Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Diplomová práce

Manuální a automatické testování softwaru

Bc. Martin Chotětický

© 2019 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Martin Chotětický

Informatika

Název práce

Manuální a automatické testování softwaru

Název anglicky

Manual and automated software testing

Cíle práce

Diplomová práce je tematicky zaměřena na problematiku testování softwarového produktu během vývoje. Hlavním cílem je testování základních funkcionalit systému prostřednictvím manuálních a automatických testů a jejich porovnání.

1) Seznámení se s problematikou testování.
2) Popsání životního cyklu testování softwaru.
3) Popsání specifik automatických a manuálních testů.
4) Analýza vhodnosti použití automatických versus manuálních testů pro určité případy.
5) Zhodnocení, doporučení a navržení dalšího postupu.

Metodika

Metodika řešené problematiky diplomové práce je založena na studiu a analýze odborných informačních zdrojů. V teoretické části jsou interpretovány získané informace na základě analýzy a syntézy dané problematiky, které jsou důležitým prvkem pro zpracování praktické části.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

testování, automatizace, software, analýza, defekt

Doporučené zdroje informací

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6

HAMBLING, Brian a Peter MORGAN. Software testing: an ISTQB-ISEB foundation guide. 2010. ISBN 978-1-906124-76-2

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 978-807-2266-364

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-802-5138-168

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Petra Pavlíčková, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 22. 11. 2018

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 29. 11. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 07. 03. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Manuální a automatické testování softwaru" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29.3.2019 ____________________________

Poděkování

Rád bych touto cestou poděkoval paní Ing. Petře Pavlíčkové, Ph.D., z katedry systémového inženýrství ČZU v Praze, za vedení práce a poskytnuté konzultace, dále participantům a v neposlední řadě své rodině za veškerou podporu při vytváření práce.

Manuální a automatické testování softwaru

Abstrakt

Práce je zaměřena na problematiku testování softwaru ve fázi vývoje za použití manuálního a automatizovaného způsobu testování a zanalyzování jejich časových a finančních náročností.

Teoretická část charakterizuje životní cyklus vývoje softwaru včetně používaných modelů a metodik. Část práce pojednává o kvalitě softwaru a životního cyklu testování softwaru. V této souvislosti práce rozebírá role při testování a způsob, jakým lze k testování přistupovat. Charakterizovány jsou dále typy testů, rozděleny dle úrovně a účelu testování, zásadní rozdíly mezi manuálním a automatizovaným testováním a testovací případ. V závěrečné části teoretické práce jsou shrnuty poznatky ze správy defektů a testovacího prostředí.

Praktická část práce řeší časovou a finanční náročnost manuálního a automatizovaného testování prostřednictvím testování dvou základních scénářů konkrétní aplikace. Na základě získání informací z teoretické části je vytvořena strategie testování a prostřednictvím dokumentace vytvořeny samotné specifikace případů užití, které slouží pro tvorbu konkrétních testovacích scénářů a skriptů.

Doba tvorby a exekuce manuálních a automatizovaných testů je autorem práce měřena a spolu s výsledky čtyř participantů provedena časová analýza vhodnosti použití manuálního a automatizovaného testování. Na základě získaných mezd odpovídajících pracovních pozic je provedena finanční analýza vhodnosti použití daného typu testování.

V závěru práce jsou zhodnoceny výsledky výzkumu s doporučenými možnostmi použití manuálních a automatizovaných testů.

Klíčová slova: testování, automatizace, software, analýza, defekt

Manual and automated software testing

Abstract

The purpose of thesis is to analyse software development, its testing in early stages using manual and automated processes and to evaluate it time consumption and financial requirements.

The thesis consists of two parts – theoretical and practical. The first mentioned part pays attention to particular stages of software development including concrete models and methodological processes which are used within practical part of this work. Regarding on that, the work describes different approaches how to test scenarios and necessary roles to be selected. Additionally, concrete test categories are mentioned depending on testing level and purpose, significant differences between manual and automated testing and a test case.

As written before, the aim of this work is to define time consumption and financial requirements regarding on the testing type used - manual or automated testing, which is the main purpose of practical part. For its demand 2 concrete testing scenarios will be used for a particular application. On the base of information described in theoretical part a unique testing strategy will be applied using specific documentation of particular test scenarios including its specifications and scripts.

Time lenght of work required to execute mentioned tasks is calculated by the author and afterwards evaluated together with results of four participants. That will be used in order to set a time consumption analysis depending on the testing procedure used – manual or automated. Financial anylisis is laboured on the base of necessary workforce required to complete certain tasks.

Results and recommendations of particular test scenarios and processing types are evaluated in the final chapter of this thesis.

Keywords: testing, automatization, software, analysis, defect

Obsah

1 Úvod **13**

2 Cíl práce a metodika **14**

2.1 Cíl práce 14

2.2 Metodika 14

3 Teoretická východiska **15**

3.1 Životní cyklus vývoje softwaru 15

3.2 Modely životního cyklu vývoje softwaru 17

3.2.1 Model velkého třesku 17

3.2.2 Model programuj a opravuj 18

3.2.3 Vodopádový model 18

3.2.4 Spirálový model 19

3.2.5 V model 20

3.2.6 W model 21

3.2.7 Inkrementální model 22

3.2.8 RAD model 22

3.3 Metodiky vývoje 22

3.3.1 Rigorózní metodiky 23

3.3.2 Agilní metodiky 24

3.4 Kvalita softwaru 29

3.4.1 FURPS 30

3.4.2 Zajišťování kvality 31

3.4.3 Řízení kvality 31

3.4.4 Verifikace a validace 32

3.5 Životní cyklus testování softwaru 32

3.5.1 Analýza požadavků 33

3.5.2 Plán testování 33

3.5.3 Vývoj testovacích případů 34

3.5.4 Nastavení prostředí 34

3.5.5 Exekuce testů 34

3.5.6 Ukončení testovacího cyklu 34

3.6 Role při testování 35

3.6.1 Tester 35

3.6.2 Test analytik 35

3.6.3 Test leader 36

3.6.4 Test manažer ... 36
3.7 Způsob testování ... 37
3.7.1 Černá skříňka ... 37
3.7.2 Bílá skříňka ... 38
3.7.3 Šedá skříňka ... 38
3.7.4 Ad hoc a exploratorní testování ... 39
3.8 Typy testů ... 39
3.8.1 Úroveň testování ... 39
3.8.2 Účel testování ... 42
3.9 Manuální a automatické testování ... 44
3.9.1 Selenium ... 45
3.10 Testovací případ ... 46
3.11 Správa defektů ... 46
3.11.1 Příčiny vzniku defektů ... 47
3.11.2 Atributy defektu ... 48
3.11.3 Priorita defektu ... 48
3.11.4 Životní cyklus defektu ... 49
3.11.5 Komunikační mapa ... 51
3.12 Testovací prostředí ... 52
3.12.1 Typy testovacích prostředí ... 53
4 Vlastní práce ... 54
4.1 Popis systému ... 54
4.2 Funkcionality systému ... 55
4.3 Plán testování ... 55
4.3.1 Přehled plánovaných testů ... 55
4.3.2 Cíl testování ... 56
4.3.3 Vstupní předpoklady ... 56
4.3.4 Výstupní předpoklady ... 56
4.3.5 Role a odpovědnosti ... 57
4.3.6 Strategie testování ... 57
4.4 Případy užití ... 59
4.4.1 Use case diagram ... 59
4.4.2 Diagram aktivit ... 61
4.4.3 Validační pravidla ... 62
4.4.4 Chybové zprávy ... 62
4.5 Specifikace případů užití ... 63
4.5.1 Analýza případu užití pro registraci ... 64

4.5.2 Analýza případu užití pro přihlášení ... 65
4.6 Testovací data ... 66
4.6.1 Způsoby vytvoření testovacích dat ... 66
4.6.2 Tvorba testovacích dat ... 68
4.7 Manuální testování ... 70
4.7.1 Tvorba testovacího případu registrace ... 71
4.7.2 Tvorba testovacího případu přihlášení ... 72
4.7.3 Exekuce testovacího případu registrace ... 73
4.7.4 Exekuce testovacího případu přihlášení ... 76
4.8 Automatizované testování ... 78
4.8.1 Tvorba testovacího skriptu registrace ... 79
4.8.2 Tvorba testovacího skriptu přihlášení ... 83
4.8.3 Exekuce testovacího skriptu registrace ... 86
4.8.4 Exekuce testovacího skriptu přihlášení ... 86
4.9 Výsledek časové náročnosti ... 87

5 Výsledky a diskuse ... 89
5.1 Časová analýza náročnosti testu ... 89
5.1.1 Registrace ... 89
5.1.2 Přihlášení ... 91
5.2 Finanční analýza náročnosti testu ... 93
5.2.1 Registrace ... 95
5.2.2 Přihlášení ... 98
5.3 Doporučení ... 101
5.3.1 Doporučení na základě typu testu ... 102

6 Závěr ... 103

7 Seznam použitých zdrojů ... 104

Seznam obrázků

Obrázek 1: Spirálový model [wikimedia.org] ... 19
Obrázek 2: V model [19] ... 20
Obrázek 3: W model [umel.feec.vutbr.cz] ... 21
Obrázek 4: Znázornění testování černé skříňky oproti testování bílé skříňky [3] ... 38

Obrázek 5: Příklad stavů životního cyklu defektu [19] 50
Obrázek 6: Příklad komunikační mapy pro správu defektů [19] 51
Obrázek 7: Logo projektu Animal Breath [Vlastní zpracování autora] 54
Obrázek 8: Životní cyklus defektu pro testovanou aplikaci [vlastní zpracování autora] 58
Obrázek 9: Use case diagram [vlastní zpracování autora] 60
Obrázek 10: Diagram aktivit [vlastní zpracování autora] 61
Obrázek 11: Tvorba testovacího případu registrace v nástroji TestRail [vlastní zpracování autora] 71
Obrázek 12: Tvorba testovacího případu přihlášení v nástroji TestRail [vlastní zpracování autora] 72
Obrázek 13: Registrační formulář [vlastní zpracování autora] 73
Obrázek 14: Vyplněný registrační formulář [vlastní zpracování autora] 74
Obrázek 15: Dokončení procesu registrace [vlastní zpracování autora] 75
Obrázek 16: Vyhodnocení testu registrace v nástroji TestRail [vlastní zpracování autora] 75
Obrázek 17: Vyplněný přihlašovací formulář [vlastní zpracování autora] 76
Obrázek 18: Horní panel po přihlášení do systému [vlastní zpracování autora] 76
Obrázek 19: Vyhodnocení testu přihlášení v nástroji TestRail [vlastní zpracování autora] 77
Obrázek 20: Výsledek exekuce testu pro registraci [vlastní zpracování autora] 86
Obrázek 21: Výsledek exekuce testu pro přihlášení [vlastní zpracování autora] 86

Seznam tabulek

Tabulka 1: Typy testovacích prostředí [19] 53
Tabulka 2: Role a odpovědnosti v projektu [vlastní zpracování autora] 57
Tabulka 3: Priorita defektů [vlastní zpracování autora] 59
Tabulka 4: Validační pravidla v registračním formuláři [vlastní zpracování] 62
Tabulka 5: Chybové zprávy [vlastní zpracování autora] 63
Tabulka 6: Testovací data pro proces registrace [vlastní zpracování autora] 69
Tabulka 7: Testovací data pro proces přihlášení [vlastní zpracování autora] 69
Tabulka 8: Časová náročnost testu pro registrace [vlastní zpracování autora] 87
Tabulka 9: Časová náročnost testu pro přihlášení [vlastní zpracování autora] 87

