Masarykova univerzita
Fakulta informatiky



IS pro správu půjček

Bakalářská práce

Martin Molnar

Brno, 2012



Masarykova univerzita
Fakulta informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Datum: 25.4.2012

Student: Martin Molnar

Program: Aplikovaná Informatika

Obor: Aplikovaná Informatika

Vedoucí práce: Mgr. Jaroslav Bayer

Název práce: IS pro správu půjček

Zadání:

Cílem bakalářské práce je analýza a implementace informačního systému pro správu půjček ve spolupráci s externím zadavatelem (společností). Dále bude práce obsahovat porovnání současných agilních metodik vývoje softwaru určených pro jednotlivce a výběr vhodné metodiky, podle které bude projekt vypracován. Výstupem práce bude kromě uceleného textu také funkční implementace informačního systému, která bude otestována a posléze nasazena v zadávající společnosti.

Základní literatura:

Alistair Cockburn. **Crystal Clear: a human-powered methodology for small teams.** Agile software development series. Addison-Wesley, 2005. ISBN: 0201699478

Matthew MacDonald. **ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně.** Encyklopedie Zoner Press. Zoner Press, 2011. ISBN: 978-80-7413-131-8

Yani Dzhurov, Iva Krasteva, and Sylvia Ilieva. **Personal Extreme Programming – An Agile Process for Autonomous Developers.** Faculty of Mathematics and Informatics, Sofia University, 2009.
URL: https://research.uni-sofia.bg/bitstream/10506/647/1/S3T2009_37_YDzhurov_IKrasteva_SIlieva.pdf

Souhlas se zadáním (podpis, datum)

Molnar, 25.4.2012
student(ka)

vedoucí
bakalářské práce

vedoucí
odpovědného pracoviště

KTP
KPGD
KPSK
KIT
* Hodící se označte!

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Martin Molnar

Vedoucí práce: Mgr. Jaroslav Bayer

Poděkování

Rád bych poděkoval svému vedoucímu práce Mgr. Jaroslavu Bayerovi a konzultantovi Mgr. Janu Gérykovi za vstřícnost a odbornou pomoc poskytnutou během psaní této bakalářské práce. Děkuji také zadavateli práce Bc. Jakubu Vojtovi za nápad a úzkou spolupráci při tvorbě praktické části práce. Za podporu v průběhu celého mého studia děkuji svým rodičům.

Shrnutí

První část této práce shrnuje současné možnosti vývoje software pomocí agilních metodik určených pro jednoho vývojáře. Zajímavé metodiky jsou krátce popsány a vzájemně porovnány.

Ve druhé části je provedena analýza a návrh informačního systému pro správu půjček s použitím zvolené metodiky. Dále je zde uvedena ukázka části vývoje a popsány zajímavé části implementace.

Klíčová slova

Agilní metodiky, Scrum, Extrémní programování, nebankovní půjčky, informační systém, ASP.NET, MVC

Obsah

1 **Úvod** 4
1.1 *Motivace* 4
1.2 *Struktura práce* 5
2 **Výhody agilních metodik** 6
3 **Agilní metodiky vhodné pro jednotlivce** 8
3.1 *Rodina Crystal metodik* 8
3.2 *Personal Kanban* 9
3.3 *Personal Extreme Programming* 9
3.4 *Personal Scrum* 10
3.5 *Zvolená metodika* 10
4 **Technologie** 11
4.1 *Zvolená platforma* 11
4.2 *Alternativy* 12
4.3 *Použitá řešení* 14
4.3.1 Klient 14
4.3.2 Server 15
4.3.3 Databáze 17
4.3.4 Nástroje 18
5 **Návrh** 19
5.1 *Zadání* 19
5.2 *Úvodní analýza* 19
5.3 *Use Cases a User Stories* 20
5.4 *Identifikované role* 21
5.4.1 Klient 21
5.4.2 Zaměstnanec 24
5.4.3 Sekretářka 25
5.4.4 Zprostředkovatel 26
5.4.5 Vymahač 27
5.5 *Deployment Diagram* 27

5.6 *Product Backlog* 28
5.7 *Sprint Backlog* 30
5.8 *Burn Down Chart* 31
5.9 *ASP.NET a MVC* 32
6 **Implementace** 35
6.1 *Správa uživatelů* 35
6.2 *Entity Framework a Code First* 36
6.3 *Ukázka vývoje* 37
6.3.1 První sprint 37
6.3.2 Druhý sprint 38
6.4 *Time Management* 38
6.5 *Testování* 39
7 **Přílohy** 40
8 **Závěr** 41

Seznam obrázků

2.1 Rozdíl mezi tradičními a agilními metodikami [14] *7*

5.1 Ukázka odlehčeného Use Case diagramu *22*

5.2 Deployment Diagram *29*

5.3 Ukázka Product Backlogu, z důvodu nedostatku místa bez schvalovacích kritérií *30*

5.4 Ukázka Sprint Backlogu *31*

5.5 Ukázka Burn down chart pro první sprint *32*

5.6 Návrhový vzor MVC, převzato z `http://msdn.microsoft.com/en-us/library/ff649643.aspx` *34*

1 Úvod

1.1 Motivace

Motivací pro vznik této práce je současný stav nabídky software určeného malým a středním firmám podnikajícím v České republice. Práce se zaměřuje na firmy pohybující se v oblasti poskytování a zprostředkování nebankovních půjček. Celosvětová finanční a ekonomická krize, která se nevyhla ani naší zemi, v posledních letech vytváří stále lepší podmínky pro vznik tohoto typu společností. Lidé se stále častěji dostávají do finančních problémů, ale banky začínají být naopak mnohem opatrnější a více si vybírají své klienty.

Pro svůj úspěšný provoz ale tyto firmy potřebují informační systém, který by jim umožnil snadno a efektivně spravovat a analyzovat velké množství specifických dat, které jejich podnikání generuje. Nemohou si však dovolit systémy navrhované na míru vzhledem k velké finanční zátěži a zároveň jsou nedostatečnou nabídkou nuceny spokojit se s používáním obecnějších systémů, které nejsou přímo určeny pro uspokojování jejich potřeb. Typickým příkladem je používání nejrůznějších skladových nebo účetních řešení pro evidenci finančních toků, případně využívání nejrůznějších tabulkových procesorů (velmi rozšířený je např. MS Excel[1]). To vede k velkému množství problémů. Správa většího objemu dat tímto způsobem je komplikovaná a neefektivní. Systémy poskytují nedostatečné nebo žádné možnosti pro analýzu získaných dat. Sdílení některých vybraných dat s klienty je složité a přidělává práci jak firmě, tak klientům a navíc neposkytuje dostatečné uživatelské pohodlí.

Cílem této práce je návrh a implementace informačního systému pro správu půjček. Systém bude určen právě malým a středním firmám, které v této oblasti podnikají, a kromě interní správy dat si

1. http://office.microsoft.com/cs-cz/excel/

klade za cíl zpřístupnit část těchto dat jejich klientům v rámci rozšíření nabídky služeb. Právě otevřenost systému i pro použití zákazníky je věc, která jej činí ojedinělým. Drtivá většina současných dostupných řešení pro zvolenou cílovou skupinu je určena pouze pro interní správu dat, případně jejich grafickou vizualizaci.

Vzhledem k tomu, že podobný systém se na českém trhu v době psaní bakalářské práce nenachází a zadavatel nemá přesnou představu, jak by měl systém vlastně vypadat, aby dokázal efektivně plnit svou funkci, byl namísto některé z tradičních metodik vývoje software zvolen agilní přístup. Kromě samotného návrhu je tedy součástí práce i porovnání současných možností agilních metodik vývoje v jednočlenném týmu.

1.2 Struktura práce

První část, která je rozdělena na druhou a třetí kapitolu, shrnuje problematiku vývoje software s použitím agilních metodik určených jednomu vývojáři, důležité metodiky porovnává a obhajuje vhodnost použití agilního přístupu a zvolené metodiky pro cílový projekt.

Ve čtvrté kapitole jsou krátce porovnány současné technologie určené pro vývoj webových aplikací a popsána důležitá řešení a nástroje použité pro vývoj aplikace.

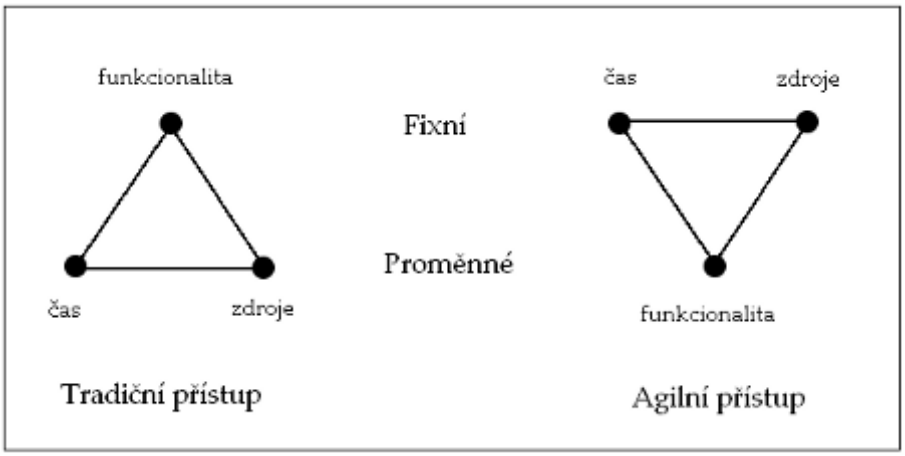
Pátá kapitola se zaobírá návrhem aplikace a v šesté jsou rozebrány důležité body implementace. Kromě toho obsahuje šestá kapitola i ukázku samotného vývoje a shrnutí poznatků získaných v rámci první fáze testovacího procesu.

