

Metodika testovania software v agilne riadenom projekte

Andrej Nad'

Bakalářská práce
2016



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2015/2016

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Andrej Nad'**
Osobní číslo: **A15793**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**
Forma studia: **kombinovaná**

Téma práce: **Metodika testování software v agilně řízeném projektu**

Téma anglicky: **A Software Testing Methodology in Agile-driven Projects**

Zásady pro vypracování:

1. Vypracujte literární rešerši na dané téma.
2. Provedte analýzu možností přístupu k testování v agilně řízeném projektu.
3. Vhodnou komparační a evaluační metodou srovnajte jednotlivé přístupy z předchozího bodu.
4. Vypracujte podrobný návod (pracovní instrukce) pro potenciálního agilního testera v projektu na základě výběru ze seznamu v předchozím bodě, případně pomocí vhodné kombinace.
5. Uvedený návrh realizujte ve vhodné webově orientované technologii.

Rozsah bakalářské práce: -
Rozsah příloh: -
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
2. AMBLER, Scott a Mark LINES. Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. Upper Saddle River, NJ: IBM Press, 2012. ISBN 9780132810135.
3. CRISPIN, Lisa a Janet GREGORY. Agile Testing: A Practical Guide for Testers and Agile Teams. Upper Saddle River, NJ: Addison-Wesley, 2009. ISBN 0321534468.
4. LERCHE-JENSEN, Steen. Agile Tester 2015: One for all, all for one. London: CreateSpace Independent Publishing Platform, 2015. ISBN 1506131859.
5. GRAHAM, Dorothy a Mark FEWSTER. Experiences of Test Automation: Case Studies of Software Test Automation. Upper Saddle River, NJ: Addison-Wesley, 2012. ISBN 0321754069.

Vedoucí bakalářské práce: **doc. Ing. František Gazdoš, Ph.D.**
Ústav řízení procesů
Konzultant: **Ing. David Janota, Ph.D.**
Datum zadání bakalářské práce: **19. února 2016**
Termín odevzdání bakalářské práce: **27. května 2016**

Ve Zlíně dne 19. února 2016


doc. Mgr. Milan Adámek, Ph.D.
děkan




prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu


Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne


.....
podpis diplomanta

ABSTRAKT

Táto práca sa zaoberá metódami a technikami agilného vývoja softwaru, popisuje ako vyzerá agilný tím, spolupráca takéhoto tímu s biznis sférou a samotné testovanie v takomto tíme. Práca uvádza agilné zásady a ich implementáciu do vývoja. Následná analýza približuje rôzne prístupy k testovaniu a uvádza výhody a nevýhody každého z nich. Výsledkom práce je web, ktorý bude pomáhať začínajúcim testerom, ako aj testerom prechádzajúcim, na agilný vývoj so samostatným prechodom a procesmi v takýchto projektoch. Výsledkom analýzy testovania sú tiež pracovné inštrukcie pre takéhoto testera. Tieto inštrukcie zahŕňajú testovanie v rámci celého životného cyklu projektu, ako aj testovanie počas jednej iterácie.

Kľúčové slová: Agilný vývoj, software, agilné testovanie, agilný tím

ABSTRACT

This work deals with methods and techniques of agile software development, it describes how agile team looks, collaboration of such team with the business sector and the testing in such a team. Work provides agile principles and their implementation in the development, following analysis presents different approaches to testing and outlines the advantages and disadvantages of each. The result is a website that will help novice testers and testers passing on agile development with transition processes in such projects. The result of the analysis of testing is also operating instructions for such a tester. These instructions include testing within the entire life cycle of the project, as well as testing within a single iteration.

Keywords: Agile development, software, agile testing, agile team

Týmto by som chcel poďakovať školiteľovi mojej bakalárskej práce doc. Ing. Františkovi Gazdošovi, Ph.D., konzultantovi Ing. Davidovi Janotovi Ph.D. za ich odborné vedenie, metodickú pomoc a cenné rady, ktoré mi poskytli pri jej vypracovávaní. Taktiež ďakujem firme CN group s.r.o. a kolegom, ktorí mi pomohli so získavaním informácií z praxe.

Prehlasujem, že odovzdaná verzia bakalárskej práce a verzia elektronická nahraná do IS/STAG sú totožné.

OBSAH

OBSAH	7
ÚVOD	9
I. TEORETICKÁ ČASŤ	10
1 AGILNÉ RIADENIE, VÝVOJ A TESTOVANIE	11
1.1 MODEL Y VÝVOJA SOFTWARE	12
1.1.1 V-Model	12
1.1.2 Vodopád	13
1.1.3 Iteratívne inkrementálny model.....	14
1.2 ÚROVNE TESTOV	15
1.2.1 Jednotkové testy (unit testy).....	15
1.2.2 Integrované testy.....	15
1.2.3 Systémové testy	16
1.2.4 Akceptačné testy.....	16
1.3 PRODUKČNÝ TÍM	16
1.4 INTERAKCIE MEDZI ZÁKAZNÍKOM A VÝVOJÁRSKYM TÍMOM.....	17
1.5 AKO FUNGUJE TESTOVANIE V TRADIČNOM A AKO V AGILNOM PROJEKTE.....	18
1.6 POUŽITIE AGILNÝCH ZÁSAD A HODNÔT PRI TESTOVANÍ.....	18
1.6.1 Poskytovanie nepretržitej spätnej väzby.....	18
1.6.2 Prinášanie hodnoty zákazníkovi	18
1.6.3 Komunikácia face-to-face.....	19
1.6.4 Keep it simple.....	19
1.6.5 Reagovanie na zmenu	19
1.6.6 Samo-organizácia	19
1.6.7 Zameranosť na ľudí	20
1.7 DRUHY TESTOVANIA.....	20
1.7.1 Testovanie so zameraním na funkcionálnosť (funkcionálne testovanie)	20
1.7.2 Testovanie so zameraním na bezpečnosť (bezpečnostné testovanie).....	21
1.7.3 Testovanie so zameraním na použiteľnosť	21
1.7.4 Stres testovanie	21
1.7.5 Load testovanie.....	22
1.7.6 Testovanie so zameraním na podporovateľnosť.....	22
2 KRITÉRIA TESTOVANIA V AGILNE RIADENOM PROJEKTE	24
2.1 MAXIMÁLNA PODPORA UŽÍVATEĽSKÝCH ZARIADENÍ	24
2.2 RÝCHLOSŤ	25
2.3 MINIMÁLNOŠŤ CHÝB.....	25
2.4 MAXIMÁLNA SPOKOJNOSŤ ZÁKAZNÍKA	25
3 ZÁKLADNÉ TECHNIKY AGILNÉHO VÝVOJA	27
3.1 EXTRÉMNE PROGRAMOVANIE - XP	27

3.1.1	Test-Driven Development (Vývoj riadený testami) – TDD.....	27
3.1.2	Acceptance Test-Driven Development –ATDD	28
3.2	SCRUM	28
3.3	KANBAN	29
3.4	DISCIPLINED AGILE DELIVERY (DAD)	30
4	TYPY TESTOVANIA.....	31
4.1	OVEROVANIE ÚLOH Z KANBANOVEJ ALEBO SCRUMOVEJ TABULE	31
4.2	PÍSANIE TESTOV V TDD	31
4.3	PÍSANIE TESTOV V ATDD.....	32
4.4	ASISTENT PROGRAMÁTORA.....	32
4.5	METÓDA 70-100.....	33
4.6	SCHVAĽUJÚCI TESTER	33
4.7	POROVNANIE TYPOV TESTOVANIA V ZÁVISLOSTI NA KRITÉRIÁCH.....	34
II.	PRAKTICKÁ ČASŤ.....	35
5	REALIZÁCIA NÁVODU PRE ZAČÍNAJÚCEHO TESTERA.....	36
5.1	OBSAH WEBOVEJ STRÁNKY	36
5.1.1	Inception	36
5.1.2	Construction	36
5.1.3	Transition.....	37
5.2	VÝBER ŠABLÓNY	38
5.3	GRAFIKA A OBRÁZKY	38
5.3.1	Titulná strana	38
5.3.2	Inception	39
5.3.3	Construction	40
5.3.4	Transition.....	41
6	ŠABLÓNY PRE SPISOVANIE SCENÁROV A REPORTOVANIE CHÝB....	42
6.1	ŠABLÓNA VO FORMÁTE .DOC/.DOCX.....	42
6.2	ŠABLÓNA VO FORMÁTE .XLS/.XLSX	43
6.3	ŠABLÓNA VO FORMÁTE .TXT.....	44
	ZÁVER	45
	ZOZNAM POUŽITEJ LITERATÚRY	46
	ZOZNAM OBRÁZKOV	47
	ZOZNAM TABULIEK	48

ÚVOD

V poslednom desaťročí sa začína čoraz viac objavovať slovné spojenie „agilný vývoj“. Hlavou ideou agilného vývoja softwaru je priniesť zákazníkovi presne to, čo chce, za čo najkratšiu dobu. Problém, ktorý sa agilný vývoj snaží vyriešiť je ten, že zákazník počas vývoja môže zmeniť svoje požiadavky, a to nie len v detailoch, ale aj v dôležitých častiach softwaru. Klasická metodika vyvíjania softwaru by v riešení rozsiahlejších zmien zlyhala úplne a v zmenách, čo i len malých detailov, by prišlo k predraženiu a výraznému časovému posunu nasadenia produktu. Práve toto efektívne rieši agilný vývoj, kedy zmena požiadaviek, v závislosti od rozsahu zmien, zaberie len pár dní, prípadne týždňov.

Zo samotnej ideí agilného vývoja je jasné, že agilne sa nedá iba vyvíjať, ale je nutné aj software agilne testovať. Tester, ktorý doposiaľ pracoval výhradne na klasických vodopádových projektoch, môže byť zmätený a nesvoj z prechodu na modernú metodiku.

Agilný tester sa pri svojej práci začne stretávať s novými pojmami, musí správne komunikovať s celým vývojovým tímom, a taktiež už nezastáva iba vedľajšiu, často krát podradne vnímanú úlohu, ale je plnohodnotný člen agilného tímu, ktorý môže výrazne ovplyvniť pracovný postup celého tímu.

Práve tento fakt ma viedol k tomu, aby som vypracoval riešenie pre potenciálneho agilného testera, ako sa bezbolestne a čo najjednoduchšie adaptovať na agilný vývoj a testovanie. Potenciálny tester v tejto práci nájde dôležité, a v agilnom tíme každodenne používané pojmy a ich vysvetlenie, rozdielne agilné techniky spolu s popisom výhod a nevýhod v oblasti testovania, spôsoby testovania ako aj zásady agilného vývoja. Taktiež môže použiť vytvorený web stránku pre lepšiu vizualizáciu a predstavu o fungovaní agilného tímu počas trvania celého projektu, ako aj počas trvania jednej iterácie. Na začiatok si môže stiahnuť niekoľko rôznych upraviteľných šablón, ktoré bude denne používať pri práci. Dozvie sa, ako tieto šablóny správne a prehľadne vyplniť tak, aby zefektívnil prácu celého tímu. Taktiež mu webová stránka ponúkne prehľad o dostupných softwaroch, ktorých používanie môže následne vo svojom tíme navrhnúť.