Tabulka 10: Časové výsledky všech participantů pro manuální testování [vlastní zpracování autora]..........88
Tabulka 11: Časové výsledky všech participantů pro automatické testování [vlastní zpracování autora]..........88
Tabulka 12: Nalezení efektivity z časového hlediska pro případ registrace [vlastní zpracování autora]..........90
Tabulka 13: Nalezení efektivity z časového hlediska pro případ přihlášení [vlastní zpracování autora]..........92
Tabulka 14: Mzdy pracovních pozic z portálu jobs.cz [vlastní zpracování autora]..........94
Tabulka 15: Přepočet měsíční mzdy na vteřiny [vlastní zpracování autora]..........94
Tabulka 16: Rozšíření o finanční složku testu pro registraci [vlastní zpracování autora]...95
Tabulka 17: Nalezení efektivity z finančního hlediska pro případ registrace [vlastní zpracování autora]..........96
Tabulka 18: Rozšíření o finanční složku testu pro přihlášení [vlastní zpracování autora]..98
Tabulka 19: Nalezení efektivity z finančního hlediska pro případ přihlášení [vlastní zpracování autora]..........99
Tabulka 20: Souhrn výsledků z analýz náročnosti testů [vlastní zpracování autora]........101

Seznam grafů

Graf 1: Časová náročnost při rostoucím počtu exekuce testu pro registraci [vlastní zpracování autora]..........91
Graf 2: Časová náročnost při rostoucím počtu exekuce testu pro přihlášení [vlastní zpracování autora]..........93
Graf 3: Finanční náročnost při rostoucím počtu exekuce testu pro registraci [vlastní zpracování autora]..........97
Graf 4: Finanční náročnost při rostoucím počtu exekuce testu pro přihlášení [vlastní zpracování autora]..........100

1 Úvod

Interakce člověka se softwarovým produktem je v dnešním světě téměř nevyhnutelná. Lidé jsou vedeni k užívání softwaru například bankami, zdravotnictvím, dopravními podniky, či využívají software pro své vlastní potřeby.

Není tajemstvím, že bezporuchovost a bezchybnost softwaru je v současném dění, které nám dnešní technologie nabízejí, brána jako samozřejmost. Procesy stojící za naplněním těchto požadavků jsou nedílnou součástí vývoje softwaru. Právě testování, které bývá často označováno jako nevděčná práce, je jedno z příčin, proč software funguje tak, jak se od něj očekává.

Diplomová práce je zaměřena na problematiku testování softwaru ve fázi vývoje, s cílem zanalyzování náročnosti manuálních a automatizovaných testů. Obsahem teoretické části je charakteristika používaných metodik vývoje, životního cyklu vývoje a testování softwaru. V práci jsou uvedeny způsoby, kterými lze k testování přistupovat, role v testovacím týmu a typy testů členěných dle úrovně a účelu testování. Definované jsou dále specifika manuálního a automatizovaného způsobu testování, správa defektů a typy prostředí, na kterých lze aplikaci testovat.

Obsahem praktické části je popis testované aplikace a souvisejících funkcionalit. Následně je vytvořen stručný plán testování a dokumentace v podobě případů užití obsahující diagramy, validační pravidla a chybové zprávy. Na základě dokumentace jsou vytvořeny specifikace případů užití, sloužící pro vytvoření manuálních a automatizovaných testů. Za pomoci kombinace doby tvorby testu a jeho exekuce je provedena časová a finanční analýza náročnosti testu. V závěru jsou zhodnoceny výsledky a doporučení vhodnosti použití manuálního a automatizovaného způsobu testování.

Motivací pro výběr diplomové práce je dlouhodobé zaměření autora na problematiku testování. Manuální testování bylo pro proces testování softwaru základním kamenem po řadu let, kdy se automatizace testů zdála nemyslitelná. Teprve rostoucí možnosti a trendy informačních technologií současné doby vedou k nárůstu zájmu o zvýšení automatizace v testování, která by přispěla k vyšší efektivitě samotného testování. I přes to je automatizované testování stále velkým otazníkem pro celou řadu projektů.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je tematicky zaměřena na problematiku testování softwarového produktu během vývoje. Dílčím cílem je popsání životního cyklu testování softwaru a charakterizování specifik automatických a manuálních testů. Hlavním cílem je testování základních funkcionalit systému prostřednictvím manuálních a automatických testů. Pro testování budou vytvořeny potřebné testovací případy ke konkrétní aplikaci. Po dokončení procesu testování bude provedena analýza vhodnosti použití manuálních a automatizovaných testů, následně vyhodnoceny výsledky a doporučení autora k použití již zmíněných testovacích způsobů.

2.2 Metodika

Metodika řešené problematiky diplomové práce je založena na studii a analýze odborných informačních zdrojů. V teoretické části jsou interpretovány získané informace na základě analýzy a syntézy dané problematiky, které jsou důležitým prvkem pro zpracování praktické části.

3 Teoretická východiska

3.1 Životní cyklus vývoje softwaru

Životní cyklus vývoje softwaru představuje sled úkolů, které definují procesy v každém kroku vývoje softwaru. Jelikož může být informační systém chápán z různých hledisek a různí autoři definují životní cyklus vývoje softwaru rozdílně, nejsou fáze životního cyklu striktně definovány. Proces životního cyklu nejčastěji obsahuje 6 hlavních po sobě následujících kroků: [1]

1. Plánování
2. Analýza
3. Návrh
4. Implementace a vývoj
5. Testování
6. Údržba

Plánování

Fáze plánování patří mezi nejdůležitější krok při vytváření úspěšného systému. Během této fáze dochází k rozhodnutí, co má být uděláno a stanovení problémů, které mají být vyřešeny za pomoci: [1]

- definování problémů, cílů a zdrojů, jako jsou například personál a náklady
- osvojení schopnosti, jak navrhnout alternativní řešení po konzultaci s klienty, dodavateli a zaměstnanci
- prostudování, jak udělat produkt lépe než konkurence

Po analýze vyplývající z fáze plánování budou k dispozici 3 možnosti: [1]

1. vytvořit nový systém
2. vylepšit současný systém
3. nechat systém takový, jaký je

Analýza

Jedná se o fázi, ve které dochází ke shromažďování a analýze veškerých požadavků z byznysu. Mělo by být jasně stanoveno a zdokumentováno, co koncový uživatel od systému očekává, a jak bude systém fungovat. Shromážděné požadavky jsou dále analyzovány a zkoumány, zda mohou být začleněny do vyvíjeného systému. Výsledkem této fáze je dokument specifikace požadavků, sloužící pro účely pokynů v další fázi projektu. [1]

Návrh

Fáze návrhu přichází po správném porozumění požadavků od zákazníka, jelikož v systému definuje veškeré elementy, komponenty, úroveň zabezpečení, moduly, architekturu a různá rozhraní a typy dat, která procházejí systémem.
Obecný návrh systému může být proveden prostřednictvím psací potřeby a papíru k určení, jak bude systém vypadat a fungovat. Poté bude vytvořen podrobný a rozšířený návrh systému a bude splňovat veškeré funkční a technické požadavky z předešlé fáze. [1]

Implementace a vývoj

Jedná se o fázi, která přichází až po úplném porozumění systémových požadavků a specifikací. Je to nejdelší fáze celého životního cyklu vývoje systému, ve které je systém vyvíjen vývojářským týmem. [1]

Testování

Hlavním cílem fáze testování je ověření, zda systém splňuje předem definované požadavky z předchozích fází. O testování systému se stará specializovaný testovací tým, který provádí všechny typy funkčních a nefunkčních testů, avšak může být prováděno i reálnými uživateli. Testování je stále důležitější součástí procesu vývoje softwaru pro zajištění spokojenosti zákazníka a vyvarování se zbytečných komplikací. Po úspěšném dokončení fáze testování je možné nasadit systém do provozu. [1]

Údržba

V této fázi je prováděna periodická údržba systému, aby se zajistilo, že systém nebude zastaralý. Představuje jakousi prevenci před různými problémy, které se mohou vyskytnout až v samotném průběhu, kdy je systém nasazen a zpřístupněn veřejnosti. Opravy se provádějí v podobě aktualizací. [1]

3.2 Modely životního cyklu vývoje softwaru

Modely životního cyklu softwaru představují a popisují vzájemné vztahy mezi jednotlivými fázemi životního cyklu softwaru. Každý model obsahuje svou vlastní metodiku, jak zaopatřit dostatečnou kvalitu systému. Modelů životního cyklu softwaru je plná řada, ačkoliv většina z nich jsou pouhé variace a kombinace nejzákladnějších a nejpoužívanějších modelů, jimiž jsou: [2]

- Model velkého třesku
- Model programuj a opravuj
- Vodopádový model
- Spirálový model

Není možné však říct, který z modelů je nejlepší, jelikož každý má své výhody a nevýhody. Dalšími modely, které budou více rozepsány, jsou modely vycházející z výše zmíněných čtyř hlavních modelů: [3]

- V model
- W model
- Inkrementální model
- RAD model

3.2.1 Model velkého třesku

Autor Ron Patton přirovnal princip modelu velkého třesku k teorii o vzniku vesmíru, odkud byl název modelu převzat. Model je založen na jednoduchosti s minimálním

zaměřením na plánování, analýzu a testování. Veškeré úsilí se týká vývoje softwaru a psaní kódu, což je ideální proces, jestliže požadavky na systém nelze přesně specifikovat a čas pro dokončení vývoje je flexibilní. [3]

3.2.2 Model programuj a opravuj

Model „programuj a opravuj" nepředstavuje příliš efektivní model, avšak oproti modelu velkého třesku je o krok vpřed, jelikož požaduje alespoň nějakou ideu o tom, jaké mají být požadavky na systém. Tým využívající tento model obvykle začíná s hrubou představou, jak bude systém fungovat, vytvoří jednoduchý návrh a poté prochází dlouhým opakujícím se procesem kódování, testování a opravování bugů. Pak v určitém okamžiku dojde k rozhodnutí, že se vývoj ukončí a dojde k vydání dokončeného systému. Vzhledem k tomu, že plánování a dokumentace je velmi málo, projektový tým může ukázat výsledky v podstatě ihned. Z tohoto důvodu tento model funguje velmi dobře pro malé projekty, které mají být rychle vytvořeny a vydány – tedy prototypy či demoprogramy. [3]

3.2.3 Vodopádový model

Projekt využívající vodopádový model prochází řadou kroků, vycházející z počáteční myšlenky po konečný systém. Model je systematický, lineárně-sekvenční, obsahující několik po sobě jdoucích kroků. Na konci každého jednoho kroku má projektový tým přehled o tom, zda jsou připraveni přejít na další krok, nebo zda zůstane na dané úrovni, dokud nebude připraven k přechodu. [3]

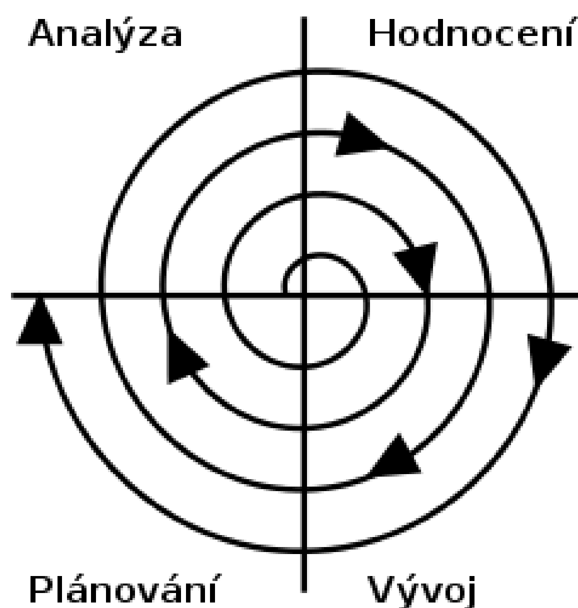
Obrovský důraz je kladen na specifikaci systému, a ačkoliv to může být limitující, tak tento model funguje velice dobře pro správně definované systémy a disciplinovaným vývojářským týmem. [3]

