Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika
Obor: Informatika

Testování softwaru v agilních projektech

BAKALÁŘSKÁ PRÁCE

Student : Robert Rojo
Vedoucí : doc. Ing. Alena Buchalcevová, Ph.D.
Oponent : Ing. Martin Čížek, MBA

2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 6. května 2015 . . . . . .

Robert Rojo

Poděkování

Rád bych poděkoval vedoucí práce doc. Ing. Aleně Buchalcevové, Ph.D. za cenné připomínky a rady k této práci. Dále bych chtěl poděkovat všem kolegům ze společnosti Orchitech Solutions, s.r.o. za to, že mi umožnili získat praktické zkušenosti v tomto oboru.

Abstrakt

Tato bakalářská práce popisuje proces testování softwaru v agilních projektech a navrhuje rozšíření metodiky MMSP (Metodiky pro malé softwarové projekty). V teoretické části práce jsou uvedeny základní pojmy a praktiky z oblasti testování, je představen agilní vývoj včetně agilního testování a je také popsána metodika MMSP. V praktické části práce je popsán referenční podnik a jsou identifikovány problémy spojené s testováním v tomto podniku. Nakonec jsou navržena a zhodnocena rozšíření metodiky MMSP za účelem vyřešení těchto problémů.

Klíčová slova

Testování softwaru, agilní vývoj, metodika, MMSP, proces vývoje softwaru

Abstract

This bachelor thesis describes the software testing process in agile projects and suggests several improvements in the MMSP methodology (Methodology for small software projects). In the theoretical part of the thesis there are mentioned basic terms and practices in the field of software testing, there is also a description of agile development and agile testing in general and the part also contains an introduction to MMSP methodology. In the practical part of the thesis, a referential business is presented along with the problems that are connected with internal testing processes. In the end of this thesis there are several improvements to the MMSP methodology suggested and evaluated.

Keywords

Software testing, Agile development, Methodology, MMSP, Software development process

Obsah

1 Úvod

Testování softwaru je nedílnou součástí procesu vývoje softwaru. Kvalitně otestovaný software přináší hodnotu nejen zákazníkovi, ale také podniku, který software vyvíjí či dodává. Protože při testování softwaru není vždy možné odhalit všechny chyby, ať už z důvodu omezeného rozpočtu projektu či různých technologických omezení, je klíčové, aby tester dokázal rozhodnout, které části aplikace je potřeba otestovat prioritně. Z důvodu existence jmenovaných omezení je v agilním vývoji někdy potřeba vypustit běžné činnosti testera, mezi které patří například tvorba testovacích scénářů. Tester se v řadě případů nemůže předem připravit na testování rozpracované části aplikace, protože neexistuje vyčerpávající popis přidávané funkcionality. Tester proto musí komunikovat s programátory, zákazníkem a také ostatními členy vývojového týmu, aby zajistil správné a požadované chování aplikace. (Crispin, a další, 2009)

Práce je určena především začínajícím testerům, kteří působí v malých podnicích vyvíjejících software na zakázku pomocí agilních metodik, a kteří během testování naráží na specifické problémy agilního vývoje, mezi které patří například nejednotná, nebo dokonce neexistující správa testů. Tato práce by jim měla pomoci vyřešit zmiňované problémy a zlepšit tak kvalitu testování v jejich podniku.

1.1 Cíle a členění práce

Hlavním cílem této bakalářské práce je navrhnout a následně zhodnotit rozšíření metodiky MMSP[1] v oblasti testování pro účely referenčního malého podniku vyvíjejícího software pomocí agilních metodik. Pro naplnění tohoto cíle byly zvoleny následující dílčí cíle:

1. Vymezit roli testování v rámci agilního vývoje softwaru
2. Popsat současný stav testování softwaru v referenčním malém podniku
3. Identifikovat problémy spojené s testováním v referenčním malém podniku
4. Navrhnout rozšíření metodiky MMSP za účelem vyřešení identifikovaných problémů
5. Zhodnotit navržené změny z hlediska míry vyřešení identifikovaných problémů

Práce je rozdělena na dvě části – teoretickou a praktickou. Teoretická část této práce je věnována vymezení základních pojmů v oblasti vývoje a testování softwaru, které jsou nezbytné pro pochopení problematiky. Dále tato část pojednává o specifických rysech agilního vývoje softwaru jak obecně, tak se zaměřením na oblast testování. Zmíněny jsou také rozdíly mezi tradičním a agilním přístupem k vývoji softwaru. Součástí teoretické části práce je také popis metodiky MMSP.

Jedním z dílčích cílů praktické části práce je identifikování problémů spojených s testováním v referenčním malém podniku. V úvodu praktické části je referenční podnik čtenáři blíže představen a jsou tak nastíněny možné nedostatky a problémy, které mohou znesnadňovat

[1] Metodika pro Malé Softwarové Projekty (MMSP) vychází z metodiky OpenUp a je doplněna o agilní přístupy a praktiky z jiných metodik. O MMSP blíže pojednává *kapitola 4 – Metodika MMSP.*

testování softwaru. Problémy jsou dále popsány, je nastíněna jejich příčina a je navrženo jejich možné řešení. Na základě možných řešení jsou dále navrženy a zhodnoceny úpravy této metodiky, za účelem vyřešení identifikovaných problémů představeného referenčního podniku.

1.2 Metody dosažení cílů

Základem pro dosažení stanovených cílů bylo nastudování uvedené literatury, získání praktických zkušeností v zaměstnání a rovněž získání teoretických a praktických znalostí absolvováním několika semestrálních kurzů, včetně kurzu *4IT354 - Základy testování SW aplikací* na Vysoké škole ekonomické v Praze. Další důležitou součástí byla analýza metodiky MMSP a analýza problémů referenčního podniku.

1.3 Vymezení problémové oblasti

Oblast testování softwaru je velice široká, proto je pro potřeby této bakalářské práce nutné zkoumanou oblast omezit. Tato práce je zaměřena především na testování vyvíjené aplikace z pohledu uživatele, částečně se také dotýká automatizace testů. Protože práce pojednává o testování v agilním vývoji, jsou do procesu testování zahrnuty některé další činnosti, mezi nimiž se nachází například kontrola kvality dokumentace. Zkoumaná oblast pokrývá uživatelské manuální a automatizované testování v softwarových projektech, v úvodu nejprve obecně, a poté se zaměřením na agilní principy.

1.4 Rešerše literatury

Rešerše prací jsou rozděleny do dvou částí. V první části jsou uvedeny akademické práce, které se alespoň zčásti dotýkají zkoumané oblasti a které mohou být pro účely této práce přínosné. Druhá část je věnována odborným knihám, zaměřeným na testování softwaru. Některé z těchto knih se přímo zabývají agilním testováním.

1.4.1 Akademické práce

Problematikou testování softwaru se zabývá velké množství diplomových a bakalářských prací. V Tab. 1-1 jsou proto uvedeny pouze některé práce, které spadají do problémové oblasti vymezené pro tuto práci.

Tab. 1-1 Výběr akademických prací zabývajících se problematikou testování softwaru (autor)

Typ práce	Rok obhajoby	Autor práce	Téma práce
DP	2014	Ing. Jan Černý	Analýza problémů agilních projektů firmy
DP	2010	Ing. Tomáš Fiurášek	Návrh metodiky testování webových aplikací
DP	2012	Ing. Lukáš Langr	Návrh realizace procesu zajištění kvality softwaru v malé firmě
DP	2010	Ing. Jakub Libík	Zlepšování softwarových procesů v oblasti testování

DP	2014	Ing. Tomáš Pavelka	Nástroje pro podporu testování a jejich praktické využití
DP	2010	Ing. Petra Rejnková	Lokalizace a přizpůsobení metodiky OpenUP
BP	2012	Bc. Yauhen Rybak	Testování softwaru
DP	2010	Ing. Robin Štolc	Porovnání komerčních a open source nástrojů pro testování softwaru
DP	2007	Ing. Jan Tryzna	Testování softwaru
DP	2014	Ing. Vladan Vachalec	Testování a kvalita softwaru v metodikách vývoje softwaru

Černý (2014) ve své práci s názvem *Analýza problémů agilních projektů firmy* v úvodu představuje agilní metodiky vývoje softwaru a blíže popisuje některé z nich. Cílem jeho práce bylo analyzovat nejčastější problémy agilních projektů vybrané firmy a vybudování interní znalostní báze za účelem předcházení problémů v budoucích projektech. Autor se zabývá zejména problémy s řízením projektu, mezi které patří například chybné odhady a podcenění projektu, nebo špatné zvolení metrik pro kontrolu stavu projektu. Jak již bylo naznačeno, práce se zaměřuje na problémy v řízení podniku a nepojednává tedy přímo o problémech spojených s testováním, jakožto součástí agilního vývoje.

Cílem diplomové práce *Návrh metodiky testování webových aplikací* autora Fiuráška (2014) bylo „sestavení metodiky testování webových aplikací pro menší softwarové společnosti“. Autor přestavuje termín Quality Assurance a poukazuje na důležitost testování při vývoji softwaru. Definuje také velké množství pojmů spojených právě s testováním. Výstupem autorovy práce je metodika, která v disciplíně testování definuje role i úlohy. V práci lze nalézt také některá doporučení pro uživatelské testování. Jednotlivé úlohy jsou ovšem popsány velmi obecně, a proto je nelze přímo použít pro účely vyřešení problémů identifikovaných v rámci této bakalářské práce.

Langr (2012) ve své diplomové práci na téma *Návrh realizace procesu zajištění kvality softwaru v malé firmě* představuje zajištění kvality, uvádí několik modelů, které definují dimenze kvality softwaru. Mezi tyto modely patří například známý model FURPS nebo McCallův model. Autor ve své práci analyzuje vybranou firmu, avšak nezaměřuje se pouze na oblast testování. Součástí autorovy analýzy jsou také problémová místa v požadavcích, v návrhu, ve vývoji, a dalších částech vývojového cyklu softwaru. Pro testování například autor doporučuje zavedení jednotné šablony pro reporty chyb.

Libík (2010) v rámci své práci *Zlepšování softwarových procesů v oblasti testování* rozebírá proces testování a popisuje jeho jednotlivé činnosti od návrhu testů, před jejich samotné provádění, až po vyhodnocení výsledků testů. Uvádí také některé nedostatky jím popisovaného implementačního balíčku, mezi které patří například chybějící popis nástroje pro automatizaci. Autor mimo jiné uvádí, že automatizace v pilotním projektu použita nebyla, stejně jako nebyl nalezen žádný vyhovující nástroj pro správu testů. Mezi kandidáty autor zařadil například RTH, TestLink, nebo Klaros-Testmanagement, u kterého uvádí, že je propojitelný se systémem

Redmine, což může být pro potřeby této práce velmi přínosné, neboť zkoumaný podnik tento systém využívá.

Pavelka (2014) se ve své práci *Nástroje pro podporu testování a jejich praktické využití* zaměřuje především na praktické využití nástroje pro podporu testování HP Quality Center 11. V teoretické části se zabývá vysvětlením základních pojmů v oblasti testování softwaru, blíže se věnuje popisu různých druhů testů a také zmiňuje některé metriky, které lze použít pro získání zpětné vazby. Uvádí příklady metrik a důsledky nedostatečného testování. Pro účely této práce může přínosná zejména část praktická, ve které autor jmenuje problémy zkoumaného nástroje, zjišťuje jejich příčiny a navrhuje možná řešení včetně jejich zhodnocení z hlediska vhodnosti a dopadu změny.

Diplomová práce autorky Rejnkové (2010) se zabývá *Lokalizací a přizpůsobením metodiky OpenUP*. Výsledkem její práce je metodika MMSP, na kterou se část této bakalářské práce zaměřuje. Autorka ve své práci mimo jiné představuje metodiku OpenUp a zvažuje její rozšiřitelnost. V praktické části se autorka věnuje tvorbě metodiky v nástroji EPF Composer. Hlavním přínosem pro tuto bakalářskou práci je výsledná metodika a její podrobný popis.

Rybak (2012) ve své bakalářské práci na téma *Testování softwaru* popisuje instalaci a praktické použití nástroje IBM Rational Functional Tester. Ukázka automatizace testování pomocí tohoto nástroje může být pro potřeby této práce přínosná.

Štolc (2010) ve své diplomové práci na téma *Porovnání komerčních a open source nástrojů pro testování softwaru* stručně popisuje teoretické základy testování softwaru, popisuje různé kategorie testovacích nástrojů včetně jejich úlohy v testovacím procesu. Zvolené nástroje dále porovnává podle předem vybraných kritérií. Zabývá se nástroji pro automatizované testování, nástroji pro sledování chyb a také porovnává nástroje pro správu testů. Hlavním přínosem pro tuto práci může být právě představení a porovnání nástrojů pro správu testů a jejich automatizaci.

Diplomová práce autora Tryzny (2007) na téma *Testování softwaru* vymezuje základní teoretické poznatky v oblasti testování, mezi které autor zařadil kategorie testování, cíle testování, výhody, ale také nevýhody testování. Součástí práce je i náhled do historie testování. Autor poskytuje i pohled na některé základní modely životního cyklu softwaru. V praktické částí práce autor představuje nástroj Mercury QuickTest Professional, popisuje postup práce s programem a jeho hodnotí jeho vlastnosti. V poslední části práce lze nalézt charakteristiku vybraného podniku a návrh metodiky pro zátěžové testování.

Vachalec (2014) v rámci své práce s názvem *Testování a kvalita softwaru v metodikách vývoje softwaru* popisuje význam kvality softwaru a představuje některé metriky pro její měření. Dále autor popisuje tradiční a agilní přístupy vývoje softwaru a jejich vztah ke kvalitě softwaru a popisuje některé testovací techniky. Hlavním cílem autorovy práce je rozšíření disciplíny testování v metodice MMSP. V metodice vytváří nové role, úlohy a pracovní produkty a porovnává vybrané nástroje pro podporu testování. Přínosem pro tuto práci je zejména způsob návrhu rozšíření metodiky.

1.4.2 Odborné knihy

V knize *Agile Testing: A Practical Guide for Testers and Agile Teams* autorky Crispin a Gregory (2009) popisují, jak by měli jednat testeři, kteří dříve působili v projektech využívajících tradiční přístupy, a kteří nově působí v agilních projektech. V úvodu knihy autorky představují myšlenku agilního přístupu a čtenáři vysvětlují, o čem vlastně agilní testování je a např. jaké role v agilním týmu vůbec vystupují. Dále zde čtenář může nalézt informace o tom, jak by měl, jako tester, jednat v rámci agilního týmu, jak by měl komunikovat a čeho by se neměl v týmu bát. Zajímavou částí knihy je rozdělení agilního testování do testovacích kvadrantů, za účelem bližšího pochopení různých typů testů a testovacích metod. Velká část knihy je věnována popisu těchto kvadrantů a vztahů mezi nimi a následující kapitoly se na tyto kvadranty velmi četně okazují. Zajímavou částí knihy jsou devátá a desátá kapitola, ve které autorky popisují, jak je možná v agilním týmu spravovat požadavky, jak psát testy, jak tyto testy automatizovat a jak testovat dokumentaci. Automatizaci je také věnována samostatná kapitola. Za zmínku stojí také 15. kapitola, která pojednává o aktivitách testera, o plánování testů a o roli testera při vytváření a odhadování uživatelských příběhů.

Patton (2001) v knize *Software Testing* představuje proces testování a uvádí, proč je důležité testovat. Seznamuje čtenáře s pojmy a praktikami v oblasti testování a popisuje jejich vzájemné vztahy, což může značně pomoci této práci právě v její teoretické části. Autor se zaměřuje na testování obecně a nebere v potaz rozdíly mezi tradičním a agilním vývojem. Autor vysvětluje, co je a co není chybou, uvádí, jak by mělo vypadat hlášení o chybě a také zmiňuje testování použitelnosti a kontrolu správnosti a přesnosti dokumentace. Dále popisuje činnosti testera, mezi které patří např. tvorba testovacích případů a reportů. Hodnotné poznatky obsahuje také část knihy popisující životní cyklus chyby. Část knihy je rovněž věnována automatizaci testování.

Autoři Morgan a další (2010) v knize *Software Testing An ISTQB–ISEB Foundation Guide*, která je určena především zájemcům o certifikaci ISTQB (International Software Testing Qualifications Board), vysvětlují základy a důvody testování a popisují, proč software selhává. V rámci kapitoly o životním cyklu softwaru popisují nástroje pro podporu testování, typy a úrovně testů a různé modely životního cyklu softwaru. V dalších kapitolách popisují detailně statické testování včetně k němu používaných nástrojů. Autoři se také zabývají problematikou správy testů, vztahem mezi rizikem a testováním, různými testovacími technikami a typy testů, na závěr pak pojednávají o testovacích nástrojích. Poslední kapitola je věnována informacím o zkouškách. Kniha je napsána velmi srozumitelně a výroky a doporučení jsou provázeny praktickým odůvodněním.

2 Testování softwaru

V úvodu této kapitoly je představen pojem *testování softwaru* a nastíněn jeho účel. V kapitole jsou dále popsány základní pojmy a praktiky z oblasti testování softwaru, které jsou společné pro vývoj tradiční i pro vývoj agilní. Jsou také představeny některé testovací techniky, a to testy statické a dynamické, testy černé a bílé skřínky. V závěru kapitoly jsou uvedeny některé důležité vlastnosti manuálního a automatizovaného testování.

2.1 Pojem testování softwaru

Testování softwaru bylo v uplynulých letech chápáno rozdílně, a tak se definice jednotlivých autorů liší.