2 Výhody agilních metodik

V této kapitole je v krátkosti vysvětlena nutnost použití ucelené metodiky pro vývoj software, objasněn rozdíl mezi běžně používanými metodikami a zdůvodněno, proč je pro účely tohoto projektu vhodné použití agilního přístupu.

Na začátku každého projektu si vývojář musí položit otázku jaké procesy a artefakty chce během vývoje používat a proč. Velké množství jich ale stále opakuje chybu, kdy se jako první pustí do vývoje aplikace. Návrh dělají až v průběhu, odhad práce řeší pomocí stanovování vágních cílových dat a velmi brzy zjistí, že se v projektu ztrácí, nevědí zda je možné provést různé úpravy, příliš se zabývají detaily a ztrácí celkový přehled. Případně zjistí, že projekt by se podle plánu měl již pomalu chýlit ke konci a práce je teprve v polovině. Tyto problémy je možné řešit pomocí použití ucelených metodik, které dávají celému vývoji určitý rámec.

Metodiky se dají podle jejich přístupu rozdělit v zásadě na tradiční neboli rigorózní a agilní [14, kapitola 2]. Druhé jmenované jsou mladší, začaly vznikat v průběhu 90. let minulého století v reakci na těžkopádnost a nevhodnost tradičních metodik pro určité typy projektů. Jejich označení agilní však vzniklo teprve po vytvoření agilního manifestu [4] v roce 2001. Zatímco tradiční metodiky jsou označovány jako těžké vzhledem ke své obsáhlosti, podrobnosti a důrazu na plánování a řízení, tak agilní jsou naopak nazývány lehkými a soustředí se na samotný produkt, čemuž podřizují všechno ostatní. To se promítá do jejich chápání fixních a proměnných součástí projektu. Pro tradiční metodiky je fixní cílová funkcionalita, kdežto čas a zdroje jsou proměnné. Agilní metodiky jako fixní berou čas a zdroje, ale funkcionalitu přizpůsobují podle možností. Dobře je to ilustrováno na následujícím obrázku.



Obrázek 2.1: Rozdíl mezi tradičními a agilními metodikami [14]

Ačkoliv by se na první pohled mohlo zdát, že tradiční metodiky vývoje jsou díky popularitě těch agilních na ústupu, není tomu tak. Pro určité typy projektů mohou být lepší volbou. Jedná se především o větší informační systémy, u kterých je předem známa požadovaná funkcionalita, dále vývoj kritických aplikací, u kterých je potřeba udělat komplexní analýzu atp.

Prototyp systému, který bude vytvořen v rámci této práce, je v základu relativně malý a vzhledem k vágním požadavkům na výsledný produkt z pohledu zákazníka se bude zřejmě v průběhu vývoje rychle a zásadně měnit. Kromě toho jsou v tomto případě fixní čas i zdroje. Lepší volbou bude tedy některá z agilních metodik.

3 Agilní metodiky vhodné pro jednotlivce

Obsahem třetí kapitoly je shrnutí současné situace v oblasti agilních metodik vývoje software určených pro jednotlivce a porovnání jednotlivých metodik.

V případě, že se vývojář rozhodne vyvíjet svojí aplikaci s použitím, v poslední době moderního, agilního přístupu, brzy zjistí, že současný stav agilních metodik pro jednoho vývojáře není příliš dobrý. Neexistuje totiž žádná běžně používaná zavedená metodika. Nejčastějším doporučením je přizpůsobit si některou z metodik určených pro malé týmy. Je to způsobeno faktem, že tato oblast zřejmě není komerčně zajímavá a vývojáři sami o sobě nemají příliš rádi svazování pravidly nebo práci navíc a raději se rovnou věnují samotnému programování.

Této neutěšené situace si v posledních letech začala všímat i akademická sféra a vzniká čím dál tím více prací, které se zabývají upravováním týmových metodik pro potřeby jednotlivce.

3.1 Rodina Crystal metodik

Crystal metodiky je souhrnné označení pro sadu metodik vyvinutých Alistairem Cocburnem [8]. Dají se rozdělit podle barev, kde každá z barev označuje potřebu odolnosti projektu proti selhání a počet lidí. Nejodlehčenější varianta vhodná pro vývoj v týmu 1 až 6 lidí se nazývá Crystal Clear [9]. Je vytvořena velmi volně a týmy jsou podporovány ve výběru pouze těch vlastností, které jim vyhovují a křížení s ostatními metodikami. Problémem použití pro jednočlenný tým je ale samotná filozofie metodiky. Soustředí se především na interakci mezi lidmi, vývoj formou hry a využití sil vzájemné komunikace, což znamená, že ve výsledku bude muset jednotlivec vynechat velkou část jejích vlastností. Dalším problémem

je volnost celého konceptu. Pro potřeby samostatného vývojáře je vhodnější kompaktní, hotová metodika, nad kterou nemusí příliš přemýšlet, a může se naplno věnovat projektu.

3.2 Personal Kanban

Druhou zavedenou agilní metodikou je upravená verze Kanbanu nazvaná Personal Kanban [5]. Původní Kanban vytvořil zaměstnanec Toyoty Taiichi Ohno pro potřeby řízení dodávek zboží. Později z toho vznikla obecná agilní metodika vývoje, která klade důraz především na vývoj just-in-time, což znamená, že vývojáři nebudou zbytečně zahlcováni a mohou se naplno věnovat dekomponovaným malým částem projektu. Personal Kanban je určen přímo jednotlivcům nebo malým týmům. Soustředí se na vizualizaci a kontrolu práce. Hlavními argumenty proti jeho použití je opět přílišná volnost a především obecnost, protože není přímo určen pro vývoj software. Problémem je také nedostatek zdarma dostupných materiálů.

3.3 Personal Extreme Programming

Jedna z prací, která se tímto problémem zabývá je od kolektivu autorů ze Sofijské Univerzity [11]. Navrhují odlehčení metodiky Personal Software Process (dále jen PSP) [12] a její úpravu pomocí některých postupů využívaných v Extreme Programming (dále jen XP) [3]. Tato modifikace je již vcelku použitelnou variantou a někteří vývojáři by její nasazení ve svých projektech mohli zvážit, nicméně vyžaduje znalost obsáhlého PSP a XP a vzhledem ke kratšímu rozsahu práce, která obsahuje spíše návrh, jak by se dalo postupovat, než samotnou metodiku, je nutné ji před použitím dotvořit.

3.4 Personal Scrum

Tuto metodiku vyvinul pro potřeby své diplomové práce v průběhu roku John Pruitt. Vznikla odlehčením a úpravou v poslední době velmi populární týmové metodiky Scrum [15]. Je velmi dobře popsána, protože kromě samotné práce autor v průběhu vytvářel i blog[1], kde popisoval jakým způsobem metodiku používá, kterým problémům čelí a následně jak a proč ji upravuje. Personal Scrum je tak velmi dobrou volbou pro samotné vývojáře a mohl by být vhodným kandidátem i pro tento projekt.

3.5 Zvolená metodika

Česky mluvící vývojář má však ještě lepší možnost. Ve stejné době jako John Pruitt vytvořil svou metodiku i Ondřej Bechný [2]. Je stejně jako Personal Scrum z větší části postavena na Scrumu. Přebírá především jeho pojetí časového rámce a iterací, které jsou reprezentovány tzv. sprinty. Sprinty jsou pro Scrum základní jednotkou vývoje. Jedná se o časové úseky, které mají pevně stanovenou délku. Ta by se měla pohybovat mezi jedním až čtyřmi týdny [15]. Dále shodně používá většinu artefaktů a část procesů. Výsledek obohacuje o některé prvky XP. Kromě samotné metodiky a ukázkového projektu obsahuje jeho práce i řadu doporučení a nástrojů, které je při vývoji vhodné použít. Důvody pro zvolení této metodiky jsou tedy především její komplexnost, ucelenost a demonstrace aplikace metodiky na příkladu. Vedlejšími faktory pro její výběr byl i osobní pozitivní vztah ke Scrumu a obohacení o prvky XP.

1. http://blog.jgpruitt.com/2011/04/10/personal-scrum/

4 Technologie

Pro vývoj každé webové aplikace je klíčové zvolit správné nástroje. Obsahem této kapitoly je tedy shrnutí současných možností vývoje webových aplikací. Dále je zde obhájen výběr platformy a krátký popis všech důležitých technologií, které byly pro vývoj aplikace použity.

Vzhledem k tomu, že Internet dosáhl dostatečné penetrace mezi běžné uživatele teprve přibližně na přelomu tisíciletí, došlo v oblasti webových technologií v předchozím desetiletí k prudkému rozvoji. Vznikla celá řada platforem a nástrojů. V současné době se situace již trochu ustálila a i když je několik zásadních inovací teprve ve stadiu příprav, dá se říci, že v současné době existují tři hlavní platformy.