I. TEORETICKÁ ČASŤ

1 AGILNÉ RIADENIE, VÝVOJ A TESTOVANIE

Jedna z mnohých charakteristík je, že hlavnú úlohu hrá samo-organizovaný, multifunkčný tím, ktorý vyvíja produkt iteratívne a inkrementálne. Produkt dodáva po pevne daných časových intervaloch a pružne reaguje na zmeny v požiadavkách od zákazníka. Agilný tím sa riadi takzvaným agilným manifestom.

Agilný manifest bol spísaný na jar v roku 2001 sedemnástimi autormi, ktorí zachytili stratégie vo forme štyroch výrokov a dvanástich podporujúcich princípov, ktoré videli dobre fungovať v praxi¹. Jednotlivé výroky sú nasledovné:

- Jednotlivci a interakcie sú prednejšie ako procesy a nástroje
- Fungujúci software je prednejší ako rozsiahla dokumentácia
- Spolupráca so zákazníkom je prednejšia ako ujednávanie kontraktov
- Reagovanie na zmenu je prednejšie ako nasledovanie plánu²

Jednotlivé podporujúce princípy sú nasledovné

1. Našou najvyššou prioritou je uspokojiť zákazníka skrz skorú a nepretržitú dodávku hodnotného softwaru.
2. Privítame meniace sa požiadavky, a to aj v neskorej fáze vývoja.
3. Dodávame fungujúci software často, v rámci týždňov až mesiacov, pričom preferujeme kratšie časové úseky.
4. Obchodníci a vývojári musia počas projektu pracovať spolu denne.
5. Stavíme projekty okolo motivovaných jedincov. Dáme im podporu a prostredie ktoré potrebujú, a budeme im dôverovať, že svoju prácu dokončia.
6. Najúčinnější a efektívny spôsob odovzdávania informácií v rámci vývojového tímu je face-to-face konverzácia.
7. Fungujúci software je hlavnou mierou postupu.
8. Agilné procesy podporujú udržateľný rozvoj. Sponzori, vývojári a užívatelia by mali byť schopní udržiavať konštantné tempo neobmedzene dlhú dobu.
9. Trvalá pozornosť k technickej dokonalosti a k dobrému dizajnu zvyšuje agilitu.

¹Ambler,Scott: Disciplined Agile Delivery, Boston: Pearson Education, 2008

²Steen Lerche-Jensen: Agile Tester 2015: One for all, all for one. CreateSpace Independent Publishing Platform, London, 2015

10. Jednoduchosť – umenie maximalizovať množstvo práce, ktoré nemusíme robiť je kľúčové.
11. Najlepšie architektúry, požiadavky a návrhy vychádzajú zo samo organizovaných tímov.
12. V pravidelných intervaloch tím uvažuje o tom, ako sa stať efektívnejším, potom podľa toho prispôsobí svoje správanie.³

1.1 Modely vývoja softwaru

Model softwarového vývoja je súhrn pravidiel, nástrojov a postupov, ktorý sa používa na riadenie, návrh a plánovanie vývoja softwaru. Tri základné a najčastejšie používané modely sú popísané nižšie, pričom za určitých podmienok je ich možné kombinovať. Pri popise jednotlivých modelov sa viac zameriam na výhody a nevýhody z pohľadu testovania.

1.1.1 V-Model

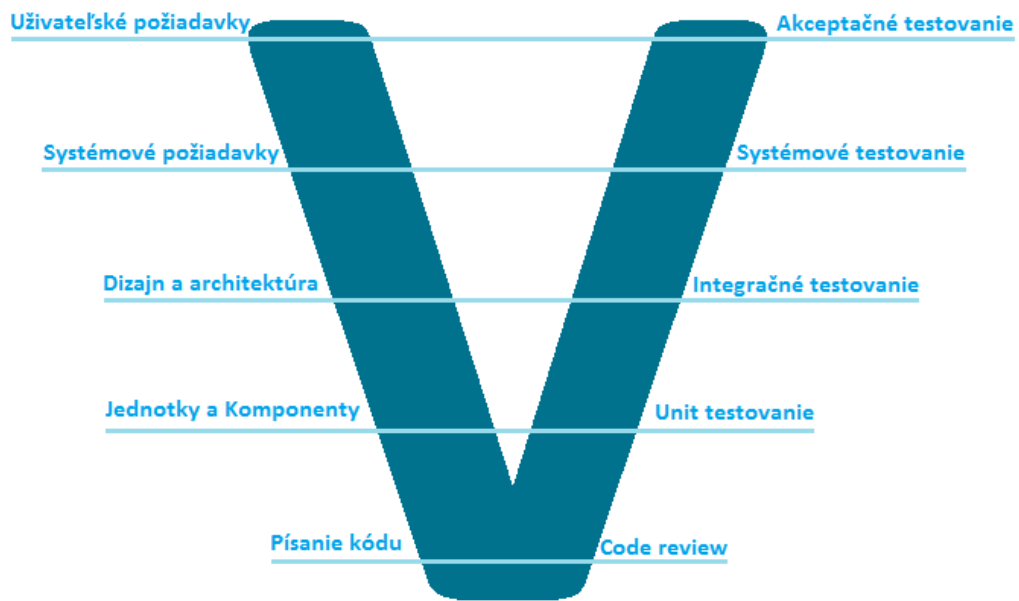
V-model je charakteristický tým, že každá vývojová aktivita má odpovedajúcu aktivitu testovaciu, čo je naznačené na prvom obrázku, a každá testovacia úroveň má svoj presne daný účel. Tento model vývoja núti tím zahrnúť testerov do projektu čo najskôr, ideálne hneď na začiatku.

Existuje mnoho variant V-modelu, avšak bežný V-model má 4 vývojové levely, ktorým odpovedajú 4 testovacie úrovne. Týmito štyrmi testovacími úrovňami sú: jednotkové, integračné, systémové a akceptačné testovanie⁴. Výhodou V-modelu je hlavne to, že paralelné testovanie s vývojom redukuje čas, každá úroveň testovania poskytuje spätnú väzbu na predošlú úroveň. Teda každé vývojové štádium má prislúchajúce verifikačné štádium.

Tieto štádiá majú rôzne ciele, testovacie techniky, napríklad techniku bielej a čiernej skrinky. Ďalej sa testujú rôzne testovacie objekty ako jednotlivé funkcie, komponenty, podsystémy a iné. Samozrejmosťou je, že každá úroveň má rôzny testovací rozsah.

³Principles behind the Agile Manifesto. Manifesto for Agile Software Development [online]. Ward Cunningham, 2001

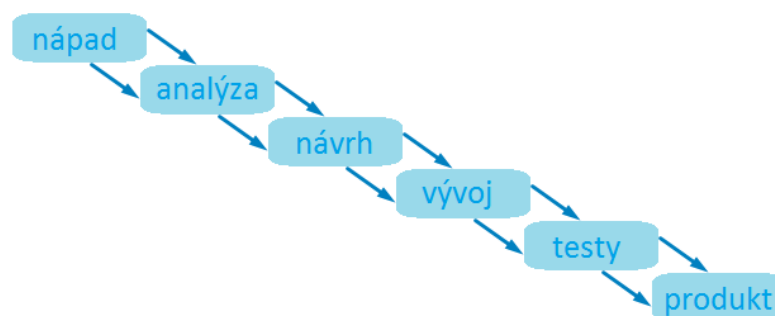
⁴Certified Tester Foundation Level Syllabus [online]. 2011, 24-27



Obrázok1: V-model

1.1.2 Vodopád

Model vodopádu je jednoduchá, elegantná metodika, ktorá v niektorých projektoch funguje veľmi dobre. Projekt vyvíjaný v modeli vodopádu postupuje akoby „dolu“ v postupnosti krokov, ktoré začínajú u prvotnej myšlienky a končia u výsledného produktu. Na konci každého z krokov vývojový tím zhodnotí, či sa môže pustiť do ďalšieho kroku. Pokiaľ projekt nie je zrelý k postupu do ďalšieho kroku, zostáva na rovnakej úrovni, pokiaľ táto úroveň nie je dokončená⁵. Ako je zobrazené na obrázku 2, tester môže začať s testovaním až potom, čo je ukončený vývoj celého produktu.



Obrázok 2: Model vodopádu

⁵Patton, Ron: Testování Softwaru. 1. Vyd. Praha: Computer Press 2002

Hlavná nevýhoda tohto modelu je, že je finančne nákladné vrátiť sa z nižšej úrovne na vyššiu. Taktiež hrozí riziko, že zákazník nedostane presne to, čo potreboval, ale to, čo vo svojich požiadavkách špecifikoval, prípadne tak ako to vývojový tím pochopil.

Jedna z výhod tohto modelu z pohľadu testera je, že sa zvyčajne nekladie dôraz na automatizované testy, nakoľko je nízka pravdepodobnosť úprav kódu vo fáze testovania a retestovania rovnakých funkcií softwaru (software sa testuje iba raz). Ďalšia výhoda z pohľadu testera je, že tester dostane do rúk dokončený produkt s kompletnou funkcionalitou a dokumentáciou.

1.1.3 Iteratívne inkrementálny model

Ako je zobrazené na obrázku 3, životný cyklus takto vyvíjaného softwaru sa pohybuje ako keby po špirále. Jedna iterácia znamená prechod viacerými aktivitami (zostavenie požiadaviek, dizajnovanie, programovanie, testovanie,..). Na začiatku každej iterácie môže zákazník pozmeniť alebo spresniť svoje požiadavky a výstupom z iterácie bude funkčná a použiteľná aplikácia alebo jej časť. To, že tím je v priamom a opakovanom kontakte s klientom, a zároveň prostredím, v ktorom bude software pracovať je veľkou výhodou pre všetkých zúčastnených. Klient má prehľad v akom štádiu je jeho produkt, vývojári môžu lepšie pochopiť ako sa software bude používať a testerí môžu napísať kvalitnejšie testovacie scenáre.



Obrázok 3: Iteratívne inkrementálny model

Na rozdiel od vodopádu, testovanie v projekte s týmto modelom zahŕňa hlavne automatizované testovanie, a to z dôvodu, že projekt sa neustále mení a neustále pribúda

nová a nová funkcionálna. Pribúda nutnosť písať regresné testy, preto sa tester v takomto projekte musí angažovať od začiatku. Zdrvivúca výhoda softwaru, ktorý je takto vyvíjaný je v čase, kedy sa produkt dostane na trh, takzvaný „time to market“. Agilne vyvíjaný software sa dokáže na trh dostať až rádovo skôr ako software vyvíjaný vodopádom. To znamená, že tá istá aplikácia by sa v prípade agilného vývoja dostala na trh za niekoľko týždňov, zatiaľ čo vo vodopáde by to trvalo mesiace.

1.2 Úrovne testov

1.2.1 Jednotkové testy (unit testy)

Už z názvu vyplýva, že sa jedná o jednoduché testy, ktoré testujú najmenšie testovateľné časti produktu. Tieto testy si píše samotní vývojári kvôli overeniu funkčnosti softwaru po zmene alebo refaktoringu. Počty unit testov sa môžu pohybovať v stovkách až tisícoch, prípadne ešte viac, v závislosti na veľkosti projektu.

Najväčšia výhoda týchto testov je, že v priebehu niekoľkých sekúnd, prípadne minút, si vie programátor overiť, či sa časť aplikácie správa tak, ako je očakávané.