V knize *Software Testing: An ISEB foundation* z roku 2010, je uvedeno, že „testování je aktivita používaná ke snížení rizika a zlepšení kvality hledáním závad". (Hambling, 2010)

Naproti tomu starší definice Glenforda Myerse v jeho knize „The Art of Software Testing" z roku 1979 říká, že „testování je proces spouštění programu, za účelem hledání chyb". (Myers, 1979)

Formálnější význam pojmu uvádí Institut pro elektrotechnické a elektronické inženýrství, IEEE. Testování popisuje dvěma definicemi. První z nich říká, že *„testování je proces provozování systému nebo jeho komponent za určitých podmínek, sledování nebo zaznamenávání výsledků a vyhodnocování sledovaného systému z nějakého hlediska."* Druhá z definic nahlíží na testování jako na *„proces analyzování softwaru, za účelem rozpoznávání rozdílů (defektů) mezi současným a požadovaným stavem a následným vyhodnocením zkoumaného softwaru"*. Z druhé definice mimo jiné vyplývá, že k testování nepotřebujeme běžící program, což je velmi důležitým poznatkem, neboť je tento princip používán při statickém testování. (Galin, 2003)

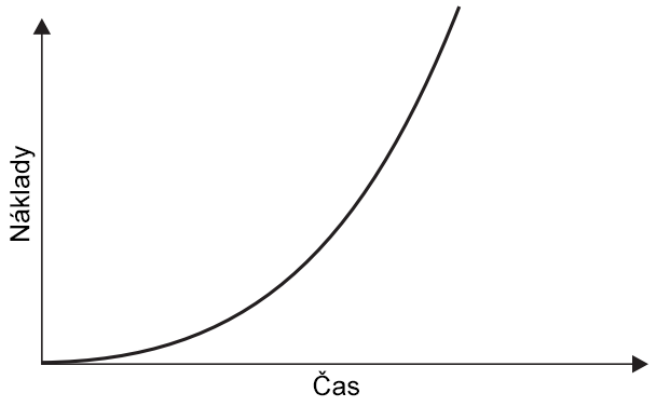
Z těchto definic je patrné, že v minulosti bylo na testování softwaru nahlíženo jako na proces, jehož hlavním úkolem bylo pouhé nalezení chyb. S postupem času začal být kladen důraz na celkovou kvalitu dodávaného softwaru. Dnes se softwarové společnosti zaměřují na software, který nejen, že obsahuje minimum chyb, ale také je uživatelsky přívětivý, intuitivní a jehož funkcionality jsou kvalitně zdokumentovány. Kromě dynamického testování je uplatňováno i statické testování požadavků a specifikací.

2.2 Účel testování

Jak bylo již naznačeno v předchozí podkapitole 2.1, testování v dnešní době neslouží pouze k odhalení chyb v softwaru, ale k zajištění celkové kvality softwaru a maximalizaci užitku pro zákazníka, kterému bude software sloužit.

Zjednodušeně lze říci, že testujeme, abychom v softwaru nacházeli chyby. Ve skutečnosti ale testujeme ještě z několika dalších důvodů: abychom se ujistili, že je kód spolehlivý; abychom se ujistili, že je kód použitelný a použitím jednotkových testů si usnadňujeme práci. (Patton, 2001)

Testováním se snažíme také o minimalizaci celkových nákladů na vývoj. Cena opravy chyby roste v čase rostoucím tempem. Na to poukazuje jak Patton (2001), tak Morgan (2010) (viz Obr. 2-1)



Obr. 2-1 Vliv času na nalezení chyby na náklady potřebné k její opravě

Testování softwaru lze rozdělit do několika dimenzí. Pro rozdělení slouží například model FURPS, který definuje 5 dimenzí kvality softwaru, z nichž se první z jich zabývá funkčními požadavky a zbývající čtyři požadavky nefunkčními:

- **Funkčnost** (Functionality) – ověřuje vlastnosti dle definovaného chování.
- **Použitelnost** (Usability) – se zabývá například vzhledem a konzistencí uživatelského rozhraní.
- **Spolehlivost** (Reliability) – ověřuje dostupnost, přesnost výpočetních operací a schopnost systému zotavit se z chyby.
- **Výkonnost** (Performance) – se zabývá otázkou doby odezvy systému, doby zotavení se z chyby, času potřebného ke spuštění a vypnutí.
- **Podporovatelnost** (Supportability) – v sobě zahrnuje testovatelnost, přizpůsobitelnost, udržovatelnost, kompatibilitu, škálovatelnost apod.

Tento model poprvé představil Robert Grady v knize *Practical Software Metrics for Project Management and Process Improvement* z roku 1992. Popis modelu je možné nalézt na webu IBM, kde je uvedeno i jeho pozdější rozšíření FURPS+, které definuje 4 další kategorie požadavků, a to požadavky designové, implementační, fyzické a požadavky rozhraní.

2.3 Základní pojmy a praktiky

V této podkapitole jsou popsány základní pojmy a praktiky z oblasti testování, které uvádějí autoři novějších publikací, zejména Ron Patton (2001).

2.3.1 Specifikace požadavků

Specifikací požadavků rozumíme dohodu mezi členy vývojového týmu. Specifikace vymezuje vlastnosti a chování produktu, který tým vytváří. Určuje, co má vyvíjený software dělat a naopak, co dělat nemá, jak bude vypadat a k čemu by měl sloužit.

Na specifikaci můžeme nahlížet z vysoké úrovně, potom mluvíme o tzv. „high-level" specifikaci. Abychom mohli získat přehled o fungování daného softwaru, musíme se na něj dívat jako na celek. Tester při tomto typu kontroly většinou zaujímá roli zákazníka. Snaží se odhadnout zákazníkovy zájmy, požadavky a jeho chování, proto je pro testera vhodné, aby se alespoň částečně seznámil s oblastí, kterou software pokrývá. (Patton, 2001) Například při testování účetního softwaru je pro testera dobré, ne-li klíčové, aby znal pojmy a principy účetnictví. Pokud tester zkoumanou oblast nezná dokonale, může se při testování lecčemu přiučit.

Do výše zmíněné specifikace vysoké úrovně Ron Patton řadí také:

- **Měřítko** – určuje, jak bude software velký, co všechno bude pokrývat a jak jeho velikost ovlivní testování.
- **Komplexnost** – se ptá, jak ovlivní komplexnost softwaru jeho testování.
- **Testovatelnost** – pokládá otázku, zda budeme mít vůbec na testování čas, znalosti, zkušenosti a zdroje.
- **Kvalitu/Spolehlivost** – pojednává o zamýšlené kvalitě a spolehlivosti vyvíjeného softwaru.

Nahlížením na specifikaci z nízké úrovně, tzv „low-level", poznáváme jednotlivé části softwaru a zkoumáme jejich vazby. Aby byla specifikace i na této úrovni správná, musí podle Rona Pattona splňovat těchto osm vlastností:

- **Úplnost** – ujišťujeme se, že nebylo ve specifikaci na nic zapomenuto.
- **Přesnost** – kontrolujeme, zda je zvolené řešení správně a zda vymezuje správně cíl.
- **Preciznost** – hlídáme, aby byl popis dostatečně určitý a byl snadno pochopitelný.
- **Konzistenci** – zajišťujeme, aby nebyl popis v rozporu sám se sebou nebo s jinými prvky specifikace.
- **Relevantnost** – ptáme se na otázku, zda je funkcionalita zákazníkem skutečně požadována či zda je nutné ji specifikovat.
- **Realizovatelnost** – vyhodnocujeme, zda je vůbec možné funkcionalitu implementovat za použití současných finančních zdrojů, lidí, nástrojů a času.

- **Nezávislost na kódu** – dáváme pozor, aby byla specifikace nezávislá na konkrétní softwarové architektuře, kódu nebo návrhu.
- **Testovatelnost** – zajímá nás, zda funkcionalitu vůbec můžeme otestovat, zda pro otestování máme k dispozici dostatek informací. (Patton, 2001)

2.3.2 Defekt (bug)

Pokud se chování softwaru nějakým způsobem odchyluje od očekávaného chování, hovoříme o defektu. Kromě pojmu „defekt" existuje celá řada označení pro selhání softwaru. Můžeme hovořit o „selhání", „nekonzistenci", „problému", „chybě" či „anomálii". V oblasti vývoje softwaru je defekt běžně označován jako „bug", v překladu brouk. Ke vzniku pojmu „bug" se váže příběh, který se stal v roce 1947 na Harvardově univerzitě. Čtyřicátá léta byla ve znamení velkých sálových počítačů, a tak nebyla náhoda, že jeden takový, Mark II, provozovali tamější inženýři právě na Harvardově univerzitě. Jednoho dne stroj ale přestal pracovat. Když technici zjišťovali příčinu problému, našli na desce se spoji mrtvého mola. Mola nejspíš přilákalo světlo a teplo, a když se dostal do útrob počítače, zkratoval elektrický obvod. V záznamu o chybě pak technici uvedli, že byl v počítači nalezen „bug". (Patton, 2001)

Očekávané chování softwaru je popsáno jeho specifikací (viz kap. 2.3.1). Odchylka od očekávaného chování ale nemusí nutně znamenat chybu v kódu. Chyba se totiž může vyskytnout i ve specifikaci. Obecně lze tvrdit, že chybou je, pokud:

- Software nedělá to, co tvrdí specifikace.
- Software dělá něco, co by podle specifikace neměl dělat.
- Software dělá něco, co specifikace nezmiňuje.
- Software nedělá něco, co specifikace nezmiňuje, ale dělat by to měl už z principu.
- Software působí složitě, těžko se ovládá a tester tuší, že by s jeho používáním mohl mít zákazník problém. (Patton, 2001)

Poslední bod je především subjektivní, a proto může být pro testera v mnoha případech těžké určit, co zákazník považuje za složité ovládání. Aby se předešlo případným nedorozuměním, je vhodné, aby byl tester v kontaktu se zákazníkem a mohl s ním ihned konzultovat změny[2], které se především dotknou uživatele, tedy změny v uživatelském rozhraní aplikace.

Patton dále vysvětluje, že čím později je defekt nalezen, tím jsou větší náklady na jeho opravu. Náklady rostou s časem exponenciálně. Defekt nalezený a opravený v počátečních fázích vývoje, v době tvorby specifikace, může stát vyvíjející firmu, řekněme, 10 korun. Pokud je defekt nalezen v pozdějších fázích[3] programování a testování, může jeho cena vzrůst na stovky až tisíce korun, a pokud chybu nalezne až zákazník, výsledná cena defektu se může pohybovat kolem 10 000 korun. (Patton, 2001)

[2] Přímá komunikace se zákazníkem je právě jednou z výhod agilního přístupu.

[3] Zde můžeme opět uplatnit výhody agilního přístupu, protože zákazník je už od začátku vývoje součástí vývojového týmu a může pomoci s odhalením chyby v raných fázích vývoje.

2.3.3 Tester

Podle (BusinessDictionary.com), je tester *„technik, který provádí předem stanovené testy na množině softwarových programů a aplikací před jejich nasazením, pro zajištění kvality jednotnosti návrhu a správného fungování. Tester používá pečlivé testovací metody, včetně rozsáhlých simulací chování koncového zákazníka, za účelem nalezení bugů, které jsou následné programátory opraveny“*.

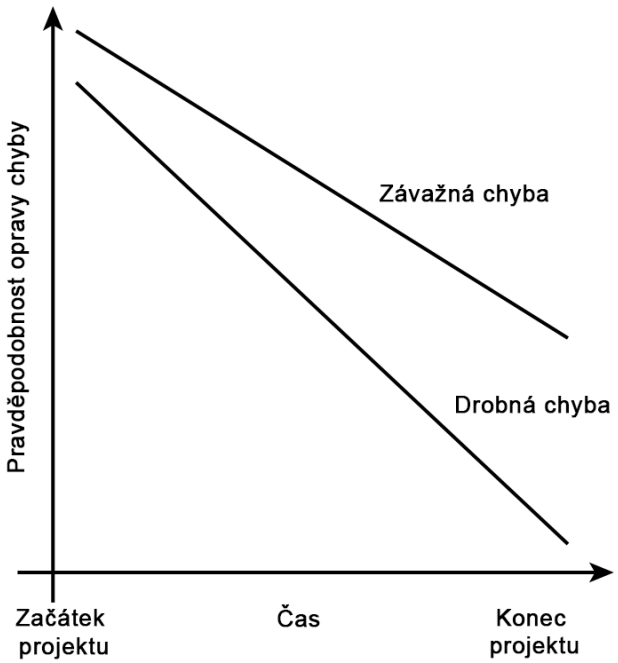
Poněkud stručnější definici uvádí (Patton, 2001), který testera popisuje jako *„osobu, jejíž cílem je nalézt bugy, nalézt je co nejdříve a ujistit se, že budou opraveny“*.

Patton navíc uvádí seznam vlastností, který by dobrý tester měl mít. Mezi tyto vlastnosti patří například kreativita, která je klíčová pro nalezení těžko odhalitelných bugů, ale také cit pro detail. Dobrou vlastností testera je také schopnost přesvědčit vývojáře, testera, nebo zákazníka o tom, proč něco chybou je, nebo proč něco jiného chybou naopak není. Tester by měl umět správně posoudit, co je chyba. U testerů zpravidla nejsou vyžadovány specifické technické dovednosti, jakými je například znalost programovacího jazyka testované aplikace, ale pokud má tester alespoň základní znalost v této oblasti, je to často výhodou. (Patton, 2001) Tester, znalý konkrétního programovacího jazyka se může nechat unést detaily na nižší úrovni, a ztratí pohled na aplikaci jako na celek či přestane uvažovat jako uživatel, ale spíše jako programátor[4].

2.3.4 Bug report

Když tester nalezne při své práci nějakou chybu, musí tuto skutečnost nahlásit vývojovému týmu formou tzv. „bug reportu“, čili hlášení o chybě. Tester by neměl nahlášení chyby odkládat. Čím dříve je chyba nahlášena, tím větší je šance, že bude také včas opravena. Vztah mezi pravděpodobností opravy chyby a fází projektu ukazuje Obr. 2-2.

[4] Tento problém se může v agilním vývoji vyskytnout poměrně často, neboť členové týmu mohou zastávat více rolí najednou. (Crispin, a další, 2009)



Obr. 2-2 Pravděpodobnost opravy chyby v čase (Patton, 2001)

Aby bylo možné chybu opravit, musí být dostatečně popsána. Patton (2001) uvádí 4 základní vlastnosti, které by hlášení o chybě mělo splňovat:

- **Stručné.** Hlášení by mělo obsahovat pouze informace, které jsou nezbytné k pochopení a nasimulování problému. Samozřejmě je často vítané uvést případné souvislosti s ostatními nahlášenými chybami, neboť mohou mít stejnou příčinu.
- **Oddělené.** Jednotlivá chybová hlášení by měla popisovat jen jednu chybu. Existují případy, kdy není snadné rozlišit, zda mají chyby stejnou příčinu. Patton v takové situaci doporučuje chyby ohlásit jednotlivě.
- **Jasné a obecné.** Hlášení by měla obsahovat jasný popis, podle kterého by měl být schopen chybu nasimulovat jak programátor, tak i ostatní testeři. Hlášení by mělo být co nejobecnější. Např. při testování webového formuláře je lepší uvést, že dojde k výjimce při zadání textu do pole pro číslo, než vypisovat konkrétní řetězce, které byly pro testování použity.
- **Opakovatelné**. Hlášení by mělo obsahovat postup k nasimulování chyby, podle něhož by mělo být možné v budoucnu popisovanou chybu vyvolat. (Patton, 2001)

Pro hlášení chyby existuje dokument s názvem *„Test Incident Report“*, vydaný jako součást standardu ANSI/IEEE 829 pro testovací dokumentaci. Tento dokument doporučuje, jak by mělo být postupováno v případě nálezu chyby v softwaru, resp. při nálezu jakékoliv neshody s očekávaných chováním. Standard uvádí několik nezbytných součástí chybového hlášení:

- **Jedinečný identifikátor**, podle něhož lze záznam o chybě dohledat, a podle něhož se na záznam můžeme odkazovat.

- **Stručný popis**, který dostatečně vystihuje nalezený problém. Součástí stručného popisu je také údaj o testovaném softwaru, jeho dílčí části a verzi. Popis by měl odkazovat na testovací případ a měl by se fakticky opírat o specifikaci nebo dokumentaci projektu.
- **Detailní popis chyby,** obsahující základní informace, jakými je datum a čas nálezu, jméno testera, informace o jeho hardwarové a softwarové konfiguraci, kroky a data nezbytná k opětovnému vyvolání chyby včetně očekávaného výsledku a skutečného výsledku.
- **Dopad chyby**, čili míru její urgentnosti a závažnosti. Vhodné je také uvést, jak se chyba promítne do testovacího plánu. (Patton, 2001)

Pokud je chyba nedostatečně, nebo dokonce nesprávně popsána a její hlášení nesplňuje výše uvedené vlastnosti, může dojít k nedorozumění mezi členy vývojového týmu, což může vést ke zvýšení celkového času potřebného k opravě chyby[5].

2.3.5 Testovací plán

Patton (2001) říká, že stejně jako si programátor před psaním kódu musí rozmyslet, jak danou funkcionalitu naimplementuje, musí si i tester svou práci naplánovat. Nemůže tedy otevřít aplikaci a začít ihned psát testovací případy a testovat. Tester se s danou aplikací musí nejprve seznámit. K dobré orientaci a správnému sestavení testovacího plánu k nové aplikaci nebo její součásti si může tester nastudovat její specifikaci nebo technický návrh[6], a pokud je aplikace již nasazena v testovacím prostředí, může tester provést průzkumné testování.

Testovací plán se skládá z vybraných testovacích případů, jejichž vhodným zvolením tester určuje, které části aplikace budou otestovány. Součástí plánu je také zvolená metoda testování. Tester se může rozhodnout například pro použití automatizovaných tesů. Testovací plán rovněž definuje použité metriky. Přístupy a metriky uvádí také standard ANSI/IEEE 829. Všechny zvolené metody jsou v testovacím plánu popsány jen velmi hrubě. Detailní popis zvolených metod je součástí *designu testů.* (Patton, 2001)

2.3.6 Design testů

Design testů detailně popisuje, kdy a jak budou použity různé druhy testů a definuje akceptační kritéria. Smyslem designu testů je určit přístup k testování jednotlivých funkcionalit. Nejsou v něm však definovány přesné kroky, ty jsou součástí *testovacího případu*. Patton (2001) vychází ze standardu ANSI/IEEE 829 a uvádí následujících 5 součástí designu testů:

1. **Identifikátory.** Design testů by měl mít přiřazen jedinečný identifikátor, podle něhož je možné dokument kdykoliv dohledat, a podle něhož se na něj lze odkazovat. Součástí designu testů by měl být také odkaz na testovací plán, případně další dokumenty.
2. **Vlastnosti, které budou otestovány.** Součástí by měl být popis vlastností softwaru, které budou otestovány. Příkladem může podle autora být „funkce sčítání v Kalkulačce“

[5] V agilních projektech se může tester s programátorem problém vyřešit ihned a záznam o chybě nemusí vůbec vzniknout. (Crispin, a další, 2009)

[6] Agilní vývoj navíc umožňuje přímou komunikaci s osobou, která specifikaci napsala. V některých případech je dokonce tester spoluautorem specifikace. (Crispin, a další, 2009)

nebo „výběr velikosti písma ve WordPadu“. Dále by zde měly být uvedeny vlastnosti, které mohou být při testování ověřeny nepřímo, jako např. vzhled některých dialogových oken. Nakonec by zde měly být uvedeny vlastnosti, které testovány nebudou, nebo budou testovány v rámci jiného designu. V takovém případě bude uveden pouze odkaz na související design testů.

