Finanční řízení projektů pro vývoj softwaru

Roman Rábel

Bakalářská práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta managementu a ekonomiky

Univerzita Tomáše Bati ve Zlíně
Fakulta managementu a ekonomiky
Ústav průmyslového inženýrství a informačních systémů
akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Roman Rábel**
Osobní číslo:	**M13497**
Studijní program:	**B6208 Ekonomika a management**
Studijní obor:	**Management a ekonomika**
Forma studia:	**kombinovaná**
Téma práce:	**Finanční řízení projektů pro vývoj softwaru**

Zásady pro vypracování:

Úvod

Definujte cíle práce a použité metody zpracování práce.

I. **Teoretická část**

- **Na základě literárních pramenů zpracujte teoretické poznatky týkající se řízení nákladů na vývoj a testování softwaru a chybovosti výsledného produktu.**

II. **Praktická část**

- **Proveďte analýzu současné realizace softwarových projektů ve firmě.**
- **Na základě zjištěných faktů navrhněte zlepšení finančního řízení uvedené firmy.**
- **Vyhodnoťte zvolené řešení v konfrontaci s výslednou chybovostí konečného produktu.**

Závěr

Rozsah bakalářské práce: **cca 40 stran**

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

GUCKENHEIMER, Sam a Juan J. PEREZ. Efektivní softwarové projekty. Vyd. 1. Brno: Zoner Press, 2007, 255 s. ISBN 978-808-6815-626.
MCCONNELL, Steve. Odhadování softwarových projektů. Vyd. 1. Brno: Computer Press, 2006, 320 s. ISBN 80-251-1240-3.
OPELT, Andreas et al. Agile Contracts: Creating and Managing Successful Projects with Scrum. Vyd. 1. Hoboken: Wiley, 2013, 282 s. ISBN 978-1-118-63094-5.
PAGE, Alan, Ken JOHNSON a Bj ROLLINSON. Jak testuje software Microsoft. Vyd 1. Brno: Computer Press, 2009, 384 s. ISBN 978-80-251-2869-5.
PATTON, Ron. Testování softwaru. Vyd. 1. Brno: Computer Press, 2002, 314 s. ISBN 80-7226-636-5.

Vedoucí bakalářské práce: **prof. Ing. Ladislav Buřita, CSc.**
Ústav průmyslového inženýrství a informačních systémů

Datum zadání bakalářské práce: **15. prosince 2016**

Termín odevzdání bakalářské práce: **15. května 2017**

Ve Zlíně dne 15. prosince 2016

L.S.

doc. Ing. David Tuček, Ph.D.
děkan

prof. Ing. Felicita Chromjaková, PhD.
ředitel ústavu

PROHLÁŠENÍ AUTORA
BAKALÁŘSKÉ/DIPLOMOVÉ PRÁCE

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen na elektronickém nosiči v příruční knihovně Fakulty managementu a ekonomiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

1. že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
2. že odevzdaná verze diplomové/bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 15.5.2017

.....
podpis diplomanta

ABSTRAKT

Cílem této bakalářské práce je provést analýzu současné realizace softwarových projektů ve firmě a navrhnout zlepšení finančního řízení v budoucnosti. V práci je soustředěna pozornost na projekt, modely vývoje softwaru a testování.

Klíčová slova: projekt, software, vývojové metodiky, testování, odhady softwarových projektu

ABSTRACT

The aim of this bachelor thesis is to analyze the current realization of software projects in the company and to suggest the improvement of the financial management in the future. The thesis is concentrating on the project, models of software development and testing

Keywords: Project, Software, Development methodology, Testing, Software project estimates

Děkuji vedoucímu bakalářské práce prof. Ing. Ladislavu Buřitovi, CSc. za odborné vedení při zpracovaní této práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

ÚVOD

V současné době jsme svědky doby, kdy jsou všechny země světa spojeny pomocí produktů mimořádně rychle rostoucího odvětví informačních technologií. Trend automatizace a centralizace je často důvodem pro to, aby se globální společnosti zaměřily na projekty softwarového vývoje, které by měly pomoci snížit náklady a zvýšit jejich konkurenční výhodu. Bohužel, většina z těchto projektů skončí nedokončená nebo není úspěšná. Existují různé důvody pro selhání projektů, ale mnohé z nich by mohly být vyloučeny přesnou finanční analýzou.

V této bakalářské práci se budu podrobněji zabývat projekty pro vývoj softwaru. Hlavním cílem v konečném důsledku je snížit náklady v podobě nového řešení finančního řízení analyzované firmy. Dalším cílem této práce je vyhodnotit zvolené řešení v konfrontaci s výslednou chybovostí konečného produktu.

V teoretické části se pokusím shrnout co je to projekt a jaké má atributy. Taky se budu věnovat metodikám vývoje, jak tradičním, tak agilním. Dále zkusím přiblížit problematiku testování a odhadu velikost softwarových projektů.

V praktické části se zaměřím na analýzu současného řízení projektů pomocí SWOT analýzy a metodou řízeného rozhovoru se pokusit zjistit nedostatky, navrhnout zlepšení a také vyhodnotit zvolené řešení vůči chybovosti výsledného produktu.

I. TEORETICKÁ ČÁST

1 CO JE TO PROJEKT?

Projekt lze definovat jako „časově omezené úsilí vynaložené na vytvoření unikátního produktu, služby nebo výstupu.“ [1, s. 20]. Na druhé straně jsou pak provozní činnosti, které slouží k udržení chodu firmy. Projekty se od provozních činností liší tím, že končí ve chvíli, kdy je dosaženo jejich cílů nebo je projekt ukončen.

Podobnou definici má také norma ISO 10006 (Směrnice pro management jakosti projektů) „Projekt je jedinečný proces sestávající z řady koordinovaných a řízených činností s daty zahájení a ukončení, prováděný pro dosažení předem stanoveného cíle, který vyhovuje specifickým požadavkům, včetně omezení daných časem, náklady a zdroji.“ [2, s. 10]

1.1 Velikost projektu

McConnell (2006, s. 96) rozlišuje velikosti projektu takto:

1.1.1 Malý projekt

Malý projekt charakterizujeme jako projekt s pěti nebo méně technickými pracovníky, ale tato charakteristika je dost neurčitá. Na malé projekty obyčejně nelze použít statisticky orientované metody, které se uplatní na projektech velkých, protože odchylky v individuální produktivitě přehluší další faktory. U malých projektů lze snadněji použít jednoduchý model personálu (použití stejného počtu zaměstnanců na celý projekt), což neplatí použití přístupů orientovaných více na algoritmy a velké projekty.

Nejlepší metody odhadování pro malé projekty bývají založené na metodě "zdola nahoru", tedy na odhadech, které vytvoří jedinci, již na daném projektu budou pak doopravdy pracovat.

1.1.2 Velký projekt

Velký projekt je projekt, jehož tým má asi 25 lidí nebo více a trvá 6 až 12 měsíců nebo více. Nejlepší metody pro velké projekty se od začátku projektu směrem k jeho konci významně mění. V počátečních fázích projektu bývají nejlepší postupy založené na metodě "shora dolů", na algoritmech a statistikách. Jsou platné v té fázi projektu, kdy ještě nejsou známí členové týmu - plány jsou postaveny na týmu, který se skládá například z "11 vývojářů seniorů, 25 běžných vývojářů a 8 testerů", a konkrétní jedinci ještě nejsou určeni.

Ve středních částech projektu vygeneruje nejpřesnější odhady kombinace metod shoda dolů a zdola nahoru, které jsou již postaveny na vlastních historických datech projektu. V pozdních fázích velkých projektů poskytnou nejpřesnější odhady metody zdola nahoru.

1.1.3 Střední projekt

Střední projekt trvá od 3 do 12 měsíců a pracuje na něm od 5 do 25 lidi. Mají výhodu v tom, že mohou použít prakticky všechny metody z velkých projektů a také některé metody z malých projektů.

1.2 Atributy projektu

Projety existují v nepřeberném množství tvarů a velikostí. Následující atributy pomáhají projekt dále definovat [3, s. 52]:

- Projekt má jedinečný účel.

Každý projekt by měl mít dobře definovaný cíl. Výsledkem bude základ pro další diskuzi a projekty, které ústí v unikátní produkt, službu nebo výstup.

- Projekt je dočasný.

Projekt má jednoznačný začátek a konec.

- Projekt se vytváří postupným rozpracováváním.