Vodopádový model se dnes již z důvodu mnoha nedostatků v praxi příliš nevyužívá. Značná nevýhoda je například v oblasti testování, které je téměř na konci celého modelu. Právě z vodopádového modelu, který představil Winston W. Royce, vychází celá řada modifikací. Ty se pokouší vyřešit nedostatky modelu. [2]

3.2.4 Spirálový model

Spirálový model byl představen Barrym Boehmem v roce 1986 a je využíván poměrně často, jelikož se ukázal jako účinný přístup k vývoji softwaru. Z velké části pokrývá největší nedostatky vodopádového modelu. Obecná myšlenka spirálového modelu spočívá v tom, že se nedefinuje detailně vše na úplném začátku, ale jen důležité funkce, které se vyzkouší, získá se zpětná vazba od zákazníků a přesune se na další krok. To se opakuje, dokud není vyhotoven finální produkt. Pro spirálový model jsou typické 4 hlavní kroky:

1. Určení cíle, alternativy a omezení
2. Vyhodnocení alternativ, identifikace a řešení rizik
3. Vývoj a verifikace další úrovně produktu
4. Plánování následující fáze



Obrázek 1: Spirálový model [wikimedia.org]

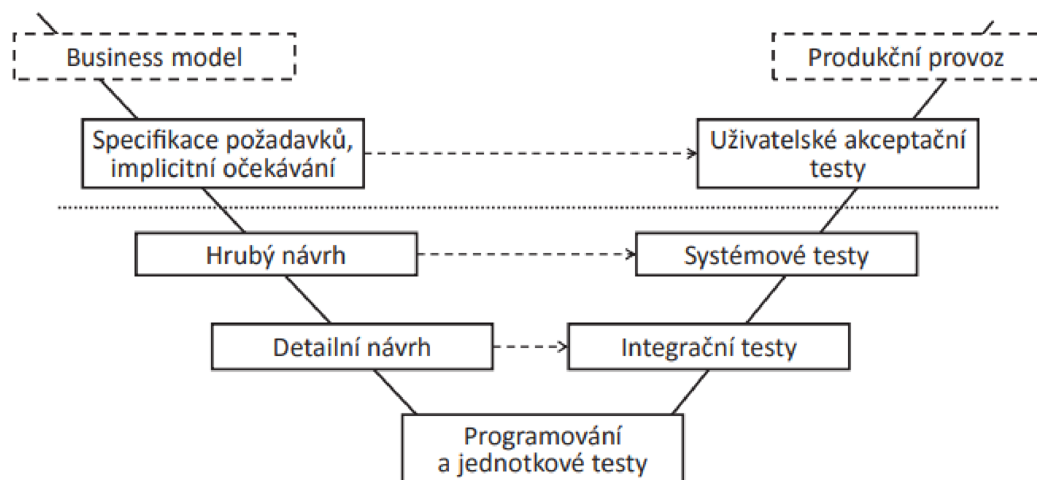
Z pohledu testování se jedná o dobrý model, jelikož je systém testován pravidelně. Umožňuje zapojení se do projektu už ve fázi návrhu, až po úplný konec projektu. Testování na konci projektu tedy představuje jakousi kontrolu, zda je vše v pořádku. [3]

3.2.5 V model

V model, někdy nazýván jako model verifikace a validace, je sekvenční a každá fáze musí být dokončena před zahájením další fáze. Znamená to, že V model vychází z výše charakterizovaného vodopádového modelu. V model však řeší nedostatek vodopádového modelu tak, že výstup z každé fáze prochází verifikací v podobě testů. Pomocí toho lze identifikovat chyby ve správnou chvíli, což snižuje náklady na jejich opravu. Právě výhoda současného testování a programování dělá V model často používaným.

Pro V model se nejčastěji využívá čtyř kroková strategie testování: [4]

- Akceptační testy
- Systémové testy
- Integrační testy
- Test jednotek (Unit testy)



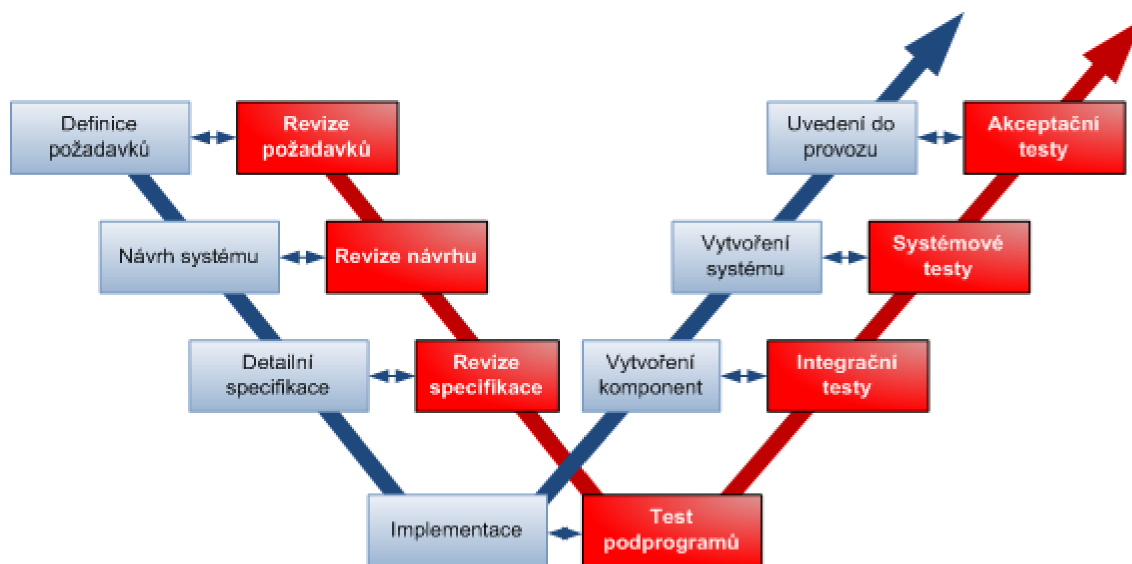
Obrázek 2: V model [19]

3.2.6 W model

W model se často využívá na problémy, které nemohou být vyřešeny pomocí V modelu, z kterého W model vychází. W model je charakteristický vztahem 1:1, který existuje mezi dokumenty a testovacími aktivitami, což zajišťuje testování systému od počátku vývoje. Použití W modelu pomáhá k tomu, aby každá fáze vývoje byla verifikována a validována. Model může být rozdělen do několika etap, které zahrnují:

- Budování test plánu a test strategie pro zajištění důkladného otestování před dodávkou
- Definování scénáře pro systém
- Příprava testovacích případů prostřednictvím specifikací a dokumentů s návrhy
- Přezkoumání testovacích případů a sdílení úprav
- Předání systému k testování různými testovacími metodikami, např. jednotkové testování, integrační testování, testování založené na specifikacích atp.
- Regresní testovací cykli a testy pro přijetí uživatelem

Ačkoliv je V model velmi účinným způsobem testování, W model je efektivnější a pomáhá získat širší pohled na testování. [14]



Obrázek 3: W model [umel.feec.vutbr.cz]

3.2.7 Inkrementální model

Inkrementální model je kombinací jednoho či více vodopádových modelů. Jeho podstatou je rozdělení požadavků do několika modulů a každý modul je pak vyvíjen odděleně. Každý modul obsahuje fázi sběru požadavků, návrh, implementaci a testování. Na konci jsou zhotovené moduly integrovány dohromady. Každý vyvinutý modul je samostatnou funkcí, která může být doručena ke koncovým uživatelům. Základ je však takový, že ostatní moduly jsou integrované jako další funkce jedna podruhé, až nakonec je doručen zákazníkovi. [5]

3.2.8 RAD model

Rapid application development, v překladu rychlý vývoj aplikací, či model rychlého vývoje, je druh inkrementálního modelu, kde jsou jednotlivé funkce a komponenty vyvíjeny paralelně prostřednictvím zkušených týmů. Konečné moduly jsou poté integrovány do samostatného prototypu, což zabraňuje selhání celého projektu. RAD model vyžaduje dostatečně velký rozpočet na projekt. RAD model nejčastěji zahrnuje následující fáze: [6]

- Obchodní modelování
- Modelování dat
- Modelování procesů
- Sestavení aplikace
- Testování

3.3 Metodiky vývoje

Metodika v obecném pojetí představuje souhrn metod a postupů, kterými se realizují konkrétní úkoly. Metodiky týkající se vývoje a údržbou informačního systému se označují jako metodiky vývoje IS/ICT. [7]

Jak lze odvodit z předešlé kapitoly, vývoj kvalitního softwaru závisí na řadě procesů, které ho značně ovlivňují. Jeden z nejpodstatnějších procesů je proces testování. V souvislosti s vývojem jsou sledovány dva hlavní směry v metodických přístupech, jimiž jsou rigorózní metodiky a agilní metodiky. Kritérium, které tyto dvě metodiky od sebe nejvíce odlišuje, se nazývá „Váha metodiky“. Kritérium váha metodiky rozděluje metodiky na těžké (rigorózní) a lehké (agilní) na základě součinu velikosti a hustoty metodiky, kde velikost představuje počet kontrolních prvků obsažených v metodice a hustota vyjadřuje míru podrobnosti a těsnosti tolerance metodiky, požadovanou detailnost a konzistenci prvků. [7]

3.3.1 Rigorózní metodiky

Rigorózní metodiky představují historicky největší skupinu metodik týkajících se podrobné deskripce procesu vývoje softwaru. Typické pro tento metodický směr je objemná dokumentace, jelikož dle rigorózní metodiky lze při budování IS/ICT veškeré procesy popsat, plánovat, řídit a měřit. Typická je taktéž charakterizace vodopádovým modelem, tedy posloupnost jednotlivých činností v průběhu vývojového cyklu, ačkoliv existují také rigorózní metodiky, které jsou založeny na inkrementálním a iterativním vývoji. Do těchto metodik patří například OPEN, RUP, EUP. Typickými rigorózními metodikami jsou pak například vodopádový model a spirálový model a ostatní modely, z nichž vycházející. [7,8]

OPEN

Metodika OPEN (Object-oriented Process, Environment and Notation) je zaměřena především na vývoj objektově orientovaných a komponentových aplikací. Stavebním kamenem metodiky OPEN byl projekt COMMA, který měl za cíl vytvořit metamodely rozličných metodik a metod objektově orientované analýzy a návrhu. Metodika OPEN je velice flexibilní, dokáže se velmi dobře přizpůsobit jak doméně, tak konkrétnímu projektu a pracovat s dovedností členů týmu. Z tohoto důvodu může být použita pro projekty jakékoliv velikosti. [7]

RUP

RUP (Rational Unified Process) je proces vývoje softwaru od společnosti IBM. Rozděluje vývojový proces na čtyři odlišné fáze zahrnující podnikové modelování, analýzu a návrh, implementaci, testování a nasazení. Tyto čtyři fáze jsou následující:

- Počáteční fáze
  - Cílem je stanovit myšlenku projektu, tedy definovat jeho cíle, požadavky, sestavení harmonogramu celého projektu. Vývojářský tým rozhoduje, zda stojí za to na projektu pokračovat, a jaké zdroje budou k tomu potřebné.
- Elaborační fáze
  - Cílem je definování architektury systému. V této fázi dochází k vytvoření určitého prototypu, pomocí kterého se provádí ověření a hodnocení architektury a požadovaných zdrojů. To vede ke zpřesnění plánu realizace systému.
- Konstrukční fáze
  - Konstrukční fáze představuje samotný návrh a realizaci systému, který je následně testován. Většinou se jedná o paralelní vývoj.
- Fáze nasazení
  - Fáze nasazení zajišťuje uživatelům použitelnost systému, včetně školení uživatelů, zřízení informační linky (helpdesku), předání dokumentů atp.