3. **Přístup k testování.** Kromě obecného popisu by zde měly být popsány testovací techniky a způsob ověžení výsledků testů. Autor uvádí následující příklad: *„Bude vytvořen testovací nástroj, který bude načítat a ukládat předem vytvořené datové soubory různých velikostí. Množství datových souborů, jejich velikost a obsah bude určen pomocí technik testů černé skřínky, doplněných o testy bílé skřínky, které dodá programátor. Úspěšnost bude ověřena porovnáním uloženého souboru s originálem pomocí nástroje pro bitové porovnávání souborů.“* (Patton, 2001)

4. **Identifikátory testovacích případů.** Součástí designu testů by měl být seznam jedinečně označených testovacích případů, například:

   Vyzkoušet nejvyšší možnou hodnotu Testovací případ č. 19875

   Vyzkoušet nejnižší možnou hodnotu Testovací případ č. 19876

   Samotný obsah nebo použité testovací hodnoty by zde neměly být pro přehlednost uváděny, protože budou součástí obsahu testovacího případu.

5. **Akceptační kritéria.** Nakonec by měl design testů obsahovat kritéria, podle kterých je možné určit, zda testovaná vlastnost v testech uspěla či selhala. Jako kritérium je možné považovat 100% úspěšnost všech testovacích případů, ale také je možné zvolit hranici nižší.

2.3.7 Testovací případ

Testovací případ, jak jej definuje standard ANSI/IEEE 829, „dokumentuje skutečné vstupní hodnoty spolu s očekávanými výstupy. Testovací případ také identifikuje případná omezení plynoucí z jeho použití.“ Cílem testovacího případu je popsat očekávané chování v závislosti na vstupních hodnotách. Každý testovací případ by měl podle zmíněného standardu obsahovat: jednoznačný identifikátor, stručný název, detailnější popis včetně všech potřebných vstupních podmínek a dat, popis očekávaného výstupu; potřebný hardware, software a jeho nastavení; neobvyklé testovací pomůcky (např. tiskárna studentských průkazů) a dále by měl specifikovat závislosti na jiných testovacích případech. (Patton, 2001)

2.3.8 Testovací data

Testovací data jsou používána k provádění testů a jsou zpravidla vytvářena jako součást testovacích případů. Tvoří množinu vstupních dat s rozličnými vlastnostmi. Mezi testovací data se mohou řadit např. údaje k testovacím uživatelským účtům, soubory určené pro ověření funkčnosti importu dat v aplikaci, údaje o testovacích platebních kartách nebo předpřipravená data pro vyplnění formulářů. Smyslem evidence testovacích dat je odstranění duplicit jak v testovacích případech, tak v chybových hlášeních. Každý záznam může mít přidělen jedinečný identifikátor, pomocí něhož se lze na data odkázat.

2.4 Testovací techniky

V závislosti na konkrétní testované aplikaci může tester zvolit několik různých testovacích technik. Výběr technik je prováděn při tvorbě testovacího plánu, a při designu testů je obsah zvolených testovacích technik blíže upřesněn.

2.4.1 Statické a dynamické testování

Statické testování se zabývá kontrolou specifikace požadavků. Při jeho provádění nedochází ke spuštění programového kódu, a proto je možné se statickým testováním začít ještě před existencí první spustitelné verze aplikace. Při statickém testování jsou ověřovány požadavky a je posuzována možnost jejich naplnění před samotnou implementací. Součástí je také kontrola dokumentace aplikace a kontrola programového kódu, tzv. „code review". Statické testování může odhalit chybu v raných fázích vývoje, kdy jsou na její opravu potřeba výrazně nižší náklady a může být prováděno jak manuálně, tak automatizovaně. (Morgan, a další, 2010)

Naproti tomu, při *dynamickém testování* je nutné kód aplikace spustit. Dynamické testování proto začíná později než statické a při jeho provádění jsou vykonávány testy podle předem připravených testovacích scénářů. V případě nalezení chyby je potřeba vynaložit větší úsilí na její opravu, což bývá spojeno s vyššími náklady. Pro dynamické testování jsou používány *testy černé, šedá a bílé skřínky*, a stejně jako u statického testování, je možné využít *manuálních i automatizovaných testů*. (Morgan, a další, 2010)

2.4.2 Testování černé a bílé skřínky

Při *testování černé skřínky* má tester k dispozici pouze uživatelské rozhraní (UI) testované aplikace, případně její programové rozhraní (API) a nemůže nahlížet do jejího zdrojového kódu. K testování tedy používá zpravidla uživatelskou dokumentaci (pokud již existuje) a další dokumenty popisující chování aplikace.

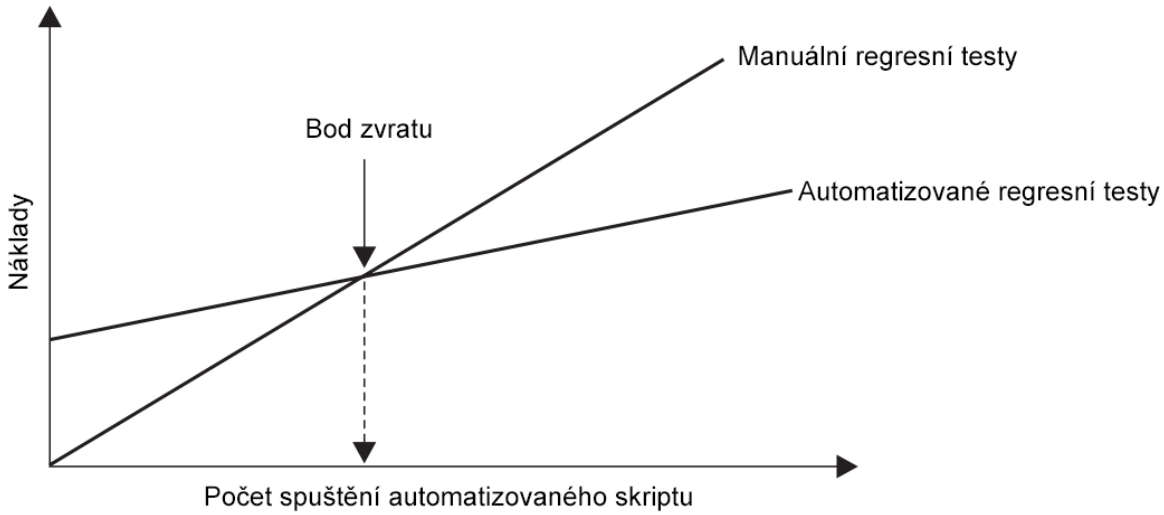
Testy bílé skřínky umožnují testerovi nahlédnout do zdrojového kódu aplikace nebo získat přístup k serveru, čímž může lépe odhalit její problémová místa nebo lépe využít automatizovaných testů. Může například narazit na nedostatečnou kontrolu vstupních polí ve formuláři nahlédnutím do programového kódu na straně serveru[7]. Tester ale v takové situaci musí znát programovací jazyk, ve kterém je aplikace vyvíjena. K testování může také využít přístup k databázi a vložit do ní vhodná testovací data. (Patton, 2001)

2.4.3 Manuální a automatizované testování

Manuální testování provádějí testeři osobně a při vývoji velkých aplikací může tento přístup zabrat velké množství času. Z důvodu časové tísně pak nemusí být řádně otestovány všechny možné případy a může dojít k přehlédnutí závažných chyb. Testeři jsou také pouze lidé a lidé dělají chyby. Některé chyby mohou testeři přehlédnout nebo je nepovažovat za závady. (Crispin, a další, 2009)

[7] (Patton, 2001) navíc definuje tzv. testování šedé skřínky, kdy tester nahlíží do zdrojového kódu na straně klienta. Jako příklad uvádí kontrolu zdrojového kódu webové stránky.

Automatizované testování řeší problém časové náročnosti na spuštění testů. Testy ale musí být důkladně připraveny a pro jejich přípravu je obvykle nutné znát nějaký skriptovací jazyk. Samotná příprava automatizovaných testů trvá výrazně déle než u manuálního testování. Automatizací testů může tester získat rychlý přehled o stavu aplikace, aniž by musel věnovat svůj čas, výsledky však musí umět správně interpretovat. Díky automatizaci regresních testů může tester zaměřit na testování kritických míst aplikace. Automatizované testy navíc pomáhají dokumentovat vyvíjenou aplikaci. (Crispin, a další, 2009)



Obr. 2-3 Návratnost automatizovaných testů

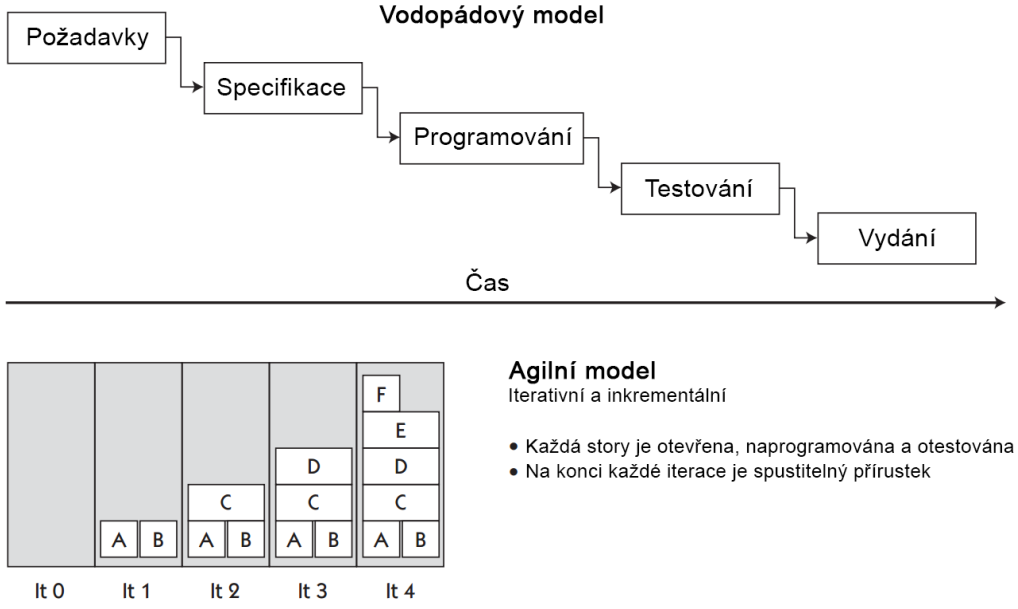
Obr. 2-3 Návratnost automatizovaných testů zobrazuje vztah mezi počtem spuštění manuálních a automatizovaných testů a náklady na jejich údržbu, provádění a vyhodnocování. Z obrázku je patrné, že je výhodné automatizovat pouze tehdy, když tento proces povede ke snížení nákladů oproti použití manuálních regresních testů.

3 Agilní vývoj softwaru

Tato kapitola popisuje specifické rysy agilního vývoje, představuje agilní manifest, jehož myšlenka tvoří základ agilního vývoje softwaru. Manifest především určuje, na co by se měl vývojový tým zaměřit a čeho by se měl po dobý vývoje softwarového produktu držet. V kapitole jsou dále uvedeny agilní principy vývoje softwaru, je začleněno testování do agilního vývoje a je představena role testera v agilním týmu. Na závěr kapitoly je představen Scrum, agilní metodika pro řízení projektu.

3.1 Tradiční versus agilní vývoj

Tradiční vývoj nejlépe vystihuje vodopádový model (viz Obr. 3-1), ve kterém fáze testování vstupuje až ke konci vývojového cyklu. Nevýhodou tohoto modelu z pohledu testování je, že jakmile je fáze započata fáze testování, už není možné měnit požadavky a specifikaci vyvíjené aplikace. Proto pokud je při testování ve specifikaci nalezena zásadní chyba, kterou není možné opravit, může proces vývoje naprosto zkolabovat. Tento problém nemožnosti „vrátit se zpět“ částečně řeší jiné modely vývoje softwaru, jako například spirálovitý model nebo model iterativního vývoje, ve kterých probíhá několik poměrně kratších vodopádových iterací. V každé iteraci jsou provedeny všechny fáze a v případě nalezení problému ve specifikaci nebo v případě změny specifikace ze strany zákazníka, je možné změny uplatnit v další iteraci.



Obr. 3-1 Rozdíl mezi tradičním a agilním modelem vývoje (Crispin, a další, 2009)

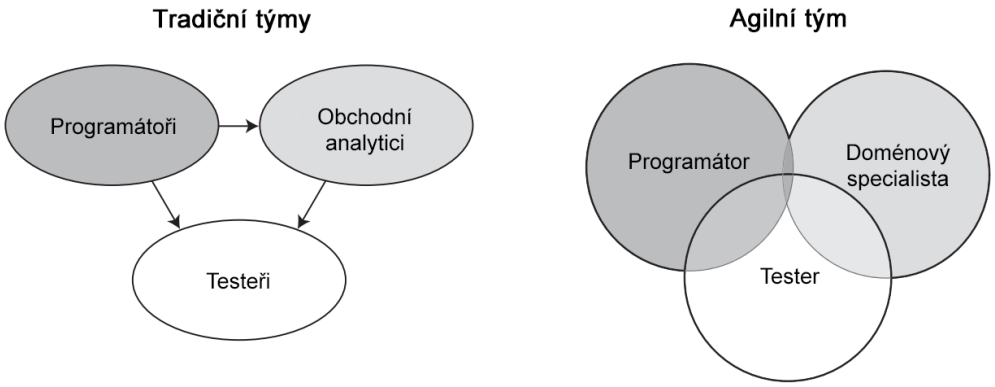
Pro agilní vývoj je charakteristický iterativní a inkrementální vývoj. To znamená, že v každé iteraci jsou prováděny všechny kroky definované ve vodopádovém modelu a na konci každé iterace vzniká funkční, nebo také spustitelný, přírůstek, tzv. inkrement. Na konci každé iterace je možné vyvíjenou aplikaci vydat a požadavky, které vývojový tým nestihl naplnit v rámci poslední iterace, je možné přesunout do iterace následující. Princip, který zajišťuje soustavné dodávání funkčního produktu, je označován anglickým výrazem „Continuous Delivery“. Agilní vývoj z teoretického hlediska nikdy nekončí.

Vývoj v agilních metodikách je podle Knesla (2009) dělen do těchto fází:

1. **Nultá iterace**, rámci které je provedena krátká analýza a jsou naprogramovány základní části produktu. V této fázi není důležité, aby aplikace řádně fungovala, ale aby bylo možné zákazníkovi předvést, jak bude produkt ve výsledku fungovat.
2. **Analýza změny**, při které jsou vybrány změny, které budou v iteraci implementovány. Tyto změny jsou analyzovány a je sestaven jejich design (návrh).
3. **Implementace požadované funkcionality**.
4. **Předvedení funkcionality zákazníkovi**.
5. (pokud není produkt hotov, zpět na bod 2)
6. **Údržba a rozvoj**, probíhající rovněž v iteracích.

Body 2 až 4 jsou označovány jako jedna iterace a opakují se do té doby, dokud není vývoj ukončen. Důvodem pro ukončení projektu může být jeho úspěšné dokončení, odložení nedostatek financí nebo změna priorit ze strany zákazníka. (Knesl, 2009)

Na rozdíly v přístupu tradičních a agilních týmů poukazují autorky Crispin a Gregory (2009), které svou myšlenku doplňují obrázkem (viz Obr. 3-2). Autorky říkají, že členové agilních týmu zastávají více funkcí a díky této vlastnosti se mohou stát „skutečnými členy týmu“.



Obr. 3-2 Porovnání struktury tradičních a agilních týmů (Crispin, a další, 2009)

3.2 Agilní manifest a jeho myšlenka

Agilní manifest byl sepsán 17 autory, mezi které patří významné osoby, jako například Jim Highsmith, Martin Fowler, nebo Alistair Cockburn. Manifest definuje priority a principy agilního vývoje softwaru vycházející ze zkušeností autorů a říká, co je v agilním vývoji důležité a zároveň nezbytné. Autoři v manifestu prohlašují, že „*budou vždy preferovat*

- *jednotlivce a interakce před procesy a nástroji,*
- *fungující software před vyčerpávající dokumentací,*
- *spolupráci se zákazníkem před vyjednáváním o smlouvě,*
- *reagování na změny před dodržováním plánu.“* (Cockburn, a další, 2001)

Autoři dále výslovně uvádějí, že jakkoliv jsou body napravo hodnotné, bodů nalevo si cení více. Z tohoto výroku ale nutně nevyplývá, že autoři stojí proti tvorbě dokumentace. Jejich myšlenka, dle mého názoru, pouze naznačuje, že pokud vyvíjený software nefunguje, nezachrání jej ani sebelepší dokumentace. Špatně zdokumentovaný software je v dlouhém období neudržitelný a velmi špatně se testuje.

Ze čtyř základních priorit agilního manifestu vyplývá, že testování v agilních projektech není vždy stejné, z důvodu existence rozdílů mezi zákazníky, a tedy i mezi vyvíjenými aplikacemi, neboť různí zákazníci mají různé potřeby.

Agilní manifest navíc definuje 12 principů, kterými se autoři řídí (Cockburn, a další, 2001):

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
10. Jednoduchost--umění maximalizovat množství nevykonané práce--je klíčová.
11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
12. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

Díky postupům definovaným v manifestu ubude práce manažera a zároveň dojde ke zlepšení výkonnosti týmu. Manažer se bude moci věnovat řízení více zaměstnanců ve větších projektech a bude se moci více zaměřit na kvalitu dodávaného softwaru, což je zapříčiněno mimo jiné tím, že není nutné sepisovat a udržovat velké množství dokumentů, které nepomáhají samotnému vývoji. Manažer na základě minulých iterací ví, kolik práce je schopen tým vykonat za určité časové období. To mu umožní lépe naplánovat budoucí iterace. Navíc může díky rozdělení práce na malé části lépe řídit celkovou rychlost vývoje. Iterativní model vývoje umožňuje změnu zadání v průběhu vývoje, což je u klasického vodopádového modelu nemožné. Produkt je možné dodávat zákazníkovi rychleji a častěji. Díky automatizaci testů může vývojový tým

zajistit vysokou úroveň kvality. Agilní přístup podporuje pouze dobře nastavené podnikové procesy a omezuje ty, které nejsou efektivní a brání komunikaci. (Knesl, 2009)

Crispin a Gregory (2009) uvádějí, že dobře méně zkušený, ale dobře organizovaný a komunikující tým, řídící se agilními principy a hodnotami, bude výkonnější než špatně fungující tým talentovaných jedinců. Z jejich myšlenky vyplývá, že pokud bude tým tvořen talentovými jedinci respektujícími tyto principy, bude velmi konkurenceschopný.

