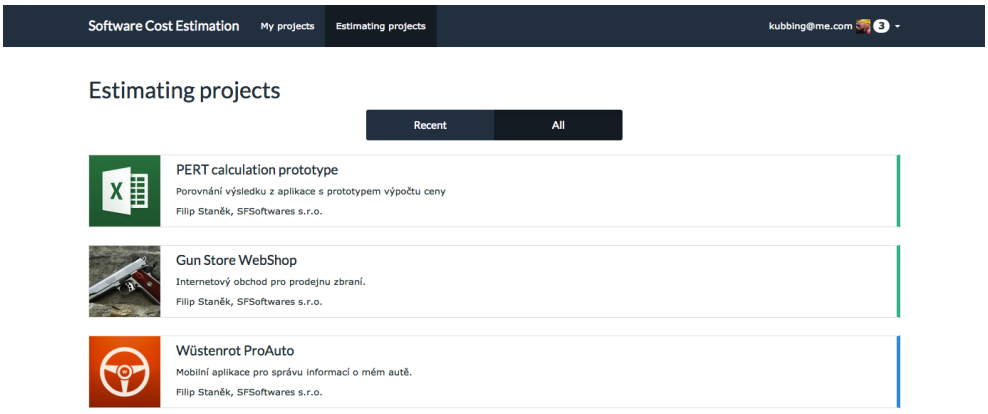
Obrázek A.4: Projektová specifikace



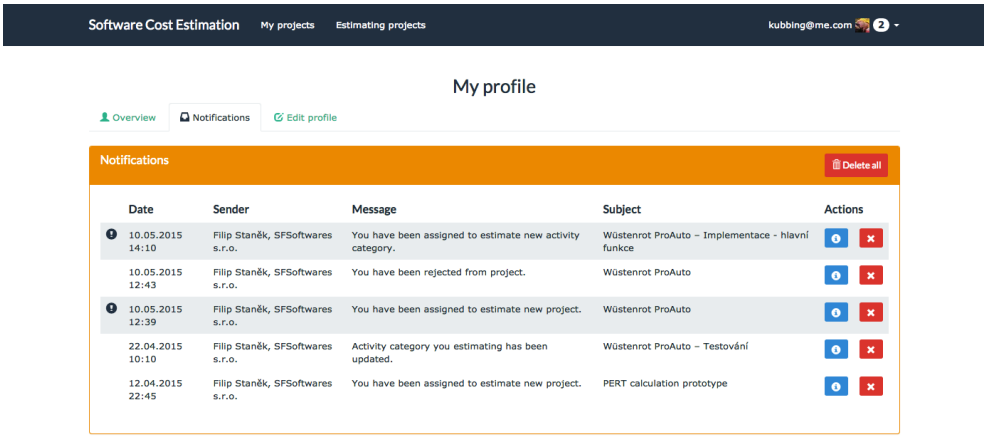
Obrázek A.5: Kategorie projektových aktivit