Jednotkové testy odhalia preklepy v kóde, zlé výstupy funkcií, prípadne zle napísané podmienky.⁶

1.2.2 Integračné testy

Ďalší typ testov sú testy integračné. Tieto testy môžeme rozdeliť na vnútorné a vonkajšie. Výsledky vonkajších integračných testov nám zodpovedajú, či je možné software spustiť na konkrétnom operačnom systéme, súborovom systéme, prípadne overia, či software spolupracuje s hardwarom. Vnútorné integračné testy informujú o spolupráci vnútorných komponentov, rozhraní a modulov. Niektoré vonkajšie moduly nie je možné používať na testovacie účely, preto je nutné naprogramovať si „stuby“⁷, ktoré budú simulovať tieto externé moduly. Samozrejme, aj tieto stuby musia byť otestované, a musia maximálne zodpovedať realite.

⁶Certified Tester Foundation Level Syllabus [online]. 2011, 24

⁷Slovo „stub“ sa do slovenčiny prekladá ako útržok, ústrižok. Avšak v oblasti vývoja softwaru sa slovo „stub“ chápe ako miniatúrny program, prípadne aplikácia, ktorý sa používa ako náhrada volanej komponenty.

Integračné testy odhalia chyby v nekompatibilitate údajov, ako je formát dátumu a času, kódovanie požiadaviek (requestov), prípadne správnosť formátu údajov zapisovaných do databáz.⁸

1.2.3 Systémové testy

Systémové testy overujú, že software ako celok funguje správne. Pod „fungovaním správne“ rozumieme, že aplikácia plní svoju úlohu presne podľa predstáv a požiadaviek zákazníka, teda že vracia správne výstupy, a zároveň sú ošetrené neštandardné situácie. K týmto testom sa môžu písať testovacie scenáre, hlásia sa chyby, tie sa následne opravujú a opravená časť sa pretestuje.

Typické chyby, ktoré tieto testy odhalia, sú napríklad nesprávna funkcionálna, slabý výkon softwaru, prípadne nedostatočná použiteľnosť aplikácie.⁹

1.2.4 Akceptačné testy

Aby sa overilo, že aplikácia presne spĺňa zákazníkove požiadavky, je potrebné vykonať akceptačné testy. Tieto testy môže vykonávať produkčný tím, ako aj sám zákazník. Zákazníkove požiadavky by ideálne mali byť k dispozícii pred zahájením vývoja a nazývajú sa akceptačné kritériá. Týmto testami sa obvykle končí fáza vývoja, a nasleduje prevzatie produktu zákazníkom. Pod akceptačné testy napríklad patrí alfa a beta testovanie.

Akceptačné testovanie odhalí nepovšimnuté chyby, prípadne chyby, ktoré sa prejavajú iba na špecifickom prostredí a špecifických podmienkach.¹⁰

1.3 Produkčný tím

Každý, kto sa podieľa na dodaní softwaru patrí do produkčného tímu. Princípy v agilne riadených projektoch povzbudzujú členov tímu, aby sa angažovali v rôznych aktivitách. To znamená, že ktorýkoľvek člen tímu môže prijať ktorúkoľvek úlohu v tíme. Členovia agilného tímu potláčajú nutnosť špecializovaných úloh tým, že si medzi sebou rozdeľujú svoje znalosti a zručnosti. Napriek tomu, každý tím si musí rozhodnúť, aké odborné

⁸Certified Tester Foundation Level Syllabus [online]. 2011, 25

⁹Certified Tester Foundation Level Syllabus [online]. 2011, 26

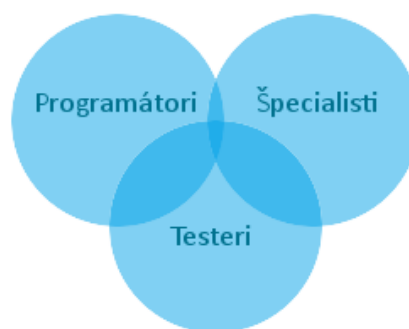
¹⁰Certified Tester Foundation Level Syllabus [online]. 2011, 26

znalosti ich projekt vyžaduje. Hoci kto z programátorov, systémových a databázových administrátorov, architektov alebo špecialistov na bezpečnosť môže byť súčasťou tímu, avšak konkrétny člen tímu môže vykonávať viac ako jednu z týchto aktivít.¹¹

Testovanie je hlavný komponent agilného vyvíjania softwaru, napriek tomu za kvalitu nezodpovedajú iba tester, ale celý produkčný tím. Tester, ďalej v tíme vývojárov pomáhajú s dodaním maximálnej biznis hodnoty (business value).

1.4 Interakcie medzi zákazníkom a vývojárskym tímom

Zákazník a vývojári celý čas úzko spolupracujú, ako je vidieť na obrázku 4, ideálne sú jeden tím so spoločným cieľom. Agilné projekty postupujú po iteráciách, ktoré trvajú zvyčajne týždeň až mesiac. Zákazník na základe výsledkov z predchádzajúcej iterácie upresňuje a prioritizuje požiadavky na ďalšie vlastnosti produktu a tím vývojárov stanoví, koľko práce je treba na zvládnutie týchto požiadaviek. Oba tímy definujú požiadavky za pomoci testov a príkladov, následne sa napíše kód tak, aby testy prešli. Tester, stoja medzi vývojármi a zákazníkmi, a musia pochopiť pohľady a požiadavky zákazníka, rovnako ako komplexnosť technickej implementácie. Niektoré agilné tímy nemajú členov, ktorí sa definujú ako tester, avšak je potrebné zaistiť niekoho, kto pomôže zákazníkovi spísať požiadavky na iterácie, uistiť sa, že testy prechádzajú a zabezpečiť automatizáciu regresných testov. Aj keď je v tíme tester, celý agilný tím je zodpovedný za tieto testovacie úlohy.¹²



Obrázok 4: Interakcie v tíme

¹¹ Crispin, Lisa: Agile testing, Boston: Pearson Education, 2008

¹² Crispin, Lisa: Agile testing, Boston: Pearson Education, 2008

1.5 Ako funguje testovanie v tradičnom a ako v agilnom projekte

V tradičnom projekte sa najskôr spíšu hrubé požiadavky na produkt (takzvaný BRUF, čo znamená Big Requirements UpFront – voľne preložené ako: „hlavné požiadavky najskôr“), z nich sa napíše špecifikácia, nasleduje samostatný vývoj produktu, a keď je vývoj úplne skončený prejde sa na testovanie spolu s opravovaním nájdených chýb. Po skončení testovacej fázy si produkt prevezme zákazník. V takto riadených projektoch býva veľmi často problém, že písanie kódu zaberie viac času ako sa pôvodne očakávalo. Z toho vyplýva, že pokiaľ je nutné dodržať dátum odovzdania produktu, testovacia fáza sa musí skrátiť. Takto vzniká riziko, že sa do produkcie dostane nedostatočne otestovaný software.

Na rozdiel od tradičnej metódy, agilne riadený projekt postupuje po inkrementálnych iteráciách, čo dáva priestor testerom, aby testovali každú časť kódu hneď potom, ako ju programátori dokončia. To znamená, že kód sa zostaví, táto malá časť kódu sa otestuje, a pokiaľ funguje správne, začne sa pracovať na ďalšej malej časti kódu. Týmto je zaručené že programátori si nevybudujú veľký náskok pred testerom, pretože úloha sa považuje za ukončenú až po otestovaní.

1.6 Použitie agilných zásad a hodnôt pri testovaní

Princíp agilného manifestu je samozrejme možné uplatniť i v testovaní. Nižšie uvediem zásady a hodnoty, ktoré sa používajú v agilných projektoch.

1.6.1 Poskytovanie nepretržitej spätnej väzby

Jeden z veľmi dôležitých prínosov testera v agilnom tíme je, že podáva spätnú väzbu od zákazníka tímu vývojárov a naopak. Častá spätná väzba výrazne zvyšuje kvalitu tým, že zákazník dostane presne to, čo požaduje. Z tohto vyplýva, že spätná väzba ušetrí zákazníkovi peniaze, nakoľko sa nezrovnalosti vyriešia skôr, a nie je potreba prerábať veľkú časť produktu, v najhoršom prípade úlohy z celej iterácie.

1.6.2 Prinášanie hodnoty zákazníkovi

Vyvíjanie softwaru agilne znamená priniesť maximálnu hodnotu zákazníkovi pomocou malých releasov, ktoré zahŕňajú presne to, čo zákazník prioritizoval. To však neznamená, že hodnota vzrastá len pridávaním nových vlastností, je potrebné sa taktiež sústrediť na celkovú funkcionálnosť produktu. Taktiež je potrebné rozhodnúť sa, čo všetko bude

v aktuálnej iterácii do produktu zakomponované. Je potrebné vyhradiť si čas na opravu chýb nájdených z minulej iterácie, vylepšenie stávajúcej funkčnosti a pridanie novej funkcionality. Správna kombinácia týchto aktivít prinesie zákazníkovi maximálnu možnú hodnotu.

1.6.3 Komunikácia face-to-face

Pokiaľ tím úplne nechápe niektorú časť projektu, napríklad novú funkcionality, tester by mal zorganizovať stretnutie so zákazníkom a programátorom. Komunikácia face-to-face platí taktiež vo vnútri tímu. Je omnoho jednoduchšie pokiaľ tím pracuje spolu v jednej miestnosti, a prípadné problémy sa môžu riešiť hneď pri objavení. Takto odpadá nutnosť pre každú jednu chybu vytvoriť bug report, opísať do neho, čo spôsobuje chybu, kedy sa chyba vyskytuje a aké má dôsledky. Toto však nie je vždy možné, preto je dobré mať osobný kontakt aspoň určitú dobu pred releasom.

1.6.4 Keep it simple

Agilné testovanie znamená, robiť čo najjednoduchšie testy, pomocou ktorých sa dá overiť, že kus daná časť softwaru funguje tak ako má, alebo že štandard kvality nastavený zákazníkom je dodržaný. Jednoduchosť však neznamená ľahkosť. Pre testera to znamená, že musí testovať „tak akurát“ pomocou najjednoduchších nástrojov a techník, ktoré sa na danú úlohu hodia.

1.6.5 Reagovanie na zmenu

Jedna z najdôležitejších zásad pri agilnom projekte. Na rozdiel od klasického vodopádu, kde sa nové požiadavky na zmeny a nové vlastnosti dodávajú až v prvom patchi, v agilnom projekte je možné tieto zmeny vykonať už v nasledujúcej iterácii. Avšak s týmto môžu mať problém testeri. Vždy, keď je nejaká funkcionality otestovaná, a zákazník príde so zmenou, je potreba danú časť software otestovať celú znova. Čiastočne túto prácu môžu zjednodušiť automatizované testy.

1.6.6 Samo-organizácia

Samo-organizácia znamená, že nie je potrebná špecializovaná pozícia na rozdeľovanie práce. Pre testera to znamená, že ťažšie automatizované testovanie môže čiastočne riešiť niekto, kto je lepší v programovaní. Samo-organizácia tiež prispieva k samostatnosti samotných členov tímu. Pri nájdení chyby v softwari je pre testera jednoduchšie, ak daný

problém rieši priamo s vývojárom, ako keď musí chyby zaznamenávať, a následne niekto musí rozhodnúť o pridelení tejto úlohy zodpovednému programátorovi.