Na začátku jsou projekty často definovány velmi zeširoka. Postupem času se jednotlivé specifické detaily stávají jasnějšími. Proto by se projekty měly vyvíjet přírůstkově. Projektový tým by měl vytvořit iniciační plány a následně je na základě nových informací detailněji aktualizovat a upravovat.

- Projekt vyžaduje zdroje, často z různých oblastí.

Zdroje zahrnují lidi, hardware, software a další majetek. Mnoho projektů překračuje hranice oddělení, aby dosáhly svého jedinečného cíle. Firma může rovněž najmout externího konzultanta, který jí poskytne další informace. Jakmile projektový tým vybere klíčové projekty k implementaci, bude pravděpodobně potřeba dodatečných zdrojů. Aby bylo možné dosáhnout projektových cílů, bude třeba přizvat lidi z jiných společností – dodavatelských a konzultačních firem. Zdroje jsou však limitované a k dosahování projektových a dalších cílů společnosti je třeba je využívat efektivně.

- Projekt by měl mít primárního zákazníka nebo sponzora.

Většina projektů má mnoho zainteresovaných stran a subjektů. Vždycky však musí existovat jeden, který bude hrát roli primárního sponzora. Sponzor projektu obvykle určuje jeho směr a poskytuje finance.

- Součástí projektu je nejistota.

Protože je každý projekt jedinečný, je někdy obtížné jasně definovat jeho cíle, odhadnout, jak dlouho bude trvat jeho dokončení, či určit, kolik bude stát. Příčinou nejistoty jsou rovněž externí faktory, jako například dodavatel, který skončí své podnikání, či člen projektového týmu, který bude nečekaně potřebovat dovolenou. Tato nejistota je jedním z hlavních důvodů, proč je projektové řízení tak náročné, zvlášť v případě projektů zahrnujících nové informační technologie.

1.3 Projektový trojimperativ

Každý projekt je omezen plánovaným rozsahem, časem a náklady. O těchto limitech se v projektovém řízení často hovoří, a to v souvislosti s takzvaným projektovým trojimperativem. Aby dosáhl úspěchu, musí projektový manažer zvažovat rozsah, čas a náklady a současně tyto často vzájemně protichůdné cíle sladit. On či ona musí uvažovat o následujícím [1, s. 23]:

- Rozsah: Jakou práci je třeba v rámci projektu udělat? Jaký jedinečný produkt, službu nebo výstup zákazník či sponzor projektu očekává? Jak bude rozsah ověřen?
- Čas: Jak dlouho by měla práce na projektu trvat? Jaký je harmonogram projektu? Jak bude tým monitorovat aktuální stav projektu ve vztahu k časovému rozvrhu? Kdo bude schvalovat změny v harmonogramu projektu?
- Náklady: Kolik by měla realizace projektu stát? Jaký je rozpočet projektu? Jak budou náklady sledovány? Kdo bude schvalovat změny rozpočtu?

1.4 Náklady na softwarový projekt

V softwarovém projektu rozlišujeme:

- náklady na software (např. programovací prostředí), hardware (případně další zařízení) včetně údržby
- náklady na cestování a školení

- náklady spojené s vynaloženým úsilím na vytvoření výrobku nebo zabezpečení služby
    - výplaty softwarovým inženýrům
    - režie: provoz prostorů, podpůrný personál, komunikace v rámci projektu, doplňkové služby (knihovna, kopírovací služby), zákonné povinnosti (pojištění)

Náklady ovlivňuje:

- velikost (složitost) projektu (vytvářeného výrobku nebo služby)
- schopnosti členy týmu a zkušenosti v problémové oblasti
- použití podpůrných prostředků
- proces vývoje dané ohraničení na výrobek (např. Spolehlivost, rychlost, paměťové nároky a jiné charakteristiky výpočetního prostředí)

Třeba odlišovat náklady a cenu výrobku pro zákazníka.

Cena = náklady + zisk

Cena často závisí na strategickém záměru organizace. Při jejím určování třeba uvažovat širší souvislosti. Cenu ovlivňuje:

- vytvoření příležitosti na trhu (pokud se přesouvá na nový trh, často je nižší cena)
- podmínky smlouvy (pokud např. Dodavateli zůstanou zdrojové texty programů a bude je moci použít na jiný projekt, stanoví se nižší cena)
- stupeň neurčitosti odhadu nákladů (často zvýšení ceny v případě nejistoty)
- pravděpodobnost proměnlivosti požadavků (jiná cena se může stanovit tehdy, pokud se počítá se změnami a jsou v ní zahrnuty, jako když se cena změn stanovuje zvlášť)
- finanční situace dodavatele a strategické rozhodnutí (někdy se stanovuje nižší cena, dokonce nižší, než jsou samotné náklady, aby se společnost udržela na trhu, cenový dumping, konkurence)
- možnosti zákazníka (cena se určí podle rozpočtu zákazníka a pak se jedná o specifikaci, zahrne se tolik funkčnosti, aby se dodržely plánované náklady)

2 MODELY ŽIVOTNÍHO CYKLU SOFTWARU

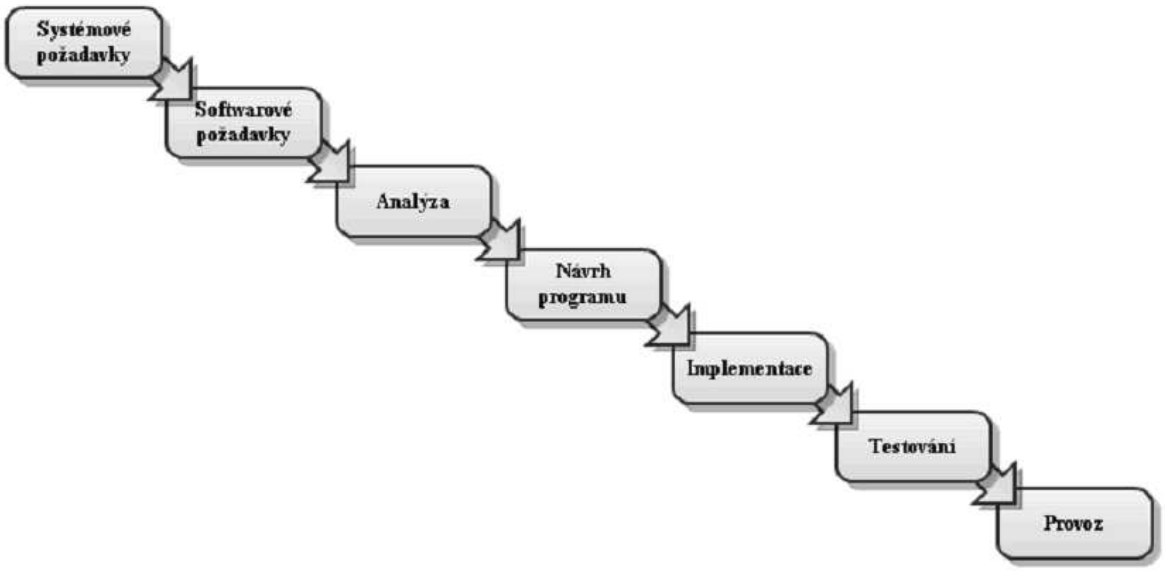
2.1 Tradiční metodiky

Tradiční metodiky jsou považovány za standartní způsob vývoje softwaru. Tyto metodiky jsou založeny na postupné řadě kroků, jako je definice požadavků, budování řešení, testování a nasazení do provozu. Tradiční metodiky vyžadují definování a dokumentaci stabilního souboru požadavků na začátku projektu. Existuje mnoho různých tradičních, ale v této práci omezím popis na tři nejvýznamnější metodiky: Vodopádový model, Spirální model a Unified Process.