4.1 Zvolená platforma

Pro potřeby této aplikace bylo zvoleno komplexní řešení, vytvořené firmou Microsoft. Hlavní předností je ucelenost a provázanost celé řady produktů. Pokud si vývojář zvolí tuto variantu může očekávat, že dostane všechny nástroje, které potřebuje pro vývoj aplikace a nemusí ztrácet čas výběrem jednotlivých součástí. V případě specifických požadavků je nahrazení některých komponent jinými možné, ale ztrácí se tím jedna ze stěžejních výhod. Na toto omezení se dá jistě nahlížet i negativně, kdy se dá říci, že Microsoft se snaží uzavřít vývojáře ve světě svých technologií, ale protože nabízené možnosti v tomto případě plně dostačují požadavkům, nebyl problém se této svobody vzdát. Další výhodou je rozsáhlá a kvalitní dokumentace[1], zahrnující i návody formou textu nebo videa. Microsoft

1. Obecně na: http://msdn.microsoft.com/cs-cz/ a pro webové aplikace zde: http://www.asp.net/

nabízí špičkové vývojové prostředí (Integrated development environment, dále jen IDE) s implementací systému automatického doplňování textu a nápovědy IntelliSense[2]. Všechny potřebné nástroje nabízí Microsoft zdarma i pro komerční využití ve verzích Express, které jsou sice ochuzené o některé nejpokročilejší funkce, určené k vývoji velkých projektů v mnohočlenných týmech, ale pro vývoj malých a středně velkých aplikací plně postačují. Pro čistě studijní účely nabízí Microsoft možnost používat i plné verze většiny produktů skrze program DreamSpark[3]. Co se týče nákladů na samotný provoz aplikace, je možné celý projekt hostovat pomocí Express nástrojů, ale lepší variantou je využít služeb nějakého webhostingu, který cenu sníží pomocí multilicencí a rozprostřením mezi velké množství zákazníků. V případě, že by aplikace nakonec postupně narostla do větších rozměrů, je možné i zakoupení potřebných nástrojů v plných verzích, protože v poměru k velikosti projektu, je pak tato částka zanedbatelná a pravděpodobně se vrátí skrze nižší náklady na další rozvoj a udržování aplikace.

4.2 Alternativy

V kontextu této práce není možné dostatečně prozkoumat alternativy pro provedení kvalifikovaného porovnání, ale přesto zde alespoň v krátkosti uvedu dva hlavní konkurenty. Prvním z nich je ekosystém, který vznikl pod záštitou Sun Microsystems a v současné době je rozvíjen firmou Oracle, která Sun odkoupila. Stojí především na jazyku Java a technologiích a frameworcích s ním spojených. Je využíván především velkými společnostmi. Hlavní výhody oproti řešení od Microsoftu jsou multiplatformita, otevřenost, nižší náklady na licence a velké zapojení komunity do vývoje samotných

2. http://msdn.microsoft.com/cs-cz/library/hcw1s69b.aspx
3. https://www.dreamspark.com/

nástrojů. Žádná z těchto výhod však není pro tento projekt zajímavá. Protože zamýšlené aplikaci plně stačí nástroje poskytované Microsoftem, není třeba využívat specifické projekty, které poskytuje komunita. Navíc se jedná o webovou aplikaci, která bude spuštěna pouze na jedné, předem vybrané platformě, takže přenositelnost není v tomto případě relevantní. Naopak se zde projevují negativa této filozofie. Především je to roztříštěnost a nekompatibilita, které nutí programátora soustředit se na výběr technologií a zajišťování jejich vzájemné kompatibility, místo na výslednou funkcionalitu aplikace. Základní prvky začínají být zastaralé a zpomalují další vývoj. To kontrastuje s rychlým rozvojem platformy od Microsoftu, který implementoval to, co se v průběhu let osvědčilo a na solidních základech rychle rozvíjí své vlastní nadstavby.

Druhou možností je volba, v současné době nejpoužívanějšího[4], programovacího jazyka PHP[5]. PHP samo o sobě je však poměrně slabým nástrojem a v případě jeho použití na větší aplikaci je třeba rozšířit jeho možnosti pomocí některého frameworku. Oblíbené jsou například Zend[6] nebo CodeIgniter[7], ale existuje celá řada dalších možností. PHP má velké množství slabin, které je třeba před jeho volbou zvážit. Je rozšířeno především díky své jednoduchosti, ale samotný jazyk byl původně navržen pro potřeby tvorby osobních stránek (odtud jeho původní název – Personal Home Page) a nepodporoval objektově orientované programování (dále jen OOP). I když byla tato možnost později přidána, nemůže se rovnat současným moderním jazykům, které byly pro OOP navrženy od začátku. PHP je čistě interpretovaný jazyk a díky tomu je také bohužel poměrně pomalý. Existují kompilátory PHP, např. HipHop for PHP[8] vytvo-

4. http://trends.builtwith.com/framework
5. http://php.net/
6. http://www.zend.com/en/
7. http://codeigniter.com/
8. https://github.com/facebook/hiphop-php/wiki/

řený vývojáři Facebooku, který kompiluje PHP kód do vysoce optimalizovaného C++, ale to je pro potřeby tohoto projektu zbytečně komplikované. V kontrastu proti dvěma výše uvedených platformám také neexistují dostatečně kvalitní nástroje, které by byly určeny přímo pro vývoj PHP aplikací.

Vedle PHP by bylo vhodné zmínit ještě dvě, v poslední době stále populárnější, platformy. Jsou jimi jazyk Python[9] s frameworkem Django[10] a jazyk Ruby[11] s frameworkem Ruby-on-rails[12]. Obě ale zatím trpí nedostatečným rozšířením, malou komunitou a horší dokumentací.

4.3 Použitá řešení

Každou webovou aplikaci můžeme v podstatě rozdělit na tři samostatné části. Vizuální reprezentace dat a skripty prováděné na straně klienta, zpracování dat pomocí skriptů prováděných na serveru a uložení, popřípadě vyhledávání nad daty v podobě databázového serveru nebo jeho alternativy.

4.3.1 Klient

Co se týče technologií, které budou použity pro klientskou část aplikace, není zde moc možností volby. Na reprezentaci struktury dat je použit HyperText Markup Language (dále jen HTML) v jeho poslední verzi 4.01[13]. Alternativy jsou pouze dvě. XHTML (eXtensible HTML), který se ukázal být slepou větví, vývoj verze 2.0 byl posléze oficiálně zastaven a v případě, že se v projektu nebudou používat XML nástroje (např. XSLT transformace), jsou jeho přínosy

9. http://www.python.org/
10. https://www.djangoproject.com/
11. http://www.ruby-lang.org/en/
12. http://rubyonrails.org/
13. Specifikace zde: http://www.w3.org/TR/1999/REC-html401-19991224/

diskutabilní. HTML5[14], který by měl spojovat oba přístupy a přinést mnoho nových věcí, je zatím pouze ve stádiu návrhu a finální verze standardu je odhadována na rok 2022 [13], takže jeho nasazení ve firemním prostředí není vhodné. Některé jeho součásti jsou však již dostatečně dobře implemetovány ve všech rozšířených internetových prohlížečích, takže např. element canvas, sloužící pro vykreslování 2D grafiky, by mohl být použit pro kreslení interaktivních grafů nebo podobnou vizualizaci dat.

Pro úpravu vzhledu jsou použity Cascading Style Sheets (dále jen CSS) v poslední verzi 2.1[15]. U CSS je situace obdobná jako u HTML. Nový standard CSS 3.0 existuje zatím pouze jako pracovní verze a jen tři jeho moduly byly zatím zveřejněny jako doporučení organizace World Wide Web Consortium (dále jen W3C). Tyto tři moduly mohou být tedy použity a případně ještě několik dalších vlastností, které jsou dostatečně podporovány.

Skripty jsou napsány v Javascriptu, což je v současné době jediný rozšířený skriptovací jazyk, který umějí internetové prohlížeče zpracovávat.

4.3.2 Server

Zatímco na straně klienta nemá vývojář prakticky možnost výběru, u technologií použitelných pro serverovou stranu aplikace je situace naprosto odlišná. Existuje celá plejáda jazyků a frameworků a velká spousta dalších je v současné době ve vývoji. Nicméně s ohledem na volbu platformy je většina technologií v tomto případě pevně daná.

14. Specifikace zde: http://www.w3.org/TR/2012/WD-html5-20120329/
15. Specifikace zde: http://www.w3.org/TR/2011/REC-CSS2-20110607/

Framework

Architektura celé aplikace je silně ovlivněna použitím frameworku ASP.NET MVC 3[16]. Tento dlouhý název v sobě skrývá kombinaci různých technologií, které jsou navzájem velmi provázány. Celý framework je postaven na kombinaci staršího serverového webového enginu Active Server Pages (dále jen ASP), ze kterého již obsahuje pouze některé pozůstatky, a frameworku .NET v poslední verzi 4. Myšlenka implementovat do ASP.NET návrhový vzor Model-View-Controller (dále jen MVC) vznikla teprve relativně nedávno a první verze vyšla v polovině roku 2009. Od té doby prošla rychlým vývojem, během něhož byly odstraněny prvotní nedostatky, a v současné době se jedná o jeden z nejmodernějších a nejkomplexnějších nástrojů pro vývoj webových aplikací. Více o ASP.NET a MVC v části 5.9.

Programovací jazyk

ASP.NET podporuje velké množství programovacích jazyků, ale největší důraz je samozřejmě kladen na programovací jazyky vyvinuté firmou Microsoft. Prvním z nich je, ze spíše historických důvodů, Visual Basic. Zvoleným a v současné době nejvíce podporovaným je C#. C# je poměrně mladým jazykem. První verze vyšla v roce 2001 a původní návrh vychází z nejrozšířenějších jazyků C++ a Java. Oproti nim se však více projevuje integrace více programovacích paradigmat. Kromě klasických dává mj. důraz i na funkcionální, deklarativní a event-driven přístup.

16. http://www.asp.net/mvc

Razor

ASP.NET nabízí podporu dvou zobrazovacích enginů. Použitým je novější Razor[17], který byl uvolněn jakožto součást poslední verze ASP.NET MVC. Razor využívá jednoduchý šablonovací značkovací jazyk, který umožňuje snadné oddělení dat od struktury šablony. S jeho pomocí je možné snadno generovat části HTML kódu.