1.6.7 Zameranosť na ľudí

Ľudia v agilnom tíme by sa mali cítiť bezpečne a nemali by sa strachovať, že ich niekto bude obviňovať z chýb. Ľudia by sa mali navzájom rešpektovať a každý by mal mať priestor na osobné rozvíjanie svojich zručností. Testeri boli väčšinou vnímaní ako menejcenní členovia tímu. Testeri, ktorí odmietajú ďalej sa vzdelávať a naberať nové zručnosti len prispievajú k názoru, že testovanie je práca, na ktorú nie je potrebná kvalifikácia. Ľudia v agilných tímoch by mali byť na rovnakej úrovni, každý môže pridať rozdielny pohľad na daný problém. Skúsený tester dokáže odhaliť chyby, ktoré neboli objavené za pomoci automatizovaných testov, tester ďalej môže klásť dôležité otázky, ktoré by ľudí bez skúseností v testovaní vôbec nenapadli.

1.7 Druhy testovania

Druhy testovania sa líšia najmä cieľom testovania a uplatnením metód. Nasledujúce druhy testovania sú v praxi najčastejšie.

1.7.1 Testovanie so zameraním na funkcionálnu funkciu (funkcionálne testovanie)

Pri tomto zameraní je nutné overiť, či software pracuje správne za normálnych podmienok. Testuje sa software ako celok a zároveň aj jeho časti. Funkcionálne testovanie nám hovorí, čo software robí a či to robí správne. V agilných projektoch sa využíva hlavne exploratory testing. Podľa knihy „Testovanie softwaru“ od Rona Pattona môžeme exploratory testing definovať nasledovne: „Testovanie softwaru, pri ktorom nemáme detaily podkladového (bežiaceho) programového kódu, sa nazýva dynamické testovanie čiernej skrinky. Je dynamické, pretože testovaný program beží – pracujeme s ním rovnako, ako keby ho mal spustený cieľový zákazník. A zároveň je to testovanie čiernej skrinky, pretože testujeme bez presných znalostí spôsobov jeho práce – máme na očiach klapky, cez ktoré dovnútra nevidíme. Zadávame vstupy, dostávame nejaké výsledky a kontrolujeme ich.“¹³

¹³Patton, Ron: Testování Softwaru. 1. Vyd. Praha: Computer Press 2002

1.7.2 Testovanie so zameraním na bezpečnosť (bezpečnostné testovanie)

Okrajovou časťou testovania funkcionality je testovanie bezpečnosti. Je potrebné odtestovať maximum vstupov, ktoré by sa dali po nasadení do produkcie zneužiť.

1.7.3 Testovanie so zameraním na použiteľnosť

Keďže software vo väčšine prípadov používajú ľudia, je nutné odtestovať, či software dokáže cieľová skupina ovládať. Inak vyzerá software zameraný na deti, a inak vyzerá používateľské rozhranie pre dispečera kamiónovej dopravy. S touto problematikou veľmi úzko súvisí testovanie dokumentácie a ostatných materiálov, s ktorými prichádza verejnosť do kontaktu. Je nevyhnutné overiť, či človek, ktorý bude produkt používať, dokáže textu pochopiť. V agilnom projekte by sa takéto testovanie malo robiť priebežne. Dôvodom je, že počas iterácie môže pribudnúť nová funkcionality, zákazník môže dodať nové šablóny a podobne.

1.7.4 Stres testovanie

Pri tomto testovaní sa software vystavuje extrémnym podmienkam. Simulácia prekročenej kapacity užívateľov, ako aj obmedzené zdroje ako je pamäť, miesto na disku a podobne ukazujú, ako si software poradí s týmito situáciami. Pri tomto druhu testovania by sa tester mal zamerať aj na to, čo sa stane, ak software zlyhá. V ideálnom prípade by nemalo dôjsť k strate dát, poškodeniu zariadenia, na ktorom software beží alebo k samotnému znehodnoteniu softwaru. Testovanie spoľahlivosti tiež zahŕňa testovanie výstupov. Je nutné overiť, či software vracia presné výstupy, či jedna sekunda naozaj trvá jednu sekundu a podobne.¹⁴

Pokiaľ sa jedná o agilný projekt, tester môže začať testovanie spoľahlivosť hneď, ako je dostupný nejaký spustiteľný kód. Testovanie s týmto zameraním je dobré vykonávať aj pri naprogramovaní novej funkcionality softwaru. Je potreba overiť, ako sa bude správať nový kus kódu, pokiaľ dôjde k zlyhaniu časti softwaru z predchádzajúcich iterácií.

¹⁴What is stress testing? *SearchSoftwareQuality* [online]. 2007. Dostupné z: <http://searchsoftwarequality.techtarget.com/definition/stress-testing>

1.7.5 Load testovanie

Výkonové testovanie prebieha za pomoci benchmarkov, ktoré porovnávajú, či už počty zvládnutých operácií alebo dobu odozvy voči inému softwaru. Počas testovania na výkon by sa tester mal zamerať na sledovanie zdrojov. Pokiaľ bude software pracovať na stroji, na ktorom spolu s ním budú spustené iné aplikácie, je nutné odtestovať, či software nevyužíva príliš mnoho dostupných zdrojov.

To znamená, že je potreba odtestovať správanie systému pri záťaži v reálnych podmienkach. Najvýhodnejšie je generovať záťaž automaticky. V agilne riadenom tíme má tester tú výhodu, že pokiaľ nie je zručný programátor, s touto automatizáciou mu pomôžu ostatní členovia tímu. S touto aktivitou úzko súvisia aj stres testy. Tieto testy majú ukázať, ako sa software bude správať pri nedostatku zdrojov ako pamäť, procesorový výkon alebo pri extrémne pomalej sieti.

V agilnom projekte tester môže testovať, či nová vlastnosť softwaru výrazne nespomalí chod ostatných častí programu. Pokiaľ v minulej iterácii zákazník nebol spokojný s celkovou rýchlosťou dodaného produktu, alebo odozvami a dobou od zadania príkazu po jeho vykonanie, tak tester by mal v priebehu aktuálnej iterácie odtestovať, či je software dostatočne rýchly a či sa naplnili zákazníkove požiadavky.

1.7.6 Testovanie so zameraním na podporovateľnosť

Tieto testy sú dôležité hlavne pri aplikáciách, ktoré budú zdieľať hardware s inými aplikáciami. Ako príklad môže poslúžiť obrázkový editor. Tester by si mal pri tomto druhu testov odpovedať na nasledujúce otázky. Podporuje náš editor štandardné formáty obrázkov? Dajú sa uložené obrázky vo formáte JPG otvoriť na inom zariadení? Bude fungovať drag-and-drop medzi ostatnými aplikáciami a našim produktom?

V tejto časti testov sa ďalej testuje to, či daná aplikácia bude fungovať aj na rôznych operačných systémoch, rôznych rozlíšeniach obrazovky alebo pri rôznej konfigurácii hardwaru.

Podstatná množina softwaru je distribuovaná do viac ako jednej krajiny. Preto je nutné, aby sa tester zamerail, či rôzne jazyky, prípadne formáty časov alebo meny zahrňujú cieľovú skupinu užívateľov.

Pri agilných projektoch môže nastať situácia, kedy zákazník zadá požiadavku na podporu iného jazyka. Keďže v prechádzajúcej verzii produktu sa s touto zmenou nepočítalo, je

dôležité odtestovať, či sa rozhrania, správy alebo emaily zobrazujú tak, ako si to želal zákazník. Ďalej bude treba overiť, či program bude akceptovať špeciálne znaky z novej jazykovej sady.

2 KRITÉRIA TESTOVANIA V AGILNE RIADENOM PROJEKTE

Každý projekt je jedinečný, to znamená, že ku každému projektu je potrebné prispôbiť spôsob vývoja a testovania. Typ projektu ako aj zameranie na konkrétnych užívateľov rozhodujú, aké kritéria budú prevažovať vo vývoji koncového produktu. Inak bude vyvíjaný a testovaný produkt pre širokú verejnosť, ako produkt pre pár operátorov, prípadne iba obyčajný program, s ktorým človek nikdy nepríde do styku. V tejto kapitole rozoberiem niekoľko hlavných kritérií testovania, ktoré sú zoradené podľa dôležitosti v tabuľke 1.

Tab1: Porovnanie dôležitosti kritérií

Kritérium	Váha, dôležitosť(1-4)
Maximálna podpora užívateľských rozhraní	1
Rýchlosť	2
Minimálnosť chýb	3
Maximálna spokojnosť zákazníka	4

Ako je možné vidieť z tabuľky vyššie, vybral som 4 kritériá, ktoré následne rozoberiem do hĺbky. Váha znamená, ako je kritérium pre testovanie dôležité, čím je číslo menšie, tým dôležitejšie je kritérium.

2.1 Maximálna podpora užívateľských zariadení

Podstatná časť vytvoreného softwaru má užívateľské rozhranie. Jedná sa najmä o aplikácie pre mobilné telefóny, programy pre počítače alebo aj webové aplikácie. Je preto nevyhnutné testovať tento software tak, aby bola zaručená funkčnosť na čo najviac zariadeniach. Tester sa preto musí vysporiadať s rôznymi desiatkami až stovkami kombinácií zariadení, operačných systémov a verziami týchto operačných systémov. Otestovať software na každej jednej kombinácii nie je v rozumnom časovom horizonte možné. Preto je potreba vybrať skupiny zariadení, ktoré sa na trhu vyskytujú najčastejšie. Napríklad, ak je potreba otestovať aplikáciu pre mobilné telefóny s androidom, tak tester by si mal zistiť, aké smartfóny sa najviac predávali v nie ďalekej minulosti (rok až

dva), a aké sa predávajú dnes, zaobstarať si tieto zariadenia s rôznymi verziami operačného systému a odtestovať software na všetkých. Nemá zmysel testovať aplikáciu zameranú na stredoeurópsky trh na smartfóne vyrábanom a predávanom iba v Indii.

2.2 Rýchlosť

Rýchlosť testovania sa dá zaručiť rôznymi spôsobmi, napríklad automatizovaním maximálneho počtu testov. Automatizácia testovania dokáže podať presný výsledok testov z minulých iterácií za minúty prípadne hodiny. Ideálne sa tieto automatické testy spúšťajú v noci, aby sa zabránilo preťaženiu serverov. Jediný problém s automatizáciou a jej dopadom na rýchlosť je, že na začiatku iterácie zaberie podstatný čas preskúmanie validity starých testov a následne písanie nových.

Testovanie, čo najrýchlejšie, bez straty kvality testovania sa dá dosiahnuť väčším počtom testerov v tíme. Následná deľba práce, kde jeden tester testuje a spisuje svoje testovanie, a druhý pomocou testovacích scenárov píše automatizované testy je efektívna. Na začiatku iterácie manuálny tester vymyslí čo sa bude ako testovať a spíše kosť svojho manuálneho testovania. Automatizačný tester zatiaľ sa stará o testy z historických iterácií a automatizuje scenáre z minulej iterácie. Od stredu až po záver iterácie manuálny tester podrobne testuje svoje scenáre, dopĺňa ich a dokumentuje ich podrobnejšie, to neznamena veľmi podrobnú dokumentáciu, ale spresnenie svojho testovania konkrétnymi krokmi, aby automatizačný tester dokázal podľa toho napísať testy.