2.1.1 Vodopádový model

Vodopádový model (někdy též vodopádový přístup) je přístup k vývoji nebo na řízení projektu, který předpokládá předem jasně daný plán, tedy sekvenční postup od analýzy. Název vychází z přirovnání posloupnosti jednotlivých fází projektu k protékání vody vodopádům. Poprvé ho publikoval Winston W. Royce v roce 1970 a to pomocí sedmi základních fází:

- Systémové požadavky (System requirements)
- Softwarové požadavky (Software requirements)
- Analýza (Analysis)
- Návrh programu (Program design)
- Implementace (Coding)
- Testování (Testing)
- Provoz (Operations)



Obr. 1 Vodopádový model

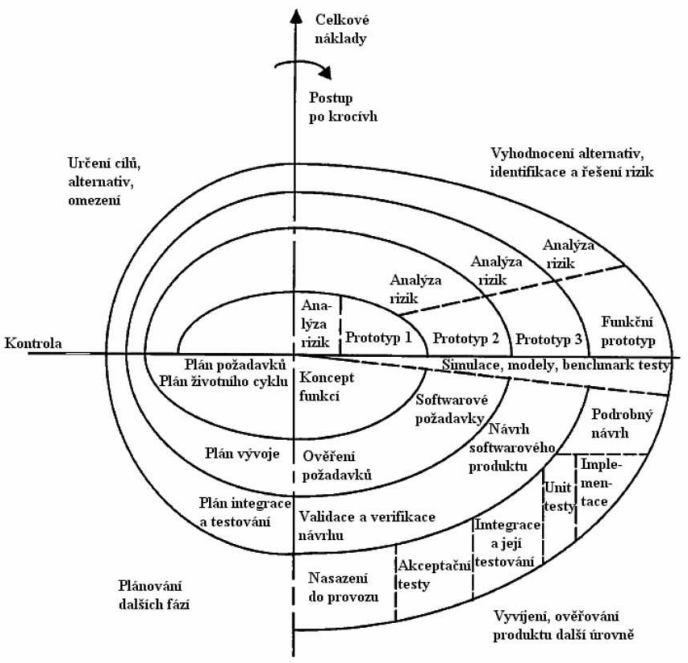
Vodopádový přístup klade důraz na plánování, termíny a časový rozvrh prací. Počítá s detailním naplánováním jednotlivých kroků a následném dodržování postupů při vývoji nebo realizaci projektu. Projektovému týmu je dán minimální prostor na změny v průběhu realizace, což je pro některé projekty nutné. Tento postup je vlastně "klasický přístup" k řízení projektů, který se drží plánu projektu. Vodopádový přístup je vhodný a užitečný v projektech, který mají jasný cíl a jasně definovatelný postup a rozdělení prací. Pro jejich naplánování se často používá Ganttův diagram. Úkolem manažera projektu je pak dodržet stanovený postup, čas a rozpočet. [4]

Vodopádový model je v protikladu vůči tzv. Agilním metodikám, které jsou naopak vhodnější pro vývojové projekty, které potřebují velkou míru inovací a upřesňování v průběhu vývoje.

2.1.2 Spirálový model

Spirálový model životního cyklu projektu vznikl jako reakce na hlavní nedostatek vodopádového modelu, který měl problém reagovat na změny požadavků během realizace projektu. Poprvé se o tomto modelu zmínil ve své práci v roce 1988 Barry W. Boehm a definoval ho jako kombinaci vodopádového modelu, proto typování, iterativní přístupu a analýzy rizik.

Požadavky mají stále na začátku realizace silné zastoupení a jejich pochopení se provádí především během prvního obkroužení spirály. Čím je proces realizace projektu ve větším okruhu spirály, tím jsou požadavky před samotným vývojem detailnější a zároveň jsou vyšší kumulativní náklady a pokrok dosažený na konci každé iterace větší. Jednotlivé cykly se skládají ze 4 fází: stanovení cílů, analýza rizik, implementace a testování a plánování další etapy. Tento model můžeme definovat také jako rizikem řízený přístup, jelikož analýze rizik se klade značná část životního cyklu. Životní cyklus připomíná vodopádový model, ale iterace a tvorba prototypů zajišťují častější zpětnou vazbu. Na konci každé iterace se provádí kontrola předchozího cyklu, včetně plánů na další cyklus a prostředků potřebných k jejich provedení. [5]



Obr. 2 Spirálový model

2.1.3 Rational Unified Process

Rational Unified Process (RUP) je metodika vývoje softwaru vytvořená společností Rational Software Corporation. Metodika RUP vychází z kolekce osvědčených praktik a postupů při vývoji softwaru. RUP je vhodnější spíš pro rozsáhlejší projekty a větší vývojové týmy, neboť klade důraz na analýzu a design, plánovaní, řízení zdrojů a dokumentaci.

RUP zavádí čtyři základní fáze vývoje, přičemž každá z nich je organizována do několika iterací. Před započetím nové iterace musí být splněna definovaná kritéria předchozí iterace. Ve fázi zahájení (inception) musí vývojáři definovat účel a rozsah projektu a jeho obchod-

ní kontext; ve fázi projektování (elaboration) je úkolem vývojářů analyzovat potřeby projektu a zákazníka a definovat základy architektury. Třetí fáze, realizace (construction), je zaměřena na vývoj designu aplikace a tvorbu zdrojových kódů, zatímco v poslední fázi, ve fázi předání (transition), dochází k předání projektu – buď zákazníkovi, nebo do dalšího vývojového cyklu.

Uvnitř každé fáze dále probíhají iterace; jejich počet závisí na konkrétních potřebách týmu a rozsahu projektu. Každá iterace je rámci RUP podrobně rozpracována; je stanoveno, kdo ji má dělat (roles), jak ji má dělat (activities) a co má být jejím výsledkem (artifacts). [6]

2.2 Agilní metodiky vývoje

Principy agilních metod jsou vysvětleny v Agilním manifestu. Stručně řečeno, agilní metody [7]:

- používají nepřetržitou komunikaci (zákazník/manažer/vývojář, manažer/vývojář a vývojář/vývojář)
- udržují důslednou sledovatelnost mezi požadavky zákazníků (stories) a softwarem
- poskytují funkční přírůstkové verze softwaru s rychlým obratem
- vylepšují neustále starý kód (refactoring)

Některé z nejpoužívanějších agilních metodik jsou EXtreme Programming (XP), Scrum, Feature-Driven Development (FDD) a Test-Driven Development (TDD).

2.2.1 EXtreme Programming (XP)

Extrémní programování je metodika pro malé až středně velké programátorské týmy (2-10), kteří se vyvíjejí softwarové projekty se zadáním, co se mohou často a rychle měnit.

Některým lidem se jeví XP jako střízlivé a praktické uvažování. Většině se jako takové nejeví. V zásadě je možné říci, že XP používá určité známé postupy, které dovádí do extrémů. [8]

XP slibuje programátorům, že:

- budou neustále pracovat na věcech, které jsou skutečně podstatné.
- s nepříznivými situacemi se nebudou vyrovnávat sami.
- budou rozhodovat o věcech, kterým rozumějí a nebudou muset dělat věci, na které nemají schopnosti.

XP slibuje zákazníkům, že:

- z každého týdne programování vytěží maximální možnou hodnotu.
- v intervalu několika týdnů uvidí pokrok, který je zajímá.
- mají možnost změnit směr projektu uprostřed vývoje, aniž to způsobilo abnormálně vysoké náklady

Jak to XP chce dosáhnout:

- Programový kód píší dvojice programátorů za jedním počítačem
- Nejprve se píší testy, pak k nim zdrojový kód.
- Napsaný kód se integruje ihned po odladění.
- Pracovat maximálně 40 hodin týdně.
- Programy jsou neustále připravované na předání zákazníkovi.
- Co je rozumné přeprogramovat (refaktorizace), provede se ihned, jak je na to čas.
- Programuje se pouze to, co je nezbytně nutné.
- Vývoj programu řídí neustále iterace.
- Každý programuje všechno.

2.2.2 Scrum

Scrum je iterativní, přírůstkový proces pro vývoj jakéhokoli produktu nebo řízení jakékoli práce. Scrum se soustřeďuje na to, jak by členové týmu měli fungovat, aby vytvořili pružnost systému v neustále se měnícím prostředí. Na konci každé iterace vytváří potenciální soubor funkcí.

Scrum nevyžaduje ani neposkytuje žádné konkrétní metody/postupy vývoje softwaru, které by měly být používány. Místo toho vyžaduje určité manažerské postupy a nástroje v různých fázích Scrum, aby se zabránilo chaosu nepředvídatelností a složitostem.

Hlavní charakteristiky Scrumu [9, s. 112]:

• Product Backlog - Toto je prioritní seznam všech funkcí a změn, které je třeba provést v systému požadovaném několika účastníky, jako jsou zákazníci, marketingové, prodejní a projektové týmy. Zodpovědnost za produkt nese jeho majitel.

• Sprinty - Sprinty jsou dlouhé 30 dnů (délka se může lišit podle projektů), je to postup přizpůsobení se měnícím se environmentálním proměnným (požadavky, čas, zdroje, zna-

losti, technologie atd.). Pracovními nástroji týmu jsou Plánovací sprint setkání, Sprint Backlog a Denní Scrum setkání.