Webový server

Pro úplnost by bylo vhodné doplnit, že celá aplikace běží na Internet Information Services (dále jen IIS) 7.5[18], což je webový server od společnosti Microsoft, a v současné době jediný server, na kterém je možné provozovat ASP.NET aplikace. Omezení na jedinou platformu již bylo diskutováno výše a bylo shledáno jako akceptovatelné.

4.3.3 Databáze

Databáze je jedním z prvků Microsoft platformy, který lze snadno vyměnit, a je možné použít i řešení, která nebyla vyvinuta přímo Microsoftem. Nicméně vzhledem k tomu, že Microsoft nabízí svůj databázový server MSSQL 2008[19] Express, který je pro cílovou aplikaci více než dostačující, zcela zdarma i pro komerční využití, není důvod ho nepoužít. Jediný problém, který v rámci vývoje nastal, vznikl při publikaci databáze na testovací server, kdy měl webhosting blíže nezjištěné potíže s provozem MSSQL Express a docházelo k opakovanému zamykání databáze, což znemožňovalo jak její použití, tak její odstranění a tedy i publikace dalších verzí. Vše se vyřešilo použitím nabízené plné verze MSSQL 2008 R2, která je s ur-

17. http://msdn.microsoft.com/en-us/library/gg549172%28v=vs.99%29.aspx
18. https://www.iis.net/
19. http://www.microsoft.com/cze/sqlserver2008/

čitými omezeními webhostingovou společností nabízena zdarma. Protože Express a plná R2 jsou samozřejmě navzájem kompatibilní, bylo rozhodnuto, že vývoj bude nadále probíhat s použitím Express edice a databáze bude nasazena na plnou verzi databázového serveru.

4.3.4 Nástroje

IDE

Nejdůležitějším nástrojem použitým při vývoji aplikace bylo IDE Visual Web Developer 2010 Express[20] (dále jen VWD). VWD je jednou z edic Microsoft Visual Studia a obsahuje kromě samotného IDE řadu dalších nástrojů a funkcí, které usnadňují vývoj webových aplikací především na platformě ASP.NET. Z těch nejdůležitějších je to například výše zmíněný IntelliSense nebo nástroj One-Click-Publish, který usnadňuje publikaci aplikace na cílový webový server.

Entity Framework

Pro vytvoření datové mezivrstvy mezi systémem a databází byl použit Entity Framework[21] (dále jen EF), což je object-relational mapping (dále jen ORM) framework určený pro ASP.NET. ORM slouží pro převod dat mezi vzájemně nekompatibilními typovými systémy a v tomto případě byl použit pro propojení s databází. Více o EF v části 6.2.

---

20. http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-web-developer-express
21. http://msdn.microsoft.com/en-us/library/bb399567

5 Návrh

Obsahem této kapitoly bude popis prvotní analýzy a návrhu systému. Jsou zde popsány všechny důležité diagramy a artefakty, které byly při návrhu použity a u každého je demonstrováno jeho použití. Kromě toho je zde uvedeno kompletní zadání projektu a identifikace rolí, které se systémem interagují.

5.1 Zadání

Zadavatel požaduje informační systém, určený malým a středním firmám, které se zabývají poskytováním nebankovních úvěrů a zprostředkováním půjček pro malé investory. Systém by měl umožňovat snadnou správu klientů a jejich dat. Měl by umožňovat přehledné sledování finančních toků uvnitř společnosti. Tyto toky představují především příjmy investic od klientů a evidenci půjček dlužníkům a jejich stav. Dále by měl umožnit klientům přístup k informacím, které o nich společnost schraňuje a k jejich osobním půjčkám, případně investicím. Systém by měl také umět přehledně zobrazit základní statistická data.

S přihlédnutím k tomu, že zadavatel do budoucna počítá s dalším rozvojem systému i nad rámec této práce, by měl být tento požadavek během návrhu a implementace zohledněn a systém by měl být připraven na další rozšíření funkcionality. Z tohoto důvodu obsahují některé oblasti návrhu neimplementované součásti.

5.2 Úvodní analýza

Hlavní náplní prvotní analýzy bylo pochopení základních požadavků klienta, které byly stručně shrnuty ve výše uvedeném zadání. Kromě toho bylo důležitou součástí i vytvoření představy o tom,

jaké typy uživatelů budou systém využívat a jaké jsou jejich hlavní požadavky. V tradičních metodikách vývoje se za tímto účelem vytváří Use Case diagram. Při agilním přístupu k problému se však objevuje několik komplikací, které činí použití tohoto prvku diskutabilním.

5.3 Use Cases a User Stories

Use Case diagram je velmi jednoduchá grafická reprezentace, která slouží pro zachycení různých způsobů interakce uživatele nebo externího systému s vlastním systémem. Používá k tomu symbolické kreslené postavičky, reprezentující uživatele, šipky, které reprezentují interakce a takzvané Use Cases. Use Case je poloformálně definovaná posloupnost kroků, která definuje jednotlivé interakce a jejich důsledky a okolnosti.

Naproti tomu základním prvkem většiny agilních metodik je User Story. User Story je jedna nebo několik málo vět, napsaných v běžném neformálním jazyce, kterému rozumí jak vývojář, tak zadavatel. Slouží k zachycení nějaké uživatelské potřeby a specifikuje, kdo jí chce naplnit a proč. Pro vyhodnocení toho, zda byla User Story úspěšně implementována, se používá neformální seznam požadavků, které musí implementace splňovat, aby byla uznána za kompletní. Obvykle bývá zapsána formou malých kartiček a například v metodice Scrum tyto User Stories vytvářejí artefakty Product Backlog a Sprint Backlog (tyto artefakty jsou detailněji popsány v částech 5.6 a 5.7).

Z výše uvedených definic by mělo být zřejmé, že Use Cases a User Stories, byť na první pohled zdánlivě plní stejnou funkci, se ve skutečnosti překrývají pouze částečně a není možné jednu z nich plně nahradit druhou a naopak. V komunitě agilních vývojářů však stále panují neshody, zda je možné, příp. vhodné, se bez jedné z nich

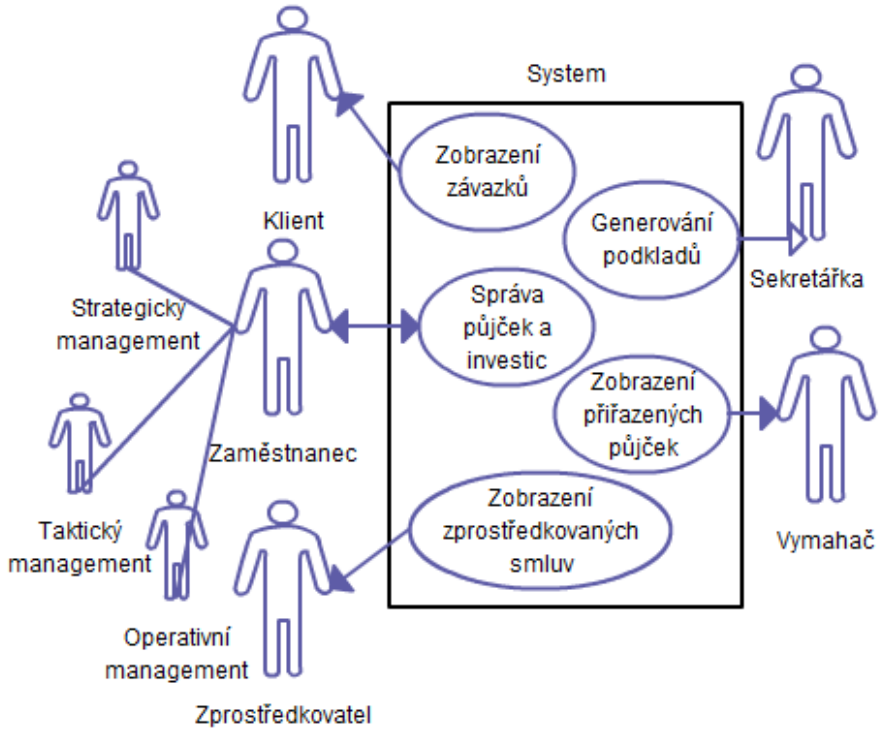
obejít. Velkým zastáncem Use Case je například Allistair Cockburn [10], který osobně velmi přispěl k rozvoji Use Case a jejich využití v Agilních metodikách vytvořením vlastní varianty [7], která je pro tyto účely vhodná. Nicméně i použití samotných User Stories má své zastánce [1] a najdou se samozřejmě i názory, že by měly být používány oba nástroje.

Po zvážení všech tří přístupů bylo rozhodnuto, že Use Cases jsou pro účely této aplikace a vývoj v jednočlenném týmu příliš komplexní. Jejich přínos je možné nahradit úzkou spoluprací se zadavatelem, kdy budou jednotlivé okolnosti a důsledky zjištěny formou domluvy, a nevzniká zde tedy potřeba je formálně zaznamenávat. Tím ovšem vyvstal problém, jakým způsobem zjistit a především přehledně graficky znázornit jednotlivé typy uživatelů. Identifikace uživatelů by šla provést v rámci vytváření jednotlivých User Stories, ale brzy se zjistilo, že zadavatel se začíná ztrácet v již definovaných rolích a docházelo k neustálým návratům a opravám. Jako vhodnější se ukázal opačný postup, kdy se nejprve identifikují jednotliví uživatelé a až podle nich se vytvoří odpovídající User Stories. K tomuto účelu byl vytvořen speciální diagram, který připomíná velmi zjednodušený Use Case, s tím rozdílem, že jsou zde zachyceny pouze hlavní potřeby daných uživatelů, které jsou jen orientační, nejsou dále specifikovány a vztahy jsou omezeny na interakce mezi systémem a uživatelem. Jsou tedy odstraněny vztahy jednotlivých Use Cases v rámci systému.