2.3 Minimálnosť chýb

Jedna z najdôležitejších úloh testovania je minimalizácia chýb. Nadá sa zaručiť to, že sa v aplikácii nenachádza ani jedna chyba, avšak je možné zvýšiť šancu, že v softwari bude minimálny počet chýb, tým že tester budú zapojení už do ranných fáz projektu. Tester by mal na začiatku skúmať požiadavky zákazníka a pred testovaním si pripraviť scenáre, kde sa môžu vyskytnúť závažné chyby. Ďalej je potreba do testovania zapojiť všetky druhy testov: jednotkové testy, integračné testy, systémové testy, akceptačné testy a prípadne zapojiť ešte nezávislé inštitúcie alebo zákazníkových testerov.

2.4 Maximálna spokojnosť zákazníka

Ďalšie z najdôležitejších kritérií je, aby bol zákazník maximálne spokojný. Testovanie v správnej miere veľmi napomáha k spokojnosti, avšak keď sa testuje veľmi dlho,

zákazníka to stojí prostriedky a odd'ahuje sa prevzatie softwaru. S problémom narastajúceho času nasadenia veľmi efektívne pomáha testovanie agilne. Oproti klasickému prístupu sa nečaká na hotový software, ale testuje sa hneď od začiatku.

Týmto postupom sa dá vzorka niekoľkých stoviek zariadení zúžiť na niekoľko desiatok.

3 ZÁKLADNÉ TECHNIKY AGILNÉHO VÝVOJA

Technika agilného vývoja je konkrétny postup, pomocou ktorého vývojový tím vytvára produkt. Cieľom agilných techník je zefektívniť a urýchliť vývoj, odstrániť čo najviac chýb už pri samotnom vývoji, a v neposlednom rade priniesť zákazníkovi presne to, čo potrebuje.

3.1 Extrémne programovanie - XP

Extrémne programovanie je agilná technika vývoja softwaru, ktorá je zameraná na prispôsobenie sa meniacim požiadavkám od zákazníka, vývoj kvalitnejšieho softwaru produktívnejšou cestou a znižovanie nákladov použitím napríklad kratších iterácií. Táto technika zahŕňa napríklad tieto praktiky:

- Párové programovanie za jedným počítačom, kde jeden človek píše kód a druhý si zatiaľ plánuje ďalšie kroky alebo svoje návrhy.
- Spoločná kancelária na podporu komunikácie medzi tímom.
- Týždňové vývojové cykly.
- Desať minútový build. Software sa musí zostaviť, nasadiť a prejsť automatizovanými testami za menej ako desať minút.
- Inkrementálny dizajn. Dizajn produktu sa mení postupne, v závislosti na zákazníkovi.
- Testy pred programovaním (takzvaný test-first approach, napríklad TDD, ktoré je opísané nižšie). Pred začatím programovania sa napíše čo najviac testov na rôznych úrovniach od jednotkových až po akceptačné testy.

3.1.1 Test-Driven Development (Vývoj riadený testami) – TDD

Spôsob TDD spočíva v tom, že pred akýmkoľvek programovaním softwaru si programátor, prípadne tester, najskôr sám napíše automatické testy. Tieto testy sú väčšinou na úrovni jednotkových testov, ale môžu byť použité aj na integračnej a systémovej úrovni. Postup pri TDD je nasledovný: Najskôr sa vytvorí test, prípadne niekoľko testov, ktoré sa zameriavajú na malú časť kódu. Tieto testy sa následne spustia a mali by všetky zlyhať kvôli neexistencii kódu. Programátor teraz začne pracovať na riešení tak, aby všetky testy prešli. Ak kód prešiel testami, prichádza na radu refaktoring. Refaktorovanie znamená vykonávanie malých úprav kódu takým spôsobom, že tieto úpravy nijako nezmenia

chovanie sa softwaru. Jedná sa napríklad o premenovávanie premenných, zvýšenie prehľadnosti kódu alebo odstránenie duplicitností. Po skončení refaktoringu sa testy spustia znovu. Tento krok je nutný kvôli overeniu, že refaktoring prebehol úspešne a software sa správa úplne rovnako ako pred ním.

3.1.2 Acceptance Test-Driven Development –ATDD

Na rozdiel od TDD, v ATDD môžu testovacie prípady písať ako vývojári tak aj tester, ako aj ostatní členovia tímu spolu s ľuďmi z biznis sféry. Tieto testovacie prípady sa vytvárajú hlavne kvôli implementácii user stories a môžu byť manuálne alebo automatizované. Prvým krokom v ATDD je analýza, pri ktorej sa diskutujú jednotlivé user stories. V tomto prvom kroku sa odhalia chyby, nejasnosti a nekompletnosti, ktoré sa následne opravujú. Ďalší krok je vytvorenie testov, ktoré vytvára buď sám tester, alebo celý agilný tím. Tieto testy sú vlastne príklady, ktoré pomáhajú tímu implementovať riešenie správne. Každý test má svoje výstupné akceptačné kritériá¹⁵, ktoré test na konci musí splniť. Ostatné kroky sú zhodné s TDD: spustenie testov, ktoré zlyhajú, naprogramovanie riešenia, spustenie testov, ktoré prejdú, refaktoring a znovu spustenie testov.

3.2 SCRUM

Scrum nepopisuje konkrétne vývojové a testovacie techniky ako XP, miesto toho popisuje spôsob ako riadiť aktivity na projekte. Pri SCRUME sa tester stretne s týmito pojmami:

- Iterácia, ktorá má pevne danú dĺžku trvania
- Prírastok produktu, software, ktorý je potenciálne doručiteľný zákazníkovi
- Produktový backlog, v ktorom sa nachádza zoznam prvkov produktu
- Šprintový backlog, v ktorom sa nachádzajú úlohy z produktového backlogu s najvyššou prioritou, ktoré sú vybrané do nasledujúceho šprintu. Tieto úlohy si z produktového backlogu vyberá tím (pull princíp).
- Definícia hotového (Definition of Done), ktorá nám udáva kritériá, podľa ktorých vieme určiť, či je šprint splnený
- Timeboxing, čo znamená, že každá úloha by mala byť splnená do určitého času

¹⁵Akceptačné kritériá udávajú hranice user story, a používajú sa na potvrdenie toho, že práca na user story môže byť ukončená. Akceptačné kritériá pokrývajú hlavne použiteľnosť, funkčnosť, výkon a spracovanie chýb a mali by byť špecifické, merateľné a hlavne testovateľné.

- Transparentnosť, pomocou ktorej vieme v akom stave je produkt alebo konkrétne úlohy. Tím rozoberá stav jednotlivých úloh na každodennom stretnutí, nazývanom denný scrum, aby každý zúčastnený v projekte mal prehľad o aktuálnom dianí.

Jednotlivé role v scrume sú nasledovné:

- Scrum master – dozerá na dodržiavanie a implementovanie pravidiel a praktík, prípadne rieši nezrovnalosti, nedostupnosť zdrojov a iné udalosti, ktoré môžu zapríčiniť, že tím nebude schopný dodržiavať tieto pravidlá a postupy.
- Majiteľ produktu – reprezentuje zákazníka, stará sa o produktový backlog
- Vývojový tím – samo-organizovaný, multifunkčný tím, ktorý nemá svojho oficiálneho tímového vedúceho, ktorý robí rozhodnutia. Tieto rozhodnutia robí tím ako celok.

3.3 Kanban

Základným princípom kanbanu je vizualizácia a optimalizácia toku práce v práve prebiehajúcim čase. Tento postup má niekoľko spoločných vecí so scrumom ako napríklad vizualizácia aktívnych úloh kvôli transparentnosti, čakajúce úlohy v backlogu, prípadne timeboxing. Kanban má výhodu oproti ostatným prístupom v tom, že dovoľuje dodávanie produktu po jednotlivých častiach radšej, ako na konci šprintu, nezávisle na čase, ale splnených úlohách.

V kanbane sa využívajú tri nástroje:

- Kanbanová tabuľa, na ktorej sa nachádzajú jednotlivé úlohy, je rozdelená na niekoľko stĺpcov, v ktorých sa nachádzajú kartičky s úlohami podľa toho, v akom sú stave.
- Limit aktuálne spracovávaných úloh. Množstvo úloh, ktoré sú naraz spracovávané je striktné limitované. Toto je možné kontrolovať pomocou maximálneho počtu kartičiek v konkrétnom stĺpci. Vždy keď sa miesto v tomto stĺpci uvoľní, môže sa do neho presunúť nová kartička s úlohou.
- Čas dodania (lead time). Kanban je navrhnutý tak, aby optimalizoval nepretržitý tok úloh minimalizovaním času dodania

3.4 Disciplined Agile Delivery (DAD)

Definícia DAD je nasledovná: „Procesový framework Disciplined agile delivery (DAD) je hybridný agilný postup na dodávanie IT riešení, kde sú prvoradí ľudia, a ktorý je zameraný na učenie.“¹⁶. O DAD môžeme povedať, že je to hybridný rámec, na ktorého základe môžeme vybudovať systém firemných procesov. DAD je rozdelený do troch fáz, ktoré majú tieto ciele: inception, construction, transition, ktoré bližšie popisujem v nasledujúcej tabuľke.

Tab2: Ciele DAD

Inception	Construction	Transition
<ul style="list-style-type: none"> - Sformovať počiatočný tím - Ujasniť si víziu projektu <ul style="list-style-type: none"> - Odsúhlasiť víziu so zúčastnenými - Prispôbiť firemným potrebám - Identifikovať počiatočnú technickú stratégiu a počiatočné požiadavky <ul style="list-style-type: none"> - Pripraviť pracovné prostredie - Zaistiť financovanie - Identifikovať riziká 	<ul style="list-style-type: none"> - Vytvoriť použiteľné riešenie - Reagovať na zmeny zúčastnených - Približovať sa k nasaditeľnej verzii - Udržiavať alebo zlepšovať úroveň kvality - Stabilizovať architektúru 	<ul style="list-style-type: none"> - Zaistiť, že riešenie je pripravené - Zaistiť, že zákazník je pripravený prijať riešenie - Nasadiť riešenie do produkcie
Priebežné ciele		
<ul style="list-style-type: none"> - Naplniť poslanie projektu - Zlepšiť schopnosti tímu - Rozšíriť existujúcu infraštruktúru 	<ul style="list-style-type: none"> - Zlepšiť pracovné prostredie - Objavovať riziká - Zlepšiť existujúcu infraštruktúru 	

¹⁶Ambler,Scott: Disciplined Agile Delivery, Boston: Pearson Education, 2008

4 TYPY TESTOVANIA

Ako je spomenuté v druhej kapitole, tak ako je každý projekt rozdielny, tak isto je rozdielny aj vývojový tím. Každý tím je inak produktívny pri rozličnom spôsobe práce, preto táto kapitola naznačí, ako sa dá efektívne testovať pri vývoji softwaru v rôznych tímoch, s rôznymi postupmi a nástrojmi.

4.1 Overovanie úloh z kanbanovej alebo scrumovej tabule

Základný predpoklad pre testovanie v takomto projekte je tabuľa s kartičkami (scrumboard alebo kanbanboard), môže byť obyčajná alebo elektronická a je rozdelená do stĺpcov. Prvý stĺpec je „backlog“ a nachádzajú sa v ňom požiadavky od zákazníka. Samo-organizovaný tím z tohto stĺpca vyberie požiadavky a presunie ich do stĺpca „implementácia“ a priradí meno programátora, ktorý túto požiadavku implementuje do projektu. Ak programátor začne na nejakej úlohe pracovať, presunie si ju do stĺpca „pracujem na:“ a po ukončení práce na tejto úlohe ju presunie do stĺpca „testovanie“. Tu prichádza na rad tester, ktorý danú úlohu odtestuje podľa testovacieho scenára, ktorý si pripravil dopredu, prípadne píše scenár priamo pri testovaní. Po skončení testovania presunie kartu buď do „overené“, ak prešla jeho testami, alebo „vrátené“, ak našiel chyby, prípadne zistil iné nezrovnalosti. Pred koncom iterácie si tester s projektovým manažérom prejdú jednotlivé karty zo stĺpca „overené“ a presunú ich do posledného stĺpca produkcia.