• Sprint Planning meeting - schůzky se nejprve účastní zákazníci, uživatelé, management, vlastník produktu a Scrum Team, kde se rozhoduje o souboru cílů a funkčnosti. Dále se Scrum Master a Scrum Team zaměřují na to, jak je produkt implementován během sprintu.

• Sprint Backlog - Je to seznam funkcí, které jsou aktuálně přiřazeny konkrétnímu sprintu. Po dokončení všech funkcí je dodána nová iterace systému.

• Denní Scrum - Jedná se o každodenní schůzku po dobu přibližně 15 minut, která je organizována tak, aby sledovala pracovní náplň týmu a řešila jakékoliv překážky, kterým tým čelí.

2.2.3 Feature-Driven Development (FDD)

Metodika FDD je založena na iterativním vývoji, který je řízen užitnými vlastnostmi produktu (feature-driven). FDD je iterativní proces s krátkými iteracemi založený na modelování. Začíná vytvořením celkového modelu a pokračuje posloupností dvoutýdenních iterací, ve kterých se provádí návrh i realizace pro jednotlivé užitné vlastnosti. Užitná vlastnost (feature) je malý výsledek užitečný z pohledu zákazníka, je srozumitelný, měřitelný a realizovatelný v rámci dvoutýdenní iterace. FDD můžeme charakterizovat jako agilní, adaptivní proces vývoje softwaru, který [10]:

• je vysoce iterativní,

• zdůrazňuje kvalitu v každém kroku,

• dodává časté, hmatatelné a fungující výsledky na každé úrovni projektu od vývojáře po vedoucího projektu,

• s minimální zátěží pro vývojáře poskytuje přesné a smysluplné informace o stavu projektu,

• je oblíbený u klientů, kteří vidí výsledky včas a mohou sami posoudit postup projektu, manažerů, kterým poskytuje přesné a smysluplné informace o stavu projektu, i vývojářů, kteří rádi pracují v malých týmech a v krátkých iteracích.

2.2.4 Test-Driven Development (TDD)

Jedna z agilních metodik, která je v praxi běžně používaná, se nazývá Vývoj řízený testy (z ang. Test driven development), kdy se vůči známému rozhraní nejprve napíší testy a až tak se vytváří samotná implementace. Výsledný kód a jeho aktualizace jsou tak neustále kontrolovány vůči očekávanému chování zapsanému v testech. Pokud dojde změnou kódu k chybě, ta je okamžitě odhalena při spuštění testů.

Princip práce při vývoji řízeném testy se shrnuje do tohoto dobře zapamatovatelného sousloví: Červená, Zelená, Refaktorování. Názvy barev v tomto sousloví mají souvislost s uživatelským rozhraním populárního nástroje pro automatizované spouštění testů, JUnit, kde červený proužek znamená, že test selhal a naopak zelený svědčí o úspěšnosti provedení testu. Popišme si tedy základní kroky podrobněji:

- Napište test, napoprvé je jasné, že test určitě selže nebo nepůjde vůbec zkompilovat.
- Rychle implementujte testovanou logiku tak, aby test proběhnul.
- Pomocí refaktorování a opakovaného spouštění testu upravte kód do přijatelné podoby.

Tato trojice je alfou a omegou programování řízeného testy. Z tohoto také plyne rovnice vývoje řízeným testy = test-first přístup + refaktorování. [11, s. 65]

3 TESTOVÁNÍ

Testování můžeme označit jako základní nástroj pro zajištění robustnosti a správnosti aplikace. Zatímco unit testy kontrolují funkčnost jednotlivých modulů aplikace, akceptační testy nabízejí komplexnější přehled a kontrolují, zda software odpovídá specifikaci a požadavkům zákazníka.

Při klasických metodách vývoje probíhá testovací fáze následovně: Manažer testování vytvoří plán testování. Začnou se znovu procházet požadavky a specifikace. Testeři začnou psát testovací skripty pro jednotlivé případy testování. Následně se čeká na dokončení softwaru, často dokud je příliš pozdě na důkladné otestování. Po otestování se začínají řešit problémy objevené během testování. Problémem při takovém vývoji je, že testeři jsou zapojeni do vývoje projektu až těsně po ukončení vývoje (vodopádový model), případně před ukončením jednotlivých iterací (spirálový model). Agilní metody vývoje softwaru se to snaží zlepšit tím, že testeři jsou součástí vývojového týmu po celou dobu vývoje a úzce spolupracují s celým týmem. Při test-driven development se testování dostává do čela a tvoří základy pro celý vývoj projektu. Testy jsou vytvářeny ještě před začátkem implementace samotné funkcionality. Toto umožňuje implementovat tak, aby funkcionalita vyhovovala testem. Může se tak značně snížit potřeba předělávat kód.

V klasických metodách vývoje je oficiálně každý zodpovědný za kvalitu. Když se však něco pokazí (software nevyhovuje specifikaci, neobjevené chyby), vzniká problém, kdo konkrétně je zodpovědný. Návrhář, který nepočítal s komplexní funkcionalitou? Vývojář, který naimplementoval málo rozšiřitelný kód? Nebo tester, který vytvořil nedostatečné testy. Při agilním vývoji však testy vytváří a provádí celý tým. Vývojář, analytik, dokonce i zákazník. Přenáší se tak odpovědnost za požadovanou funkcionalitu z testerů na celý tým. Jelikož každý vývojář vytváří testy, zvyšuje se celková testovatelnost softwaru. Dá se tedy říci, že testování představuje nejsilnější stránku agilního vývoje softwaru. Ne všechny požadavky na software se však musí vztahovat na funkcionální stránku softwaru a dají se snadno otestovat. [12]

3.1 Způsoby testování podle stavu aplikace

Podle toho, zda je k testování nutné spuštění aplikace, rozlišujeme statické a dynamické testy.

3.1.1 Statické testy

Statické testy nevyžadují běh software. Využívají se tedy zejména v raných fázích životního cyklu software, kdy ještě není vytvořen funkční prototyp aplikace. Lze je použít ještě před začátkem psaní kódu na kontrolu specifikace požadavků a analýzu zdrojového kódu.

3.1.2 Dynamické testy

Dynamické testy vyžadují spuštění aplikace. Potřebují proto alespoň spustitelný prototyp software. Používají se v pozdějších fázích vývoje a jsou zaměřeny na provoz software.

3.2 Postupy testování podle znalosti kódu

Podle míry znalosti kódu rozlišujeme dva postupy testování – testy bílé skřínky (angl. White box testing) a testy černé skřínky (angl. Black box testing). Občas se můžeme setkat s kombinací obou kategorií, ta bývá nazývána jako testy šedé skříňky (angl. Grey box testing).

3.2.1 White box testování

Testování bílé skříňky se zaměřuje na vnitřní logiku a strukturu systému, proto se někdy označuje také jako strukturální. Testovaný systém si můžeme představit jako průhlednou skříňku, do které vidíme a známe všechny její součásti. testování bílá skříňka se zakládá na důkladné znalosti implementace systému. protože vyžaduje znalost kódu aplikace, může probíhat okamžitě jako je vytvořena testována část kódu. Při tomto testování se může provádět analýza toku dat, toku řízení, struktury kódu a zpracování výjimek a chybových stavů na otestování požadovaného chování systému.

3.2.2 Black box testování

Při testování černé skříňky tester vychází ze specifikace nebo požadavků systému, bez znalosti o vnitřní struktuře. Testovaný systém představuje černou skříňku, která je určitým způsobem nakonfigurována a přijímá vstupy, většinou přes uživatelské rozhraní nebo webový prohlížeč. Tester zkouší zadávat různé vstupy a sleduje výstupy, které porovnává z očekávaných výsledků.

Výše nákladů má zde logaritmickou stupnici – to znamená, že s postupem času roste vždy na desetinásobek. Chyba, kterou jsme odhalili a opravili ještě v raných stádiích sestavování

specifikace, nemusí stát téměř nic, nebo v našem příkladu 10 centů. Jestliže stejnou chybu objevíme až během kódování (programování) nebo testování softwaru, bude stát 1 až 10 dolarů. A jestliže ji najde až zákazník v hotovém produktu, vyskočí náklady klidně na 100 dolarů i více.

3.3 Rozdělení testů podle způsobu jejich provádění

Podle toho, kdo a jak testy provádí, rozlišujeme:

3.3.1 Manuální testování

Testování prováděné přímo testery. Jednotlivé testy jsou prováděny a vyhodnocovány manuálně. Tester podle připravených podkladů provede všechny potřebné kroky a zaznamená výsledek.