Každá jednotlivá fáze je ukončena časovým okamžikem, ve kterém musí být splněny cíle dané fáze. Poté dochází k rozhodování.

Metoda vývoje RUP poskytuje strukturovaný způsob, kterým si firmy mohou představit vytváření softwarových programů. Vzhledem k tomu, že poskytuje konkrétní plán pro každý krok procesu vývoje, pomáhá předcházet ztrátě zdrojů a snižuje neočekávané náklady na vývoj. [9]

3.3.2 Agilní metodiky

Podstatou agilních metodik je agilní přístup, tedy pružná reakce na změnu, pružné rozvrhnutí práce v průběhu vývoje a ověření výstupu s uživateli. Zásluhou agilního

přístupu je vývoj postaven na týmové spolupráci, otevřenosti změnám a v neposlední řadě otevřené a časté komunikaci.

Agilní metodiky se využívají především u tvorby velmi složitých a komplexních softwarů, kdy je obtížné detailněji definovat požadavky na úplném začátku projektu. Ty jsou tvořeny na základě zkušeností s prototypy postupně během jednotlivých iterací vývoje. K průběžnému sepsání požadavků často vypomáhá i zpětná vazba uživatelů. [10]

Agilní metodiky definují 10 hlavních principů:

1. Včasné a kontinuální dodání softwaru, který zákazníkům přináší hodnotu.
2. Změna požadavků je akceptovatelná i v pozdějších fázích vývoje, jelikož tím může zákazník získat konkurenční výhodu.
3. Každodenní spolupráce vývojářů s uživateli.
4. Motivovaní jedinci mající podporu vedení jsou klíčovým faktorem úspěchu.
5. Osobní komunikace je v oblasti vývojového týmu nejefektivnějším způsobem pro přenos informací.
6. Hlavní mírou úspěchu je fungující software.
7. Pro agilní procesy je předpokládán „zdravý“ vývoj.
8. Kvalitní návrh i perfektní technické řešení.
9. Podstatným požadavkem je jednoduchost řešení.
10. Samoorganizující se tým přináší nejlepší architektury, požadavky a návrhy. [7]

Mezi populární agilní metodiky lze zařadit například Scrum, Extrémní programování či DSDM. [7]

SCRUM

Metodika SCRUM byla vytvořena autory Kenem Schwaberem, Jeffem Sutherlandem a Mikem Beedlem a využívá agilní přístup vycházející z přesvědčení, že vývoj softwaru je empirický proces, což vyžaduje zcela odlišný způsob řízení. Název Scrum byl převzat z ragbyového prvku mlýn (skrumáž), což mělo zdůraznit adaptivnost a samoorganizaci. Scrum metodika je především zaměřena na řízení projektů, při vývoji softwaru bere v potaz nepředvídatelnost a nutnost monitorování, které naplňuje prostřednictvím denních konzultací či třiceti denních iterací označovaných jako „sprint“. Na konci každého sprintu je dodána vybraná skupina užitných vlastností. [7]

Rozsáhlost týmu využívající Scrum metodiku by měla být od čtyř do patnácti lidí, kteří v ideálním případě sídlí na stejném místě. Velikost a lokalita týmu však není pravidlem. Rozlišují se dvě skupiny účastníků a to „Pigs“ a „Chickens“. Do první skupiny patří osoby, které přímo souvisí s vývojem aplikace (např. Product Owner, Scrum Master), kdežto do skupiny „Chickens“ patří osoby, které za vývoj přímo nezodpovídají (např. stakeholdeři, či manažeři). [11]

Definovány jsou čtyři fáze životního cyklu:

- Plánovací fáze
    - Jedná se o fázi, ve které se specifikují první požadavky, plán dodávky a vize.
- Vynášecí fáze
    - Připojování nefunkčních požadavků do soubor požadavků (tzv. backlogu)
- Fáze vývoje
    - V této fázi dodává tým každých 30 dní funkcionalitu s nejvyšší prioritou. Každý člen týmu pracuje na přiděleném úkolu a zúčastňuje se denních meetingů, které monitorují stav projektu. Na konci každé iterace jsou předvedeny výsledky a jejich zhodnocení.
- Fáze dodávky
    - Ve fázi dodávky dochází k předávání produktu samotným uživatelům. [7]

Extrémní programování

Extrémní programování, často značené jako XP, je snadná, účinná a nepříliš riziková metodika vývoje softwaru. Je založena na vysoké disciplinovanosti s včasnou a nepřetržitou zpětnou vazbou vyplývající z krátkých cyklů a veškeré běžné činnosti vývoje softwaru jsou dovedeny do extrému. Plánování v XP bývá často rychlé, avšak zohledňuje přírůstkový přístup, tedy dochází v průběhu projektu ke změnám a přírůstkům plánu. Značnou výhodou je pružné určování termínů implementace funkcí a reagování na změny v zadání. K testování v XP metodice se značně využívají automatické testy napsané programátory či zákazníky, které slouží pro sledování pokroku ve vývoji a k zachycení různých nedostatků. [12]

Typický průběh vývoje extrémním programováním je následující: [13]

- Zadání
    - Jde o stručný popis funkčních požadavků a akceptačních kritérií v podobě uživatelských příběhů (user story), které píšou zákazníci za účelem vyjádření svých potřeb, na které by se měl systém zaměřit. Požadavky jsou kategorizovány dle významnosti a vývojáři pak odhadují čas nezbytný pro implementaci.
- Plánování
    - Plánovací dodávky – zákazník definuje požadované vlastnosti vývojářům, ti odhadují jejich náročnost a na základě odhadu se určí hrubý plán projektu. Hrubý plán projektu je zpřesňován po každé jedné další iteraci.
    - Plánování iterace – jedná se o týmové plánování, při kterém se určí, co bude vyvíjeno v rámci dvoutýdenní iterace.
- Návrh
    - Extrémní programování je založené na nejjednodušším možném řešené, tzn. že složité řešení se ihned nahrazuje jednodušším a žádná funkčnost systému není přidávána předčasně, ale právě tehdy, kdy je pro systém nezbytná.
- Implementace
    - Extrémní programování využívá takzvané párové programování. To spočívá v tom, že u počítače se sedí v párech, kdy jeden z dvojice přemýšlí o implementaci dané metody a druhý z dvojice o zjednodušení implementace. Metoda párového programování je velmi efektivní a vede ke komunikaci v týmu, což je jedna z hodnot extrémního programování.
- Testování
    - Pro každou část kódu je napsán jednotkový test, a to ještě dříve, než je napsán samotný kód. Na nalezenou chybu jsou napsány další testy zabraňující její šíření a opakování se v následujících fází vývoje. Z uživatelských příběhů se pak píší akceptační testy. Jelikož v XP je prováděno velmi často regresní testování, využívá se často automatických testů.

Autor metodiky XP, Kent Beck, stanovil také čtyři hodnoty, které vedou k úspěšnému vedení projektu s využitím extrémního programování. [12]

„Budeme úspěšní, když budeme mít styl zachovávající důslednou sadu hodnot, které slouží potřebám člověka i společnosti: komunikace, jednoduchost, zpětná vazba a odvaha.“ [12]

Komunikace ve vývojářském týmu je nezbytným aspektem, bez kterého by byla práce na projektu mnohonásobně složitější. Metodika extrémního programování udržuje komunikační toky prostřednictvím mnoha postupů, které by bez komunikace nemohli fungovat. Jedná se především o párové programování, testování jednotek či odhadování úkolů. Tyto a další různé aktivity vedou právě ke komunikaci celého vývojářského týmu, především pak programátorů, manažerů a zákazníků. [12]

Předností, na kterou se metodika XP zaměřuje, je jednoduchost. Ovšem hledat nejjednodušší možné řešení není vždy nejsnadnější práce. Právě mezi první a druhou hodnotou, tedy mezi komunikací a jednoduchostí, existuje vzájemně se doplňující vztah. Více komunikace vede k prozření, co je potřeba dělat a čím jednodušší je zamýšlený systém, tím méně je o jeho vývoji potřeba komunikovat. [12]

Zpětná vazba v XP se sleduje v různých časových hledisek v rámci testování. Prvotní zpětná vazba se týká programátorů, kteří píší testy jednotek pro systémovou logiku. V tomto časovém hledisku jde o minuty až dny, během kterých programátor získá úplnou zpětnou vazbu o současném stavu systému.

Další zpětnou vazbou jsou testy funkcionality. Zjišťuje se zde i rychlost celého týmu, zda odpovídá plánu. Zákazníci mohou revidovat časový rozvrh z pravidla po dvou až třech týdnech. Také zpětná vazba je úzce propojena s ostatními hodnotami, například čím více máme zpětné vazby, tím je snadnější a efektivnější komunikace. [12]

Jak již bylo výše zmíněno, metodika extrémního programování vyžaduje dovedení běžných činností do extrému. K tomu je někdy zapotřebí vytvářet odvážná rozhodnutí, například při opravě chyby, která může vést k naprosté a okamžité nefunkčnosti většiny testů a následnému úsilí na celkovou opravu. [12]

DSDM

Dynamic Systems Development Method, neboli zkráceně DSDM, je agilní metodika vyvinuta v 90. letech na území Velké Británie. Je využívána jak v Evropě, tak i ve spojených státech a její předností je výborně propracovaný systém školení a zakládání si na kvalitní dokumentaci. Jak název napovídá, jedná se o dynamickou metodiku, což znamená, že disponuje schopností přizpůsobit se v průběhu vývoje různým změnám.