Z praxe môžem tvrdiť, že niektoré testy, ako napríklad automatické priradenie platby konkrétnemu zákazníkovi je nutné zautomatizovať, aby sme vedeli, či nová funkcionálna nenarušila túto vlastnosť. Presne kvôli tomu je vytvorený stĺpec „požiadavka na regresný test“, kam po dotestovaní presúvam takéto kartičky s úlohami. Iné testy, ako napríklad požiadavka na zmenu obrázku v emaile nie je potrebné automatizovať, nakoľko s týmito emailami prichádzam denne niekoľkokrát do styku, ich automatizácia by zbytočne zabrala čas na vytvorenie a udržiavanie testu, a tiež čas pri spustení regresných testov.

4.2 Písanie testov v TDD

Pred TDD slúžilo testovanie softwaru iba na overenie, či daný kód odpovedá požiadavkám zákazníka, takzvaný prístup „test-last“. Pri TDD je testovanie v úplne inej pozícii, a to niečoho, podľa čoho sa produkt bude vyvíjať, takzvaný prístup „test-first“. Testy v takomto projekte nemajú za úlohu preukázať, prípadne overiť, že programátor spravil svoju prácu dobre, ale slúžia pre návrh a implementáciu riešenia problému do produktu.

Tester v takto riadenom projekte by mal disponovať pokročilejšími znalosťami z programovania. Pred tým, ako sa k úlohe vôbec dostane programátor, musí tester pripraviť, v závislosti na rozsahu úlohy, desiatky až stovky jednotkových testov. Tieto testy potom odovzdá programátorovi, ktorý musí napísať kód tak, aby všetky testy prešli. Programátor následne kód zrefaktoruje, a spustí testy znovu. Úloha sa považuje za dokončenú až po prejdení všetkých testov po refaktoringu.

Nakoľko TDD je zamerané výhradne na testy v kóde, nie je ním možné pokryť všetky druhy testov. Taktiež nie je v silách testera, prípadne programátora, úplne pokryť testami vyvíjanú vlastnosť.

4.3 Písanie testov v ATDD

Podobne ako TDD, aj ATDD stojí na základoch „test pred kódom“, teda jedná sa o „test-first“ metódu. Na začiatku si celý tím spolu sadne a na základe analýzy sa vytvoria testovacie scenáre. Analýza by mala u všetkých členov tímu odstrániť nejasnosti a zaručiť, že každý člen tímu bude vedieť, o čo v konkrétnej úlohe pôjde. Tester pri analýze nemá špeciálne postavenie, nakoľko spolu s ostatnými premýšľa nad otázkami typu: „Ako sa bude daný prípad testovať?“, „Na čo si máme dať pozor?“, a iné. Nasleduje samostatné vytvorenie akceptačných testov, ktoré validuje zástupca z biznis sféry. Zvyšok prebieha rovnako ako pri TDD. To znamená: spusti kód, ktorý spadne, oprav kód, spusti kód, ktorý prejde, refaktoruj, spusti kód, ktorý prejde.

4.4 Asistent programátora

Tester v takejto pozícii čaká, kedy programátor dokončí rozrobenú úlohu a jej riešenie hneď odtestuje. Po odtestovaní ju buď schváli, alebo vráti k prepracovaniu a opravené riešenie odtestuje znovu. Po prípadnom schválení riešenia zautomatizuje svoje testy. Takáto pozícia testera v tíme je vhodná najmä pre začínajúcich junior testerov, ktorí majú aspoň malé skúsenosti s programovaním.

Výhodou takéhoto testera v tíme je, že programátor si nemusí po sebe dôkladne testovať svoju prácu, a môže sa venovať programovaniu ďalších požiadaviek. Ďalšia výhoda je, že tester je pevnou súčasťou tímu, takže odpadá nutnosť písania dokumentácie ku každej vlastnosti produktu.

Nevýhodou takéhoto testera je, že musí poznať produkt po technickej aj užívateľskej stránke. Musí sa starať o testovacie dáta a poznať presné kroky, ako sa zo vstupu dopracovať k výstupu. Takýto tester zvládne manuálne testovať kód po troch až štyroch programátoroch. Pokiaľ svoje testovacie prípady musí automatizovať, počet testerov v tíme je nutné zvýšiť.

4.5 Metóda 70-100

Pri tejto metóde je dobré používať nástroj na vizualizáciu práce. Jadro spočíva v tom, že každá úloha po dokončení programovania je hotová iba na 70%. Až tester po jej otestovaní môže tvrdiť, že úloha je skončená na 100%. Takto má tím prehľad, ktorá úloha je v akom stave, a koľko chýba k jej úplnému dokončeniu.

4.6 Schvaľujúci tester

Pred koncom každej iterácie sa do tímu pripojí tester, ktorý si sadne spolu s programátorom. Programátor predvedie svoju prácu testerovi, ktorý sa aktívne zapája a navrhuje svoje postupy ako danú časť kódu otestovať. Po dotestovaní, tester buď dané riešenie schváli alebo ho nechá prepracovať. Po skončení všetkých úloh jedného programátora v danej iterácii sa tester presunie k ďalšiemu. Opravené chyby si znova obaja za jedným počítačom spolu prejdú a po schválení tím odovzdá produkt zákazníkovi.

Nevýhodou tohto postupu je, že tester nie je pevnou súčasťou tímu a nebýva s tímom v kontakte počas celej iterácie. Kvôli nesúdržnosti tímu je potreba písať viac dokumentácie, podľa ktorej si tester vytvorí svoje testovacie prípady.

Výhodami môže byť napríklad to, že tester môže takto testovať viac projektov naraz, nemusí riešiť programovanie automatických testov, ktoré si naprogramujú samotní vývojári po tomto testovaní.

4.7 Porovnanie typov testovania v závislosti na kritériách

Z nižšie uvedenej tabuľky (Tab. 3), kde 1 značí najlepší výsledok, môžeme vypožorovať, že testovanie pomocou kanbanovej tabule zaručí spokojnosť zákazníka, avšak nie je rýchle, nakoľko tester bude pravdepodobne testovať na čo najviac zariadeniach. Písanie testov v TDD nezohľadňuje užívateľské zariadenie ale kód, zaručí tak v ňom minimálnosť chýb, avšak nie je veľmi rýchle, nakoľko je potreba písať stovky jednoduchých testov ešte pred napísaním kódu. Testovanie pomocou ATDD je na tom veľmi podobne, jediný rozdiel je v tom, že sa môžu zahrnúť rozdielne užívateľské rozhrania. Toto, ako aj nutnosť zídania sa celého tímu s ľuďmi z biznis sféry a prebranie všetkých akceptačných kritérií spôsobí, že vývoj bude trvať dlhšie. Asistent programátora je veľmi rýchla metóda, nakoľko funkcionality je odtestovaná hneď po naprogramovaní, to isté platí aj o metóde 70-100. Nakoniec je tu schvaľujúci tester. Tester s programátorom väčšinou testujú iba na jednom zariadení, rýchlosť nie je ideálna, nakoľko programátor miesto programovania testuje, to je však vyvážené minimom chýb, nakoľko jednu vec testujú dvaja ľudia.

Tab3: Porovnanie typov testovania v závislosti na kritériách

	Maximálna spokojnosť zákazníka	Minimálnosť chýb	Rýchlosť	Maximálna podpora užívateľských rozhraní
Kanbanová tabuľa	2	3	4	1
Písanie testov v TDD	1	2	3	4
Písanie testov v ATDD	1	2	4	3
Asistent Programátora	3	2	1	4
Metóda 70-100	3	2	1	4
Schvaľujúci tester	1	2	3	4

Toto porovnanie je však orientačné a dosť všeobecné, nakoľko každá úloha je niečím špecifická. Pri niektorých typoch úloh, ako responzivnosť stránky je nutné sa zameriavať na úplne iné veci ako pri výmene dát vnútri aplikácie.

Nakoľko ani jedna z metód nie je sama o sebe ideálna, je potreba ich kombinovať a nájsť správne vyváženie medzi rýchlosťou, minimálnosťou chýb, uspokojením zákazníka, a aj užívateľa tým, že aj na jeho neštandardnom zariadení bude aplikácia fungovať správne.

II. PRAKTICKÁ ČASŤ

5 REALIZÁCIA NÁVODU PRE ZAČÍNAJÚCEHO TESTERA

Svoj návrh návodu pre začínajúceho testera som sa rozhodol realizovať pomocou značkovacieho jazyka HTML5¹⁷, kaskádových štýlov CSS¹⁸, a Javascriptu. Túto zostavu jazykov som si zvolil preto, že je kompatibilná so všetkými bežnými zariadeniami a operačnými systémami. Taktiež je pomerne jednoduché s kombináciou týchto jazykov tvoriť efektne vyzerajúce, responzívne¹⁹ a moderné webové stránky.

5.1 Obsah webovej stránky

Webová stránka je rozdelená na úvodnú stránku a tri podstránky: Inception, Construction, Transition.

5.1.1 Inception

Cieľom fázy Inception je dosiahnuť súbeh medzi všetkými zúčastnenými stranami o cieľoch a životnom cykle projektu.

Tab4: Popis práce testera vo fáze Inception

Čo robí tester	Výstup práce testera
Výber vhodného software na reportovanie bugov	Adaptácia, akceptovanie a používanie softwaru tímom
Zostavenie šablón pre testovacie scenáre	Šablóna, ktorá bude prehľadná a ľahko použiteľná ako pre testera tak aj pre vývojára
Hľadanie nevytvorených chýb	Zoznam vecí, ktoré nie sú úplne domyslené, alebo sú sporné, prípadne zbytočné

5.1.2 Construction

V tejto fáze sa aktívne vyvíja software, s najväčšou pravdepodobnosťou tu tester bude pracovať v iteráciách, ktoré trvajú niekoľko týždňov.

¹⁷HTML5. W3C [online]. 2014.

¹⁸W3schools: CSS Tutorial [online]. 2016.

¹⁹Responzívnu webovou stránkou rozumieme dokument, ktorý je optimalizovaný pre rôzne zariadenia a ich zobrazovacie možnosti. Jednotlivé prvky dokumentu vyzerajú inak na počítači, tablete alebo telefóne v závislosti od rozlíšenia obrazovky. Responzívny dizajn zaručuje, že dokument bude prehľadný, čitateľný alebo inak použiteľný na rôznych zariadeniach.