5.4 Identifikované role

5.4.1 Klient

Role klient reprezentuje dva základní typy uživatelů – dlužníka a investora. Hlavním cílem těchto uživatelů je získat díky systému přehled o svých osobních údajích, které o nich firma má, a kontrolovat



Obrázek 5.1: Ukázka odlehčeného Use Case diagramu

své půjčky, příp. investice. Tyto typy uživatelů se systémem pracují pouze pasivně. Využívají ho jen na získání informací, žádné údaje do systému nevkládají, ani nemají možnost zobrazená data data nějak modifikovat. Pokud si budou chtít vyžádat úpravu některých parametrů jejich závazků nebo osobních údajů, musí využít přímého kontaktu se zadavatelem. Toto opatření dává zadavateli plnou kontrolu nad současným stavem jeho klientů, což může být podle vyjádření zadavatele velmi důležité, zvláště pak u dlužníků.

Při definici této role se ukázalo, jak velký rozdíl v chápaní jednotlivých rolí může mezi zadavatelem a vývojářem nastat. Kromě

problému s modifikací dat, kdy by z pohledu vývojáře bylo lepší umožnit uživatelům alespoň částečnou interakci se systémem, se jádrem problému ukázal být spor o překrývání těchto dvou typů uživatelů. Zadavatel ze začátku trval na tom, že dlužník nikdy nebude investorem a naopak, protože žádný z jeho dlužníků u něj nemá investované peníze. Z pohledu aplikace je však mnohem flexibilnější umožnit uživateli mít v systému zaregistrované půjčky i investice a zároveň je rozdíl v obtížnosti implementací obou řešení zanedbatelný. Nakonec se ukázalo, že stačí otočit otázku. Z původní: „Může mít nějaký dlužník zájem o investování svých peněz?“, intuitivně (a chybně, viz posl. odstavec této části) vyplývá, že ne, protože každý dlužník raději jako první uhradí své závazky a až poté se bude zajímat o možnost zhodnocení přebytku prostředků. Jenže je možné zeptat se i opačně, tedy: „Může se nějaký investor dostat do situace, kdy bude potřebovat půjčku?“. A tady už tak jednoznačná odpověď neexistuje. Modelová situace, která rozhodla o spojení těchto dvou typů uživatelů do jedné role, vypadala následovně:

- Investor vloží své prostředky na účet firmy a firma mu poskytne zhodnocení jeho úspor zprostředkováním půjčky s fixní dobou návratnosti.
- Investor se dostane do finanční tísně, ať už špatným odhadem svých možností a investováním příliš vysoké částky, nebo z nejrůznějších osobních důvodů, jako je ztráta zaměstnání, nečekané výdaje apod.
- Nyní potřebuje řešit svou situaci a obrátí se na firmu ve snaze získat část svých peněz zpátky.
- Firma nemůže zrušit poskytnutý úvěr a ani nemůže pomoci klientovi z vlastních prostředků, protože buď jich nemá dostatek, nebo na sebe nechce brát riziko spojené s převzetím

části zprostředkovaného úvěru a vyplacením této části investorovi.

- Firma nabídne investorovi krátkodobou půjčku z prostředků získaných od jiného investora.

Investor tímto způsobem získává čas na řešení své finanční situace a zároveň mu výnos z dlouhodobé investice pokryje náklady spojené se splácením úroku. Protože investoři jsou, zvláště pro menší nebo začínající firmy, velmi důležití (větší část jejich prostředků pochází právě od nich), může být úrok dokonce snížen tím, že v rámci zachování dobrých vztahů si firma z této půjčky nebude nárokovat žádnou provizi, čímž se úrok sníží na zisk investora, který prostředky poskytl.

Z výše uvedeného vyplývá, že investor by tedy měl mít větší množství možností než dlužník. Nicméně zadavatel byl zaskočen přehlédnutím výše uvedeného scénáře a proto byla znovu prodiskutována otázka, zda může mít dlužník zájem o investování peněz. Díky tomu byla nalezena mnohem závažnější chyba v původních úvahách. Dlužník může splatit své půjčky, následně začít investovat přebytky a stát se tak velmi snadno investorem. Výsledná dohoda tedy byla, že typy uživatelů dlužník a investor budou spojeny do jedné role. Zde je vhodné uvést ještě drobný implementační detail. Většina klientů bude mít v systému pouze jeden typ závazku. Z tohoto důvodu budou investorům skryty půjčky a dlužníkům investice až do té doby, než jim bude v systému vytvořen alespoň jeden závazek odpovídajícího typu.

5.4.2 Zaměstnanec

Tato role v rámci implementace zahrnuje všechny zaměstnance firmy, kteří mají oprávnění se systémem pracovat. Zaměstnanci mají

přístup do celého systému a mohou s ním aktivně interagovat. Vkládají a spravují uživatele, vytvářejí a spravují půjčky a investice, nahrávají smlouvy a mají přístup ke všem statistickým výstupům. Takový přístup plně vyhovuje menším a začínajícím firmám, které mají pouze několik zaměstnanců a všem plně důvěřují. S tím, jak firma roste, však vzniká potřeba vytvořit větší množství specializovaných rolí a omezovat jejich přístup k citlivým informacím, které pro svou práci nepotřebují.

Z tohoto důvodu je v návrhu zahrnuto rozdělení role zaměstnanec na tři specializace pro budoucí využití. Strategický management má za úkol řízení firmy na nejvyšší úrovni. Stará se o nastavení parametrů půjček a investic, sledování a analyzování statistických dat, případně činí důležitá rozhodnutí určující budoucí směřování firmy. Menší a střední firmy zpravidla mívají jen jednočlenný strategický management a tato pozice obvykle patří majiteli firmy. Druhou specializací je taktický management. Ten zastává nejčastěji také jen jeden člověk a funguje jako zástupce majitele. Může tedy zčásti nebo dočasně zastupovat strategického managera, ale jeho hlavním úkolem je starost o běžný chod firmy, dohled nad ostatními zaměstnanci a uzavírání důležitých kontraktů. Na nejnižším stupni se nachází operativní management. Má za úkol běžné činnosti, jako jsou komunikace s klienty, uzavírání smluv nebo shánění investic a půjček.

5.4.3 Sekretářka

Vedle výše uvedených zaměstnanců firma také využívá služby administrativní pomocnice – sekretářky. Může buď přímo pracovat ve firmě jako zaměstnanec, nebo fungovat v roli externího spolupracovníka. Jejím úkolem je starost o nejrůznější administrativu spojenou s běžných chodem společnosti, ale může se starat například i o účetnictví nebo jednoduché právní služby. Sekretářka nemá do systému přístup a přímo s ním neinteraguje, ale v příštích verzích

systému jí budou moci zaměstnanci prostřednictvím systému generovat potřebné podklady. Typickým příkladem může být dokument, který bude obsahovat informace o pohybech finančních prostředků, na základě kterého bude moci sekretářka vypracovat účetnictví. Zadavatel by za užitečnou funkcionalitu považoval i generování dokumentů, které by šly jednoduše importovat do některého z existujících účetních systémů, ale to zatím existuje pouze ve formě nápadu s velmi nízkou prioritou.

5.4.4 Zprostředkovatel

Na pomezí klienta a zaměstnance stojí zvláštní role zprostředkovatel. V současném návrhu se jedná o externího spolupracovníka, který pro firmu vyhledává a doporučuje klienty. V případě, že si jím doporučený kontakt vezme od firmy půjčku nebo do firmy vloží investici, je zprostředkovatel uveden u dané smlouvy a odměněn určitou procentuální částkou z hodnoty tohoto kontraktu. Prozatím nezodpovězenou otázkou zůstává, zda se zprostředkovatelem může stát i klient nebo za určitých okolností třeba zaměstnanec firmy. Klient se zprostředkovatelem stává ve chvíli, kdy firmě doporučí například svého známého a u některých firem mohou být zaměstnanci motivování procentuálním bonusem z uzavřené smlouvy, čímž se v podstatě také sami stávají zprostředkovateli. Po diskuzi se zadavatelem bylo nakonec rozhodnuto, že tyto případy systém prozatím nebude brát v úvahu a role zprostředkovatele bude omezena jen na jasně určené externí spolupracovníky. V případě, že se v průběhu testovacího provozu objeví problémy s tímto úhlem pohledu, může být role zprostředkovatele pozměněna.

5.4.5 Vymahač

I když se každá společnost poskytující úvěry snaží prostřednictvím všech dostupných nástrojů prověřit své klienty, aby odhalila případné problémy, určitá míra rizika při půjčování peněz vždy zůstane. V případě, že má klient snahu své závazky vyrovnat, vycházejí mu firmy obvykle vstříc. Nebývá to z jejich strany projevem nějakého altruismu, ale spíše snahou vyhnout se zdlouhavému, nákladnému a často nepříliš úspěšnému vymáhání. Pro ty ostatní případy však nějakým způsobem spolupracují s vymahači.

Jedním z prvních rozšíření systému bude právě podpora této role. V současné době je návrh její implementace v podstatě hotový, dolaďují se už jen poslední detaily. Vymahači by měli, podobně jako klienti, se systémem interagovat pouze pasivně. Po přihlášení do systému by získali informace o problémových klientech a jejich závazcích, které jim byly přiřazeny nějakým zaměstnancem. Podobné případy jsou řešeny individuálně a přiřazení bude předcházet osobní komunikace vymahače se zaměstnancem, takže dalšími věcmi se již v rámci systému není nutno zabývat.