3.3 Fáze agilního vývoje softwaru

Jak bylo uvedeno v podkapitole 3.1, každá iterace se skládá z analýzy, implementace, a předvedení změny zákazníkovi. Níže je uveden popis jednotlivých fází jedné iterace.

3.3.1 Analýza změny

V této fázi analytik vývojového týmu zpracovává zákazníkovy požadavky do další iterace. Existují případy, kdy je plán iterace předem daný, také je v některých případech potřeba nejprve zpracovat nedokončené úkoly z předchozí iterace. Dále dochází k zohlednění úprav, které mohou zlepšit kompatibilitu s dalšími změnami. V této fázi je dále určeno, které úkoly budou v průběhu iterace řešeny. Při analýze jsou často využívány prototypy a názorné příklady, které mají za cíl přinést jasný pohled na výsledek práce. V této fázi jsou připravovány akceptační testy, jsou navrhovány a připravovány změny datových modelů a logiky aplikace, nezbytné pro splnění nově zadaných úkolů. (Knesl, 2009)

3.3.2 Implementace změny

V této fázi dochází k samotnému zpracování úkolů a implementaci požadovaných změn. Důležitou roli zde hraje samořízení týmu. Manažer pravidelně kontroluje stav úkolů a jejich plnění a snaží se odstraňovat problémy, které vývojovému týmu brání v jejich úspěšném dokončení. Důraz je kladen zejména na rovnocenné postavení v týmu. Tým má jako celek společný cíl, a tím je dodání hotového produktu včas. (Knesl, 2009)

3.3.3 Předvedení změny zákazníkovi

Ke konci iterace jsou zákazníkovy předvedeny dokončené a otestované změny. Zákazník si může změny vyzkoušet, zhodnotit je a na základě toho se rozhodnout, jaké změny budou uplatněny v příštích iteracích. Změny musí být viditelné či měřitelné (např. v případě požadavku na zrychlení aplikace). Nedokončené změny nesmí být na konci iterace přítomny ve výsledném produktu a zákazník o nich musí být informován. (Knesl, 2009)

3.4 Testování jako součást agilního vývoje

Stejně jako programátoři dělají více, než jen pouhé přepisování specifikace do programového kódu, testeři v agilních projektech provádějí více činností, než jen samotné testování vyvíjené aplikace. V zájmu každého člena agilního týmu je dodávat výsledný produkt zákazníkovi v té nejvyšší možné kvalitě tak, aby přinesla oběma stranám co nejvyšší hodnotu. Hlavním úkolem testera je zajistit požadovanou úroveň kvality. (Crispin, a další, 2009)

Testování neprovádí jen samotní testeři, nýbrž všichni členové vývojového týmu. Samozřejmě nemůžeme předpokládat, že bude mít programátor stejné znalosti a zkušenosti v oblasti testování jako tester, a naopak, nemůžeme od testera očekávat, že bude schopen vyprodukovat vysoce kvalitní programový kód. Testeři stojí ve většině případů na straně zákazníka a bojují za jeho zájmy, kdežto programátoři, stojící na straně druhé, se snaží přijít s co nejefektivnějším řešením, které nemusí být vždy v souladu se zákazníkovými zájmy. Klíčová je tedy dohoda mezi vývojáři a zákazníkem. (Crispin, a další, 2009)

V agilním vývoji probíhá testování neustále a má různé podoby. Programátoři používají jednotkové testy jako nástroj ke kontrole vlastního kódu. Testeři provádějí nejen uživatelské testování aplikací, ale také testování jejich dokumentace. Zákazník pak provádí vlastní akceptační testy. Testeři a manažeři ověřují proces testování a vyhodnocují jeho efektivitu, za účelem zlepšení tohoto procesu.

3.5 Role testera v agilním týmu

Aby bylo možné správně definovat roli testera, je potřeba pro začátek testera jako osobu charakterizovat. Ron Patton uvádí několik obecných vlastností, které by měl dobrý tester mít. Podle něj tester měl být:

- **Zvědavý.** Tester se nebojí neznáma a je zvědavý, co nový software dělá a jak funguje.
- **Schopný řešit problémy**. Pokud něco nefunguje, tester si s tím často umí poradit.
- **Neústupný.** Tester se nenechá jen tak odradit těžko opakovatelnou chybou.
- **Kreativní.** Tester potřebuje zjistit, co všechno se může při používání softwaru pokazit.
- **Téměř puntičkářský.** Tester prahne po dokonalosti, ale ví, kdy už jsou jeho požadavky nedosažitelné.
- **Správný ve svých rozhodnutích.** Tester musí vědět, jak dlouho bude testování trvat a zda neočekávané chování aplikace je skutečně chybou.
- **Taktní a diplomatický.** Tester je vždy nositelem špatných zpráv. Proto by měl programátorům o chybách říkat taktně a profesionálně a neměl by se chybám v softwaru vysmívat.
- **Přesvědčivý.** Když tester věří, že je jeho nález chybou a ne nutné jej opravit, musí o tom umět přesvědčit i ostatní členy týmu, kteří mohou mít jiný názor. (Patton, 2001)

Tester v agilním týmu provádí nejen běžné činnosti, jakými jsou validace, verifikace, tvorba a správa testovacích případů; příprava, spouštění a vyhodnocení automatizovaných testů; průzkumné testování apod. Agilní testeři komunikují se zákazníkem a poskytují mu informace o implementovaných funkcionalitách, diskutují o případných změnách vyvíjeného softwaru, které se týkají především uživatele a pomáhají definovat jak funkční, tak nefunkční požadavky. Na základě vzniklých požadavků tvoří testy, které slouží k ověření požadovaného chování. V agilním týmu provádějí testování zpravidla všichni jeho členové, proto za testera označujeme osobu, která velmi široce spolupracuje jak s vývojovým týmem, tak se samotným zákazníkem, resp. vlastníkem produktu.

Testeři by neměli sloužit jen jako síto určené k odchytávání chyb. Snadno by se totiž mohlo stát, že si programátoři na situaci zvyknou a začnou produkovat nekvalitní kód, který si po sobě ani letmo nezkontrolují a který je plný banálních chyb. Testeři by v takové situaci byli zaplaveni velkým množstvím drobných chyb a pro jejich velký počet by nemohli v dostatečně krátkém čase ověřit ani základní funkcionality. Dlouhodobé testování nekvalitního kódu by se mohlo rovněž projevit na výkonu testera. Tester, který musí denně nahlásit 20 drobných chyb, týkajících se překlepů či nedostatečně ošetřených vstupů ve formuláři, může snadno přehlédnout skutečnost, že ve stejném formuláři zcela chybí jedno z nepovinných polí.

Autorky (Crispin, a další, 2009) uvádějí 10 agilních principů, které jsou důležité pro testera. Tyto principy ale mohou platit obecně, i pro ostatní členy vývojového týmu. Tester by podle nich měl:

- nepřetržitě poskytovat zpětnou vazbu,
- přinášet hodnotu zákazníkovi,
- podporovat osobní komunikaci,
- mít odvahu,
- udržovat věci jednoduché,
- uplatňovat nepřetržité zlepšování,
- reagovat na změny,
- být schopný organizovat sám sebe,
- být zaměřený na lidi,
- umět užívat si svou práci.

V porovnání s obecnými rysy testerova chování, které popisuje Patton (2001), je v agilním vývoji navíc kladen zvláštní důraz na nepřetržitou zpětnou vazbu, přinášení hodnoty zákazníkovi a schopnost reakce na časté změny.

3.6 Scrum

Scrum je v současnosti jednou z nejpoužívanějších agilních metodik pro řízení projektu, která poskytuje základ a cestu k dosahování obchodních cílů na základě spolupráce. Scrum byl vyvinut pro potřeby vývoje softwaru, ale je možné jej využít i v oblastech výuky, marketingu apod.

Koncept Scrumu byl poprvé představen v roce 1986 jako součást práce s názvem *New New Product Development Game*. Autoři této práce, Hirotaka Takeuchi a Ikujiro Nonaka, definovali svůj přístup jako „flexibilní a holistickou strategii vývoje“. Přístup přirovnávali ke hře ragby, protože stejně jako tam, se snaží jeden „víceúčelový“ tým dostat míč za cílovou čáru. Tento princip byl, a je, v kontrastu s tradičním lineárním přístupem. (Scrum Alliance, 2014)

Scrum dodržuje hodnoty definované v agilním manifestu (viz kap. 3.2 Agilní manifest a jeho myšlenka).

Scrum *upřednostňuje jedince a interakce před procesy a nástroji*. Je založený na týmové spolupráci, kterou využívá pro přinášení obchodní hodnoty, a umožňuje efektivní interakci mezi členy týmu, kteří spolupracují na dosažení společného obchodního cíle. Pro dosažení stanoveného cíle vývojový tým provádí některé činnosti, které jsou pro fungování Scrumu klíčové. (Scrum Alliance, 2014)

Tým zpravidla:

- určuje, jak vykoná zadanou práci,
- práci vykonává,
- zjišťuje, co výkon práce znemožňuje,
- bere odpovědnost za vyřešení všech zjištěných problémů s výkonem práce,
- spolupracuje s ostatními částmi podniku na problémech, které nemůže přímo ovlivnit.

Scrum dává přednost *fungujícímu softwaru před vyčerpávající dokumentací*. Na konci každého sprintu (iterace) je dodán funkční příspěvek, který sice nemusí pokrýt dostatek požadovaných vlastností, ale již obsažené vlastnosti musí mít kvalitu výsledného produktu.

Spolupráce se zákazníkem před vyjednáváním o smlouvě je další charakteristikou Scrumu. Jeho cílem je podporovat a usnadňovat spolupráci členů týmu. Ti se společně snaží najít nejlepší řešení, jak produkt vytvořit a dodat. Tým, obvykle prostřednictvím Product Ownera, spolupracuje se všemi zainteresovanými stranami a přizpůsobuje vizi produktu tak, aby produkt přinášel co nejvyšší možnou hodnotu.

Posledním rysem agilního manifestu, který Scrum dodržuje, je upřednostňování *reagování na změny před dodržováním plánu*. Týmy, které využívají Scrum, často plánují. Jejich plány se nemusí týkat jen následujícího sprintu, ale mohou pokrývat delší časové období. Zpravidla se jedná o plány celého projektu nebo jeho jednotlivých fází. Tyto plány týmu pomáhají přijímat změny a vytvářet hodnotu, avšak by neměly být týmem jen slepě následovány, neboť samotný myšlenkový proces při tvorbě plánu je důležitější než plán samotný. Plán je potřeba v čase doplňovat o nové poznatky a požadavky, protože směr vývoje projektu se často mění, a díky těmto úpravám plánu je týmu umožněno lépe uspět. Scrum využívá především zpětné vazby, pro dosažení nejlepšího možného výsledku práce. (Scrum Alliance, 2014)

Scrum navíc definuje 5 vlastních hodnot, které týmu pomáhají k úspěchu a umožňují podniku, který jej využívá, objevit své přednosti:

- **Soustředění**. Díky tomu, že se tým soustředí na malé množství věcí najednou, můžou být výsledky jeho práce mnohem kvalitnější a mohou být dodány dříve.
- **Odvaha.** Týmová spolupráce přináší odvahu postavit se větším výzvám.
- **Otevřenost**. Spolupráce umožnuje členům týmu vyjádřit, jaké problémy a obavy jim stojí v cestě při plnění úkolů, aby mohly být tyto problémy společně vyřešeny.
- **Odhodlanost**. Díky značné kontrole nad svými činy, je vývojový tým mnohem odhodlanější.

- **Úcta**. Díky otevřenosti mohou členové týmu sdílet své úspěchy a neúspěchy, což jim napomáhá k vzájemné úctě a uznání. (Scrum Alliance, 2014)

Scrum začíná v momentě, kdy zákazník požaduje nějaký produkt. Produktem může být nová softwarová aplikace nebo nová funkcionalita přidávaná do existující aplikace. Vývoj je řízen inkrementálně v krátkých časových úsecích – s*printech***.** Sprinty mají pevnou délku a bývají dlouhé 1 až 4 týdny. V každém sprintu je vyvíjen a dodáván funkční přírůstek produktu. Každý takový přírůstek je v celku rozlišitelný, přináší viditelné zlepšení a splňuje předem stanovená akceptační kritéria.

4 Metodika MMSP

Tato kapitola představuje Metodiku pro malé softwarové podniky a popisuje její obecné zaměření a principy. Zbývající část kapitoly je věnována popisu disciplín, rolí a pracovních produktů, které metodika definuje.

4.1 Popis metodiky MMSP

MMSP je metodikou pro menší softwarové projekty. MMSP byla vytvořena pro potřeby výuky softwarového inženýrství na Vysoké škole ekonomické v Praze, ale autorka Rejnková (2011a) uvádí, že je možné metodiku použít i v reálných projektech v praxi. Metodika vznikla v rámci autorčiny diplomové práce na téma *Lokalizace a přizpůsobení metodiky OpenUP*.

MMSP, jak již napovídá název zmiňované práce, vychází zejména z metodiky OpenUp, konkrétně verze 1.5.1.1, a je vytvářena a publikována v rámci projektu *Eclipse Process Framework*. MMSP je inspirována i dalšími metodikami, zejména agilními. Vychází také z jiných publikací, které se věnují agilním praktikám a efektivní spolupráci v rámci projektového týmu. (Rejnková, 2011a)

4.2 Zaměření a principy metodiky

Rejnková (2011a) uvádí, že metodika MMSP je vhodná především malé projekty, jež mohou být charakterizovány těmito specifickými rysy:

- *„délka projektu do 6 měsíců,*
- *do 10 členů týmu,*
- *projekt resp. vyvíjený systém, není extra kritický.“* (Rejnková, 2011a)

Metodika kromě výše zmíněných specifik dále respektuje principy agilního modelování dle Scotta W. Amblera. Ambler (2014) uvádí 14 nejlepších praktik, z nichž pro potřeby metodiky použila autorka těchto 9:

1. *„**Aktivní účast zainteresovaných stran na projektu** – Zainteresované strany by měly každodenně spolupracovat s vývojovým týmem a poskytovat mu potřebné informace.*
2. ***Návrh architektury řešení** – Na začátku každého agilního projektu je nutné identifikovat architekturu, která je aplikovatelná pro vytvářené softwarové řešení.*
3. ***Nepřetržitá dokumentace** – V průběhu celého životního cyklu je nutné vytvářet kvalitní dokumentaci projektu.*
4. ***Dokumentovat co nejpozději** – Dokumentaci je vhodné vytvářet co nejpozději je to možné, což zabrání archivování spekulativních a často měnících se informací a pomůže zaznamenání pouze toho, co je opravdu důležité a stabilní.*
5. ***Iterativní modelování** - Na začátku každé iterace je vhodné modelovat, alespoň jako součást plánování.*

6. ***Dostatečně dobré artefakty*** *– Modely či dokumenty je nutné vytvářet tak, aby byly pro danou situaci dostačující. Jakékoliv nadbytečné informace jsou zbytečné.*
7. ***Více modelů*** *– Je vhodné vytvářet různé typy modelů, protože každý z nich nahlíží na znázorňovanou realitu z jiného úhlu pohledu a je použitelný v jiné situaci.*
8. ***Požadavky dle priorit*** *– Agilní týmy zpracovávají požadavky dle priorit, které jsou určeny zainteresovanými stranami a sledují co nejvyšší návratnost investic.*
9. ***Předvídání požadavků*** *– Na začátku agilního projektu je nutné identifikovat rozsah projektu a vytvořit prvotní seznam požadavků setříděný dle priorit."* (Rejnková, 2011a)

Autorka dále doplňuje, že metodika je vhodná především pro vývoj doplňkových systémů a systémů podporujících fungování organizace. Pro vývoj systémů, které jsou kritické pro poslání podniku, nebo systémů, na kterých závisí životy lidí, není metodika příliš vhodná.

4.3 Oblasti metodiky

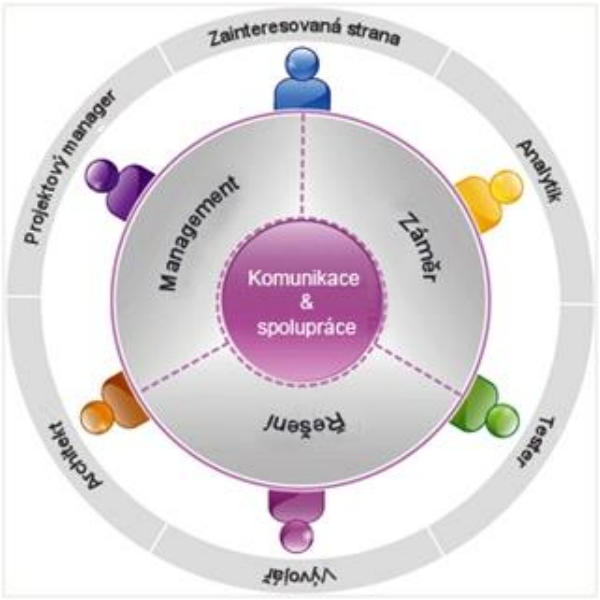
MMSP je rozdělena do pěti oblastí. Kromě úvodu zde lze nalézt popis rolí, disciplín, pracovních produktů a životního cyklu. Jednotlivé oblasti jsou blíže charakterizovány v následujících podkapitolách.

4.3.1 Úvod do metodiky

První z oblastí je *Úvod do metodiky*, ve kterém autorka uvádí, čím tato metodika je, k čemu slouží a jak s ní pracovat. Jsou zde uvedeny zaměření a principy metodiky a také zdroje, ze kterých autorka při tvorbě metodiky čerpala.