V praxi je metodika DSDM založena na 9 hlavních principech:

- Aktivní spolupráce s uživatelem
- Tým s kompetencí rozhodovat
- Hojné dodávky produktů
- Silná podpora podnikových cílů
- Využití iterativního a inkrementálního vývoje
- Dynamika při vývoji
- Stanovení požadavků na hrubé úrovni
- Testování během celého životního cyklu
- Kooperace mezi členy týmu

Metodika DSDM prochází těmito fázemi:

- Studie proveditelnosti
- Byznys studie
- Funkční model – sběr funkčních požadavků
- Návrh – zpodrobnění požadavků a návrh řešení
- Implementace – realizace návrhu, školení a další činnosti

Hlavní fáze (funkční model, návrh a implementace) má iterativní průběh. [7]

3.4 Kvalita softwaru

Pod slovem „kvalita“ si různí lidé dokáží představit odlišný koncept. Pro každého představuje kvalita určitého produktu rozdílný výsledek a spojit tyto názory do jednoho uceleného bývá často obtížné. Právě na toto téma se zaměřil David Garvin ve své práci založené na rozboru vnímání kvality, ve které byla kvalita analyzována z pěti hledisek:

- Transcendentální hledisko
- Hledisko uživatele
- Hledisko výroby
- Hledisko produktu
- Hledisko ceny

Odborné definice na kvalitu softwaru jsou součástí mezinárodních norem. Těmi nejpodstatnějšími, avšak již zastaralými, jsou například normy ISO/IEC 9126 Softwarové inženýrství – jakost produktu, či ISO/IEC 14598 Softwarové inženýrství – hodnocení softwarového produktu. Právě zastaralost a dlouhodobou nekonzistentnost těchto norem řeší postupné nahrazování jednotného systému norem ISO/IEC 25000 - 25099 v rámci novějšího projektu SQuaRe. Jedna z norem v tomto projektu, ISO/IEC 25010, definuje kvalitu softwaru jako míru, do které jsou softwarovým produktem splněny stanovené a implicitní potřeby, jestliže je produkt používán za stanovených podmínek. Hodnocení kvality softwaru je zde složeno z charakteristik, mezi které patří funkčnost, účinnost, kompatibilita, použitelnost, bezporuchovost, bezpečnost, udržovatelnost a přenositelnost. Každá z těchto charakteristik obsahuje i své vlastní podcharakteristiky, které představují předmět měření. Mezi ně patří i testování, z čehož vyplývá, že testování představuje informaci o kvalitě produktu. [4]

3.4.1 FURPS

FURPS představuje populární model kvality obsahující 5 atributů kvality, jejichž první písmena v anglickém tvaru slova dávají dohromady název modelu. Mezi tyto atributy patří:

- Functionality (funkčnost)
  - Atribut představující požadované funkcionality, schopnosti a bezpečnostní aspekty systému
- Usability (použitelnost)
  - Atribut představující vnímání systému uživatelem – např. jednoduchost použití, konzistence, estetika atd.

- Reliability (spolehlivost)
    - Atribut představující správnost výstupu, četnost selhání, závažnost selhání, doba bezporuchového provozu atd.
- Performance (výkonnost)
    - Výkon za určitých podmínek, odezva systému apod.
- Supportability (podporovatelnost)
    - Atribut představující kombinaci určitých vlastností (škálovatelnost, testovatelnost, aj.)

Jak lze vydedukovat, pouze jeden atribut se týká funkčních požadavků. Zbylé 4 atributy jsou pak mimofunkční. V současnosti se lze setkat s rozšířeným modelem, jehož název je FURPS+, obsahující další kategorie jako jsou omezení návrhu, požadavky na implementaci, požadavky na rozhraní a požadavky na fyzické vlastnosti. [4]

3.4.2 Zajišťování kvality

Zajišťování kvality softwaru není pouze samotné testování, jak bývá často nesprávně publikováno. Zajišťování kvality softwaru neboli Software Quality Assurance (SQA) je ve skutečnosti zcela zaměřeno na kvalitu procesů úplného životního cyklu softwaru, což logicky ovlivňuje výslednou kvalitu softwarového produktu. Hlavními aktivitami jsou definice, zavádění procesů včetně kontrol, zda jsou dodržovány a klasifikace s cílem nalezení možných vylepšení.

Za cíl zajišťování kvality se dá považovat předcházení vzniku defektu, což je hlavním rozdílem od řízení kvality softwaru. [4]

3.4.3 Řízení kvality

Řízení kvality softwaru (ang. Software Quality Control) je dalším populárním pojmem, jehož význam bývá také často zaměňován. Oproti zajišťování kvality je řízení kvality zaměřeno na výstupy ze samostatných procesů (dokumentace, kód atp.). Tyto výstupy jsou dále kontrolovány, zda odpovídají specifikacím a požadavkům. Řízení kvality

se tedy oproti zajišťování kvality zabývá nalézáním defektů, jejich odstraňování a kontrolování správnosti po provedení změn. [4]

3.4.4 Verifikace a validace

Zajišťování a řízení kvality softwaru souvisí s další dvojicí pojmů, jimiž jsou verifikace a validace, často označovány jako V&V aktivity. Tyto pojmy jsou na sobě vzájemně silně závislé a nelze je mezi sebou zaměnit, jelikož každý z nich představuje něco zcela jiného.

Obecně lze prohlásit, že aktivity verifikace a validace ověřují správnost a shodu se specifikacemi, naplnění uživatelských potřeb a stanoveného účelu. Tyto pojmy bývají často spatřeny v podobě neformální, avšak velmi populární, definice dle Barryho W. Boehma, který jej publikoval takto:

„Verifikace: Vytvářím produkt správně?“

„Validace: Vytvářím správný produkt?“ [4]

Na základě těchto otázek je zřejmé, že verifikace je proces ověření souladu produktu se specifickými požadavky, zatím co validace představuje potvrzení, že systém funguje dle očekávaní uživatele, který daný produkt testuje v podobě akceptačních uživatelských testů. Perfektní grafické zobrazení je uvedeno na obrázku č. 2, kde horizontální tečkovaná čára představuje rozdělení testování ze strany verifikace a ze strany validace. Výjimkou bývají systémové testy, které mohou být testovány v obou případech. [4]

3.5 Životní cyklus testování softwaru

Životní cyklus testování softwaru je testovací proces, který je prováděn systematickým a plánovaným způsobem a v kterém jsou vykonávány různé činnosti s cílem zlepši kvalitu výrobku. Následující kroky jsou součástí životního cyklu testování softwaru, a každý jeden krok má své vlastní vstupní kritéria:

- Analýza požadavků
- Plán testování
- Vývoj testovacích případů
- Nastavení prostředí
- Exekuce testů
- Ukončení testovacího cyklu

V ideálním případě je posloupnost výše zmíněných kroků dodržována, tedy přechod na další krok nebude umožněn do doby, dokud nebude aktuální krok dokončen. V praxi to však pravidlem není. [18]

3.5.1 Analýza požadavků

Analýza požadavků je úplně prvním krokem v životním cyklu testování softwaru. Tým pro zajištění kvality softwaru (Quality Assurance) se pokouší porozumět požadavkům z hlediska toho, co bude testováno. Ve chvíli, kdy by došlo ke konfliktu z důvodu špatného porozumění či chybějícího požadavku, snaží se QA tým problém vyřešit s různými zúčastněnými stranami, jako je například business analytik, systémový architekt, klient atp. [18]

3.5.2 Plán testování

Plán testování je nejdůležitější fází životního cyklu. Právě zde je definována veškerá strategie testování, i to je důvodem, proč občas bývá tato fáze nazývána jako strategie testování. O vytvoření plánu testování se nejčastěji stará test manager (manažer testování), kdy určuje odhady úsilí a nákladů na celý projekt. Obsahem dále bývá například přehled plánovaných testů, cíl testování, vstupní a výstupní předpoklady atd. Tato fáze typicky začíná po dokončení sběru požadavků. Výsledkem plánu testování je test plán. [18]

3.5.3 Vývoj testovacích případů

Po dokončení plánu testování zahájí QA tým vývoj testovacích případů. Testovací tým připraví podrobné testovací případy a pokud je potřeba, zajistí i testovací data určené k testování. Ve chvíli, kdy jsou testovací případy připraveny, dochází k jejich přezkoumání. To provádí většinou kolegové na stejné úrovni, či vedení. [18]

3.5.4 Nastavení prostředí

Nastavení testovacího prostředí je zásadní součástí v životním cyklu testování softwaru. Právě testovací prostředí rozhoduje o tom, za jakých podmínek bude software testován. Tato fáze je v pořadí za fází vývoje testovacích případů, ale prakticky ji lze zahájit souběžně, jelikož se do nastavování testovacího prostředí testovací tým příliš nezapojuje. O nastavení prostředí se povětšinou starají vývojáři, v některých případech zákazník. Testovací tým by měl připravit případy smoke testů, aby ověřil připravenost nastaveného prostředí. [18]

3.5.5 Exekuce testů

Po úspěšném vývoji testovacích případů a nastavení testovacího prostředí lze zahájit fázi exekuce testů. Testování testovacích případů provádí nejčastěji testeři a test analytici, avšak v některých situacích se zapojuje celý tým. Jakmile je testovací případ otestován, tester jej označí odpovídajícím statusem. [18]

3.5.6 Ukončení testovacího cyklu

Během vývoje softwaru je samozřejmé, že se v aplikaci nacházejí chyby a výsledek všech testovacích případů nebude ihned vyhodnocen jako bezchybný. Nalezené chyby se zadávají do systému pro zadávání defektů a následně jsou opraveny vývojáři, případně je pozměněn testovací případ v závislosti na chybě. V závěru se provádí setkání všech členů a zhodnocují se kritéria pro dokončení cyklu. [18]

3.6 Role při testování

V každém pracovním týmu jsou jisté role a pozice pracovníků, kteří práci vykonávají. Stejně tomu tak je právě v testovacím týmu. Testování softwaru nereprezentuje pouze exekuci testů, ale celou řadu různých činností a procesů, na jejichž základně jsou role často rozděleny. Nejčastěji a ideálně provádí testování softwaru celý tým pracovníků, avšak jejich pracovní pozice se liší. Počet členů by měl být dostatečně velký v závislosti na velikosti projektu a pro každý jeden projekt se může struktura měnit dle potřeb projektu. Nejčastěji jsou však v testovacím týmu role testera, test analytika, test leadera a test managera.

3.6.1 Tester

Role testera se může lehce lišit v závislosti na velikosti a charakteru firmy, ve které práci vykonává. Obecně však provádí exekuci manuálních testů, jejichž testovací scénáře připravil test analytik. Tester musí být dobře seznámen s aplikací, znát její funkčnost. K tomu ve většině případů poslouží funkční specifikace, nebo testera seznámí s aplikací analytik. Tester musí být pečlivý a svědomitý, především při hledání chyb, jejich zadávání do bug trackingu a sledování jejich životních cyklů. Po opravě chyby a nasazení na dané prostředí je tester předurčen k přetestování dané chyby, aktualizace jejího stavu a stavu samotného testovacího případu. Tester dále vytváří a spravuje potřebná testovací data, která k testování využívá. [15]

V případě, že daný projekt využívá automatizovaného testování, provádí tuto činnost tester. Ten se ale od obyčejného testera velmi liší. Obvykle automatizované testy provádí specializovaný pracovník, který se zaměřuje pouze na tento typ činnosti. [4]

3.6.2 Test analytik

Jak již název poukazuje, hlavní oblastí, kterou se test analytik zabývá, je test analýza. Ta spočívá v porozumění vyvíjené aplikace na základě dokumentace od analytiků a následné tvorby dokumentů pro testování. V test analýze jsou tvořeny testovací případy

na základě požadavků. Z testovacích případů jsou dále definované testovací scénáře, které pokrývají jak pozitivní, tak negativní výsledek testu. Test analytik při své práci se scénáři shromažďuje požadavky na testovací data a stanovuje riziko a prioritu scénářů. Po dokončení všech testovacích případů určuje test analytik podrobnější plán testů. Test analytik se také zapojuje do samotného testování a provádí veškeré pracovní aktivity jako tester. [4]

3.6.3 Test leader

Jednou z hlavních a úplně prvních činností, kterou test leader neboli vedoucí testování provádí, je vytvoření plánu testování (tzv. test plan), který musí odpovídat zvolené strategii testování. Samotný test plan musí být test leaderem v průběhu kontrolován, zda je dodržován a v případě změny je nutné provést korektivní akci. Další podstatnou činností je zadávání a rozdělování práce pro jednotlivé členy týmu (testery a test analytiky) a kontrola jejich plnění. Obvykle vytváří pravidelné reporty o aktuálním stavu testování a nalezených chybách. V rukou test leadera by mělo být nastavení a správa systému pro zadávání defektů. Také test leader se zapojuje do samotného testování.