3.3.2 Automatizované testování

Testy prováděné prostřednictvím specializovaných softwarových nástrojů. Jednotlivé testy jsou připraveny a nahrány jako skripty do automatizovaných nástrojů.

3.4 Cena za kvalitu

Podle standartu ISTQB vstupují do ceny za kvalitu (či přesněji „ceny za nekvalitu“) následující kategorie nákladů [13, s.18]:

- Náklady využití na prevenci defektů – sem patří např. školení vývojářů, náklady na zavedení kódovacích standartu.
- Náklady na nalezení defektů – zahrnují veškeré testovací od plánování testů, přípravy testovacích případů, revize specifikací, přípravu testovacího prostředí, jejich vykonávání až po reporting.
- Náklady vyvolané řešením defektů v době testování (angl. cost of internal failure) – sem patří náklady na opravu defektů ve specifikacích, opravy defektů v kódu a na ně nabalené náklady dalších rolí (nasazování do prostředí, přetestování, atd.)
- Náklady vyvolané selháním v produkčním prostředí (angl. cost of external failure) – zahrnují nejen přímé náklady na veškeré aktivity související s opravou případných produkčních defektů a distribucí opraveného systému uživatelů. Velkou po-

ložkou v této škatulce mohou být náklady na sankce (např. smluvní pokuta účtovaná odběratelem), náklady na ztrátu reputace atd.

Náklad na zajištění kvality před nasazením do produkčního prostředí jsou dané součtem první tří položek:

Cena zajištění kvality = Náklady na prevenci + Náklady na nalezení defektů + Náklady na opravu

Přínos těchto investic do kvality před uvedením systému do produkčního provozu vůči možným nákladům, které by bylo nutné vynaložit, kdybychom tyto investice/aktivity nerealizovali, pak ukazuje cena za kvalitu:

Cena za kvalitu = Náklady vyvolané možným produkčním selháním – Cena zajištění kvality

Pro argumentaci opodstatnění investic do aktivit pro zajištění kvality je tedy pochopitelně nutné, aby byla cena za kvalitu kladným číslem. Zjednodušeně řečeno: má smysl testovat jen do té míry, pokud jsou náklady na testování nižší než náklady na řešení případných produkčních incidentů.

3.5 Chyby v softwaru

Jak nám říká Murphyho zákon programování - neexistuje program, který by byl zcela prost chyb. Dodatek k tomuto zákonu poté praví, že každé odstranění libovolné chyby zanese do programu chybu jinou, skrytou a zákeřnější. To znamená, že není možné vytvořit program, který by byl zcela bez chyb. Libovolný softwarový produkt bude vždy obsahovat nějaké chyby.

O chybě hovoříme, když je splněna jedna nebo více z následujících pravidel [12, s.14]:

- Software nedělá něco, co by dle specifikace měl dělat.
- Software dělá, něco, co by podle údajů specifikace neměl dělat.
- Software dělá něco, o čem se specifikace nezmiňuje.
- Software dělá něco, o čem se specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo – podle názoru testera – jej koncový uživatel nebude považovat za správný.

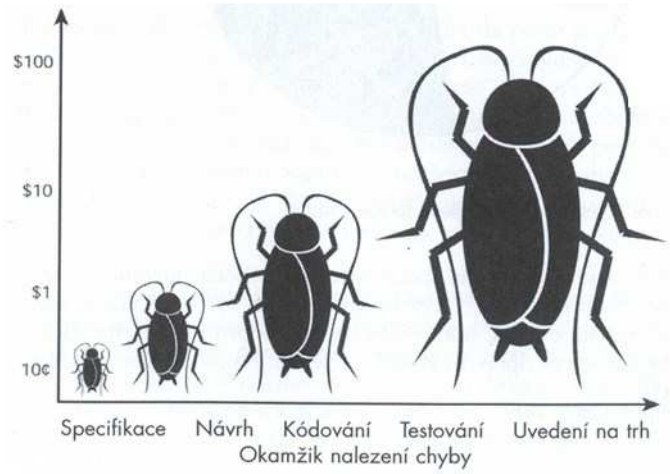
Některé druhy chyb:

- algoritmická chyba

- chyba syntaxe
- chyba výpočtu a přesnosti
- chyba dokumentace
- chyba stresu nebo přetížení
- chyba kapacity nebo meze
- časová nebo součinnostní chyba
- chyba propustnosti nebo výkonu
- chyba zotavení
- chyba HW a systémového SW
- chyba nedodržení standardů a procedur

3.5.1 Nárůst nákladů na vývoj softwaru

Historicky je dáno, že proces identifikace a odstranění chyb v procesu vývoje softwaru reprezentuje více než polovinu nákladů na celý vývoj. V závislosti na použité metodě testovací aktivity zabírají od 30 do 90 procent práce vydané k vytvoření funkčního programu. Včasné odhalení chyby může výrazně redukovat náklady. Chyby mohou být klasifikovány na základě toho, kde byly objeveny, to znamená v jednotlivých etapách životního cyklu vývoje softwarového produktu. Konkrétně se jedná o etapy požadavky návrh, kódování, testování modulů, integrační testování, systémové testování, instalační a akceptační testované a provozování a údržba. Na obrázku vidíme, jak postupem času rostou náklady na opravu těchto chyb.



Obr. 3 Rostoucí náklady na opravu chyb

4 ODHADY V SOFTWAROVÝCH PROJEKTECH

U softwarových projektů je problém mít představu o tom, jak dlouho každá aktivita trvá, protože zde panuje značné množství nejistoty a i malé změny v technologii či týmu mohou mít zásadní dopad na trvání aktivity. Proto mluvíme o odhadech.

Při plánování třeba odpovědět na tyto otázky:

- jaké úsilí je třeba vynaložit na provedení určité činnosti,
- kolik času je třeba k provedení činnosti,
- jaké jsou náklady na provedení činnosti.

Odhad je nejpravděpodobnější hodnota, kterou bude nabývat určitý ukazatel. Odhad může být

- empirický: analýza údajů za účelem stanovení vztahů mezi jednotlivými charakteristikami
- analogický: použití známých měření na odhad
- teoretický: vytvoření numerického modelu, který se ověřuje zpravidla empiricky
- heuristický: rozšíření ostatních metod, použití expertních systémů

Náklady a zdroje je třeba znát na začátku, ale přesně určit jejich potřebu je možné až při provádění projektu, resp. při jeho ukončení. Odhad je vždy zatížen určitou chybou, upřesňuje se s postupem projektu. Kvalitu odhadu lze stanovit až porovnáním očekávaných a skutečných hodnot (pokud je chyba menší než 20% už se mluví o dobrém odhadu). Cílem odhadu je předpověď nejpravděpodobnějšího budoucího vývoje sledované charakteristiky projektu (nákladů, času trvání). Splnění odhadu není cílem projektu. S odhadem souvisí měření (přiřazování čísel nebo symbolů atributem objektů). Měření je důležité při rozhodování a také při určení přesnosti odhadu. Podrobněji viz kapitola Měření v softwarovém projektu. Proces předpovědi a její vylepšování musí zahrnovat následující činnosti:

1. Výběr charakteristik, které se budou odhadovat

2. Provedení odhadu (předpovědi)

3. Vyhodnocení správnosti odhadu (zpětná vazba)

Metody odhadu v softwarovém projektu:

- shora dolů: možnost podcenění "drobných" problémů (tzv. implementační detaily)

- zdola nahoru: možnost podcenění činností týkajících se celého systému (např. integrace, řešení rizik); je náročnější, třeba mít už nějaký návrh systému, tj. rozdělení do komponent.

4.1 Techniky odhadu

4.1.1 Modelování

Rozlišujeme

1. parametrické modely
2. neparametrický modely (např. metody strojového učení, neuronové sítě)

Snaha o určení, které (měřitelné) parametry ovlivňují délku trvání činností a odhalit vztah mezi těmito parametry a délkou trvání činností. Modely závisí na použitém procesu, tj. pokud se změní proces třeba zpravidla změnit i model.

4.1.2 Posouzení expertů

Použijí se historické údaje z předchozích ukončených projektů. Jedná se často o dost nespolehlivý postup. Lze zlepšit použitím údajů ze skutečně podobného projektu a zajištěním, aby jednotlivci sepisující odhad byli experti v dané oblasti. Postupuje se zpravidla shora dolů.