4.3.2 Role

Další oblast metodiky definuje *role*, které jsou jednotlivým členům vývojového týmu přiřazeny. Role a jejich zaměření znázorňuje *Obr. 4-1 Role a jejich zaměření* , ze kterého je patrné, že některé osoby, například testeři, se podílí na více úlohách. Kromě těchto šesti rolí metodika obsahuje roli „Nedefinovaná", která označuje osoby, které na projektu spolupracují, ale do žádné z těchto rolí nespadají. (Rejnková, 2011a)



Obr. 4-1 Role a jejich zaměření (Rejnková, 2011a)

4.3.3 Disciplíny

Další oblast metodiky, *disciplíny*, charakterizuje kolekci úloh, které se vztahují k určité oblasti zájmu, což je typické spíše pro tradiční vývoj. Autorka proto upozorňuje, že *„zařazení úloh do vybraných disciplín je považováno za efektivní a přehledný způsob jejich organizace, který usnadňuje pochopení jejich vzájemných vztahů. Je však nutné si uvědomit, že na základě z této klasifikace není možné plánovat životní cyklus projektu“*. (Rejnková, 2011a)

MMSP definuje následující disciplíny:

- **Řízení projektu.** V této disciplíně jsou vysvětleny základní metody a postupy potřebné pro výkon role Projektového manažera, za účelem podpory a motivace týmu a efektivní spolupráce se všemi zainteresovanými stranami.
- **Požadavky.** Tato disciplína poskytuje způsoby, jakými lze požadavky na vyvíjený software efektivně získat, analyzovat, specifikovat a spravovat.
- **Architektura.** Tato disciplína určuje, jak má být na základě definovaných požadavků navrhována a zpřesňována architektura vyvíjeného softwaru.
- **Vývoj.** V této disciplíně je popsáno, jakým způsobem by mělo být navrženo a implementováno technické řešení tak, aby bylo v souladu s definovanou architekturou a specifikovanými požadavky.
- **Řízení změn a konfigurací.** Hlavním cílem této disciplíny je zajistit konzistenci všech pracovních produktů, které jsou v průběhu životního cyklu projektu vytvářeny a také efektivně zaznamenávat a řídit požadavky na změny.
- **Testování.** Účelem této disciplíny je získání zpětné vazby o vyvíjeném softwaru prostřednictvím plánování, přípravy, provádění a vyhodnocování testů. (Rejnková, 2011a)

Pro účely této práce je důležité blíže popsat právě disciplínu *Testování*. Ostatní disciplíny jsou s touto disciplínou z velké části provázány vzniklými artefakty, jakými jsou např. Popis architektury, Soupis požadavků, Případy užití, ale také Plán projektu a Plány jednotlivých iterací. S artefakty vzniklými při testování pracují i ostatní členové týmu, ne jen Testeři. Testeři za ně ale ve všech případech odpovídají.

4.3.4 Pracovní produkty

V této oblasti metodiky je poskytován souhrnný přehled všech pracovních produktů (artefaktů), které by měly být v průběhu vývoje vytvářeny a udržovány. Pracovní produkty jsou zde popsány jak z hlediska obsahu, tak i následků, které by mohly nastat jejich absencí. Dále jsou charakterizovány situace, ve kterých není nutné dané artefakty vytvářet. Součástí MMSP jsou též šablony, obsahující instrukce k jejich vyplnění, které definují doporučenou strukturu pracovních produktů.

4.3.5 Životní cyklus

Oblast *životního cyklu* popisuje proces vývoje softwaru dle metodiky MMSP. Cyklus je tvořen posloupností fází, iterací, aktivit a úloh. Jejich popis znázorňuje *Tab. 4-1 Prvky životního cyklu vývoje softwaru*

Tab. 4-1 Prvky životního cyklu vývoje softwaru (Rejnková, 2011a)

Název	Popis	Příklady
Úloha	Popis, jak pracovat, aby bylo dosaženo stanovených cílů či byly vytvořeny určité pracovní produkty. Úlohy jsou vykonávány rolemi a jsou obvykle definovány jako série dílčích kroků.	Návrh architektury, Řízení iterace, Provedení testů
Aktivita	Množina souvisejících úloh.	Zahájení projektu, Testování řešení
Iterace	Opakovatelná sekvence dílčích aktivit.	Iterace Zahájení, Iterace Rozpracování
Fáze	Dílčí část životního cyklu vývoje softwaru, která je tvořena jednou nebo větším počtem iterací této fáze.	Fáze Konstrukce, Fáze Zavedení

Životní cyklus je podle metodiky MMSP rozdělen do čtyř fází: zahájení, rozpracování, konstrukce a zavedení. V jednotlivých fázích může být provedeno několik jejích iterací, které se mohou lišit délkou i podrobností prováděných úloh. Autorka dále uvádí, že by jednotlivé iterace měly trvat 2 až 4 týdny, neboť takto zvolené časové období poskytne členům týmu dostatek času pro vypracování dílčích cílů bez stresu a rovněž jim umožní získat včas zpětnou vazbu. (Rejnková, 2011a)

4.4 Testování v metodice MMSP

Proces testování v metodice popisuje disciplína *Testování*, ve které jsou uvedeny základní úlohy, které provádí osoba charakterizovaná rolí *Tester*. V průběhu těchto úloh dochází ke vzniku, změně a opakovanému použití několika pracovních produktů.

4.4.1 Disciplína testování

Disciplína *Testování* popisuje proces efektivního získání zpětné vazby o vyvíjeném softwaru. Pro tento účel jsou v disciplíně definovány 3 úlohy: *Plánování testů*, *Příprava testů* a *Provedení testů*. V každé iteraci vývojového cyklu jsou prováděny všechny tyto úlohy. V některých případech je možné provést více těchto testovacích iterací v rámci vývojové iterace. Tato disciplína umožňuje zejména zajištění včasné a časté zpětné vazby, objektivní měření zlepšení kvality v jednotlivých přírůstcích (inkrementech), identifikaci nesrovnalostí a chyb řešení a ověření, zda provedené změny negativně neovlivnily již otestované části softwaru. (Rejnková, 2011b)

Disciplína je složena z úloh:

- Plánování testů,
- Příprava testů,
- Provedení testů.

V rámci úlohy *Plánování testů* je vymezena část softwarového systému, která bude testována, jsou určeny základní testovací techniky, je vytvořen orientační harmonogram testování a jsou určeny osoby odpovědné za přípravu a provedení různých typů testů. Plánování testů by mělo proběhnout vždy na začátku iterace. Při plánování jsou dále definovány strategie a přístupy k testování, jednotlivé typy a úrovně testů, jsou určeny oblasti pro automatizované a manuální testování, je definován základní způsob reportování chyb. Výstupem této úlohy je pracovní produkt *Plán testů.*

Následující úloha, *příprava testů,* je zahajována v případě, kdy je nutné otestovat nový přírůstek vyvíjeného softwaru, pro který ještě nebyly vytvořeny testovací podklady. Příprava testů začíná sepsáním S*eznamu testovacích nápadů*, obvykle ještě před existencí spustitelné či testovatelné verze softwaru. Na základě testovacích nápadů jsou vytvořeny T*estovací případy*, které je možné přesně vytvořit zpravidla, až když software lze spustit. Během této úlohy mohou být nalezeny nesrovnalosti se specifikací, což je předmětem k diskuzi s projektovými analytiky. V této úloze jsou navíc vytvářeny *Testovací sady* či skripty pro automatizované testy a jsou připravována *Testovací data*.

V úloze *Provedení testů* jsou vybrány vhodné *testovací sady*, ze kterých jsou vykonávány jednotlivé testy podle *Testovacích případů.* Mimo těchto testů jsou prováděny tzv. free testy, které nejsou přesně popsány, ale jsou testerem vykonávány za účelem odhalení slabých míst testovaného softwaru. V neposlední řadě jsou prováděny retesty již opravených chyb. V případě nalezení chyby je tato skutečnost zaznamenána do T*estovacích sad*. Nalezená chyba je ohodnocena stupněm závažnosti, tedy mírou dopadu chyby na uživatele, a stupněm důležitosti, který určuje míru nutnosti opravy chyby. Po dokončení všech testů v rámci iterace je vhodné vypracovat Z*áznam výsledků testů*, který shrnuje informace o provedených testech a nalezených chybách. (Rejnková, 2011b)

4.4.2 Pracovní produkty v oblasti testování

V průběhu testování dochází k vytváření několika pracovních produktů. Pracovní produkty jsou dokumenty, které uchovávají informace potřebné k P*lánování* či P*rovádění testů* nebo slouží jako výstupní dokumenty obsahující výsledky provedených testů

MMSP definuje tyto pracovní produkty:

- **Plán testů.** Obsahuje všechny důležité informace o testování, které má být provedeno pro danou verzi vyvíjeného softwaru. Definuje, co bude otestováno, jaké testy budou prováděny. Součástí plánu testů je strategie testování, způsob reportování chyb a také harmonogram testování včetně určení odpovědných osob.

- **Seznam testovacích nápadů.** Je vytvářen v rámci přípravy testů a přehledně zachycuje problémové oblasti vyvíjeného softwaru, které by bylo vhodné otestovat. K tvorbě testovacích nápadů slouží primárně případy užití, které jsou vytvořeny v rámci specifikace. Součástí testovacích nápadů nejsou jen úspěšné průchody, ale i chybové stavy, ke kterým může dojít např. při nevyplnění povinných polí ve formuláři. Pro inspiraci při tvorbě je možné použít testovací nápady z minulých, podobných projektů.

- **Testovací případ.** Popisuje přesné kroky, které by měl tester provést pro ověření dané funkcionality. Součástí testovacího případu je popis očekávaného chování systému na testerovy akce.

- **Testovací sada.** Je soubor Testovacích případů, které jsou logicky seskupeny, za účelem kompletního otestování určité části softwaru. Testovací případy se v sadě mohou vyskytnout několikrát, avšak pokaždé s jinými vstupními parametry.

- **Testovací data.** Představují různorodou množinu dat, která je používána pro testování. Testovací data nemusí být vytvářena, pokud stačí pro účely testování pouhé parametry uvedené u daného případu užití.

- **Záznam výsledků testů.** Je pracovním produktem, který obsahuje souhrnné informace o provedených testech a jejich výsledcích. Kromě statistiky úspěšných a neúspěšných testů by měl obsahovat i podrobný report chyb. Bez záznamu výsledků testů neexistují žádné doklady o provedení testů a není možné určit, v jaké fázi se testování právě nachází. V případě automatizovaných testů mohou být reporty generovány automaticky, avšak vždy my měly obsahovat odbornou interpretaci výsledků. (Rejnková, 2011b)

4.4.3 Role testera v metodice

Tester je osoba odpovědná za testování. Tester identifikuje, definuje a provádí všechny potřebné testy, včetně jejich zaznamenání a analýzy jejich výsledků Výsledky testů pak komunikuje s ostatními členy vývojového týmu. Metodika doporučuje, aby testování vykonávalo více osob. Nejzkušenější z nich by měla být odpovědná za vytváření šablon a školení zvolené testovací metodiky. Samotné provádění testů pak mohou vykonávat méně zkušené osoby. (Rejnková, 2011b)

Úlohy a pracovní produkty spojené s rolí testera zachycuje *Obr. 4-2 Úlohy a pracovní produkty role Tester*

Obr. 4-2 Úlohy a pracovní produkty role Tester (Rejnková, 2011b)

5 Problémy agilního přístupu v praxi a jejich řešení

Tato kapitola je věnována popisu podniku, který je dále použit jako reference při identifikaci možných problémů spojených s testováním, vznikajících zejména při agilním vývoji softwaru. Identifikované problémy tohoto referenčního podniku jsou vyjmenovány, blíže popsány, zhodnoceny z hlediska míry jejich závažnosti a je navrženo jejich řešení.

5.1 Charakteristika referenčního podniku

Podnikem je malá softwarová firma, která podle Doporučení 2003/361/ES ze dne 6. května 2003 spadá do kategorie *malý podnikate*l. Podniky v této kategorii zaměstnávají méně než 50 zaměstnanců a mají roční obrat menší než 10 mil. EUR. (CzechInvest, 2014)

Podnik se mimo jiné zabývá zakázkovým vývojem software, zejména webových aplikací. Pro řízení projektu je použita upravená metodika Scrum s délkou iterace 1 měsíc. Vývoj je tak řízen agilně, a tedy iterativně a inkrementálně. Výstupem každého sprintu je fungující verze aplikace, doplněná o nové funkcionality a zbavená nalezených chyb, a protože podnik udržuje se zákazníkem stálý obchodní vztah, vývoj aplikace nikdy nekončí.

5.1.1 Požadavky

Požadavky mají formu uživatelských příběhů, tzv. „user stories“. Uživatelské příběhy pocházejí z Extrémního programování, kde byly definovány v roce 1988 jako součást plánování, a říkají, podobně jako případy užití, co má být uděláno, pro koho to má být uděláno a hlavně proč to má být uděláno. (Agile Alliance, 2013) Obecným příkladem uživatelského příběhu může být:

„Jako administrátor chci kontaktní formulář, abych se včas dozvěděl o chybách systému“. (Šochová, 2011)

Uživatelské příběhy mohou nabývat různých podob. V publikacích lze nejčastěji nalézt požadavky ve formě

„Jako Uživatel, chci Funkcionalitu, abych dostal Business Value“, (Šochová, 2011)

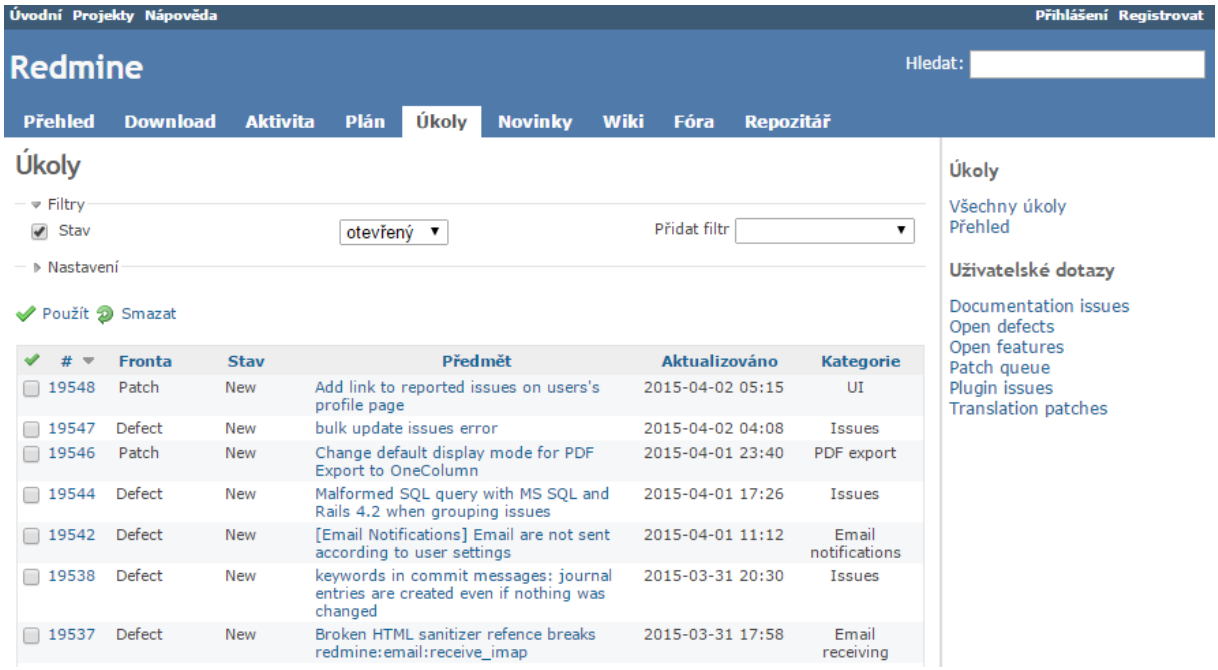
ale lze se setkat i s jinými formami. Uživatelem je myšlena osoba nebo skupina osob, které budou danou funkcionalitu, jež jim přinese obchodní hodnotu, využívat.

Požadavky přicházejí buď přímo od zákazníka, nebo od členů vývojového týmu. Protože testeři často zaujímají roli uživatele, jsou to často oni, kdo v průběhu vývoje nejen ověřují naplnění požadavků, ale rovněž vytvářejí nové požadavky. Důvodem vzniku nového požadavku může být kupříkladu chybějící mobilní verze webové stránky. Jestliže se s mobilní verzí webu původně nepočítalo, nejedná se o chybu (bug), ale právě o nový požadavek, který může být vývojovým týmem buď schválen, nebo zamítnut, závislosti na domluvě se zákazníkem.

5.1.2 Správa požadavků a chyb

Pro správu požadavků a chyb je v podniku využíván Redmine. Redmine je webová aplikace s otevřeným zdrojovým kódem, určená ke správě projektů. Každý člen vývojového týmu (včetně zákazníka) má do této aplikace přístup a může do jisté míry do jejího obsahu zasahovat.

V Redminu jsou definovány úkoly, které tým v průběhu vývoje plní. Úkoly jsou členěny podobně jako v metodice Scrum, lze zde nalézt úkoly typu Epic, Story, Task, ale také Bug. Součástí je rovněž *wiki*, ve které je možné udržovat dokumentaci vyvíjeného softwaru. K úkolům je možné psát komentáře a připomínky, přikládat obrázky a jiné datové soubory.



Obr. 5-1 Issue tracking systém Redmine (Redmine.org, 2014)

5.1.3 Průběh testování

V podniku působí pouze jeden tester. Jak již bylo zmíněno v kapitole 3.4, testování se do jisté míry účastní všichni členové týmu, ale tester je osoba provádějící uživatelské testování a z větší části i odpovídající za funkčnost dodávaného softwaru.

Tester provádí následující činnosti:

- validuje a verifikuje vyvíjený software,
- komunikuje se zákazníkem a s programátory,
- vytváří, kontroluje a průběžně doplňuje uživatelské dokumentace aplikací,
- spravuje soubor testovacích případů,
- hlásí nalezené chyby.

V počátcích vývoje nové aplikace (příp. rozšíření stávající aplikace) si tester důkladně prostuduje specifikaci aplikace, zejména její technický návrh, poznamená si problémové části

aplikace a sestaví základní případy užití. Při této činnosti nemusí nutně vzniknout žádné artefakty, protože tato fáze má za úkol pomoci testerovi ke snazší orientaci v testované aplikaci. Specifikace je tvořena analytiky a vývojáři na seniorské pozici. Tester se ve většině případů do tvorby specifikace nezapojuje, avšak může k ní vznést připomínky, které tvůrci specifikace posoudí a případně provedou potřebná opatření.

Když je hotová kostra aplikace a část kódu je testovatelná, provádí tester validaci a verifikaci funkcionalit. Vzhled aplikace v tuto chvíli není konečný a velmi často se mění, a proto je při testování kladen důraz na základní funkčnost aplikace. Tester kontroluje, zda jsou všechny součásti, které jsou označeny za hotové, skutečně v aplikaci přítomny, že rozložení ovládacích prvků je smysluplné a ovládání je intuitivní. Pokud je k dispozici grafický návrh aplikace, je použit jako reference. Vzhled a použitelnost tester prodiskutovává se zákazníkem, a také s ostatními členy vývojového týmu. Zároveň sepisuje první testovací případy, v čemž mu pomáhají dříve promyšlené případy užití a uživatelské příběhy.