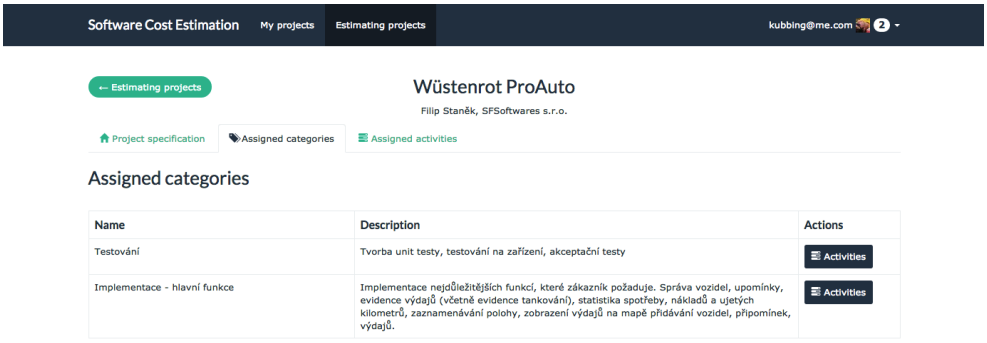
Obrázek A.6: Nastavení projektu



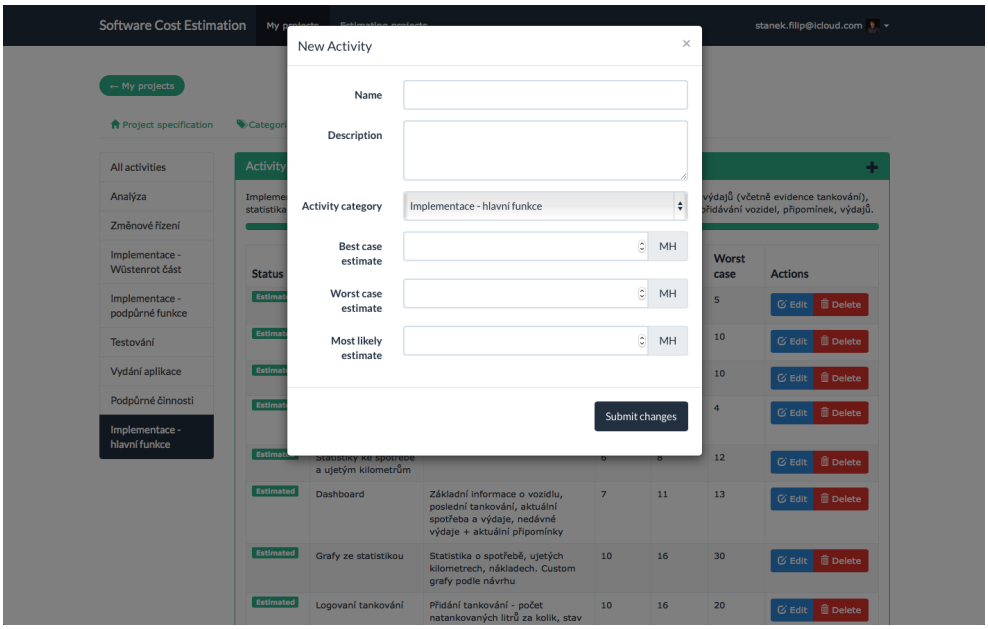
Obrázek A.7: Seznam odhadovaných projektů



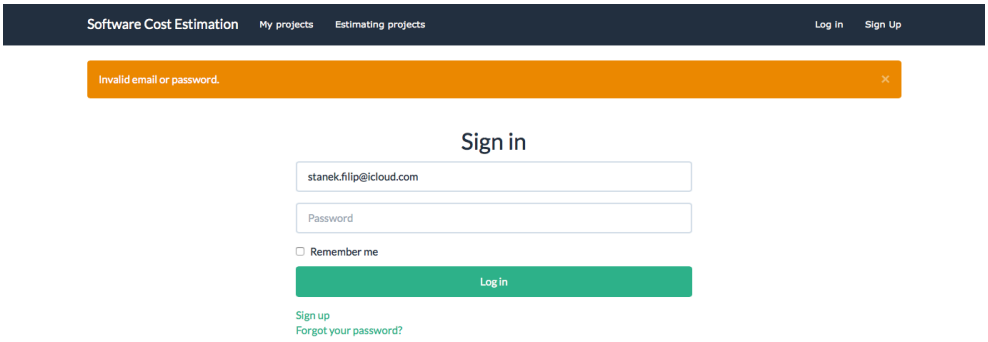
Obrázek A.8: Seznam notifikací



Obrázek A.9: Zobrazení přidělených kategorii na odhadovaném projektu



Obrázek A.10: Formulář na vytvoření nové aktivity



Obrázek A.11: Přihlašovací obrazovka

Příloha B

Export odhadu z projektu ProAuto

Project overview

Estimated effort:	**319,43 MH – 354,57 MH**
Estimated cost:	**101 740,41 CZK – 131 076,26 CZK**
Confidence level:	98 %
Estimates coverage:	99 %