Použití DELPHI metody při expertním odhadu nákladů [14]:

1. Experti dostanou specifikaci softwaru a formulář odhadu.

2. Diskuse o produktu a metodách odhadu.

3. Vytvoření individuálních odhadů.

4. Zápis odhadů do tabulky a vrácení expertem (identifikují se pouze expertovy osobní odhady).

5. Distribuce odhadů a diskuse.

6. Přehodnocení odhadů - cyklus pokračuje bodem 3, dokud nedojde k dohodě.

4.1.3 Délka textu programu

Je to nejjednodušší metrika rozsahu programu. Jsou různé přístupy pro měření délky textu programu. Určitý způsob se zpravidla váže na jednu organizaci a jeden programovací jazyk. [15, s. 25]

Nejčastěji se používají tyto způsoby vyjádření délky programu:

- počet oddělovačů (;)
- počet dodaných příkazů (DSI - Delivered Source Instructions)
- počet vytvořených řádků, komentáře a prázdné řádky se nepočítají (LOC - Lines of Code)
- počet znaků v textu programu
- počet bajtů potřebných pro uchování programu
- počet stran výpisu programu.

4.1.3.1 Výhody

- jednoduchost a široká použitelnost
- možnost použití historických dat (neboť existují)
- možnost automatického sledování

4.1.3.2 Nevýhody

- potíže při sledování různých jazyků
- nepostihuje jakost a může záviset na stylu zápisu.

4.1.4 Funkční body (angl. Function points)

Vychází ze specifikace požadavků (pro aplikace komunikující s uživatelem) a určuje počet a rozsah uživateli přístupných služeb systému (tyto aspekty lze stanovit dost přesně už na začátku projektu).

Používá se na odhad rozsahu softwarového systému, zejména pro interaktivní databázově-orientované systémy (např. Obchodní aplikace). Nehodí se pro systémy se složitým vnitřním zpracováním. Technika funkčních bodů nezávisí od programovacího jazyka. [15, s. 32]

4.1.4.1 Výhody

- možnost odvození přímo ze specifikace
- nezávislost na programovacím jazyce a technologii (používají se tabulky převodu funkčních bodů na řádky textu zdrojového programu pro jednotlivé programovací jazyky)

4.1.4.2 Nevýhody

- dost spoléhá na uživatelské posuzování rozsahu (nemusí vždy odpovídat skutečnosti)
- potíže a subjektivita při určování charakteristik a jejich stupně složitosti
- nedá se automatizovat
- při opětovném používání softwarových součástek nevyjádří skutečnou produktivitu, resp. náklady.

4.1.5 Body případů použití (angl. Use case points)

Rozšíření techniky funkčních bodů na odhad rozsahu softwarového projektu. Technika nezávisí od programovacího jazyka. [15, s. 36]

4.1.5.1 Výhody

- možnost automatizace procesu
- možnost stanovení průměru času implementace na jeden případ použití v rámci organizace
- dobré oddělení metriky rozsahu od velikosti týmu

4.1.5.2 Nevýhody

- potřebujeme mít vytvořené případy použití (může to být až 10-20 % úsilí v projektu)
- velké jednotky pro podrobné plánování
- různé odhady složitosti případů použití (co se počítá za transakci)
- některé z faktorů neovlivňují celý projekt, a tedy vnášejí do odhadu chybu

II. PRAKTICKÁ ČÁST

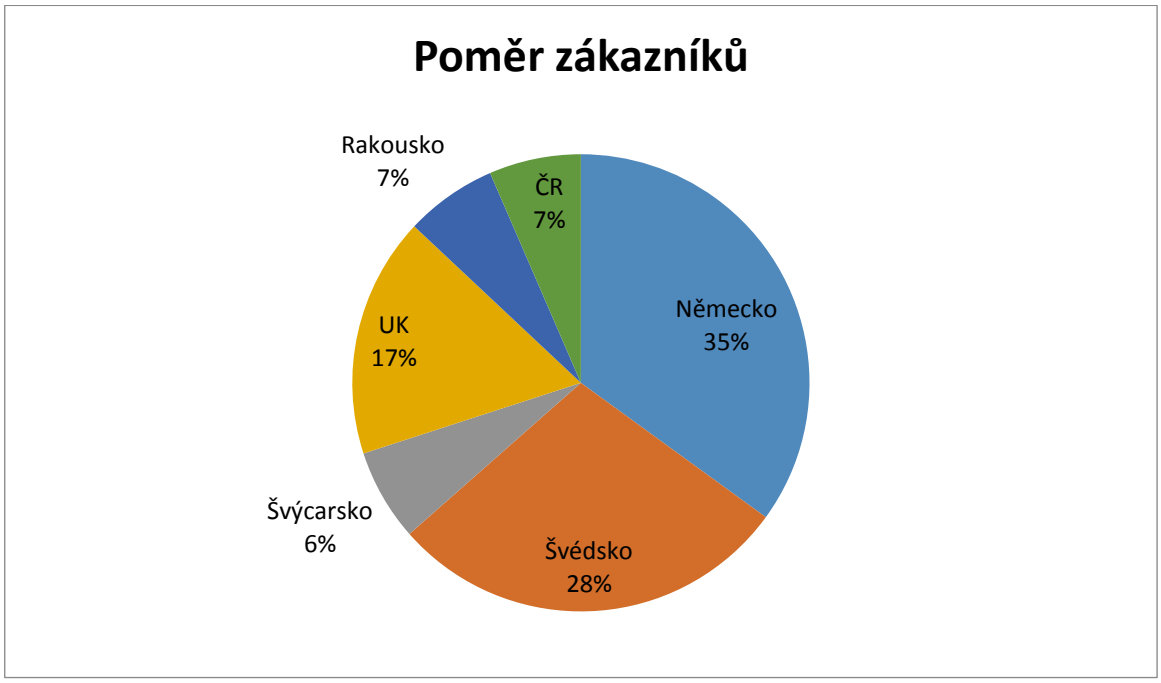
5 CHARAKTERISTIKA FIRMY

Společnost byla založena před 20 lety a nyní zaměstnává více než 100 zaměstnanců v České a Slovenské republice. Firma je odborníkem v oblasti vývoje softwaru, který je její hlavní obchodní činností pro klienty v Evropě (viz. obrázek níže).

Nabízí také služby IT Management Consulting a QA/Testing.

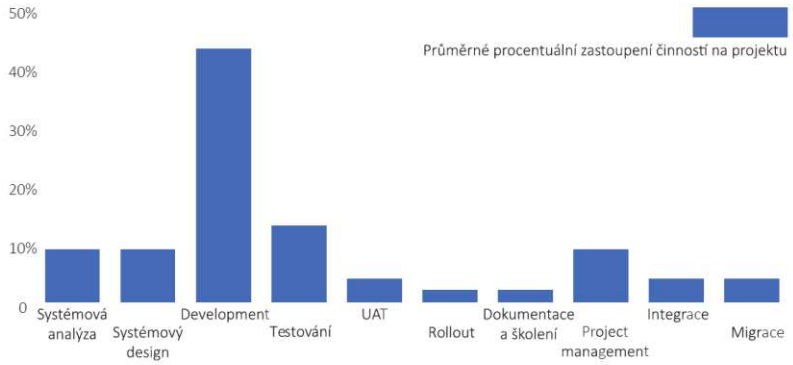
Zákazníci jsou předními firmami v oblastech, jako jsou finance, logistika, média a vydavatelství, bezpečnost a telekomunikace.

V tomto roce také firma plánuje rozběhnout vzdělávací program na pracovišti, dlouhý přibližně 6 měsíců, během kterého zaměstnanci procházejí několika fázemi. Každá fáze je zaměřena na získání specifických zkušeností a budování potřebných dovedností pro profesionální mladé vývojáře a testery.