Za další milník můžeme označit moment, kdy jsou naimplementovány a otestovány hlavní funkcionality, a je z velké části ustálen vzhled aplikace. Touto dobou bývají opraveny nejzávažnější chyby a aplikace je z větší části stabilní. Do aplikace jsou navíc přidávány nové vlastnosti, které nebývají plně popsány technickým návrhem. Tester musí kontrolovat, zda nové prvky nezpůsobují v aplikaci problémy a zda jsou se zbytkem aplikace konzistentní po stránce vzhledové i po stránce funkční.

Ke konci sprintu jsou zpravidla všechny uživatelské příběhy otestovány a tým se zaměřuje na opravy chyb. Náplní testerovy práce jsou tak především retesty dříve nalezených a již opravených chyb. Tester podle potřeby aktualizuje uživatelskou dokumentaci aplikace tak, aby odpovídala aktuálnímu stavu aplikace. Nezávažné problémy, které není možné včas vyřešit, jsou přesunuty do dalšího sprintu.

5.1.4 Hlášení chyb a jejich řešení

Když tester, nebo jakýkoliv jiný člen týmu, nalezne chybu, zpravidla ji ihned zadá do Redminu (viz kap. 5.1.2). Pokud si tester není jistý, zda se jedná o chybu, projedná nejprve nalezený problém s ostatními vývojáři. Pokud potřebuje ke správnému zadání chyby znát více informací, může také kontaktovat zákazníka a vyžádat si od něj bližší informace. Typickým příkladem je chybějící nebo očividně nesprávný text v aplikaci. Tester by měl nejen uvést, proč si myslí, že je text nesprávný, ale také by měl uvést jeho správné znění. Nejedná-li se o pouhý překlep a není-li znění textu specifikováno, kontaktuje tester zákazníka a požádá ho o dodání tohoto textu.

Podoba chybových hlášení je z části dána nastavením Redminu. Pro bug tracker jsou kromě běžných polí (Identifikátor, datum a čas, projekt apod.) využívána následující pole:

- **Autor** – Osoba, která chybu nahlásila, většinou tester nebo zákazník.
- **Tracker** – Určuje typ úkolu, tedy rozlišuje Bug, Story, Epic a Task.
- **Priorita** – Nízká, normální, vysoká, urgentní, okamžitá.
- **Stav** – Nový, přiřazeno, vyřešeno, otestováno, zavřeno, zamítnuto.
- **Kategorie** – Kategorie, do které chyba spadá (objednávky, administrace atd.)

- **Předmět** – Stručný a výstižný popis problému.
- **Popis** – Detailní popis problému včetně kroků k nasimulování, údajů nezbytných k nasimulování, specifických postřehů, důsledku chyby, očekávaného chování, výpisu z aplikačního logu.

Kromě zmíněných polí je možné k chybovému reportu přiložit datový soubor, který může obsahovat snímek obrazovky s popisovanou chybou (velmi užitečné při nálezu grafické chyby nebo překlepu). V případě chyby spojené s importem či exportem dat lze přiložit importovaný, resp. exportovaný soubor.

Proces nahlášení a následného zpracování chyby vyobrazuje *Příloha 1 Proces nahlášení a zpracování chyby*.

5.2 Identifikované problémy referenčního podniku

V této kapitole jsou představeny problémy referenčního podniku, které byly identifikovány testerem a dalšími členy vývojového týmu. Problémy jsou blíže popsány, je zhodnocena jejich závažnost a posouzena složitost jejich vyřešení. Součástí popisu jednotlivých problémů je navržené řešení, které má za úkol odstranit, nebo alespoň zmírnit, vliv identifikovaných problémů.

5.2.1 Přehled identifikovaných problémů

Tab. 5-1 zobrazuje přehled identifikovaných problémů, které souvisí s testováním softwaru v referenčním podniku. Každý problém je blíže popsán v samostatné podkapitole. U každého identifikovaného problému je v tabulce uvedena jeho míra závažnosti a předpokládaná složitost vyřešení. Oba tyto ukazatelé jsou pouze subjektivní, neboť objektivní zhodnocení problémů by bylo velmi složité a vyžadovalo by hlubší studii na toto téma.

Tab. 5-1 Seznam nalezených problémů, které se týkají testování. (autor)

ID	Problém	Závažnost	Složitost vyřešení
P1	Nedostatečný popis uživatelských příběhů	Vysoká	Nízká
P2	Nejednotná správa testovacích případů	Nízká	Vysoká
P3	Absence automatizovaného testování	Střední	Vysoká
P4	Chybějící uživatelské dokumentace u některých projektů	Vysoká	Střední
P5	Absence testovacích reportů	Nízká	Nízká
P6	Chybějící metriky v procesu testování	Střední	Střední
P7	Nejednotnost chybových hlášení	Střední	Střední

5.2.2 Nedostatečný popis uživatelských příběhů

Hlavním problémem při testování podle uživatelských příběhů je jejich nedostatečný popis. Tester se z jejich popisu obvykle nedozví příliš mnoho informací o výsledném řešení dříve, než začne s testováním. Pokud požadovaná změna není popsána v některém z dokumentů, například v technickém návrhu, nemůže tester spolehlivě určit, zda jsou všechny nové změny žádané,

fungují správně a zda žádná z požadovaných změn nechybí. Může se také stát, že daná funkcionalita je zmíněna v nějakém dokumentu, ale je buď nedostatečně podrobně popsána, nebo její popis není aktuální.

Aby tester získal potřebné informace, musí se zpravidla obrátit na programátora/analytika, což zabere určité množství času oběma. Nastává-li tato situace často, přirozeně roste i celkový čas strávený nad těmito konzultacemi, což může mít nepříznivý účinek na dokončení sprintu. Jejich osobní konverzace, která je v agilním vývoji velmi častá, není v budoucnu zpětně dohledatelná.

Problému by se dalo předejít vyčerpávajícím popisem funkcionality ještě před začátkem implementace, ale taková myšlenka jde proti agilním principům. Jak má daná věc fungovat, nemusí být navíc předem známé, protože se programátor může rozhodnout funkcionalitu v době psaní kódu drobně upravit, aby zajistil její správné chování nebo uživatelskou přívětivost.

Vhodným řešením jmenovaného problému je okomentování výsledného řešení po jeho implementaci. Popis řešení se stane součástí úkolu a bude zpřístupněn jak testerovi, tak zákazníkovi a ostatním členům vývojového týmu. Ještě před samotným testováním může kdokoliv k úkolu vznést připomínky. Popis bude stručný, výstižný a bude členěn na dvě části. První část bude popisovat dopad řešení na uživatele. V druhé části budou obsaženy informace o technickém řešení z pohledu aplikace, což může programátorům pomoci také při revizi kódu. Navíc, pokud programátor ví o možném problému, který by mohl při testování nastat, může jej uvést jako doplňující informaci a pomoci tak případnému odhalení chyby.

5.2.3 Nejednotná správa testovacích případů

Většina testovacích případů, které zkoumaný podnik využívá, je vytvořena v tabulkách Google Sheets, jejichž výhodou je snadná možnost spolupráce několika členů týmu najednou a také jejich přenositelnost. Existují ale další formy testovacích případů, které podnik používá. Mezi ně patří testovací případy sepsané na wiki stránkách v Redminu (viz kap. *5.1.2 Správa požadavků a chyb*), jiné testy jsou obsaženy v dokumentech napsaných v jazyce LaTeX. Hlavním problémem tohoto řešení je jeho nejednotnost. Na testovací případy je potřeba se během testování a hlášení chyb odkazovat. Různé testovací scénáře a případy obsahují různé informace a popisují testy jiným jazykový stylem.

Správa testů by měla být jednotná a snadno kontrolovatelná. Vývojovému týmu pak dokáže odpovědět na otázky (Crispin, a další, 2009):

- Které testy byly automatizovány?
- Které testy je ještě potřeba automatizovat?
- Které testy jsou nyní používány jako součást regresní sady?
- Jaké testy pokrývají konkrétní oblast aplikace?
- Jak je funkcionalita XYZ navržena a jak by měla fungovat?
- Kdo napsal uvedený testovací případ a kdy? Kdo jej naposledy upravil?
- Jak dlouho je uvedený test součástí regresní sady?

Testy jsou přirozenou součástí vývoje softwaru a slouží jako součást dokumentace aplikace, a proto by měly být přehledně organizovány. (Crispin, a další, 2009)

Ke správě testovacích případů existuje velké množství nástrojů. Srovnání některých nástrojů lze nalézt v diplomové práci Ing. Robina Štolce (2010) s názvem *„Porovnání komerčních a open source nástrojů pro testování softwaru“* nebo v diplomové práci Ing. Vladana Vachalce (2014) s názvem *„Testování a kvalita softwaru v metodikách vývoje softwaru“*.

Možným řešením tohoto problému by bylo zvolení a zavedení některého z nástrojů pro správu pracovních produktů, spojených s testováním, zejména testovacích případů, testovacích sad a testovacích dat. Údržbu nástroje a jeho obsahu by měl na starosti tester. Informace v nástroji by byly dostupné i ostatním členům týmu.

5.2.4 Absence automatizovaného testování

Pro agilní vývoj jsou typické drobné přírůstky funkcionalit. Každá malá změna ale může mít zásadní dopad na celkovou funkčnost vyvíjeného systému. Proto pro kontrolu již hotových a otestovaných částí systému, provádí tester pravidelně tzv. regresní testování. Testování již hotových částí ale může zabrat mnoho času, a tak je tento proces vhodné automatizovat. Crispin a Gregory (2009) uvádějí, že *„díky automatizaci mohou lidé dělat to, v čem jsou nejlepší.“* Automatizované testy navíc slouží jako dokumentace vyvíjeného systému.

Zavedení automatizovaných testů stojí velké úsilí a je časově náročné, a proto by měl vývojový tým před jeho zavedením zvážit, zda se mu tato investice vyplatí. Výsledky špatně navržených automatizovaných testů je složité interpretovat a obvykle vyžadují přítomnost zkušené osoby. Problémem při zavádění automatizace může být také starý programový kód. Aby bylo možné proces testování automatizovat, musí aplikace obvykle splňovat určité předpoklady, mezi které patří například jedinečné identifikátory tlačítek a formulářových polí ve webových aplikacích. Zavedení takových změn zpětně ale vyžaduje velké množství úsilí a obvykle se nevyplatí. (Crispin, a další, 2009)

Testování lze automatizovat několika způsoby. Nejběžnějším příkladem jsou jednotkové testy, které jsou už v podniku přirozenou součástí vývoje a jsou udržovány programátory. Tester může navíc pro účely regresního testování použít několik dalších typů nástrojů:

- **Nástroje pro tvorbu testovacích dat.** Do této kategorie se řadí nástroje, které testerovi mohou pomoci s vymýšlením smysluplných dat. Testovací data by měla být různorodá. Nutnost vymýšlet testovací data testera zdržuje, byť nepatrně. Testovací data, která jsou smysluplná, navíc pomáhají ve vymýšlení případů užití. Záznamy o chybě pak vypadají lépe, než když jsou ve snímcích obrazovky použity nesmyslné řetěze typu „asdf123123“. Na druhou stranu lze využít nástrojů, které jsou schopny vytvořit řetězce ze speciálních znaků, za účelem důkladného otestování vstupních míst aplikace. Testovací data lze v určitých případech získat z produkční databáze, která ovšem nemusí obsahovat vhodná data. Produkční data je navíc potřeba zbavit citlivých údajů. Za zmínku stojí on-line nástroj s otevřeným zdrojovým kódem, který je dostupný na `http://www.generatedata.com/`. Pomocí něj lze vytvářet velké množství různých dat v mnoha souborových formátech. Dokáže také vytvořit SQL skript pro snadné nahrání testovacích dat (viz níže).

- **Nástroje pro nahrávání testovacích dat**. Velké množství dat může způsobit výrazné zpomalení aplikace. Tento problém je nutné odhalit během testování, protože by mohl výrazným způsobem snížit hodnotu pro zákazníka. Pokud máme k dispozici nástroj, který umožňuje taková data vytvořit, je nutné je do aplikace nějakým způsobem přenést. Ne vždy aplikace obsahuje funkci importu dat, a tak je často potřeba dostat data do aplikace jinak. Ruční i automatizované vyplňování dat, ať už přes formulář aplikace nebo přes volání webových služeb, je relativně pomalé. Nejjednodušším způsobem je přímý zásah do databáze.

- **Nástroje pro vyplňování formulářů.** Pro snadnější testování webových formulářů je možné využít doplňků či vestavěných funkcí webových prohlížečů, které dokáží automaticky vyplnit formuláře. Použití takových nástrojů je klíčové při testování formulářů, neboť výrazně zrychluje a usnadňuje testerovi práci. Automatické vyplňování webových formulářů podporuje například prohlížeč Google Chrome. Pro desktopové aplikace je možné využít některých nástrojů podporujících zaznamenávání a přehrávání akcí, viz níže.

- **Nástroje pro zaznamenávání a přehrávání akcí uživatele.** Pro testování webových aplikací je možné využít nástroj Selenium (`http://www.seleniumhq.org/`), pomocí něhož lze zaznamenávat akce uživatele a ty poté v prohlížeči opět přehrávat. U každé akce lze definovat i očekávanou reakci testované aplikace. Pro desktopové aplikace s grafickým uživatelským rozhraním na platformě Windows je možné použít nástroj AutoIt, dostupný na adrese `https://www.autoitscript.com/`. Pro Linux existuje například alternativní nástroj AutoKey.

- **Nástroje pro testování API**. Při testování programového rozhraní aplikace je potřeba dodržet přesnou formu vstupních dat. Pro snadnou obsluhu existují nástroje s grafickým uživatelským rozhraním, které přinášení i možnost rychlé a pohodlné změny testovacích dat a umožňují jednotlivá volání ukládat či automaticky vyhodnocovat odpovědi aplikace na základě předem stanovaných pravidel. Příkladem nástroje na testování webových služeb je SoapUI. Jedná se o nástroj podporující velké množství protokolů, mezi které patří SOAP, REST, HTTP a další. Nástroj je zdarma dostupný na adrese `http://www.soapui.org/` a mezi jeho vlastnosti patří také tvorba a vyhodnocení automatizovaných testovacích scénářů.

Zkoumaná metodika MMSP přímo úlohy spojené s automatizovaným testováním nedefinuje. Tyto činnosti je proto potřeba přidělit některé z rolí. Protože automatizace vyžaduje specifické znalosti, měla by pro tyto potřeby vzniknout nové role, které se bude zabývat čistě automatizací.

5.2.5 Chybějící uživatelské dokumentace u některých projektů

Při tvorbě testů mohou testerovi pomoci již existující podklady, mezi které patří i dokumentace projektů. V agilním vývoji se ale může stát, že z časových nebo rozpočtových omezení není možné vytvořit dokumentaci, která by dostatečným způsobem pokrývala požadovanou oblast. Testování části aplikace, o které tester předem nic neví je problematické.

Pokud je potřeba vytvářet uživatelskou dokumentaci aplikace, je pro tuto činnost vhodným kandidátem právě tester, neboť zná potřeby uživatele systému a ví, s jakou částí aplikace by

mohl mít uživatel problém. Při psaní dokumentace se navíc může důkladně zamyslet nad tím, jestli dávají popisované vlastnosti smysl a zda je chování i vzhled nových součástí konzistentní se zbytkem aplikace. Tvorba dokumentace probíhá podle agilních principů v rámci iterace co nejpozději, zejména z důvodů stálých změn. Popisovány by tak měly být jen řádně otestované části aplikace.

5.2.6 Absence testovacích reportů

Na konci sprintu by měly být sestaveny testovací reporty. Ty by měly napomáhat předcházení stejným chybám v budoucnosti, což uvádí například Patton (2001). Nepřítomnost testovacích reportů zpravidla nemá tak velký vliv na kvalitu produktu, jako ostatní zmíněné problémy. Slouží nicméně jako zpětná vazba, která pomáhá odhalit nedostatky v procesu vývoje softwaru.

MMSP definuje šablonu pro testovací report s názvem *záznam výsledků testů*, jejíž součástí je tabulka se současným počtem nových chyb a celkovým počtem chyb. Údaje jsou členěny podle stupně závažnosti chyby.

Testovací report by měl být obsáhlejší a měl by obsahovat více ukazatelů (viz kap. *5.2.7. Chybějící metriky v procesu testování*). Také by jej bylo vhodné považovat za samostatný pracovní produkt, jehož autorem by byl právě tester. Sestavený report by sloužil nejen testerovi, ale i ostatním členům týmu, zejména projektovému manažerovi.

5.2.7 Chybějící metriky v procesu testování

V návaznosti na předchozí problém, *Absence testovacích reportů*, by měly být v testovacím reportu vhodné metriky, které by vypovídaly o výsledcích testování. Protože jsou testovací reporty sestavovány jen na konci iterace, je vhodné vytvořit způsob, kterým by bylo možné okamžitě nahlédnout na vývoj testování nejen v rámci iterace, ale také v rámci celého vývoje projektu. Důležitým ukazatelem je například počet chyb, které se vyskytnou po nasazení aplikace do produkčního provozu. (Crispin, a další, 2009)

Pro vyřešení tohoto problému je potřeba vybrat vhodné metriky, na základě nich provést sběr dat a data vhodně interpretovat. Výsledky tohoto procesu se tak stanou součástí nově vzniklého testovacího reportu. Výběr metrik bude mít, stejně jako testovací report, na starosti tester

5.2.8 Nejednotnost chybových hlášení

Způsob hlášení nalezených chyb je popsán v kapitole *5.1.4 Hlášení chyb a jejich řešení*. Všechny důležité informace o chybě jsou v Redminu ukládány do jednoho pole – Popis. Do tohoto pole může uživatel vložit prakticky cokoliv. To způsobuje nejednotnost jednotlivých hlášení, zvláště pokud chyby nahlašuje několik různých uživatelů. Chybová hlášení se stávají součástí dokumentace projektu, a proto by měla mít jednotnou strukturu a jednotný styl. Chybové hlášení by mělo být stručné, úplné a výstižné. Pokud testovaná aplikace vyžaduje přihlášení uživatele, je potřeba uvést i potřebné přihlašovací údaje. V případě grafické chyby je velmi vhodné přiložit i snímek obrazovky.

Přestože se může při nahlašování chyby zdát, že jsou všechny informace jasné, po delší době tomu tak být nemusí. Nekritické chyby mohou být přesunuty do dalších sprintů. Oprava chyby

tak může být zahájena například po dvou letech od jejího nahlášení. Z důvodu nedostatečného popisu pak není možné chybu nasimulovat a vývojový tým může být mylně přesvědčen, že byla chyba v minulosti opravena v rámci jiného úkolu.