Tab5: Popis práce testera vo fáze Construction

Čo robí tester	Výstup práce testera
Vybranie úloh do iterácie	Zoznam úloh, ktoré bude po naprogramovaní možné testovať
Programovanie automatických testov	Zrevidované testy Opravené testy Nové automatické testy z poslednej iterácie
Písanie testovacích scenárov pred začatím testovania	Hrubý popis testovacieho scenáru, môže byť aj vlastnými slovami, prípadne naznačený zoznam krokov
Pretestovanie opravených chýb a zvyšných úloh	Všetky úlohy z iterácie budú naprogramované a odtestované, rovnako budú opravené nájdené chyby, alebo budú zaradené do nasledujúcej iterácie
Testovanie s vývojármi	Schválenie nasadenia aktuálne testovanej verzie

5.1.3 Transition

Vo fáze Transition sa odovzdáva produkt zákazníkovi. Pre testera bude nutné zaistiť vonkajšiu krásu produktu, ako aj všetkej jej dokumentácie, manuálov, rozhraní, a iných súčastí softwaru.

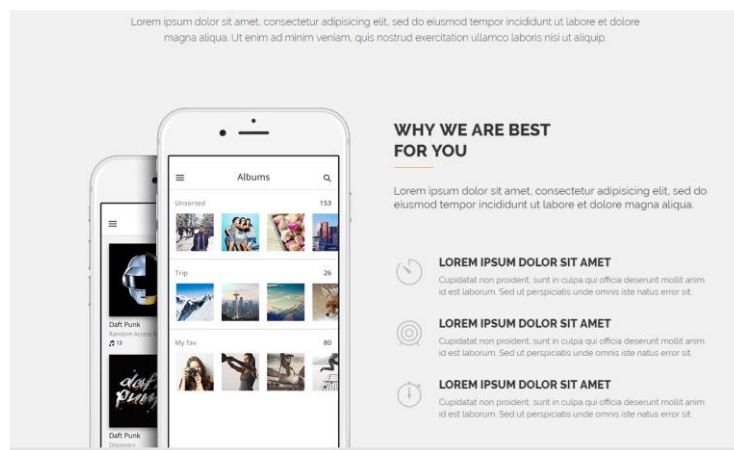
Tab6: Popis práce testera vo fáze Transmition

Čo robí tester	Výstup práce testera
Testovanie vzhľadu, gramatické a iné chyby	Celistvé užívateľské prostredie
Testovanie užívateľského manuálu, priloženej dokumentácie	Manuál, ktorý je presný a správny

5.2 Výber šablóny

Nakoľko je na internete voľne a zadarmo dostupných nespočetné množstvo šablón, rozhodol som sa použiť a následne upraviť vhodnú šablónu. Po prezretí niekoľkých šablón, som vybral túto: <https://dl.dropboxusercontent.com/u/74113979/html/landing/bent.zip>. Šablóna po prvom spustení obsahuje základné obrázky a pseudotext „Lorem ipsum“.

Samotná webová demo stránka obsahovala rôzne typy podstránok, ktoré neboli vhodné pre implementáciu môjho návrhu. Preto som ich zmazal ako aj z HTML dokumentu, tak aj z CSS súboru. Naopak podstránka „INCEPTION“ mala veľmi vhodné rozloženie prvkov (viď obrázok 6), preto som rovnaký dizajn použil aj na podstránku „TRANSMITION“. Vymenil som však rozloženie textu a obrázku.



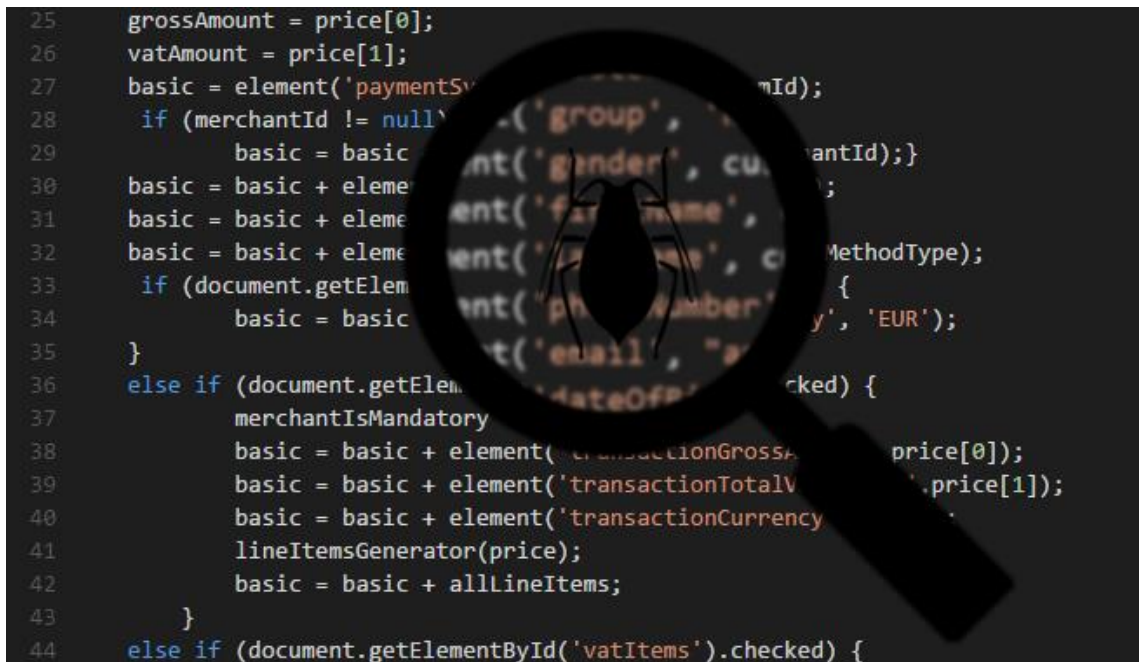
Obrázok 5: Screenshot zo šablóny Bent

5.3 Grafika a obrázky

Všetku grafiku som si vytvoril sám. K tvoreniu som používal voľne a zadarmo dostupné nástroje Gimp a Inkscape. Časti obrázkov, ktoré som sám nevytvoril sú taktiež voľne dostupné pre širokú verejnosť.

5.3.1 Titulná strana

Obrázok na titulnej strane je tvorený javascriptovým kódom, ktorý mi denne slúži na generovanie testovacích dát. Kód je akože pod lupou, pod ktorú je navyše vidieť „bug“. Toto znázorňuje, že chyby sa môžu vyskytovať aj v jednoduchých aplikáciách. Našťastie, v mojom generátore som chybu zatiaľ neobjavil.



Obrázok 6: Titulný obrázok

5.3.2 Inception

Obrázok na tejto podstránke zobrazuje časť screenshotu s mnou používaného nástroja na vizualizáciu, reportovanie a manažment úloh na projekte. Nad ním je časť šablóny, ktorú bežne používam na reportovanie chýb, alebo písanie testovacích prípadov a scenárov.

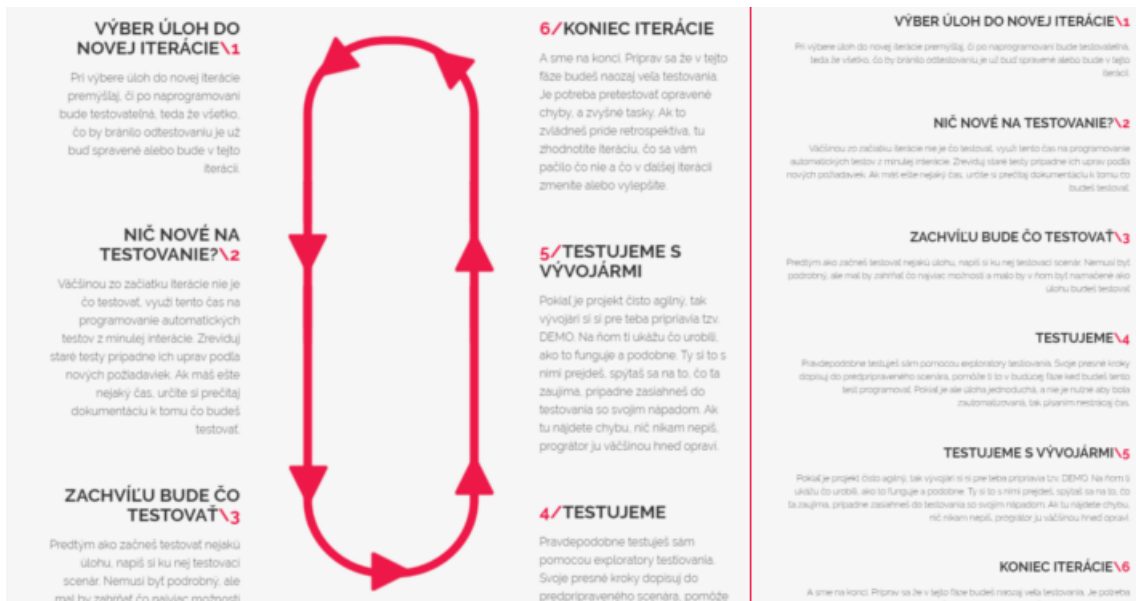


Obrázok 7: Inception

5.3.3 Construction

Na tejto podstránke sa nachádza obrázok, ktorý má znázorňovať iteráciu. Celý je vytvorený vo vektorovej grafike a následne prevedený na klasický maticový obrázok. Šípky znázorňujú, akým smerom postupuje iterácia, teda aj jednotlivé aktivity testera.

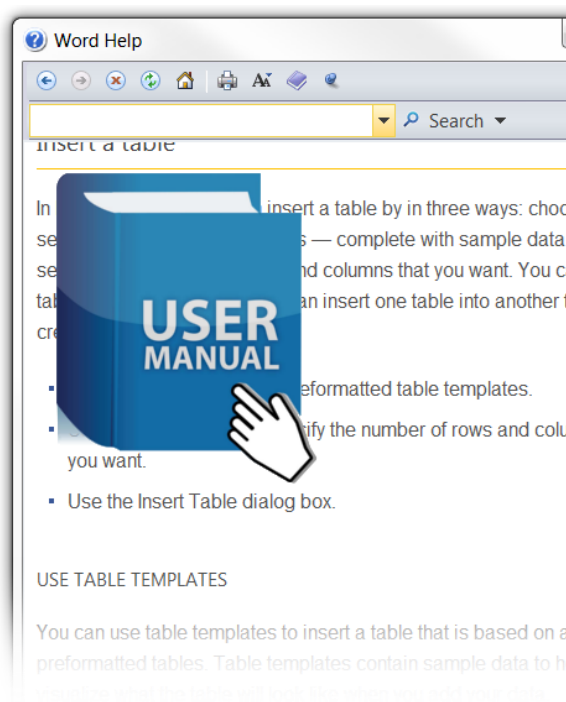
Tento obrázok sa však nezobrazí pri rozlíšení pod 992px. Je to z toho dôvodu, že by sa zobrazil medzi jednotlivé texty, tým pádom by celá táto podstránka bola rozhádzaná a nedávala by zmysel. V pôvodnej šablóne to však riešené nebolo, preto som musel upraviť CSS súbor tak, aby sa pri rozlíšení 992px a menšom obrázok nezobrazil. Zároveň by boli rozhádzané aj jednotlivé kroky, ktoré tester v rámci fáze Construction absolvuje. Preto bolo nutné vytvoriť vlastný štýl tak, aby pri nižšom rozlíšení zmizol obrázok a zároveň sa kroky zoradili pod seba. Screenshot nižšie zobrazuje na ľavej strane štandardné rozloženie stránky, a na pravej je rozloženie stránky v zariadeniach s menšou uhlopriečkou ako 993px.