5.5 Deployment Diagram

Jako druhý byl pro zvýšení celkové přehlednosti vytvořen Deployment Diagram. Zatímco odlehčený Use Case se pohyboval spíše v teoretické rovině, tato grafická reprezentace mapuje části systému na fyzické výpočetní zdroje. Hardwarová část je kvůli velikosti aplikace poměrně jednoduchá, softwarová část je ovlivněna použitým návrhovým vzorem MVC a nasazením EF. V návrhu jsou zahrnuty také dva nápady, o jejichž implementaci se zatím pouze uvažuje. Prvním z nich je nahrazení běžného Hypertext Transfer Protocol[1] (dále jen HTTP) zabezpečeným a šifrovaným Hypertext

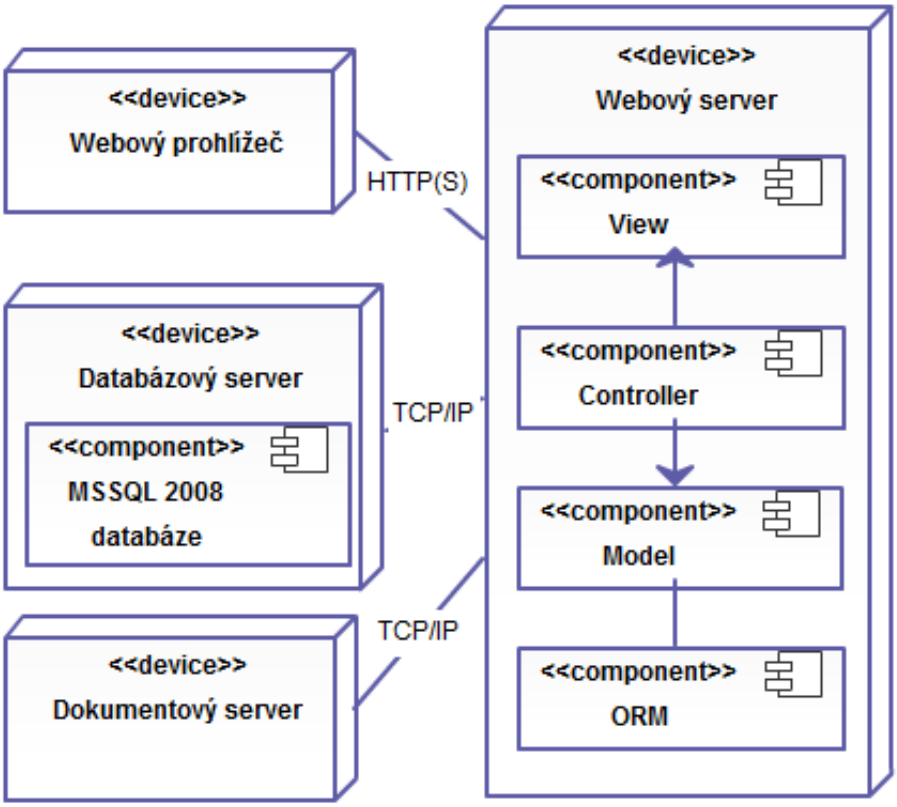
1. http://tools.ietf.org/html/rfc2616

Transfer Protocol Secure[2] (dále jen HTTPS). To zatím nebylo pro vývoj a testovací fázi potřeba, ale s nasazením do ostrého provozu přichází problém s přenášením osobních, případně jinak citlivých údajů, skrze nešifrované spojení a jako nejsnadnější řešení se jeví právě použití HTTPS. Druhý otazník zatím visí nad použitím speciálního serveru určeného pouze pro dokumenty. V budoucnu se totiž množství dokumentů ukládaných systémem zřejmě citelně navýší a bylo by vhodné oddělit dokumentové úložiště od webového serveru. V úvahu ale zatím připadají i jiná řešení, jako je např. přesun dokumentů do databáze.

5.6 Product Backlog

Použitá metodika je z větší části postavena na agilní metodice Scrum a přebírá tedy i jeho tři základní artefakty. Product Backlog je uspořádaný seznam požadavků a funkcí systému. Ty jsou reprezentovány jednotlivými položkami, které se rozdělují na čtyři základní typy. Nejčastějším typem je požadavek, který je v tomto případě realizovaný pomocí User Stories. Specifikem User Story je, že pokud není dostatečně samovysvětlující, jsou uvedena rozšířená kritéria pro její akceptování. Dalším typem jsou znalosti. Tato položka pokrývá nutnost učit se nové technologie a postupy. Samotný typ technologie pak slouží k zachycení požadavku na implementaci pomocných struktur a funkcionalit, které nejsou přímo požadovanými vlastnostmi systému. Typickým příkladem je například základní kostra systému, propojení s databází atp. Posledním typem je chyba. V případě, že se v průběhu vývoje objeví nějaký zásadní nedostatek původního návrhu, způsobující chybu většího rozsahu, je označen takto a přidán do Product Backlogu. Pokud je chyba menšího rozsahu, měla by být vyřešena v rámci sprintu.

2. http://tools.ietf.org/html/rfc2818



Obrázek 5.2: Deployment Diagram

Product Backlog je vytvářen společnými silami vývojáře a zadavatele. O seřazení položek rozhoduje s konečnou platností zadavatel. Položky jsou řazeny shora od nejdůležitější po nejpostradatelnější a je u nich určen hrubý časový odhad. Kromě tohoto čísla se zadavatel obvykle rozhoduje podle přínosu pro jeho podnikání, závislostí na ostatních položkách atd. Protože se jedná o agilní vývoj, není Product Backlog neměnným finálním dokumentem, vytvořeným během úvodní analýzy. Místo toho je v průběhu vývoje opra-

vován a doplňován. Product Backlog byl vytvořen pomocí tabulkového procesoru Calc[3] z kancelářské sady LibreOffice.

	A	B	C	D
1	**Název**	**Typ**	**Náročnost**	**User Story**
2	Seznámení s technologiemi	Znalosti	45	Úvodní seznámení s konkrétními technologiemi, čtení dokumentace a tutoriálů.
3	Kostra systému	Technologie	10	Vytvoření základní kostry systému.
4	Nasazení systému	Technologie	5	Nasazení systému na testovací server.
5	Správa uživatelů	Požadavek	14	V roli Zaměstnanec chci přidávat, odebírat a upravovat všechny uživatele v systému.
6	Správa půjček	Požadavek	19	V roli Zaměstnanec chci přidávat, odebírat a upravovat všechny půjčky v systému.
7	Správa investic	Požadavek	19	V roli Zaměstnanec chci přidávat, odebírat a upravovat všechny investice v systému.
8	Prohlídka závazků	Požadavek	7	V roli Klient chci zobrazit svá osobní data, závazky a jejich stav.
9	Nahrávání smluv	Požadavek	4	V roli Zaměstnanec chci nahrávat k půjčkám a investicím odpovídající smlouvy.
10	Stahování smluv	Požadavek	2	V roli Zaměstnanec chci mít u každé půjčky zobrazeny příslušné smlouvy.
11	Zobrazení statistik	Požadavek	15	V roli Zaměstnanec chci zobrazit základní statistické údaje o změnách a finančních tocích v systému.
12				
13	**Celkový čas**		**140**	

Obrázek 5.3: Ukázka Product Backlogu, z důvodu nedostatku místa bez schvalovacích kritérií

5.7 Sprint Backlog

Sprint Backlog je seznam požadavků, které by měly být implementovány v průběhu příštího sprintu. Vytváří ho sám vývojář, bez zásahů zadavatele, vždy na začátku výběrem horních položek z Product Backlogu, které časové odpovídají délce sprintu. Tyto položky se pak dále dekomponují na menší, snadněji splnitelné úkoly, které by měly ideálně mít časovou náročnost mezi 4 až 16 hodinami [2, str. 8]. Protože se jedná o dostatečně malý projekt a sprinty jsou poměrně krátké, byly položky dekomponovány o něco jemněji než bývá obvyklé. Jejich náročnost se tedy pohybuje v rozmezí 2 až 14

3. http://www.libreoffice.org/features/calc/

hodin Stejně jako u Product Backlogu byl pro jeho vytvoření použit nástroj LibreOffice Calc.

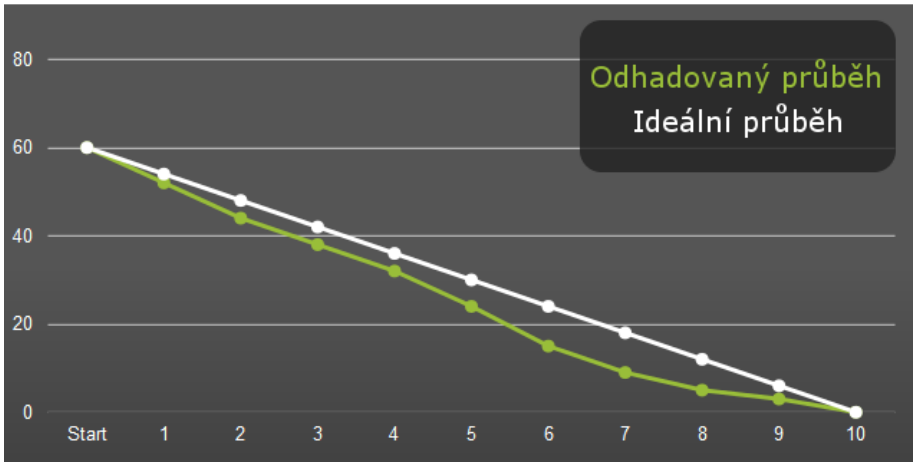
	A	B	C	D	E
1	**Název**	**Úkoly**	**Odhad náročnosti**	**Reálná náročnost**	**Zbývá**
2	Seznámení s technologiemi		**45**	**40**	**0**
3		ASP.NET MVC	12	10	0
4		Razor	2	2	0
5		Entity Framework	8	5	0
6		Code First	2	1	0
7		Membership	5	8	0
8		Tutoriály	10	9	0
9		MSSQL Express	6	5	0
10					
11	Kostra systému		**10**	**10**	**0**
12		Základní rozložení stránky	3	2	0
13		Napojení na databázi	2	4	0
14		CRUD akce pro uživatele	5	4	0
15					
16	Nasazení systému		**5**	**10**	**0**
17		Publikování systému	5	0,5	0
18		Publikování databáze	0	9,5	0
19					
20	**Sprint**		**60**	**60**	**0**