MMSP definuje šablonu s názvem *Reporty chyb*, která je součástí pracovního produktu *Záznam výsledků testů*. Její struktura je velmi podobná chybovým hlášením v Redminu, které podnik využívá. Pro popis chyby slouží jedno pole, do kterého jsou vkládány všechny informace potřebné k nasimulování. Samotná struktura popisu není definována. (Rejnková, 2011a)

Pro zajištění jednotné struktury chybových hlášení je potřeba doplnit šablonu pro reporty chyb. Pole *Popis* bude mít následující strukturu:

- **Prostředí**. Při vývoji bývá k dispozici několik prostředí, ve kterých je aplikace spouštěna. Mezi takové patří například:
    - **Vývojové** – obsahuje vždy nejnovější změny.
    - **Předváděcí** – obvykle obsahuje starší verzi aplikace a podmínkami se snaží přiblížit prostředí produkčnímu.
    - **Produkční** – na něm se nachází produkční instalace aplikace.
- **Verze aplikace/sestavení.** V praxi se při testování může stát, že není jedno z prostředí aktuální, a tedy neobsahuje poslední změny. Díky uvedené verzi aplikace nebo sestavení je možné zamezit případným nedorozuměním, neboť nahlášená chyba se v nejnovější verzi už nemusí vyskytovat.
- **Klientské prostředí**. Zde může být uvedena verze operačního systému, typ a verze webového prohlížeče nebo jiného potřebného softwaru, například JRE (Java Runtime Environment).
- **Potřebné údaje.** Pokud je k nasimulování vyžadováno přihlášení uživatele, je nutné uvést potřebné přihlašovací údaje. Pokud testovaná aplikace rozlišuje několik uživatelských rolí s různými oprávněními, je vhodné uvést také roli uživatele. Dalším příkladem mohou být údaje testovací platební karty nebo soubor určený k importu do aplikace.
- **Kroky k nasimulování.** V bodech popsané kroky k nasimulování. Vedle kroků mohou být uvedeny poznámky k chování testované aplikace v jednotlivých krocích.
- **Očekávané chování**. Popis očekáváného chování, který je obvykle obsažen v testovacích případech.
- **Skutečné chování**. Popis, obvykle chybného, současného chování aplikace, který je výsledkem provedených kroků.
- **Doplňující informace.** Zde mohou být uvedeny možné souvislosti a postřehy, které mohou s chybou souviset. Pokud si je tester téměř jistý původem problému, může na konec hlášení doplnit informaci o možné příčině. Tato informace by neměla být považována za absolutně pravdivou, může ale pomoci při řešení problému.

Nehodící se položky lze vynechat. Položky, které se stanou běžnou součástí hlášení, ale u výjimečných případů je nelze zjistit, by měly být v takovém případě označeny jako *„neznámé*“, aby došlo k zachování jednotné struktury hlášení.

5.2.9 Shrnutí a zhodnocení problémů

Výše uvedené problémy ovlivňují kvalitu procesu testování různým způsobem a představují různý stupeň ohrožení pro referenční podnik. Za nejzávažnější a zároveň nejsnáze řešitelný problémem považuji problém *P1 – Nedostatečný popis uživatelských příběhů*. Naopak za nejhůře řešitelný problém považuji P5 – Absence automatizovaného testování, neboť je tato činnost velmi nákladná a vyžaduje speciální znalosti. Vyřešení zmíněných problémů by pomohlo zajistit kvalitnější a efektivnější proces testování softwaru v podniku. Rozšířením metodiky o navržené změny by následně bylo možné uplatnit velkou část disciplíny *Testování* v uvedeném podniku. Změny v metodice popisuje následující kapitola *Rozšíření metodiky MMSP*.

6 Rozšíření metodiky MMSP

Cílem této kapitoly je rozšířit metodiku MMSP (viz kap. 4) s ohledem na problémy referenčního podniku, které byly popsány v kapitole 5.2. Kapitola obsahuje také zhodnocení navržených rozšíření z hlediska míry vyřešení těchto problémů. V následujících podkapitolách jsou uvedeny nově vzniklé role, úlohy a pracovní produkty v metodice. V závěru kapitoly je uveden také očekávaný přínos těchto změn.

6.1 Role

Tato část kapitoly popisuje úpravy původních rolí *Tester* a *Vývojář* a představuje nově vzniklou roli *Správce automatizace*.

6.1.1 Tester

Tester provádí nově tyto úlohy, za které rovněž odpovídá:

- Výběr nástroje pro správu testů,
- Údržba nástroje pro správu testů,
- Výběr vhodných testovacích metrik,
- Sběr dat na základě vybraných metrik,
- Interpretace dat z metrik,
- Tvorba testovacího reportu,
- Tvorba a správa uživatelské dokumentace,
- Výběr a zavedení nástroje pro správu testů,
- Údržba nástroje pro správu testů,
- Kontrola popisu implementace.

Většina nově vzniklých úloh vyžaduje schopnosti a informace, které má k dispozici právě Tester, proto je tato role jejich vhodným vykonavatelem. Díky výborné znalosti vyvíjeného systému je navíc možné, aby Tester vytvářel a spravoval uživatelskou dokumentaci. Tester se stále zabývá pouze manuálním testováním, proto není potřeba, aby nabýval znalostí z oblasti automatizace.

6.1.2 Správce automatizace

Provádí úlohy:

- Analýza a tvorba automatizovaných testů,
- Spouštění automatizovaných testů,

- Vyhodnocování automatizovaných testů.

Automatizace vyžaduje speciální schopnosti a znalosti, a proto není možné výše jmenované úlohy přiřadit k roli Tester. Automatizované testování navíc nemusí probíhat vždy, proto je možné v takovém případě z metodiky jednoduše vyřadit celou roli Správce automatizace, včetně souvisejících úloh a pracovních produktů.

6.1.3 Vývojář

Vývojář navíc provádí úlohu *Kontrola popisu implementace*, která zahrnuje případné doplnění chybějícího popisu.

6.2 Úlohy

Následující podkapitoly popisují nové úlohy, vzniklé na základě navržených řešení. Součástí popisu každé úlohy je odkaz na problémy, které úloha přímo nebo částečně řeší.

6.2.1 Výběr nástroje pro správu testů

Cílem je vybrat nejvhodnější nástroj, který co nejlépe podpoří proces testování. Díky jednotnému nástroji získají členové týmu, zejména Testeři, přehled o všech vytvořených a provedených testech a budou moci budoucí změny v aplikaci lépe promítnout do testovacích scénářů. Vhodným propojením tohoto nástroje s nástrojem pro zaznamenávání chyb, může tým docílit mnohem efektivnější spolupráce. S výběrem nástroje může testerovi pomáhat Analytik nebo Projektový manažer. Zavedení této úlohy pomůže odstranit problém *P2 – Nejednotná správa testovacích případů*.

Vykonává: Tester

Dále může vykonávat: Analytik, Projektový manažer

Povinné vstupy: -

Nepovinné vstupy: Vize, Požadavky

Výstupy: Popis nástroje pro správu testů

6.2.2 Údržba nástroje pro správu testů

Cílem je zajišťovat funkčnost a dostupnost vybraného nástroje. V případě, že se na projektu podílí více testerů, je potřeba, aby někdo z nich kontroloval konzistenci tetovacích případů, testovacích dat a jiných pracovních produktů, které jsou nástrojem spravovány. Vývoj nové verze aplikace může znamenat nejen nutnost změny několika těchto pracovních produktů, ale také jejich uspořádání v rámci nástroje pro správu testů. Tester se stane rovněž správcem tohoto nástroje a bude odpovídat za jeho obsah a správný chod. Částečnou správu může provádět také Projektový manažer, může mít na starosti přidělování práv k přístupu jednotlivým osobám, které mají zájem nástroj využívat. Zavedení této úlohy pomůže udržovat vybraný nástroj a zajistit tak, aby byl minimalizován vliv problému *P2 – Nejednotná správa testovacích případů*.

Vykonává: Tester

Dále může vykonávat: Projektový manažer

Povinné vstupy: Popis nástroje pro správu testů

Nepovinné vstupy: -

Výstupy: -

6.2.3 Výběr vhodných testovacích metrik

Cílem je vybrání množiny metrik, které budou nejvhodněji a nejpřesněji vystihovat aktuální stav testování v projektu. Vybrání správných metrik je klíčové pro rozhodování, a tak se této činnosti může věnovat i projektový manažer. Tato činnost napomůže vyřešení problému *P6 – Chybějící metriky v procesu testování*.

Vykonává: Tester

Dále může vykonávat: Správce automatizace, Projektový manažer

Povinné vstupy: Plán testů

Nepovinné vstupy: -

Výstupy: Seznam vybraných metrik

6.2.4 Sběr dat na základě vybraných metrik

Cílem je sběr dat, které byly pořízeny za pomoci sestavených metrik. Na základě těchto metrik jsou Testerem v průběhu testování sbírána data, která jsou dále vyhodnocována. Sběr dat může být prováděn automatizovaně, proto je Správce automatizací vhodnou osobou, která může Testerovi pomoci se sběrem dat. Proces je také v zájmu Projektového manažera, ten je schopen poskytnout přístup k některým potřebným datům o vývoji a stavu projektu. Sběr dat a jeho následná interpretace pomáhá vyřešení problému *P6 – Chybějící metriky v procesu testování*.

Vykonává: Tester

Dále může vykonávat: Správce automatizace, Projektový manažer

Povinné vstupy: Seznam vybraných metrik

Nepovinné vstupy: -

Výstupy: Data a informace získané z metrik

6.2.5 Interpretace dat z metrik

Cílem je vyhodnocení dat, které byly pořízeny pomocí sestavených metrik. Tato část je pro úspěšnost vyřešení problému *P6 – Chybějící metriky v procesu testování* kritická. Nesprávnou interpretací dat může Tester získat naprosto odlišnou představu o stavu testování. Tuto činnosti by proto měl vykonávat jen zkušený Tester. Ideálně by se mělo jednat o stejnou osobu, která metriky navrhla nebo pomohla sestavit. V některých případech může tuto roli zastoupit

Projektový manažer nebo Správce automatizace, pokud mají dostatečné zkušenosti nebo jim zkušený Tester připravil pomocné podklady. Tyto informace slouží k získání okamžitého přehledu o stavu testování a měly by být vyhodnocovány každý den. Na konci vývojové iterace je sestaven Testovací report, který shrnuje informace získané za celou dobu testování.

Vykonává: Tester

Dále může vykonávat: Správce automatizace, Projektový manažer

Povinné vstupy: Seznam vybraných metrik, Data a informace získané z metrik

Nepovinné vstupy: -

Výstupy: Data a informace získané z metrik

6.2.6 Tvorba testovacího reportu

Cílem je sestavení kompletního testovacího reportu. Report je sestaven na základě několika dalších vstupů. Jedním ze vstupů je soubor již vyhodnocených dat nasbíraných z metrik. Účelem Testovacího Reportu je shrnout informace z metrik, které byly v rámci předcházejících úloh získány. Report je sestavován vždy na konci iterace. Jeho tvorba přímo řeší problém *P5 – Absence testovacích reportů*. Protože pro sestavení reportu je většina informací již vyhodnocená, není tento proces zvláště složitý a může být vykonáván i méně zkušenými testery.

Vykonává: Tester

Dále může vykonávat: Správce automatizace

Povinné vstupy: Seznam vybraných metrik, Data a informace získané z metrik

Nepovinné vstupy: Plán testů, Plán projektu

Výstupy: Testovací report

6.2.7 Analýza a tvorba automatizovaných testů

Cílem je vytvořit sadu automatizovaných testů, které budou dále spouštěny a vyhodnocovány. Jedinou osobou, která může tento úkol vykonávat, je Správce automatizace, neboť úloha vyžaduje specifické znalosti v oblasti automatizace a skriptování v některém z použitých nástrojů. Součástí této úlohy je také výběr vhodného nástroje pro automatické testování. Při výběru nástroje je nutné dbát na možné budoucí změny procesech testování. Výstupem této úlohy je Plán automatizace a soubor Automatizovaných testů. Samotná úloha přímo napomáhá k vyřešení problému *P3 – Absence automatizovaného testování*.

Vykonává: Správce automatizace

Dále může vykonávat: -

Povinné vstupy: Plán testů, Seznam testovacích nápadů, Testovací data

Nepovinné vstupy: Testovací případ

Výstupy: Plán automatizace, Automatizované testy

6.2.8 Spouštění automatizovaných testů

Cílem je spustit sadu automatizovaných a získat jejich výsledky. Samotné spuštění testů obvykle nevyžaduje zvláštní znalosti, a proto tuto činnost může kromě Správce automatizace vykonávat také Tester. Úloha přímo pomáhá k vyřešení problému *P3 – Absence automatizovaného testování.* Výsledky automatizovaných testů navíc pomáhají řešit problém *P7 – Nejednotnost chybových hlášení*, díky tomu, že výstup automatizovaných testů má vždy stejnou strukturu a v případě nalezení chyby spuštěním těchto testů je možné výstup z nástroje připojit k Reportu chyb.

Vykonává: Správce automatizace

Dále může vykonávat: Tester

Povinné vstupy: Plán automatizace, Automatizované testy

Nepovinné vstupy: -

Výstupy: Výsledky automatizovaných testů

6.2.9 Vyhodnocování automatizovaných testů

Cílem je interpretovat výsledky automatizovaných testů a promítnout je do testovacího reportu. Z důvodu náročnosti provádí úlohu pouze Správce automatizace, který získané poznatky posoudí a doplní jejich závěry do Testovacího reportu, což napomáhá vyřešení problému *P3 – Absence automatizovaného testování* a také problému *P5 – Absence testovacích reportů.*

Vykonává: Správce automatizace

Dále může vykonávat: -

Povinné vstupy: Výsledky automatizovaných testů

Nepovinné vstupy: Testovací plán, Případy užití

Výstupy: Testovací report

6.2.10 Tvorba a správa uživatelské dokumentace

Cílem je vytvářet a spravovat uživatelskou dokumentaci projektu. Úlohu provádí pouze Tester, neboť je pro tvorbu dokumentace potřeba znát popisovaný systém a vědět o všech změnách, které byly do systému zavedeny během vývoje. Tester tuto činnost provádí na konci iterace, kdy jsou všechny požadavky naplněny a především otestovány. Ve tvorbě dokumentace mu mohou pomoci existující Případy užití. Zavedením tohoto procesu dojde k vyřešení problému *P4 – Chybějící uživatelské dokumentace některých projektů*. Protože mohou existovat požadavky na tvorbu dokumentace k již existujícím částem vyvíjeného systému, může Tester ve volném pracovním čase doplňovat uživatelské dokumentace existujících projektů.

Vykonává: Tester

Dále může vykonávat: -

Povinné vstupy: Plán iterace, Požadavky, Seznam požadavků na změnu, Případy užití

Nepovinné vstupy: Testovací plán

Výstupy: Uživatelská dokumentace

6.2.11 Výběr a zavedení nástroje pro správu testů

Cílem je vybrat nejvhodnější nástroj pro správu testů. Úloha pomáhá k vyřešení problému *P2 – Nejednotná správa testovacích případů*, Výběr vhodného nástroje je klíčový a měl by být velmi dobře promyšlen. Nástroj by měl vybírat zkušený Tester za pomoci Projektového manažera, případně Správce automatizace. Zvolení nevhodného nástroje se může projevit až po delší době. Je tedy nutné nástroj dostatečnou dobu provozovat ve zkušebním režimu a do ostrého provozu jej zařadit až po důkladném posouzení ze strany několika zkušených členů týmu.

Vykonává: Tester

Dále může vykonávat: Projektový manažer, Správce automatizace

Povinné vstupy: Plán testů

Nepovinné vstupy: -

Výstupy: Popis nástroje pro správu testů

6.2.12 Údržba nástroje pro správu testů

Cílem je udržovat nástroj pro správu testů a aktualizovat jeho obsah. Předpokladem této úlohy je již existující nástroj pro správu testů, jehož popis, včetně informací o možném přístupu, je obsažen v pracovním produktu Popis nástroje pro správu testů. Činnost údržby je prováděna v průběhu celého testování, neboť je nástroj používán pro účely uchovávání testovacích případů a odkazování se ně. Samotné údržba není náročná na znalosti a zkušenosti, proto ji může vykonávat i méně zkušený Tester. Díky tomuto nástroji je možné plně vyřešit problém *P2 – Nejednotná správa testovacích případů.*

Vykonává: Tester

Dále může vykonávat: Správce automatizace

Povinné vstupy: Popis nástroje pro správu testů

Nepovinné vstupy: Plán testů, Testovací případ, Testovací sada, Testovací data, Záznam výsledků testů

Výstupy: -

6.2.13 Kontrola popisu implementace

Cílem je kontrola správného vyplnění popisu implementace a jeho případná oprava. Vývojář po dokončení změny v danému úkolu doplní stručný a výstižný Popis implementace. Tester při testování zkontroluje i tento popis a případný problém nahlásí Vývojáři. Ten buď Popis implementace doplní sám, nebo o to požádá Testera (za předpokladu, že k tomu má Tester dostatek informací).

Vykonává: Tester, Vývojář

Dále může vykonávat: -

Povinné vstupy: Seznam pracovních položek, Požadavky, Případy užití, Popis implementace

Nepovinné vstupy: Testovací případ

Výstupy: Popis implementace

6.3 Pracovní produkty

V této části kapitoly jsou popsány nově vzniklé a upravené pracovní produkty. Pracovní produkty jsou navázány na jednotlivé úlohy, u kterých je uvedeno, který identifikovaný problém řeší.

6.3.1 Report chyb

Tento dokument slouží k podrobnému popisu výsledků testování. Pro Report chyb je možné použít šablonu, která je již součástí MMSP. Pouze dojde k vyplnění pole Popis tak, jak je definováno v kapitole *5.2.8 Nejednotnost chybových hlášení*.

Odpovědná osoba: Tester

Upravuje: Tester, Správce automatizace

Slouží jako vstup pro: Zhodnocení výsledků iterace

Je výstupem z: Tvorba testovacího reportu

6.3.2 Popis řešení

Tento dokument slouží k podrobnému popisu výsledků testování. Obsah Popisu řešení je nastíněn v kapitole *5.2.2 Nedostatečný popis uživatelských příběhů.*

Odpovědná osoba: Vývojář

Upravuje: Tester, Vývojář

Slouží jako vstup pro: Plánování testů, Provedení testů, Příprava testů

Je výstupem z: Implementace řešení

6.3.3 Seznam vybraných metrik

Tento dokument slouží jako seznam vybraných metrik. Seznam může mít podobu jednoduché tabulky. U každé metriky by však měl být uveden název, autor a velmi detailní popis včetně postupu získání a vyhodnocení dat.