Obrázok 8: Construction

5.3.4 Transition

V tejto fáze tester prichádza do kontaktu s nefunkčionálnymi časťami projektu, ako je dokumentácia. Preto aj obrázok zobrazuje elektronický užívateľský manuál ako aj grafiku tlačeneého manuálu, ktorý sa väčšinou dodáva k špecializovanému softwaru.



Obrázok 9: Transition

6 ŠABLÓNY PRE SPISOVANIE SCENÁROV A REPORTOVANIE CHÝB

Pre testera, ako aj pre celý tím je výhodné, pokiaľ sa pre písanie testovacích scenárov a reportovanie chýb používa tá istá šablóna alebo dokument. Zainteresovaní ľudia tak vedia, kde hľadať jednotlivé údaje, ktoré sú potrebné buď na naprogramovanie automatického testu, na pretestovanie opravenej chyby, alebo na znovu vyvolanie chyby. Takéto šablóny je možné používať v desiatkach rôznych formátov, preto som vybral tri najčastejšie formáty, a príklad šablóny vypracoval. Samozrejme tieto šablóny je možné stiahnuť si z webového rozhrania v sekcii Inception, a následne si ich prispôsobiť potrebám svojho testovania.

6.1 Šablóna vo formáte .doc/.docx

Takúto šablónu dokáže otvoriť väčšina pokročilých nástrojov na spracovanie textov ako napríklad Microsoft Word alebo Libre Office. Hneď v prvom riadku sa vyplní názov projektu, ideálne je vyplniť ho raz spolu s niektorými ostatnými poliami, šablónu uložiť a ďalej s týmto krokom nestrácať čas. Pre lepšiu orientáciu sa v ľavej hornej šablóne vyplní identifikačné číslo úlohy alebo chyby, priorita úlohy, jej názov a krátky opis. V pravej časti sú údaje o tom, kto testovací scenár navrhol, kedy, a zároveň kedy a kým bol vykonaný.

Pod touto hlavičkou nasleduje časť, v ktorej sa vyplnia takzvané preconditions, teda podmienky pred spustením testu. Nasleduje hlavná časť s jednotlivými krokmi, a pod nimi sú postconditions, teda podmienky, v akom stave sa má systém po testovaní nachádzať. Na konci šablóny je ešte pole, do ktorého sa dajú vpísať prídavné informácie. Taktiež je tam možné vložiť okrem textu screenshoty alebo grafy. Nižšie je obrázok toho, ako má vyzerat' správne vyplnená, prehľadná šablóna spolu s nahlásenou chybou.

Project name: MůjPrvýProjekt

Test Case ID:	0001	Designed by:	Andrej Nad'
Test Priority:	HIGH	Designed date:	15.2.2016
Test Title:	Prihlasovanie do portálu	Executed by:	Andrej Nad'
Description:	Pokus o prihlásenie sa	Executed date:	19.3.2016

Preconditions:	<ol style="list-style-type: none"> 1. Užívateľ je odhlásený 2. Používateľské konto je aktivované
-----------------------	--

Step:	Test steps:	Test data:	Expected result:	Actual result	Status
1	Vyplniť meno	Username: Demo	Okno bude zelené	Okno je zelené	OK
2	Vyplniť heslo	Password: Demo	Heslo bude: ****	Heslo je: Demo	FAIL
3	Kliknúť na Login	-	Používateľ bude prihlásený	Používateľ je prihlásený	OK

Postconditions:	<ol style="list-style-type: none"> 1. Používateľ bude prihlásený 2. V súbore main.log nebude žiadny nečakaný error 3. Používateľ uvidí hlášku „TERAZ SI PRIHLÁSENÝ“
------------------------	--

Additional information:	Prihlasovanie trvalo až +- 10 sekúnd, čo je veľmi dlhá doba!
--------------------------------	--

Obrázok 10: Vyplnená šablóna vo formáte .docx

6.2 Šablóna vo formáte .xls/.xlsx

Táto prípona súborov je typická pre tabuľkový procesor Microsoft Excel. Tento formát som zvolil preto, lebo je dostatočne prehľadný, jednoduchý a prostredie MS Excel disponuje funkciami, ktoré word nepodporuje. Šablóna je rozdelená na jednotlivé listy, takže v jednom testovacom prípade je možné mať niekoľko testovacích scenárov. Šablónu je samozrejme možné upraviť do tvaru, ktorý najviac vyhovuje pre daný testovací prípad, preto vyzerá zámerne inak, ako šablóna vo formáte .docx. Príklad takejto šablóny je zobrazený na obrázku 12.

	A	B	C	D	E
1	Test Case:		Prihlasovanie do portálu		
2	Id:		1		
3	Pre-condition:		Odhlásený užívateľ, aktivované		
4					
5	Step	Name	Test steps	Expected result	Request
6	1.1				
7	1.2				
8	1.3				
9	1.4				
10	1.5				
11	1.6				
12	1.7				
13					

Obrázok 11: Prázdna šablóna vo formáte .xlsx

6.3 Šablóna vo formáte .txt

Tento formát je najuniverzálnejšou a najjednoduchšou voľbou. Na jeho editáciu a zobrazenie postačia integrované nástroje v bežných operačných systémoch, vrátane príkazového riadku, čo je pri predošlých šablónach nemožné. Samozrejme má aj nevýhody. Tou hlavnou je, že sa priamo v šablóne nemôžu ukladať obrázky, grafy, prepojenia alebo iný obsah okrem holého textu. Nižšie je na obrázku uvedený ten istý testovací scenár vyplnený do obvyčajnej textovej šablóny.

```
 9 .....10 .....20 .....30 .....40 .....50 .....60 .....70 .....80
1 Task ID: | 0001
2 Task Title: | Prihlasovanie do portalu
3 Task Created: | 2016-02-15
4 Status: | FAIL
5 -----
6 PRECONDITIONS:
7 • Uzivatel je odhlaseny
8 • Uzivatelske konto je aktivovane
9
10 -----
11 SCENARIO:
12 • Vyplnit meno -> uzivatel vidi zelene okno OK
13 • Vyplnit heslo -> Heslo je vo formante **** FAIL
14 • Kliknut na login -> Uzivatel je prihlaseny OK
15
16 -----
17 TEST DATA:
18 • Credentials:
19   o Username: Demo
20   o Password: Demo
21 • URL: 192.168.45.21/login.php
22
23 -----
24 TEST RESULTS:
25 • Uzivatel je sice prihaseny, ale heslo sa pri zadavani zobrazilo ako text,
26   nie ako ****
27 -----Executed: 2016-03-19-----
```

Obrázok 12: Šablóna vo formáte .txt

ZÁVER

Táto práca sa zaoberá testovaním v agilnom prostredí, jeho prostredím a popisom malej aplikácie, ktorá takýto typ testovania môže uľahčiť.

Prvá kapitola pojednáva o najzaužívanejších modeloch vývoja softwaru, o ich výhodách, nevýhodách a použití. Popisuje úrovne testov, charakterizuje produkčný tím a interakcie medzi ním a zákazníkom a analyzuje rozdiely v testovaní v agilnom a tradičnom projekte. Ďalej je tu možné nájsť, aké sú agilné zásady, prečo sú dôležité a vyžadované. Na konci prvej kapitoly je možné zistiť, aké spôsoby testov sú pri agilných projektoch najpoužívanejšie.

Druhá až štvrtá kapitola analyzuje možnosti prístupu k testovaniu. Dajú sa tu nájsť výhody a nevýhody testovania podľa každého kritéria, techniky vývoja a testovania a typy testovania. Na konci druhej kapitoly je prehľadná tabuľka, v ktorej potenciálny agilný tester nájde, ktoré kritérium je najviac a najmenej dôležité. Tieto kritériá nemusia byť len pre celý projekt, ale aj pre konkrétnu úlohu alebo dokonca podúlohu. Na konci štvrtej kapitoly potom nájde, ktoré kritérium má aký vplyv na typ testovania.

Podrobný návod na testovanie softwaru a priblíženie aktivít v rámci jednej iterácie, ale aj celého projektu je v piatej kapitole tejto práce. Je tu možné nájsť priebeh testovania od začatia prvých prác na projekte, až po testovanie po skončení vývoja.

Uvedený návrh na postupy a metódy testovania v agilne riadenom projekte som vypracoval ako responzívnu webovú stránku. Tester si tak môže prezrieť tento web na svojom mobilnom telefóne, tablete alebo počítači bez straty ovládateľnosti alebo prehľadnosti. Nájde tu sťahovateľné a upraviteľné šablóny, ktoré sa v priebehu práce naučil správne vyplňať.

Hlavným cieľom bakalárskej práce je pomôcť ako aj úplnému začiatníkovi, tak aj skúsenému testerovi, ktorý prechádza z klasického do agilného riadeného projektu. Takýto tester síce pozná úrovne testov a druhy testov, avšak nemusí poznať agilné zásady a techniky používané v agilných projektoch. Vypracovaná analýza môže pomôcť v orientácii v spôsobe testovania v agilných projektoch, navrhnutá aplikácie vrátane priložených šablón zase zrýchli získanie rutiny pri práci v projekte a tým zvýšenie produktivity testera hlavne v počiatočných štádiách životného cyklu vývoja softwaru.

ZOZNAM POUŽITEJ LITERATURY

- [1] Ambler, Scott. Lines, Mark: Disciplined Agile Delivery, Boston: Pearson Education, 2012
- [2] Steen Lerche-Jensen: Agile Tester 2015: One for all, all for one. CreateSpace Independent Publishing Platform, London, 2015
- [3] Principles behind the Agile Manifesto. *Manifesto for Agile Software Development* [online]. Ward Cunningham, 2001 [cit. 2016-05-25]. Dostupné z: <http://www.agilemanifesto.org/principles.html>
- [4] Certified Tester Foundation Level Syllabus [online]. 2011, , 24-27. Dostupné z: <http://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html4>
- [5] What is stress testing? *SearchSoftwareQuality* [online]. 2007. Dostupné z: <http://searchsoftwarequality.techtarget.com/definition/stress-testing>
- [6] Patton, Ron: Testování Softwaru. 1. Vyd. Praha: Computer Press 2002
- [7] Crispin, Lisa. Gregory, Janet: Agile testing, Boston: Pearson Education, 2008
- [8] HTML5. *W3C* [online]. 2014. Dostupné z: <https://www.w3.org/TR/html5/>
- [9] *W3schools: CSS Tutorial* [online]. 2016. Dostupné také z: <http://www.w3schools.com/css/>

ZOZNAM OBRÁZKOV

Obrázok 1: V-model	13
Obrázok 2: Model vodopádu	13
Obrázok 3: Iteratívne inkrementálny model	14
Obrázok 4: Interakcie v tíme	17
Obrázok 5: Screenshot zo šablóny Bent	38
Obrázok 6: Titulný obrázok.....	39
Obrázok 7: Inception	40
Obrázok 8: Construction.....	41
Obrázok 9: Transition	41
Obrázok 10: Vyplnená šablóna vo formáte .docx.....	43
Obrázok 11: Prázdna šablóna vo formáte .xlsx	43
Obrázok 12: Šablóna vo formáte .txt.....	44

ZOZNAM TABULIEK

Tab1: Porovnanie dôležitosti kritérii	24
Tab2: Ciele DAD	30
Tab3: Porovnanie typov testovania v závislosti na kritériách	34
Tab4: Popis práce testera vo fáze Inception	36
Tab5: Popis práce testera vo fáze Construction.....	37
Tab6: Popis práce testera vo fáze Transmition.....	37