Obr. 4 Poměr zákazníků podle příjmů z dané země v minulém roce

5.1 Poměrové zastoupení činností na projektu



Obr. 5 Poměrové zastoupení činností na obvyklém projektu firmy

5.2 Životní cyklus projektu

Obr. 6 Softwarový vývoj

Přístup firmy k rozvoji projektů je přizpůsoben způsobu práce zákazníka, ať už se jedná o agilní modely nebo vodopádový model. Firma se zaměřuje na následující jazyky a operační systémy:

- Java (J2EE, J2SE, EJB), Groovy, Graily
- .NET (C #, VB)
- C / C ++
- PHP
- Python
- Javascript
- Linux (RedHat, Debian, Suse...)
- Windows (Desktop, Server, Phone)
- iOS / Android

5.3 Testování softwaru

Firma zajišťuje velké testovací oddělení, kde provádí externí testování softwaru vyráběného zákazníky i softwaru, který si vyrábí sama. Poskytuje i následující služby:

- Automatizované testování
- Prohlížení kódu (např. CodeItRight)
- Test jednotky (například JUnit)
- UI testování (např. QF-Test, Selenium, UFT, TestComplete)
- Řízení zkušební dokumentace (Testlink, Centrum kvality)
- Analýza testů
- Příprava zkušebních plánů a specifikací
- Reportování chyb (např. Mantis, Bugzilla, Redmine) + Statistika

6 CENA ZA SOFTWARE

Do ceny nemusí být v nabídce zahrnuto:

- Hardware a provozní prostředí systému
- Čas cesty členů týmu k zadavateli nebo do datového centra
- Výdaje na dopravu a parkovné
- Licence operačních systémů a databází
- Licence komponent třetích stran
- Vlastnictví kódu zadavatelem
- Vlastnictví dokumentace analýzy zadavatelem
- Zřízení a provoz testovacích prostředí souběžných s ostrým provozem
- Zálohování systému
- Instalace systému u zadavatele
- Nákup specializovaných zařízení pro testování
- Školení
- Následná systémová a uživatelská podpora
- Kolik bude stát další případný vývoj (udává se hodinová sazba vývojáře)

Hodinová sazba programátora

Neupřesněnou položkou bývá hodinová sazba programátora (vývojáře), protože ji lze chápat vice způsoby. Běžné se sazby pohybuji v rozmezí:

- Samostatný programátor na živnostenský list 300-1000 Kč/hod
- Programátor české středně velké sw. společnosti 800-1500 Kč/hod
- Programátor větší sw. společnosti nebo specialista 1300-3000 a vice Kč/hod

Sazba programátora běžné zahrnuje:

- Plat programátora
- Vzdělávaní, profesní školeni
- Osobni hardware a software (notebook, mobilní telefon, operační systém, vývojový software, utility)
- Vybavení vnitřní IT infrastruktury (sítě, vnitropodnikové servery a systémy, testovací se servery a systémy)
- Náklady na provoz kanceláři, připojeni, energie

- Rozpuštění nákladů na režijní pracovníky (účetní, finance, administrativa, management)
- Zisk společnosti

Konečná cena v nabídce nebývá úplně konečná. Je důležité vědět, co je v ní vše započítáno a co zadavatel bude muset doplatit.

7 SWOT ANALÝZA

Silné stránky	Slabé stránky
a) Jazykové znalosti b) Zkušenosti vývojových leaderů c) Flexibilita **d)** Vzdělávací procesy	a) Nedostatek informací b) Malá zastupitelnost pozic a kapacity c) Věkový průměr **d)** Manuální testování
Příležitosti	**Hrozby**
a) Dotace na technologie a vzdělávání b) Rozvoj do dalších segmentů trhu (např. sociální sítě) c) Pokračovat v expanzi v EU **d)** Zajištění dlouhodobé věrnosti zákazníků	a) Nesystémová práce b) Zpoždění vývojových projektů c) Zastaralé technologie d) Vstup konkurenční firmy na trh

8 ANALÝZA SOUČASNÉ REALIZACE PROJEKTŮ

Cílem práce bylo na základě analýzy současné realizace softwarových projektů v podniku identifikovat vzniklé problémy a navrhnout jejich možná finanční řešení.

Pomocí dotazníku a řízeného rozhovoru použité přímo ve vybraném podniku jsem se pokusil analyzovat proces řízení projektu, zjistit nedostatky a navrhnout soubor řešení k jejich odstranění. Formulované otázky (viz. - Příloha 1) byly sestaveny na základě určeného cíle práce a následně zodpovězeny projektovým manažerem ve vybraném podniku. Zdrojem informací byl samotný podnik a čas strávený v něm.

Informace jsem zpracoval na základě odpovědí z rozhovoru i z vlastních zjištění a znalostí získaných vlastním pozorováním během doby setrvání v podniku.

8.1 Zhodnocení výsledků výzkumu

Na základě výsledků z řízeného rozhovoru provedeného s projektovým manažerem vybraného podniku jsem zjistil problémy, které jsou identifikovány v následující části práce.

Problémy/plusy ve fázi plánování projektů:

- Za všechny naplánované projekty jsou zodpovědné 3 osoby.

Jelikož podnik má rozpracovaných několik projektů současně, odpovědnost za ně převezme pouze jedna osoba a to vrcholový manažer projektů.

- Nedostatečná analýza rizik.

Tento uvedený problémy se vyskytuje u všech realizovaných projektů a nebyl dosud řešen ze strany podniku.

- Roste zájem o agilní vývoj.

Plusem přivlastňování agilních metodik vývoje je, že testeři začínají být součástí projektu hned na začátku a můžou se zapojit to plánování od prvního dne.

Problémy ve fázi realizace projektů:

- Chaotické kontrolní setkání a setkání řešící problémy.

Tento problém se vyskytuje hlavně na straně hlavního projektového manažera, ale mohl jsem ho zpozorovat i ze strany zúčastněných osob setkání. Všichni účastníci nevěděli, o čem bude dané setkání, a tedy nebyli schopni promptně reagovat na položené otázky.

- Žádná předpříprava na setkání.

Tento problém se dotýká projektového týmu, kterému v některých případech není předem oznámen záměr setkání, a tudíž jim není umožněno se připravit na toto setkání předem.

- Preferování autoritativního manažerského stylu pouze "shora dolů" ze strany projektového manažera.

Problémy ve fázi ukončení a zhodnocení projektů:

V této fázi jsem nezaznamenal žádné větší problémy, protože proces ukončení probíhá tak jak má, tedy produkt (v našem případě software) je "odzkoušen" a předán zákazníkovi do užívání. V případě zjištěných nedostatků nebo nespokojenosti investora s konečným produktem ihned dochází k řešení vzniklých problémů. Je ukončena a doplněna veškerá dokumentace ohledně projektu, přičemž probíhá i zhodnocení projektu, poučení se ze vzniklých problémů během plánování i realizace projektu a jsou podány návrhy jak se jim v budoucnosti vyhnout. Je poskytována i záruka, která závisí na druhu softwaru.

8.2 Návrhy na zlepšení

Podnik pracuje na několika projektech současně. Tyto projekty se nacházejí v různých fázích svého životního cyklu, a proto vyžadují i samostatný přístup. Vrcholový projektový manažer nemůže podrobně sledovat a kontrolovat každý a jeden, i když v konečném důsledku je za ně odpovědný. Do popředí se nám dostat problém, že v určitém bodě bude odpovídat za takové množství projektů, které nebude schopen zvládnout, nakolik budou vyžadovat jeho neustálou přítomnost a kontrolu a on nebude schopný průběžného hodnocení úkolů a ani kritického posouzení všech důležitých detailů. V podniku je jediná osoba pověřená vrcholovým managementem, proto navrhujeme přerozdělení větší odpovědnosti na jednotlivé liniové manažery a očekáváním svědomitého plnění zadaných úkolů.

Riziko představuje přirozenou součást řízení projektu, a proto by mělo být zahrnuto i do projektového plánu a mělo by být s ním počítáno v každé oblasti. Na základě analýzy projektů jsme zjistili, že daný podnik vypracování analýzy řízení rizik i jejich kontroly značně podceňuje. Každý projekt je do určité míry spojen s rizikem. Jsou rizika, která ovlivnit víme a ty, které ovlivnit neumíme jako například změna politických otázek nebo změna legislativy.

Identifikace rizika by měla proběhnout hned na začátku projektu i při hodnocení průběhu během realizace a při přijímání každého závažnějšího rozhodnutí. Je dobré využít vlastní

zkušenosti i údaje z předešlých námi zrealizovaných projektů. Zaznamenají se potenciální rizika, které vznikají při interakcích činností, procesů, projektovou organizací i zainteresovanými stranami. Všechna určená rizika následně procházejí kvalitativní i kvantitativní analýzou a procesem jejich posouzení, začlení se do úrovně rizik podle jejich závažnosti.

Při realizaci několika projektů je velmi náročné naplánování kontrolních setkání v rámci životního cyklu projektu. Investor požaduje kvantum informací ohledně postupu projektu od řídícího projektového manažera a ten zas od svého projektového týmu. Je velmi problematické sladit několik kontrolních setkání při různých projektech.