Odpovědná osoba: Tester

Upravuje: Tester, Správce automatizace

Slouží jako vstup pro: Sběr dat na základě vybraných metrik

Je výstupem z: Výběr vhodných testovacích metrik

6.3.4 Data a informace získané z metrik

Tento dokument pro sběr dat a informací získaných z metrik. Stejně jako seznam metrik, mohou být tyto informace zaznamenány do tabulek. Struktura tabulek nebude jednotná, protože každá metrika potřebuje různá vstupní data a poskytuje jiný výstup.

Odpovědná osoba: Tester

Upravuje: Tester, Správce automatizace

Slouží jako vstup pro: Interpretace dat z metrik

Je výstupem z: Sběr dat na základě vybraných metrik

6.3.5 Testovací report

Tento dokument slouží k podrobnému popisu výsledků testování. V podstatě se jedná o doplněný dokument Záznam výsledků testů, který je v metodice již obsažen.

Odpovědná osoba: Tester

Upravuje: Tester, Správce automatizace

Slouží jako vstup pro: Řízení změn

Je výstupem z:Interpretace dat z metrik, Tvorba testovacího reportu,

6.3.6 Plán automatizace

Tento dokument slouží detailnímu naplánování automatizovaného testování. Součástí dokumentu by měl být i popis nástroje, který bude pro účely automatizace použit. V dokumentu budou zaznamenány části systému, pro které bude použita automatizace, bude uveden důvod a odhadnuta časová náročnost tvorby testů. Součástí plánu bude také seznam jednotlivých automatických testů.

Odpovědná osoba: Správce automatizace

Upravuje: Správce automatizace

Slouží jako vstup pro: Spouštění automatizovaných testů

Je výstupem z: Analýza a tvorba automatických testů

6.3.7 Výsledky automatizovaných testů

Tento dokument slouží jako interpretace výsledků automatických testů. Každý dílčí výsledek musí být řádně provázán s odpovídajícím testem a musí u něj být uveden možný důvod a dopad. Součástí tohoto dokumentu je také shrnutí a závěr, který plyne ze získaných výsledků.

Odpovědná osoba: Správce automatizace

Upravuje: Správce automatizace

Slouží jako vstup pro: Tvorba testovacího reportu

Je výstupem z: Spouštění automatizovaných testů

6.3.8 Automatizované testy

Tento dokument představuje soubor automatizovaných testů. Testy budou mít zpravidla podobu skriptů, proto budou ukládány do samostatných datových souborů, nebo budou přímo součástí vybraného specializovaného nástroje na tvorbu a spouštění těchto testů. Struktura je silně závislá na použitém skriptovacím jazyce, a proto ji nelze v rámci metodiky pevně určit. Testy by však měly splňovat obecné předpoklady popsané v kapitole *5.2.4 Absence automatizovaného testování*.

Odpovědná osoba: Správce automatizace

Upravuje: Správce automatizace

Slouží jako vstup pro: Spouštění automatizovaných testů

Je výstupem z: Analýza a tvorba automatizovaných testů

6.3.9 Uživatelská dokumentace

Tento dokument slouží jako uživatelská příručka produktu. Podoba dokumentace je závislá na konkrétním popisovaném systému. Dokumentace by měla mít jednotný vzhled, který je v souladu se všemi dokumenty, které podnik v rámci své činnosti produkuje. Důležitou součástí je také historie revizí dokumentace.

Odpovědná osoba: Tester

Upravuje: Tester

Slouží jako vstup pro: Tvorba a správa uživatelské dokumentace

Je výstupem z: Tvorba a správa uživatelské dokumentace

6.3.10 Popis implementace

Tento dokument slouží k popisu výsledné implementace řešení. Po vytvoření se stane součástí dokumentace projektu. Tento pracovní produkt vytváří Vývojář zpravidla po dokončení

Implementace řešení. Jak by měl tento pracovní produkt vypadat, je popsáno v kapitole *5.2.2 Nedostatečný popis uživatelských příběhů*.

Odpovědná osoba: Vývojář

Upravuje: Tester

Slouží jako vstup pro: Plánování testů, Příprava testů

Je výstupem z: Kontrola popisu implementace

6.3.11 Popis nástroje pro správu testů

Tento dokument slouží k popisu nástroje pro správu testů. Součástí tohoto dokumentu jsou přístupové údaje a matice práv uživatelů zvoleného nástroje. Dokument by měl obsahovat instrukce k obsluze zvoleného nástroje.

Odpovědná osoba: Tester

Upravuje: Tester, Manažer projektu

Slouží jako vstup pro: Údržba nástroje pro správu testů

Je výstupem z: Výběr a zavedení nástroje pro správu testů

6.4 Zhodnocení změn

Navržené změny upravují metodiku MMSP za účelem odstranění, nebo alespoň zmírnění identifikovaných problémů (viz kap. *5.2 Identifikované problémy referenčního podniku*). Pro vyřešení konkrétních problémů by měla být přijata odpovídající opatření, která vyházejí z nově vzniklých úloh, s nimi souvisejících rolí a pracovních produktů. Navržená opatření nemusí být podnikem přijata všechna a v plné míře. Mohou také sloužit jen jako odrazný můstek při vlastním přizpůsobování metodiky MMSP, nebo některé její upravené verze.

Pokud by byly správně zavedeny všechny zmíněné změny, bylo by zamezeno všem identifikovaným problémům. Tím by podnik mohl dosáhnout lepšího postavení na trhu díky efektivnější spolupráci. Mezi další přínosy je možné zařadit: lepší kvalifikovanost členů vývojového týmu, kvalitnější dokumentaci projektů a přehlednější vývoj testování v projektech. Zavedení automatizace testů umožní testerům věnovat pozornost kritickým částem vyvíjeného systému. Tyto změny mohou mít také příznivý dopad na finanční situaci podniku.

Tohoto stavu nebude však možné v praxi dosáhnout z důvodu časové a finanční nákladnosti navržených změn. Některé změny navíc nemusí být v souladu s interními procesy podniku. Podnik, který má zájem o eliminaci těchto problémů, musí zvážit, která opatření je pro něj výhodné zavést, k čemuž mu může napomoci v úvodu kapitoly zmíněná *Tab. 5-1 Seznam nalezených problémů, které se týkají testování. (autor)*

7 Závěr

Hlavním cílem této bakalářské práce bylo navrhnout a následně zhodnotit rozšíření metodiky MMSP v oblasti testování pro účely referenčního malého podniku vyvíjejícího software pomocí agilních metodik.

Hlavního cíle bylo dosaženo splněním pěti dílčích cílů práce. Prvním dílčím cílem bylo vymezit roli testování v rámci agilního vývoje softwaru, čehož bylo dosaženo v teoretické části práce. Pro tento účel bylo navíc nutné blíže představit základní pojmy a praktiky z oblasti testování softwaru a uvést některé důležité informace o agilním vývoji. Dalším dílčím cílem bylo popsat současný stav testování softwaru v referenčním malém podniku. Tohoto cíle bylo dosaženo v praktické části, a to charakterizováním vybraného podniku a popisem procesů spojených s testováním softwaru. Na základě tohoto popisu byly v praktické části identifikovány problémy spojené s testováním softwaru, čímž byl naplněn třetí dílčí cíl práce. Tyto problémy byly dále blíže představeny a bylo nastíněno jejich možné vyřešení. Čtvrtým dílčím cílem bylo navrhnout rozšíření metodiky MMSP za účelem vyřešení identifikovaných problémů. Pro splnění čtvrtého cíle bylo nutné nejprve MMSP blíže popsat v teoretické části práce a poté na základě identifikovaných problému tuto metodiku rozšířit v rámci části praktické. Součástí navržených rozšíření bylo pojednání o možném přínosu těchto rozšíření a o možných problémech, které mohou zamezit jejich úplnému zavedení. Cílů, které byly na začátku práce vymezeny, se mi podařilo dosáhnout.

Za hlavní přínosy práce považuji návrh řešení zmíněných problémů a samotné rozšíření metodiky MMSP. Navržené změny mohou pomoci metodiku využít v agilních týmech, kde je kladen značný důraz na kvalitu procesu testování a na využití automatizovaného testování, za účelem usnadnění rutinní práce. Navržené změny mohou být přínosné pro potřeby výuky a mohou napomoci dalšímu rozšiřování metodiky MMSP či podobných metodik. Dalším přínosem této práce je značné rozšíření mých znalostí a zkušeností v oblasti testování softwaru, které mi pomohou, nebo již pomáhají, v naplňování role testera v mém zaměstnání.

Práci je možné převzít a doplnit o další množinu problémů, které se mohou vyskytovat jiných typech podniku. Zajímavým by bylo posouzení účinnosti změn v různých podnicích. Tato práce by mohla být dále rozšířena o výběr a vyčerpávající zhodnocení nástrojů, které byly v práci zmíněny. Práce by mohla sloužit tako odrazný můstek pro začínající testery, kteří se nově stali součástí agilního vývoje a testování jim znesnadňují některé jmenované problémy.

Terminologický slovník

Termín	Význam (zdroj)
Artefakt	Pracovní produkt. Dokument ve vývoji, sloužící k popisovanému účelu. (autor)
Code review	Proces, při kterém si programátoři navzájem kontrolují kvalitu kódu. Jedná se o jednu z metod statického testování. (Patton, 2001)
Epic	Část vlastnosti vyvíjeného systému, která je popsaná zákazníkem a je součástí Produktového backlogu. Je dále dělena na menší části (Story). (Crispin, a další, 2009)
Extrémní programování	Jedna z agilních metodik. (autor)
Inkrement	Přírůstek. Zpravidla nová funkcionalita, vznikající během iterace. (autor)
Iterace	Krátká opakující se část vývojového cyklu (obvykle 1 až 4 týdny), na konci které vzniká produkt produkční kvality, který je možné dodat zákazníkovi. (Crispin, a další, 2009)
JRE (Java Runtime Environment)	Prostředí nutné pro spouštění aplikací napsaných v programovacím jazyce Java. (autor)
Metodika	Souhrn doporučených praktik a postupů v procesu vývoje softwaru. (autor)
Produktový backlog	Seznam všech nápadů a změn, které by bylo dobré v budoucnu implementovat. Pojem vychází z metodiky Scrum. (Crispin, a další, 2009)
Retest	Proces ověření, zda byla oprava chyby úspěšná. (autor)
Scrum	Agilní metodika pro řízení projektu. (autor)
Sprint	Jedna iterace v metodice Scrum. (autor)
Story	Krátký popis požadované vlastnosti vyvíjeného systému, který přináší hodnotu zákazníkovi nebo koncovému uživateli. Může se dále dělit na menší části (Task). (Crispin, a další, 2009)
Task	Dílčí části jedné Story. Obvykle svým rozsahem nepřesahují jeden den práce. (Crispin, a další, 2009)
Uživatel	Osoba, která bude vyvíjenou aplikaci používat.
Validace	Kontrola, zda je vyvíjený produkt možné používat a má požadované vlastnosti. (autor)
Verifikace	Kontrola, zda je produkt vyvíjen správným způsobem. (autor)
Vlastník produktu (Product Owner)	Jedna z rolí v metodice Scrum, která udržuje komunikaci se zákazníkem a má na starosti určování priorit ve vývoji. (Crispin, a další, 2009)
Wiki	Skupina webových stránek, které je možné několika uživateli vytvářet, spravovat a zároveň využívat k získání informací. (autor)
Zákazník	Osoba či organizace, pro kterou je software vyvíjen. (autor)

Seznam literatury

Agile Alliance. 2013. User Stories. *Guide to Agile Practices.* [Online] Agile Alliance and Institut Agile, 2013. [Cited: Duben 2, 2015.] http://guide.agilealliance.org/guide/user-stories.html.

Ambler, Scott. 2014. Agile Modeling Best Practices. *Agile Modeling.* [Online] Ambysoft Inc., 2014. [Cited: Duben 13, 2015.] http://www.agilemodeling.com/essays/bestPractices.htm.

BusinessDictionary.com. What is software tester? definition and meaning. *BusinessDictionary.com.* [Online] WebFinance, Inc. [Cited: Březen 29, 2015.] http://www.businessdictionary.com/definition/software-tester.html.

Cockburn, Alistair, a další. 2001. Manifest Agilního vývoje software. *Manifest Agilního vývoje software.* [Online] 2001. [Citace: 22. Únor 2015.] http://www.agilemanifesto.org/iso/cs/.

Crispin, Lisa a Gregory, Janet. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams.* Boston : Addison-Wesley Professional, 2009. 9780321534460.

CzechInvest. 2014. Aplikační výklad pro vymezení pojmů drobný, malý a střední podnikatel a postupů pro zařazování podnikatelů do jednotlivých kategorií. *CzechInvest.* [Online] 1. 7 2014. [Citace: 5. Únor 2015.] http://www.czechinvest.org/data/files/definice-maleho-a-stredniho-podniku-2-1112.pdf.

Eeles, Peter. 2005. Capturing Architectural Requirements. *IBM developerWorks.* [Online] IBM, 15. Listopad 2005. [Citace: 5. Duben 2015.] http://www.ibm.com/developerworks/rational/library/4706.html.

Galin, Daniel. 2003. *Software Quality Assurance.* London : Pearson Education, 2003. 9780201709452.

Hambling, Brian. 2010. *Software testing: An ISEB foundation.* Londýn : British Computer Society, 2010. 9781906124762.

Knesl, Jiří. 2009. Agilní vývoj: Úvod. *Zdroják.* [Online] Devel.cz Lab, s.r.o, 11. Prosinec 2009. [Citace: 20. Duben 2015.] http://www.zdrojak.cz/clanky/agilni-vyvoj-uvod/. 1803-5620 .

Morgan, Peter, a další. 2010. *Software Testing: An ISTQB-ISEB Foundation Guide.* 2. 2010. 978-1906124762.

Myers, Glenford. 1979. *The Art of Software Testing.* New York : Wiley, 1979. 9780471043287.

Patton, Ron. 2001. *Software Testing.* Indianapolis : Sams Publishing, 2001. 9780672319839.

Redmine.org. 2014. Issues - Redmine. *Redmine.* [Online] 2014. [Citace: 2. Duben 2015.] http://www.redmine.org/projects/redmine/issues.

Rejnková, Petra. 2011a. *Metodika MMSP.* [Online] 2011a. [Citace: 11. Duben 2015.] http://mmsp.czweb.org/.

—. 2011b. *Lokalizace a přizpůsobení metodiky OpenUP.* Praha : Vysoká škola ekonomická v Praze, 2011b. Diplomová práce.

Scrum Alliance. 2014. Core Scrum. *Scrum Alliance.* [Online] Scrum Alliance, Inc., Srpen 15, 2014. [Cited: Duben 18, 2015.] https://www.scrumalliance.org/why-scrum/core-scrum-values-roles.

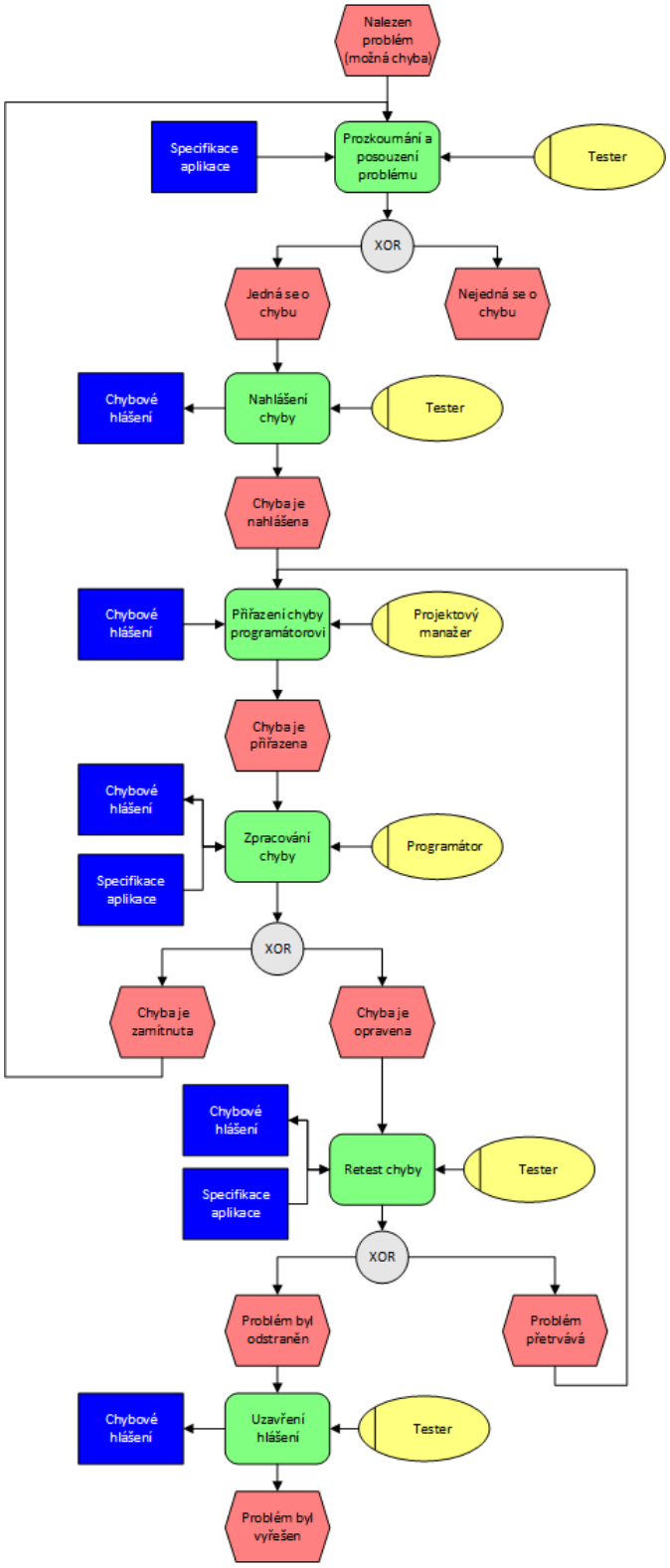
Šochová, Zuzana. 2011. Proč píšeme User Story? | Zuzi's blog. *Soch.cz.* [Online] 2. Listopad 2011. [Citace: 21. Únor 2015.] http://soch.cz/blog/management/agile/proc-piseme-user-story/.

Seznam obrázků a tabulek

Obrázky

Tabulky

Přílohy



Příloha 1 Proces nahlášení a zpracování chyby