Obrázek 5.4: Ukázka Sprint Backlogu

5.8 Burn Down Chart

Burn Down Chart je jednoduchý graf, jehož zavedení umožňuje vývojáři snadné sledování průběhu aktuálního sprintu. Na ose X je promítnutý čas, na ose Y celková odhadovaná náročnost zbývajících úkolů ve sledovaném sprintu. Graf je každodenně aktualizován. V případě, že se začne výrazně odchylovat od ideálního lineárního průběhu, může vývojář některé úkoly vracet zpět do Product Backlogu, nebo z něj naopak další úkoly přibírat. U správně odhadnutého a provedeného sprintu by měla křivka reprezentující objem zbývající práce protnout osu X poslední den sprintu a neměla by se příliš odchylovat od lineárního průběhu. Po zvážení zkušeností tvůrce použité metodiky bylo upuštěno od snahy automatizovat generování grafu použitím nějakého nástroje. K zachycování

Burn Down Chartu byl zvolen obyčejný čtverečkovaný papír a propisovací tužka. Tento postup se nakonec ukázal jako naprosto vyhovující potřebám jednoho vývojáře a zároveň bylo pro jeho provádění zapotřebí minimální množství času. Vybraný graf byl následně pro účely prezentace vytvořen pomocí online služby[4].



Obrázek 5.5: Ukázka Burn down chart pro první sprint

5.9 ASP.NET a MVC

Použitý framework ASP.NET nabízí pro vývoj webové aplikace tři samostatné programovací modely. Každý je určený na jiný typ projektů, nabízí velmi odlišné pohledy na věc, ale zároveň je v případě nutnosti možné je kombinovat. Nejjednodušší z nich jsou Web Pages. Slouží pro rychlý vývoj malých webových projektů, které nemají za úkol zpracovávat větší množství dat. Typickou aplikací jsou blogy nebo osobní webové stránky. Pro středně velké projekty jsou tu Web Forms, které jsou postaveny na event-driven přístupu a bu-

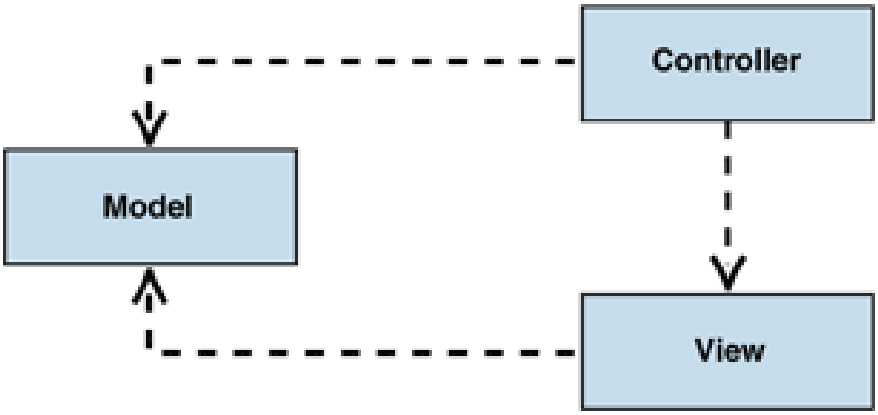
4. www.burndown-charts.com

dou nejvíce vyhovovat programátorům zvyklým na vývoj ve Win Forms. Pro tento projekt byl však zvolen třetí přístup a to ASP.NET implementace návrhového vzoru MVC.

MVC je návrhový vzor, který pomáhá zvyšovat přehlednost kódu, usnadňuje údržbu, popřípadě vývoj velkých systémů a jeho aplikace výrazně zjednodušuje testování. Toho všeho je dosaženo pomocí rozdělení aplikace na tři logické celky:

- Model – Reprezentuje stav aplikace. Obsahuje datové struktury, z nichž některé obvykle mapuje na databázi.
- View – Přebírá vstupní informace od Controlleru a zobrazuje je uživateli.
- Controller – Vyhodnocuje uživatelské vstupy, aktualizuje model (čímž mění stav aplikace) a předává informace celku View.

Kromě výhod vyplývajících ze samotné podstaty tohoto návrhového vzoru je ASP.NET MVC také mnohem bližší vývojářům, kteří mají nějaké zkušenosti s vývojem webových aplikací, lépe reflektuje bezstavovost HTTP, je mladší a i když jsou pro rychlé prototypování Microsoftem doporučovány spíše Web Forms, byla upřednostněna čistota návrhu, kterou umožňuje právě MVC.



Obrázek 5.6: Návrhový vzor MVC, převzato z `http://msdn.microsoft.com/en-us/library/ff649643.aspx`

6 Implementace

V této kapitole je shrnuta implementace systému, respektive její důležité prvky. Dále je zde uvedena ukázka vývoje jedné z jeho částí. Navzdory tomu, že se jedná o projekt relativně malého rozsahu, zde není prostor pro kompletní popis celého průběhu vývoje. Z tohoto důvodu bude v práci zahrnuta pouze demonstrace formou shrnutí specifické první iterace a popisu druhé iterace, která byla zvolena, protože obsahuje největší část implementace funkcionality.

6.1 Správa uživatelů

Správa uživatelů je jednou z typických funkcí, kterou musí v nějaké podobě implementovat většina webových služeb. ASP.NET proto nabízí vývojářům vlastní komponentu Membership, která naprogramování této funkcionality výrazně usnadňuje. Membership umožňuje vývojáři snadno spravovat uživatelské informace potřebné pro autorizaci a autentizaci. Podporuje i správu rolí a je velmi snadno konfigurovatelný a rozšiřitelný. Jeho použití však přináší i drobné komplikace. Protože je tato komponenta podle objektově orientovaného přístupu správně zapouzdřená, je třeba s ní interagovat pouze pomocí rozhraní. Toto neumožňuje používat údaje z tabulek patřících této komponentě například jako cizí klíče ve svých vlastních tabulkách.

Dalším problémem je, že v modelu je potřeba vytvořit kromě třídy, která je mapovaná do databáze, ještě jednu zděděnou. Ta bude obsahovat jak údaje získané přes rozhraní komponenty Membership, tak data získaná z vlastní databáze aplikace. S tímto potomkem pak bude pracovat samotná aplikace a při zápisu jeho obsahu do databáze se část dat uloží pomocí rozhraní, následně se odstraní ztrátovou konverzí na rodičovský typ a ten je již možné snadno

uložit do vlastní databáze. Jak již bylo řečeno výše, Membership je velmi snadno upravitelná komponenta, takže je například možné ukládat data jiným způsobem než do databáze, je možné si tento způsob přímo vytvořit nebo lze ukládání dalších dat pro potřeby jednodušších aplikací zprostředkovat pomocí rozšíření Profiles.

6.2 Entity Framework a Code First

K vytvoření datové mezivrstvy mezi vlastní aplikací a databází nabízí Microsoft svůj EF postavený na ADO.NET[1]. EF zjednodušuje přístup k datům tím, že umožňuje vývojáři vyhnout se nutnosti pracovat přímo s databází, což bývá často složité a vzhledem k rozdílnosti typových systémů nepraktické. Namísto toho je možné získávat data pomocí pomocí dotazů na silně typované třídy a nechat EF, aby se postaral o samotnou interakci s databází. Kromě toho, díky přístupu EF ke vztahům, není nutné psát složité dotazy na spojování tabulek. EF nabízí tři způsoby jakými lze vytvořit model datových objektů. Jsou to automatické převedení z již existující databáze, vytvoření modelu pomocí model designeru, nebo nejnovější, preferovaný a zvolený přístup nazvaný Code First. Code First spočívá v tom, že programátor popíše model pomocí vytvoření tříd a specifikací speciálních atributů. EF se pak sám postará o vytvoření modelu a jeho namapování na databázi.

Použití EF velmi ulehčilo tuto část vývoje. Hlavním přínosem byla schopnost EF samostatně generovat databázi podle tříd a značek. Tuto schopnost trochu omezuje nutnost celou databázi před vygenerováním smazat, protože EF nedokáže modifikovat strukturu již existující databáze, takže je třeba do ní po vygenerování opět nahrát testovací data. Nicméně všechny tyto kroky lze automatizovat a podmíněně provádět při spuštění aplikace. Dalším přínosem bylo

1. http://msdn.microsoft.com/en-us/library/h43ks021.aspx

také usnadnění testování, kdy odpadá nutnost psát netypované textové SQL dotazy.

6.3 Ukázka vývoje

Po konci každého sprintu by mělo následovat jeho zhodnocení, ve Scrumu nazývané Sprint Retrospective. Jeho cílem je seznámit tým s postupem prací na projektu, v případě samostatného vývojáře spíše získat přehled o tom, nakolik byl provedený sprint úspěšný. Z této zpětné vazby by se měl vývojář poučit a poopravit své plány pro následující sprinty.