Expected duration:	**337 MH**	Estimated activities:	**53**
Expected cost:	**116 408,33 CZK**	Incomplete activities:	**0**

Overview of categories

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Analýza	19,5	27,8	41,5	28,67	8 600,00 CZK
Implementace - podpůrné funkce	46,5	67,5	100	69,42	26 378,33 CZK
Implementace - Wüstenrot část	21	31	47	32	12 160,00 CZK
Implementace - hlavní funkce	68	99,5	143	101,5	38 570,00 CZK
Podpůrné činnosti	21	26,5	39	27,67	8 300,00 CZK
Testování	31	43	66	44,83	11 208,33 CZK
Vydání aplikace	4,5	6,5	9	6,58	1 975,00 CZK
Změnové řízení	16	26	38	26,33	9 216,67 CZK
Total	227,5	327,8	483,5	337	116 408,33 CZK

Overview of activities

Analýza

Description: Analýza zadání, sběr požadavků, tvorba finální specifikace, tvorba wireframes

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Tvorba finální kalkulace	1,5	2	3	2,08	625,00 CZK
Analýza použitelných technologii	1	1,3	2	1,33	400,00 CZK
Analýza možnosti řešení	4	6	7	5,83	1 750,00 CZK
Tvorba cenové nabídky	1	1,5	2	1,5	450,00 CZK
Tvorba odhadu	2	2,5	4	2,67	800,00 CZK
Tvorba specifikace požadavků	3	5,5	8	5,5	1 650,00 CZK
Sběr požadavků	3	4	7	4,33	1 300,00 CZK
Tvorba protypu wireframe	4	5	8,5	5,42	1 625,00 CZK
Total	19,5	27,8	41,5	28,67	8 600,00 CZK

Implementace - podpůrné funkce

Description: Přihlašování, registrace, úprava uživatelského profilu, průvodce aplikací

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Vytvoření modelových tříd pro data ze serveru	10	12	15	12,17	4 623,33 CZK
Startup Service	6	8	10	8	3 040,00 CZK
Synchronizace dat	6	10	15	10,17	3 863,33 CZK

Navigační prvky v aplikaci	6	8	12	8,33	3 166,67 CZK
Editace profilu uživatele	1,5	2	4	2,25	855,00 CZK
Přihlašovací obrazovka	2	3	5	3,17	1 203,33 CZK
Registrace uživatele	1	2	4	2,17	823,33 CZK
Příprava mechanismů k synchronizaci	2	4	5	3,83	1 456,67 CZK
Vytvoření ApiWrapperu	5	6	8	6,17	2 343,33 CZK
Návrh a příprava databáze	1,5	3,5	6	3,58	1 361,67 CZK
Přihlášení přes facebook	2	4	6	4	1 520,00 CZK
Přihlášení - standardní	1,5	2	4	2,25	855,00 CZK
Wizard	2	3	6	3,33	1 266,67 CZK
Total	46,5	67,5	100	69,42	26 378,33 CZK

Implementace - Wüstenrot část

Description: Mapa Poboček s popisem a otevírací dobou, sjednání pojištění, informace o pojištění, promo zprávy

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Promo zprávy	5	7	10	7,17	2 723,33 CZK
Seznam poboček	2	3	5	3,17	1 203,33 CZK
Detail pobočky	3	4	6	4,17	1 583,33 CZK
Informace o pojištění	1	2,5	5	2,67	1 013,33 CZK
Callback formulář	3	4	5	4	1 520,00 CZK
Mapa poboček	5	7,5	11	7,67	2 913,33 CZK
Splash promo	2	3	5	3,17	1 203,33 CZK
Total	21	31	47	32	12 160,00 CZK

Implementace - hlavní funkce

Description: Implementace nejdůležitějších funkcí, které zákazník požaduje. Správa vozidel, upomínky, evidence výdajů (včetně evidence tankování), statistika spotřeby, nákladů a ujetých kilometrů, zaznamenávání polohy, zobrazení výdajů na mapě přidávání vozidel, připomínek, výdajů.