Ve větších firmách je projekt rozdělen do více částí (např. backend[1] a frontend[2]), na kterých pracují samostatné skupiny pracovníků, které mají svého test leadera. V menších firmách pak zodpovídá test leader za celý tým. [4]

3.6.4 Test manažer

Test manažer neboli manažer testování, uzavírá vrchol pomyslné pyramidy rolí při testování. Právě test manažer je odpovědný za celý projekt v rámci testování a odpovídá vedení projektu, případně vyššímu managementu. Striktně spolupracuje s test leaderem, kontroluje a schvaluje jeho práci, například při tvorbě test planu. Mezi nejčastější činnosti

[1] Backend – část webové aplikace neviditelné pro běžného uživatele

[2] Frontend – část webové aplikace viditelné pro běžného uživatele

patří stanovení kritérií kvality produktu, správa klíčových dokumentů, koordinace akceptačního testování, volba přístupu k testování a stanovení jeho cílů a také rozhoduje o sestavení testovacího týmu a případných změnách. Často vyjednává s managementem, vývojovým týmem a zákazníkem. Pro svůj testovací tým zajišťuje veškeré nástroje potřebné k testování. [4]

3.7 Způsob testování

Jak již bylo mnohokrát zmíněno, testování během vývoje softwaru je nepostradatelný proces a je nezbytné jej využít po celou dobu vývoje pro dosažení požadované kvality softwaru. Existuje celá řada pohledů na proces testování, kategorizace testů a také způsobů, na kterých je testování založeno. Právě způsob testování je nejčastěji dělen na testování takzvané černé a bílé skříňky.

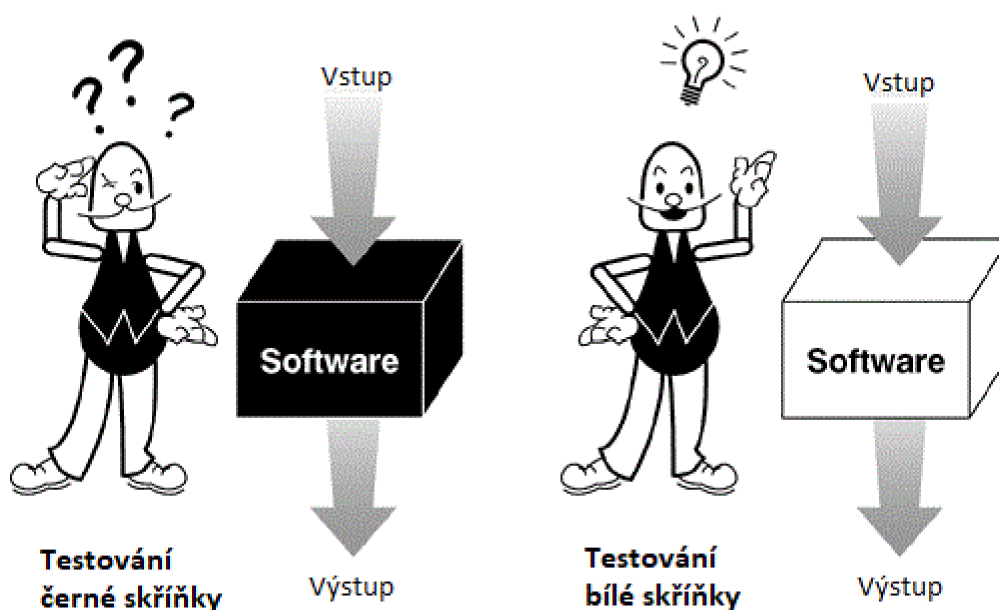
3.7.1 Černá skříňka

Testování černé skříňky, často využíváno anglického názvu black-box, je založeno na skutečnosti, že je tester seznámen pouze s tím, co má testovaný software dělat. Jinými slovy nevidí do té pomyslné skříňky, aby zjistil, jak samotný software funguje. Není seznámen s kódem aplikace a s jejím vnitřním fungováním. Jestliže tester zadá určitý vstup, dostane určitý výstup. Perfektní příklad představuje softwarová aplikace kalkulačka. Tester testující dle způsobu černé skříňky zadá nějakou operaci a dostane matematický výsledek. Při testování v černé skřínce nezáleží na tom, jaké se provádí operace a algoritmy během výpočtu. Podstatný je výstupní hodnota na konci výpočtu.

Tento způsob se nejčastěji využívá u testování zákazníkem, který nepotřebuje vidět do kódu softwaru a testuje simulaci běžného použití aplikace. [3]

3.7.2 Bílá skříňka

Při testování bílé skříňky, anglicky white-box, má softwarový tester přístup ke kódu programu s možností jej prozkoumat. To mu může výrazně pomoci s testováním. Tento fakt vede k tomu, že tester do pomyslné skříňky vidí. Rizikem v oblasti testování bílé skříňky je neobjektivní testování. Pokud tester vidí do kódu, pak může přizpůsobit testy tak, aby odpovídali operacím v kódu. [3]



Obrázek 4: Znázornění testování černé skříňky oproti testování bílé skříňky [3]

3.7.3 Šedá skříňka

Rozhodnout mezi způsobem testování, kdy obě možnosti mají své světlé i stinné stránky, může být velice náročné. Z tohoto důvodu vznikla další možnost, která by měla reprezentovat zlatou střední cestu. Touto možností je testování takzvané šedé skříňky neboli gray-boxu. Řešení spočívá v návrhu testů orientovaných z pohledu zákazníka, což je převzato z testování černé skříňky, a provádění testů je pak založeno na testování bílé skříňky, kdy uživatel využívá možnosti nahlížení do kódu, či ověřuje výsledky operací skrze dotazy do databáze. [2]

3.7.4 Ad hoc a exploratorní testování

Ad hoc testování je dalším způsobem testování, které je vždy prováděno nesystematicky, bez plánování a náhodně. Cílem je nalézt co největší množství chyb v testovaném softwaru. Běžně k tomuto způsobu testování není vyžadována žádná dokumentace. Často je označováno jako neefektivní testování bez strategie. Každopádně mnohdy úspěšnost tohoto testování předčí očekávání, proto je využíváno jako doplněk k formálnějším technikám testování.

Zpočátku terminologie spojovala k ad hoc testování ještě jeden termín, a to exploratorní testování. V současnosti je již způsob testování mezi těmito termíny definován odlišně. Ačkoliv je mezi oběma způsoby silná spojitost, existuje několik rozdílů, které jej rozlišují. Také exploratorní testování má za cíl nacházet všechny možné defekty napříč aplikací, hlavní rozdíl je však v systematičnosti. Oproti ad hoc testování je exploratorní testování systematické, tester nejdříve zkoumá samotnou aplikaci a mapuje její funkce a data. Poté jsou identifikované rizikové oblasti a dochází ke zvolení strategie testování. Exploratorní testování provádí typicky zkušený tester s patřičnými znalostmi. [4]

3.8 Typy testů

Testy lze kategorizovat dle mnoha hledisek, avšak nejdůležitější kategorizace jsou dle úrovně testování a účelu testování. Testy dle úrovně testování jsou řazeny na základě specifických testovacích cílů a testy dle účelu testování jsou testy potřebné k dosažení těchto cílů. [16]

.

3.8.1 Úroveň testování

Typy testů kategorizovaných dle úrovně vývoje lze snadno odvodit z V-modelu na obrázku č. 2, kde každý dokument na levé straně modelu je testován daným typem testu na pravé straně modelu. Právě tyto dokumenty jsou základem, na kterém jsou testy vytvořeny. Výraz „úroveň testů“ či „testovací úroveň“ označuje zaměření testování a typy problémů, které s vysokou pravděpodobností odhalí. Typické úrovně jsou: [16]

- Jednotkové testy (testy komponent)
- Integrační testy
- Systémové testy
- Uživatelské akceptační testy

Jednotkové testy

Předtím, nežli je možné testovat jednotkové testy, musí být logicky napsán kód, jak lze vydedukovat z obrázku č. 2, ve spodní části V modelu. Jednotkové testy právě tento kód testují. Obecně je kód zapsán do části komponenty, či jednotky, které jsou obvykle konstruovány izolovaně kvůli integraci v pozdější části. Jednotky se také nazývají jako programy, moduly nebo již zmíněné komponenty.

Testování jednotek a samotné jednotkové testy jsou určeny k tomu, aby bylo zajištěno splnění a korektnost kódu napsaného pro danou jednotku před integrací s jinými jednotkami. Vytváření testů probíhá za podpory specializovaných frameworků. Testy vytváří a provádí především vývojář, který kód napsal, nebo osoba, která má napsanou specifikaci programu. Defekty nalezené a opravené během testování jednotek se většinou nezaznamenávají. [16]

Integrační testy

V momentě, kdy jsou jednotky neboli komponenty připraveny, přechází se do etapy, ve které se jednotlivé komponenty seskupí a vytvoří systém. Odborně je tento proces nazýván jako integrace. Účelem integračního testování je odhalit chyby v rozhraní, tedy v komunikaci mezi jednotlivými komponentami aplikace, mezi komponentou a systémem, komponentou a hardwarem, nebo dokonce komponentou a rozhraním jiného systému. Výsledkem úspěšného integračního testování zpravidla bývá dostatečně stabilní systém. [16]

Systémové testy

Po zjištění a ověření toho, že veškeré komponenty pracují korektně pospolu, přichází na řadu systémové testování. To se zaměřují na testování systému v podobě celku, který by měl užívat konečný zákazník, tudíž se testovací tým snaží otestovat systém z pohledu

zákazníka, nikoliv z pohledu vývojáře pracujícího s kódem. Fáze systémového testování je velmi důležitá, jelikož se precizně ověřuje splnění požadavků, které byly specifikované zákazníkem. Systémové testy ověřují soulad očekávaného chování se skutečným, validují výstupy, testují možné pozitivní, ale i negativní situace. Typicky nastává několik cyklů, při kterých jsou prováděny funkční i nefunkční testy, zaznamenány a opravovány defekty včetně jejich přetestování v dalším cyklu. Systémové testování by mělo být dokončené s přesvědčením, že produkt bude zákazníkem akceptovatelný. [4]

Uživatelské akceptační testy

Po dokončení systémového testování dochází na další velice podstatnou fázi, kterou je akceptační testování. Jedná se o validační aktivitu, která je prováděna uživateli za stranu zákazníka. Odborně je toto testování označováno zkratkou UAT (User Acceptance Testing)

Podstatou akceptačních testů je odhalit, zda produkt, jenž byl dodán, splňuje takzvaná akceptační kritéria, která byla definována zákazníkem jako požadovaná a ověřitelná podmínka pro přijetí produktu. Akceptační kritéria jsou vždy závazně sjednána mezi oběma stranami předem, proto za předpokladu nesplnění těchto kritérií může zákazník produkt odmítnout.

Jak bylo uvedeno výše, samotné testování akceptačních testů provádí uživatelé na straně zákazníka, a to standardně dle takzvaného plánu akceptačního testování, který je poskytnut od zákazníka, či naopak dodán spolu s produktem dodavatele. Testuje se softwarový produkt jako celek, včetně různých dodaných výstupů (manuál, šablony dokumentů atp.). Akceptační testování je rozdílné od předešlých typů testování a nejen tím, kým je práce vykonávána, ale v jejím způsobu. Testeři prioritně nehledají defekty v aplikaci, ale zaměřují se na scénáře, které zahrnují jejich každodenní aktivity z běžného provozu. [4]

3.8.2 Účel testování

Typy testů kategorizovaných dle účelu testování lze nejčastěji rozdělit na funkční a nefunkční testy. Existují však i další možné testy, které budou společně dále rozepsány v této podkapitole. [16]

Funkční testy

Funkční testy, jak již z názvu vyplývá, jsou zaměřeny na funkcionalitu systému. Jejich úkolem je ověřit, zda systém odpovídá všem předem specifikovaným funkčním požadavkům, jež byly předem sepsány. Jelikož funkční testy ověřují správnost chování všech funkcí, které byly implementovány, mají podstatný vliv na bezporuchovost dokončené aplikace. Právě to vede k velkému důrazu a pečlivosti testování funkčních testů během celého cyklu testování.