Setkání by se měli zúčastnit pouze kompetentní osoby, tedy ti, kterých se problém výsostně dotýká. Je třeba si předem ujasnit (týká se hlavního projektového manažera), koho na daném setkání opravdu potřebuje a kdo je tam pouze "do počtu" a nemá žádnou kompetenci se k danému problému vyjadřovat (například testeři k refaktorizaci kódu). Je třeba si předem ujasnit, proč se setkání koná, tedy zda jde o kontrolní setkání nebo setkání za účelem vyřešení závažného problému. Jaký je očekávaný výsledek daného setkání například změny v plánu nebo návrh řešení zadaného problému, zda jeho samotné vyřešení. I tuto oblast by měl zajistit zejména projektový manažer. Je třeba zvážit, zda je setkání nezbytné nebo je možné ho vynechat, to by mělo být zejména v kompetenci projektového manažera.

ZÁVĚR

Hlavním cílem této bakalářské práce byla navrhnutí zlepšení finančního řízení uvedené firmy. Tato práce se zaměřuje na projekty, modely životního cyklu softwaru a způsoby testování a důsledky při jeho neefektivnosti.

V praktické části se ukázalo, že velký důraz je kladen na komunikaci roste zájem ze strany testerů o implementování agilních metodik vývoje, aby se stali součástí projektu hned na počátku a mohli si stejně jako vývojáři vytvořit podrobný plán.

Výsledná odpověď je, že neexistuje nejvhodnější metodika, ale záleží od rozsahu projektu, rozpočtu, délky trvání, velikosti týmu, počtu účastníku, očekávaných výstupů, atd.

Dělal jsem na více projektech, kde byl aplikovaný klasický vodopádový model a bylo to podle všeho správné řešení. A setkal jsem se i s projekty, kde byl oficiálně Scrum + TDD. Podle mých zkušeností závisí více na schopnostech managementu a programátorů než na konkrétní metodě vývoje.

Platí následovné: při řízení malého týmu, se snažme využít jeho silné stránky jako je rychlost a přímočarost komunikace a bleskově zpětná vazba. Platí "komunikace před procesy". Zde mohou agilní metody jako Scrum dost naučit. Naopak u velkého projektu se silně začne projevovat přínos procesů a všeobecná znalost manažerských technik, které jdou nad rámec vývoje SW resp. IT.

SEZNAM POUŽITÉ LITERATURY

[1] SCHWALBE, Kathy, 2011. *Řízení projektů v IT - Kompletní průvodce*. 1.vydání. Praha: CPRESS. ISBN 978-80-251-2882-4.

[2] ČSN ISO 10006 ed. 2 (010333). *Systémy managementu jakosti - Směrnice pro management jakosti projektů*. Praha: Český normalizační institut, 2004.

[3] Guckenheimer, Sam a Juan J. PEREZ, 2007. *Efektivní softwarové projekty*. 1. vydání. Brno: Zoner Press. ISBN 978-80-86815-62-6.

[4] MANAGEMENTMANIA, 2015. *Vodopádový model*. ManagementMania.com. [online]. [cit. 2017-03-12]. Dostupné z: https://managementmania.com/cs/vodopadovy-model-waterfall-model

[5] SLIDEGUR, 2017. *Spirálový model*. Slidegur.com. [online]. [cit. 2017-03-12]. Dostupné z: http://slidegur.com/doc/1627030/spirálový-model

[6] KADLEC, Václav, 2003. *Nevěříte Extrémnímu programování? Zkuste klasiku: Rational Unified*. Zive.cz. [online]. [cit. 2017-03-19]. Dostupné z: http://www.zive.cz/clanky/neverite-extremnimu-programovani-zkuste-klasiku-rational-unified-process/sc-3-a-112889/

[7] BECK, Ken et al., 2001. *O Manifestu*. Agilemanifesto.org. [online]. [cit. 2017-03-12]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html

[8] KADLEC, Václav, 2003. *Extremismus nemusí být na škodu: extrémní programování*. Zive.cz. [online]. [cit. 2017-03-19]. Dostupné z: http://www.zive.cz/clanky/extremismus-nemusi-byt-na-skodu-extremni-programovani/sc-3-a-111714/

[9] OPELT, Andreas et al., 2013. *Agile Contracts: Creating and Managing Successful Projects with Scrum*. 1. vydání. Hoboken: Wiley. ISBN 978-1-118-63094-5.

[10] BUCHALCEVOVÁ, Alena, 2005. *Metodika Feature-Driven development neopouští modelování a procesy, a přesto přináší výhody agilního vývoje*. nb.vse.cz. [online]. [cit. 2017-04-02]. Dostupné z: http://nb.vse.cz/~buchalc/clanky/tsw2005.pdf

[11] BECK, Kent, 2004. *Programování řízené testy*. Praha: Grada Publishing. ISBN 80-247-0901-5.

[12] PATTON, Ron, 2002. *Testování softwaru*. 1. vydání. Brno: Computer Press. ISBN 80-7226-636-5.

[13] BUREŠ, Miroslav et al., 2016. *Efektivní testování softwaru*. 1. vydání. Praha: Grada Publishing. ISBN 978-80-247-5594-6.

[14] MANAGEMENTMANIA, 2015. *Metoda Delphi*. ManagementMania.com. [online]. [cit. 2017-04-15]. Dostupné z: https://managementmania.com/cs/metoda-delphi

[15] MCCONNELL, Steve, 2006. *Odhadování softwarových projektů*. 1.vydání. Brno: Computer Press. ISBN 80-251-1240-3.

[16] PAGE, Alan, Ken JOHNSON a Bj ROLLINSON, 2009. *Jak testuje software Microsoft*. 1.vydání. Brno: Computer Press. ISBN 978-80-251-2869-5.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DSI Delivered Source Instructions

FDD Feature-Driven Development

QA Quality assurance

HW Hardware

IEC International Electrotechnical Commission

ISO International Organization for Standardization

ISTQB International Software Testing Qualifications Board

IT Information technology

LOC Lines of Code

RUP Rational Unified Process

SW Software

TDD Test-Driven Development

XP EXtreme Programming

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: ŘÍZENÝ ROZHOVOR ROZDĚLENÝ DO TEMATICKÝCH BLOKŮ

Podnik

1. Jaký je strategický cíl podniku?

2. Na jaké projekty je převážně zaměřen váš podnik?

3. Věděli byste definovat silné, slabé stránky vašeho podniku?

4. V čem vidíte příležitosti a co byste označili za hrozby pro váš podnik?

5. Věděli byste odhadnout poměr úspěšně a neúspěšně zrealizovaných projektů?

6. Jakou má váš podnik pověst na trhu?

7. Odmítli jste někdy nějakou nabídku realizovat projekt?

8. Kolik projektů máte momentálně rozpracovaných?

Plánování projektů

1. Jak u vás v podniku probíhá proces plánování projektů?

2. Zadá vám investor nebo zadavatel projektu podmínky na co klade největší důraz při realizaci projektu, tedy kvalita, čas nebo prostředky?

3. Převezmete odpovědnost za projekt, který naplánoval někdo jiný?

4. Sestavujete plány sám nebo necháte ostatní, aby si sestavili vlastní plány?

Realizace projektů

1. Máte jako vedoucí manažer projektového řízení problém s komunikací s lidmi?

2. Jak často se setkáváte ohledně realizovaných projektů?

3. Realizujete nějakou předúpravu před daným setkáním nebo řešíte problémy až na samotném setkání?

4. Víte lidí dostatečně motivovat?

5. Vytváříte si při plánování a realizaci projektu i rezervy? (Finanční, časové, lidské zdroje)

6. Jako hlavní projektový manažer máte přehled o všech realizovaných projektech.

7. Jak budete postupovat v případě, že se Vám něco stane a nebudete schopen delší dobu vykonávat dohled nad projekty?

8. Jak u vás probíhá proces kontroly realizovaného projektu?

9. Jak často provádíte kontrolu, tedy porovnávání současného stavu projektu s plánem?

10. Jak postupujete v případě, že odchylky oproti plánu jsou velmi velké?

11. Nedělá Vám problém kontrolovat několik projektů najednou?

Ukončení projektu

1. Jaký je u vás proces ukončení projektu?

2. Jak řešíte nespokojenost se realizovaným projektem u zákazníka?

3. Jak řešíte vzniklé nedostatky po zrealizování projektu a jeho předání zákazníkovi?