6.3.1 První sprint

Úvodní sprint byl specifický tím, že jeho stěžejním cílem nebyla implementace funkcionality, ale hlubší seznámení se s použitými technologiemi, studium dokumentace a posléze pomocných tutoriálů. Původně měl, stejně jako ostatní sprinty, trvat 40 hodin, ale již na jeho začátku se ukázalo, že použité technologie jsou velmi odlišné od vývoje při použití PHP, což si vyžádalo větší časovou dotaci, než bylo původně odhadnuto. Protože nebylo možné ani nutné postup v učení nějak přesně měřit, byl nakonec sprint, namísto rozdělení do dvou, pouze prodloužen na 60 hodin. Před koncem se objevil zásadní problém s publikací databáze na webový server, který byl způsoben jeho provozovatelem. Problém se po komunikaci s podporou webhostingu podařilo odstranit a vzhledem k tomu, že nasazení samotné aplikace proběhlo již hladce, nenarušila tato komplikace upravený časový rozvrh sprintu.

6.3.2 Druhý sprint

Náplní druhého sprintu bylo implementovat velkou část základních vlastností systému. Nejprve bylo realizováno rozšíření správy uživatelů, protože na nich závisejí všechny ostatní součásti systému. Velkým usnadněním bylo použití komponenty Membership, které na druhou stranu přineslo některé komplikace popsané v části 6.1. Následně byla naprogramována správa půjček. Kvůli částečné závislosti na investicích by bylo možná vhodné naprogramovat tyto části v opačném pořadí, ale protože konečné slovo ohledně pořadí položek v Product Backlogu má zadavatel, bylo přihlédnuto k jeho přání. To se navzdory o něco vyšší časové náročnosti nakonec ukázalo jako správný postup, protože když zadavatel testoval správu půjček, vymyslel několik úprav systému investic, na které si buď během analýzy nevzpomněl, nebo ho napadly, až když si sám vyzkoušel, co všechno systém správy půjček umí. Jako poslední bylo vytvořeno zobrazování osobních informací a závazků klientům. Tato část byla vzhledem k minimální funkcionalitě vyhodnocena jako jednoduchá, což se také v průběhu implementace potvrdilo a nedošlo tedy k žádným závažným problémům.

6.4 Time Management

Velkým problémem, kterému čelí každý vývojář pracující sám, je organizace času. Do jisté míry může pomoci použití některé z metodik vývoje software, ale protože ve výsledku na sebe a svou práci dohlíží především on sám a je také tím, kdo stanovuje cílová data a časové odhady, může být velmi těžké si práci správně rozložit a dostatečně se na ni soustředit. To se plně projevilo i v tomto projektu, kdy v průběhu a především na začátku vznikaly problémy s posouváním cílových dat, motivací a nevhodným pracovním prostředím. Problémy se podařilo vyřešit použitím jednoduché time manage-

ment techniky zvané Pomodoro [6]. Pomodoro vzniklo v roce 1980 z nápadu studenta, který měl problémy s přípravou na nadcházející zkouškové období. Svou základní myšlenkou má velmi blízko k agilním metodikám, především používáním časového rámce a iterativního vývoje. Z tohoto důvodu se také stalo oblíbenou volitelnou součástí některých z nich. V oblibě ho mají především metodiky obsahující programování ve dvojicích, např. XP. Pomodoro spočívá v rozdělení času na 25 minutové úseky, mezi kterými je 5 minut dlouhá pauza. Po každých čtyřech úsecích následuje delší pauza, doporučuje se 15 až 20 minut. Jediným nástrojem potřebným k jeho aplikaci je libovolný přístroj schopný zvukově upozornit na konec nastaveného časového úseku.

6.5 Testování

V době odevzdání práce byla aplikace nasazena do interního testovacího provozu v rámci firmy. Bylo nalezeno malé množství chyb technického rázu, což bylo zřejmě způsobeno třemi faktory. Hlavním z nich byl fakt, že se na vývoji podílel jediný vývojář a nevznikaly tedy chyby při spojování jednotlivých částí systému dohromady, příp. špatným pochopením cizího kódu. Důležitá ale byla jistě i jednoduchost a především čistota návrhu usnadněná použitím návrhového vzoru MVC. V případě, že bude systém dále růst, především pokud bude potřeba najmout další programátory, bylo by vhodné naprogramovat alespoň základní sadu automatických testů pro usnadnění testování.

Z uživatelského hlediska se projevila špatná ergonomie některých ovládacích prvků a v určitých případech neintuitivní chování. Dále uživatelé kritizovali celkový vzhled aplikace, ale to se u prototypu dalo očekávat. Po ustálení funkcionality bude tento problém vyřešen pomocí externí spolupráce s profesionálním grafikem.

7 Přílohy

Elektronické přílohy se nacházejí v Archivu IS MU a jsou zorganizovány do následující adresářové struktury:

`/project/` – Kompletní Visual Web Developer projekt, obsahující i zdrojové kódy

`/compiled/` – Zkompilovaný projekt připravený k nasazení na webový server

`/diagrams/` – Diagramy a artefakty použité v průběhu vývoje

`/bp_source/` – Zdrojové LaTeX soubory tohoto dokumentu

`/bp.pdf` – Tento dokument v elektronické verzi

8 Závěr

V rámci této práce byl úspěšně navržen a implementován funkční prototyp informačního systému pro správu půjček určený malým a středním firmám podnikajícím v České republice. Pro testovací účely je dostupný na url: `http://xloan.aspone.cz`. Tento systém by v budoucnu mohl zacelit trhlinu na lokálním trhu a uspokojit poptávku po tomto typu aplikace. Na příkladu systému byl zároveň demonstrován návrh a průběh vývoje podobných webových služeb. Kromě toho práce obsahuje shrnutí problematiky vývoje software v jednočlenném týmu za použití současných agilních metodik a v krátkosti popisuje otestování aplikace a její přijetí zadavatelem.

Je třeba zmínit, že systém je stále ve stádiu vývoje a zadavatel si ho přeje dále rozvíjet a po dosažení dostatečné kvality ho případně nabízet k prodeji dalším firmám pomocí modelu Software as a Service. Prostoru a nápadů k dalšímu vývoji je dostatek, což dokazuje i samotný návrh systému, který počítá s budoucími rozšířeními nad rámec současné implementace. Hlavním cílem bude nadále zvýšení bezpečnosti s ohledem na velké množství osobních a jinak citlivých údajů, se kterými systém pracuje. Vylepšení a rozšíření se dočká také práce s dokumenty. Velmi užitečné by bylo například automatické generování smluv podle šablony a údajů zákazníka nebo generování souborů ve formátu podporovaném některým rozšířeným účetním systémem.

Literatura

[1] Use cases vs user stories in Agile development. 2012, [online]. [cit. 2012-05-13].
URL: http://www.boost.co.nz/blog/agile/use-cases-or-user-stories/

[2] BECHNÝ, O.: *Návrh metodiky vývoje softwaru z pohledu samostatného vývojáře*. Diplomová práce, Fakulta informatiky, Masarykova univerzita, leden 2011, [online]. [cit. 2012-03-05].
URL: http://is.muni.cz/th/73071/fi_m/

[3] BECK, K.: *Extreme programming explained: embrace change*. XP series, Addison-Wesley, 2000, ISBN 978-0201616415.

[4] BECK, K.; a kol.: Manifest Agilního vývoje software. 2001, [online]. [cit. 2012-04-15].
URL: http://agilemanifesto.org/iso/cs/

[5] BENSON, J.; Barry, T. D.: *Personal Kanban: Mapping Work*. CreateSpace, 2011, ISBN 978-1453802267.

[6] CIRILLO, F.: *The Pomodoro Technique*. Lulu.com, 2009, ISBN 978-1445219943.

[7] COCKBURN, A.: Structuring use cases with goals. Technická zpráva, Journal of Object-Oriented Programming, 1997, [online]. [cit. 2012-05-14].
URL: http://alistair.cockburn.us/Structuring+use+cases+with+goals

[8] COCKBURN, A.: *Agile software development*. Agile software development series, Addison-Wesley, 2002, ISBN 978-0201699692.

[9] COCKBURN, A.: *Crystal Clear: A Human-Powered Methodology For Small Teams*. Addison-Wesley, 2004, ISBN 978-0201699470.

[10] COCKBURN, A.: Why I still use use cases. 2008, [online]. [cit. 2012-05-14].
URL: `http://alistair.cockburn.us/Why+I+still+use+use+cases`

[11] DZHUROV, Y.; KRASTEVA, I.; ILIEVA, S.: Personal Extreme Programming – An Agile Process for Autonomous Developers. říjen 2009, ISBN 978-9549526622, [online]. [cit. 2012-03-20].
URL: `https://research.uni-sofia.bg/bitstream/10506/647/1/S3T2009_37_YDzhurov_IKrasteva_SIlieva.pdf`

[12] HUMPHREY, W. S.: *PSP(sm): A Self-Improvement Process for Software Engineers*. Addison-Wesley, 2005, ISBN 978-0321305497.

[13] JAMES, J.; HICKSON, I.: HTML 5 Editor Ian Hickson discusses features, pain points, adoption rate, and more. 2008, [online]. [cit. 2012-05-10].
URL: `http://en.wikipedia.org/wiki/Html5#cite_note-techrepublicref-23`

[14] ŠKODA, O.: *Agilní metodiky vývoje softwaru*. Diplomová práce, Fakulta informatiky, Masarykova univerzita, červen 2009, [online]. [cit. 2012-03-05].
URL: `http://is.muni.cz/th/98591/fi_m/`

[15] SCHWABER, K.; BEEDLE, M.: *Agile Software Development with Scrum*. Series in Agile software development, Pearson Education International, 2008, ISBN 978-0132074896.