Nejčastěji se funkční testy provádí v integrační, systémové a akceptační úrovni testování. Testování funkčních testů při vývoji přináší oproti ostatním typům velké množství nalezených chyb. [17]

Mezi funkční a často využívané testy se řadí také bezpečnostní testy (security testing). Tento okruh testů má na starost ověřování ochrany dat proti neoprávněnému přístupu. Typické pro tyto testy je také autentizace, autorizace a dostupnost dat. [4]

Nefunkční testy

Nefunkční testy, jak opět může být z názvu zřejmé, slouží k testování různých vlastností systému nesouvisejících s jeho funkcemi. I přes netestování funkční stránky systému jsou nefunkční testy velice důležité pro správné fungování systému.

Největší využití nefunkčních testů se vztahuje především k testování takzvaných performance testů. Ty slouží k ověření a testování zátěže aplikace, především při zvýšeném počtu souběžně pracujících uživatelů. Mezi nefunkční testy ověřující výkonnost aplikace patří také load testy, které simulují zátěž dlouhodobě, aby bylo ověřeno, že systém z dlouhodobého hlediska zůstane stabilní a použitelný. Podobným typem jsou tzv. stress testy neboli testy hraniční zátěže, kdy se testuje chování systému při extrémní zátěži. Cíl stress testů je zjištění všech možných omezení systému. Další obdobným typem jsou testy

spolehlivosti (reliability testing), které zjišťují časovou délku schopnosti běhu systému, aniž by došlo k selhání. [4]

Regresní a konfirmační testy

V odvětví vývoje softwaru je běžné, že v průběhu je některá část systému více či méně upravena. Ať je důvodem oprava defektu, či rozšíření funkčnosti, nebo úplně jiná změna, vždy dochází k riziku zrození zcela nového defektu v již otestované části systému. Právě tento fakt je podnětem pro takzvané regresní testování. Cílem regresního testování je ověření, že změna konkrétní části systému neměla vliv na funkčnost ostatních součástí a zůstali ve stejném stavu.

Regresní testování se provádí typicky na jednotkové, integrační a systémové úrovni a vždy se testují již existující testovací případy. Pokud by se pro regresní testování psali nové testy, bylo by obtížné zachytit rozdíly mezi aktuálními a dříve získanými výsledky. Nicméně prakticky nedochází k testování celého systému, jelikož regresní testování je velice nákladná činnost, nejen z časové stránky. Dochází tedy k segmentaci testovacích případů, které budou v rámci regrese testovány. Segment těchto testů by měl pokrýt možná co nejvíce oblastí, na které by změna měla mít dopad.

Mimo jiné je v regresním testování využíván tzv. balík regresních testů. Ten obsahuje právě testy, které pokrývají veškeré součásti aplikace. Ačkoliv to není pravidlem, tak regresní testování ideálně využívá automatizace. Ta je více přiblížena v kapitole 3.9.

Testy, které se zabývají ověřením a potvrzením správnosti opravy nalezené chyby se odborně nazývají konfirmační testy. [4]

Smoke a sanity testy

Názvem „smoke testy“ jsou označovány sady testů, které jsou prováděny za účelem zjištění dostatečné stability systému a funkčnosti všech jeho hlavních částí. Jejich hlavním cíle tedy není nalézání defektů, jako u klasického regresního testování, ale spíše prozkoumávání připravenosti systému. Jelikož jsou tyto testy exekvovány vždy po nasazení určitého buildu, bývají mnohdy automatizovány. Právě výsledek těchto testů relevantně ovlivňuje rozhodnutí, zda má smysl pokračovat v testování na daného buildu, či

testování pozastavit. Povětšinou bývají sestaveny v pozdní fázi integrace a často bývají hlavním kritériem pro přechod na fázi systémového testování.

Dalším a velmi příbuzným typem ke smoke testům jsou takzvané sanity testy. Ty se od smoke testů odlišují pouze v důkladnějším provádění těchto testů. [4]

End-to-End testy

Tento typ testů, který je běžně zkráceně označován jako E2E, je založen na sledování dané entity, jako jsou objekty, data apod., během celé doby její životnosti v systému.

Nejčastěji se jejich využití nachází v průběhu akceptačního testování jako scénáře, které zahrnují aktivity uživatelů od úplného prvního kroku až po ten úplně poslední. Cílem těchto testů je prokázat správnost chování jako celku. Pro příklad se může jednat o následující sekvenci aktivit: Přihlášení → Získání dat → Provedení konkrétní aktivity → Odhlášení.

Využití E2E testů se však nachází i v systémových testech, především pro testování skrze vícero propojených podsystémů. [4]

3.9 Manuální a automatické testování

Manuální testování představuje takové testování softwaru, při kterém jsou testy prováděny lidským faktorem. Využívá se především v situacích, kdy provedení daného testu vyžaduje lidský úsudek, který by případný automatizovaný skript nebyl schopen řádně vyhodnotit. Manuální testování bylo základním kamenem pro proces testování v obecné rovině po řadu let, kdy se automatizace testů zdála nemyslitelná. Teprve rostoucí možnosti a trendy informačních technologií současné doby vedou k nárůstu zájmu o zvýšení automatizace v testování, která by přispěla k vyšší efektivitě samotného testování. Testovací či projektové týmy pak mnohdy stojí před rozhodnutím, zda zainvestovat do automatizace či nikoliv. [22]

Oproti tomu automatické či automatizované testování lze obecně a jednoduše definovat jako testování prostřednictvím určitého softwaru, který vykonává jednotlivé kroky namísto lidského testera. Nelze však říci, že by docházelo k plnohodnotnému nahrazení za lidský faktor, jelikož automatizovaný test musí být někým napsán. Pravdou

je, že poptávka po automatizovaných testerech, tedy osobách, které jsou schopné automatizované testy obstarat a obsluhovat, stále roste. Tomu nahrává i fakt, že automatizace se velmi rychle vyvíjí s příchodem nových technologií a nástrojů.

Hlavním důvodem, proč se automatizované testování využívá, je zvýšení efektivity testování, jelikož manuální návrh, provedení i analýza výsledků testovacích případů mnohdy zabírá příliš mnoho času. [22]

3.9.1 Selenium

Nejpoužívanějším nástrojem pro automatizované testování je Selenium, které se skládá z několika navzájem se doplňujících komponent. Mezi komponenty patří:

- Selenium IDE
- Selenium RC
- Selenium WebDriver
- Selenium Grid.

Selenium je vyvinuto v programovacím jazyku Java a je zcela přenositelný (lze ho použít na různých platformách). [21]

Selenium IDE

Jedná se o komponentu dostupnou v podobě plaginu do internetového prohlížeče Mozilla Firefox. Představuje nejsnazší způsob, jak vytvořit automatizovaný test, avšak pro jeho zaměření na pouhý jeden internetový prohlížeč není tento typ Selenia využíván. [21]

Selenium RC

Selenium RC je založeno na serveru představující proxy server pro instance internetového prohlížeče, které na svůj počin spouští a vypíná. Výhodou je podpora mnoha programovacích jazyků a připravenost knihoven funkcí pro jednotlivé jazyky. [21]

Selenium WebDriver

Selenium WebDriver představuje oproti Selenium RC snazší přístup k vytváření testů, jelikož není nutné využívat Selenium server pro spouštění testů. Pomocí WebDriveru je volán každý prohlížeč sám o sobě. [21]

3.10 Testovací případ

Testovací případ, známý populárním anglickým názvem „test case“, představuje určité akce vykonávané s danou softwarovou komponentou. Ten se využívá jak pro manuální, tak pro automatické testování, avšak v odlišné podobě. Pro manuální testování se vytváří seznamy kroků, které jsou zapotřebí provést, a očekávaných výsledků. Pro automatické testování se používá spíše termín „testovací skript“, jehož náplní je sada programových instrukcí. Automatické testovací případy by měly, na rozdíl od manuálních, sami rozpoznat výsledek testu.

Obecně je testovací případ dokument, který popisuje určitou činnost či funkčnost, která by měla být otestována. Podrobněji je testovací případ zdokumentován v praktické části diplomové práce, a to v kapitole 4.7. [20]

3.11 Správa defektů

V počátku této kapitoly je vhodné definovat základní pojmy, které bývají často nesprávně zaměňovány a to:

- Chyba
  - Chybu může provést člověk, tedy vývojář či analytik, během výkonu své práce.
- Defekt
  - Defekt, nebo také bug, je nesoulad aktuálního chování s chováním očekávaným. Zdrojem defektu bývá chyba, které se vývojář či analytik dopustil. Nalezení defektu vede k nápravě chyby, neboť chybovat je lidské.

- Selhání
    - V důsledku defektu může docházet k selhání systému.
- Incident
    - Dle definice ISTQB reprezentuje incident určitou událost, kterou je zapotřebí dále prozkoumat. V praxi však bývají tímto pojmem označovány vady systému nalezené v produkčním prostředí.

Při vývoji nového softwaru je pochopitelné, že bude docházet k chybám, především pokud se jedná o více komplexní systém, na kterém pracuje větší počet lidí. V takových projektech je samozřejmostí nástroj pro správu defektů, takzvaný bug tracker či bug tracking software.

V průběhu životního cyklu defektu se do kontaktu s defektem dostane téměř celý testovací a vývojářský tým. Osobou, která defekt objeví a zadá do bug trackeru, bývá nejčastěji tester. Mnohdy se v projektu nachází role analytik defektů, či defekt koordinátor, který má za úkol správu defektů. Při nově vytvořeném defektu testerem posoudí, zda se jedná o relevantní defekt a odstraní případné duplicitní defekty. V případě, že je zadaný defekt nejasný či nekompletní, vrací jej k osobě, která defekt reportovala, aby potřebné informace dodala. Pokud je defekt v pořádku, je předán analytikem defektů kompetentním vývojářům k opravě. Vývojář opravuje defekty a dodává informace o opravách. [19]

3.11.1 Příčiny vzniku defektů

Co se příčin vzniku defektů týče, může jich být celá řada a za jejich zrozením nemusí vždy stát vývojář. Opět lze vycházet z V modelu na obrázku č. 2, kdy realizátor veškerých aktivit nacházejících se na levé straně modelu může zrealizovat chybu, která může být v budoucím vývoji zdrojem defektu. Mezi nejčastější příčiny vzniku defektů tedy patří: chybná či neúplná specifikace požadavků, nesprávná interpretace požadavků do funkčního designu, chybně interpretovaný funkční design a následně technický design, nedostatečný výkon aplikace, nevhodná ovladatelnost, nesprávně pochopené požadavky či design vedoucí k defektům v testovacích scénářích a testovacích skriptech. [19]

3.11.2 Atributy defektu

Každý založený defekt by měl v ideálním případě obsahovat určité atributy. Některé z atributů jsou nepovinné, jiné však nepostradatelné. Následující seznam představuje nejčastější atributy nacházející se v defektu:

- Název (Summary) – stručný popis defektu
- Stav – stav, ve kterém se defekt nachází
- Priorita defektu
- Popis defektu – popis defektu nejčastěji obsahuje postup k vyvolání defektu, popis očekávaného chování systému a popis skutečného chování systému
- Prostředí
- Řešitel a reportér

3.11.3 Priorita defektu

Se založením nového defektu je zapotřebí určit prioritu daného defektu. O to by se měl postarat samotný tester, který defekt založil, případně opravit či updatovat osoba s vyšší odpovědností (např. test analytik, defekt koordinátor, test manažer). Také priority, kterých mohou defekty nabývat, patří do seznamu aktivit, které by měly být stanoveny již na začátku projektu. Výčet nejčastějších priorit se nachází v následujícím seznamu:

- 1 – „Blocker“ (Blokující)
- 2 – „Critical“ (Kritická)
- 3 – „Major“ (Důležitá)
- 4 – „High“ (Vysoká)
- 5 – „Minor“ (Nižší)
- 6 – „Trivial“ (Triviální)

Často se také využívá zkrácené verze v následující podobě:

- 1 – „High“ (Vysoká)
- 2 – „Medium“ (Střední)
- 3 – „Low“ (Nízká)

Někdy bývají číselné hodnoty nahrazeny alfabet znaky, tedy A, B, C atp.