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Úprava a detail vozidla	3	4	5	4	1 520,00 CZK
Tabulky položek s filtrací	5	6	10	6,5	2 470,00 CZK
Mapa s výdaji	4	6	10	6,33	2 406,67 CZK
Notifikace uživatele v čase připomínky	2	2,5	4	2,67	1 013,33 CZK
Statistiky ke spotřebě a ujetým kilometrům	6	8	12	8,33	3 166,67 CZK
Dashboard	7	11	13	10,67	4 053,33 CZK
Grafy ze statistikou	10	16	30	17,33	6 586,67 CZK
Logovaní tankování	10	16	20	15,67	5 953,33 CZK
Výdaje	12	16	20	16	6 080,00 CZK
Přidání vozidla	4	7	9	6,83	2 596,67 CZK
Připomínky	5	7	10	7,17	2 723,33 CZK
Total	68	99,5	143	101,5	38 570,00 CZK

Podpůrné činnosti

Description: Project management, schůzky a komunikace se zákazníkem, komunikace mezi členy projektového v týmy

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Komunikace mezi členy týmu	3	3,5	5	3,67	1 100,00 CZK
Schůzky se zákazníkem	6	6	8	6,33	1 900,00 CZK
Rozdělování úkolů	2	4	6	4	1 200,00 CZK
Komunikace se zákazníkem	10	13	20	13,67	4 100,00 CZK
Total	21	26,5	39	27,67	8 300,00 CZK

Testování

Description: Tvorba unit testy, testování na zařízení, akceptační testy

Estimator: Jakub Hladík, Hornet

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Unit testy komunikace s backendem	3	4	6	4,17	1 041,67 CZK
Oprava chyb	3	10	25	11,33	2 833,33 CZK
Testování aplikace na zařízení	15	17	20	17,17	4 291,67 CZK
Unit testy klíčových funkcí	10	12	15	12,17	3 041,67 CZK
Total	31	43	66	44,83	11 208,33 CZK

Vydání aplikace

Description: Vydání aplikace na appstore, distribuce buildů zákazníkovi

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Tvorba popisu aplikace do AppStoru	1,5	2	3	2,08	625,00 CZK
Distribuce buildu na testflight	2	3	4	3	900,00 CZK
Zaslání aplikace na appstore	1	1,5	2	1,5	450,00 CZK
Total	4,5	6,5	9	6,58	1 975,00 CZK

Změnové řízení

Description: Rezerva na změny funkcí a implementaci požadavků, které nejsou uvedeny ve specifikaci.

Name	Best case (MH)	Most likely (MH)	Worst case (MH)	Expected (MH)	Estimated Cost
Rezerva na změny vzniklé odlišným pochopením zadání	2	3	6	3,33	1 166,67 CZK
Rezerva na přidání funkcí	4	8	12	8	2 800,00 CZK
Rezerva na změny od zákazníka	10	15	20	15	5 250,00 CZK
Total	16	26	38	26,33	9 216,67 CZK

Příloha C

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

CRUD Create Read Update Delete

DRY Don't repeat yourself code

ERB Embedded Ruby

MD Man-day – člověkoden

MH Man-hour – člověkohodina

MVC Model View Controller

OOP Objektově orientované programování

PaaS Platform as a Service

PERT Program evaluation and review technique

REST Representational State Transfer

URL Uniform Resource Locator

WBS Work Breakdown Structure

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `software-cost-estimation-app` ..... zdrojové kódy aplikace
  - `test_database.dump` ..... export testovací databáze
  - `attachments`
    - `PERT_prototype.xlsx` ..... prototyp výpočtu ceny
    - `ProAuto_project_output.pdf` ..... výstup z testovacího projektu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Staněk_Filip_2015.pdf` ..... text práce ve formátu PDF