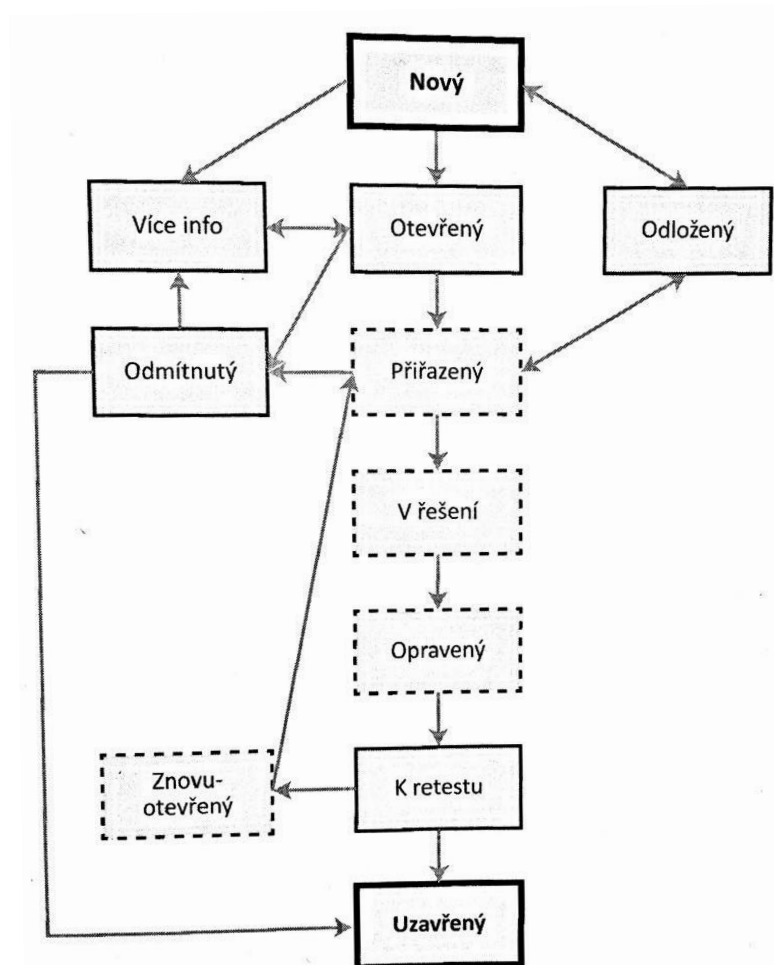
3.11.4 Životní cyklus defektu

Životní cyklus defektu by měl být společně s atributy defektu, řízení změn, zodpovědností účastníků a konfigurací nástroje na správu defektů dobře promyšlen již na začátku projektu a každý kompetentní pracovník by měl být s tímto seznámen.

Životní cyklus defektu by měl být v ideálním případě zrealizovaný na základě kontextu daného projektu, což jej může zjednodušit, případně rozšířit jeho stavy. Stanovený cyklus, tedy jeho stanovené stavy, není vhodné v průběhu projektu nadále měnit.

Pomyslných stavů, kterých mohou defekty nabývat, existuje celá řada a z logického pohledu vyplývá, že čím více stavů zařazených do životního cyklu defektu určitého projektu, tím lépe lze rozlišit i ty nejjemnější rozdíly ve stavech, avšak jejich správa bude velmi nepřehledná. Právě nepřehlednost způsobuje špatnou práci s defekty, což může mít pro projekt negativní výsledky. Nejčastěji používanějšími stavy však jsou: nový, otevřený, v řešení, opravený, uzavřený a odložený. [19]

Na obrázku č. 5 je vyobrazen příklad životního cyklu defektu, který může být samozřejmě zjednodušen, ale také rozšířen. [19]

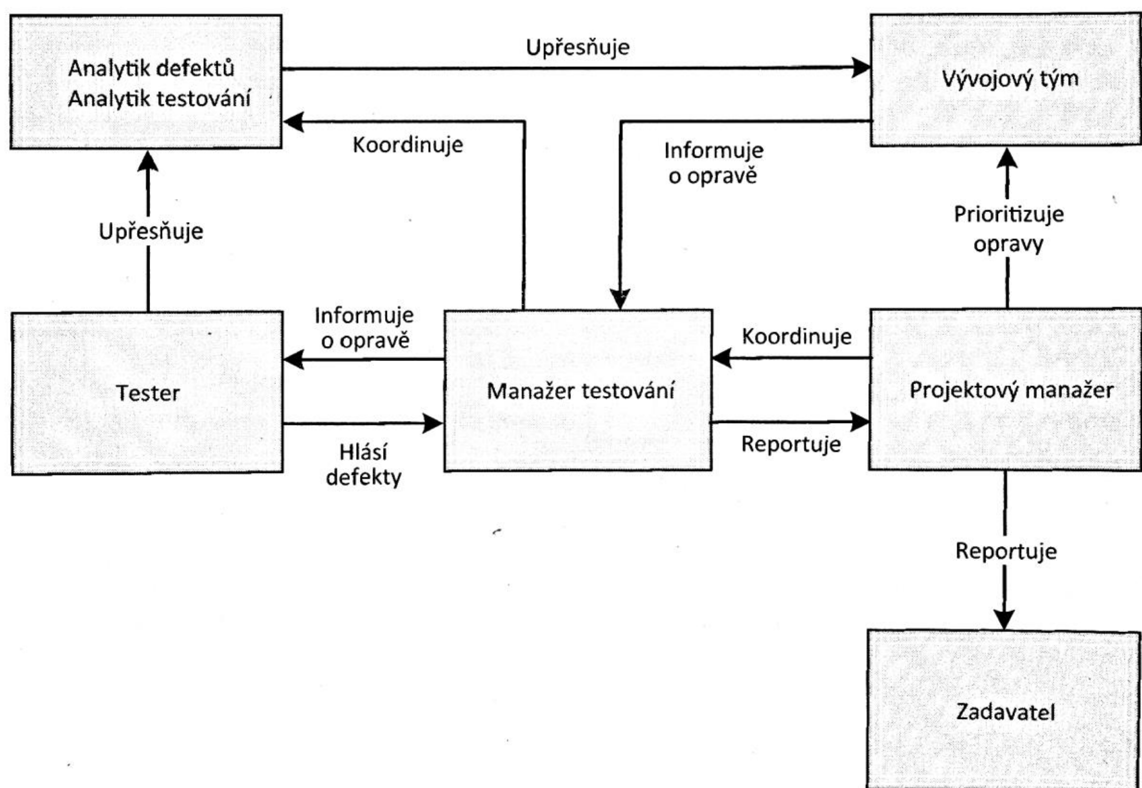


Obrázek 5: Příklad stavů životního cyklu defektu [19]

3.11.5 Komunikační mapa

Během finalizace projektu, kdy se vykonávají poslední testy, opravují a uzavírají defekty, dochází často ke stresujícím a vypjatým situacím. Pro tyto případy se často vytváří komunikační mapa.

Komunikační mapa zefektivňuje komunikaci v průběhu práce tím, že zobrazuje informační toky působící mezi jednotlivými subjekty. Pro úplné zefektivnění by měli být s komunikační mapou seznámeni všichni členové týmu již v ranní fázi projektu. Příklad komunikační mapy pro správu defektů je k prohlédnutí na obrázku č. 6. Stejně jako u životního cyklu defektu, také komunikační mapu lze upravit dle libosti projektu. [19]



Obrázek 6: Příklad komunikační mapy pro správu defektů [19]

3.12 Testovací prostředí

Pro testování softwaru při vývoji je nezbytný pojem testovací prostředí. Testovací prostředí je definováno jako prostředí, ve kterém je hardware, software, simulátory a nástroje včetně vybavení, které je důležité pro vykonávání testů. Je možné jej velice zjednodušeně popsat jako prostředí, které by mělo být určitým způsobem odvozené od cílového produkčního prostředí. Ovšem s příchodem nových metodik a technik si už projekt při vývoji nevystačí se snadným modelem, kdy se software nejdříve vyvine a teprve poté se testuje. Fakt, že v rámci životního cyklu vývoje softwaru se provádí různé typy testů, je důvodem, proč existuje řada různých testovacích prostředí. Jak již bylo zmíněno, testovací prostředí obsahuje nástroje, které vypomáhají s vykonáváním testů. Mezi tyto nástroje patří například nástroje pro automatizované testování, nástroje pro zátěžové testy, nástroje pro přístup do databáze, nástroje pro vytváření testovacích dat aj.

O testovacím prostředí by se mělo v ideální případě začít jednat již na začátku projektu, zhruba ve fázi prvotní analýzy požadavků. Řešitelé by měli mít alespoň představu o počtu testovacích prostředí. Pokud by se jednání o testovacím prostředí posunulo do pozdějších fází, pravděpodobně by došlo ke zvýšení jednak finančních nákladů, ale také časových možností, jelikož testovací tým potřebuje mít testovací prostředí k dispozici v požadovaném čase. [19]

3.12.1 Typy testovacích prostředí

Typů testovacích prostředí je několik. Mezi sebou se liší v určitých atributech jako je například funkce, shodnost s produkčním prostředím, správa prostředí, výkonost nebo samozřejmě název. Typy nejčastějších testovacích prostředí jsou pak včetně atributů sumarizované v následující tabulce:

Název prostředí	Zkratka	Funkce	Shoda s produkčním prostředím	Výkonnost	Správa prostředí	Integrovanost
Pískoviště	-	Prototypování, ověřování nástrojů	Ne	Ne	vývoj	Ne
Developerské	DEV	Vývojové testy	Ne	Ne	vývoj	Ne
Systemtest	SYS	Vývojové testy integrace, systémové testy	Ano (SW)	Ne	vývoj	Ano
Integrační	INT	Systémové integrační testy, uživatelské akceptační testy	Ano (SW)	Ne	produkce	Ano
Předprodukční	PRE	Zátěžové testy, technické akceptační testy	Ano (SW i HW)	Ano	produkce	Ano
Podpora produkce	PRS	Simulování produkčních chyb a jejich retestování	Ano	Ne	produkce	Ano
Školící	EDU	Školení uživatelů	Ne	Ano	produkce	Ano
Produkce	PRO	Produkční prostředí	-	Ano	produkce	Ano

Tabulka 1: Typy testovacích prostředí [19]

Dle výše uvedené tabulky lze vydedukovat, že v rámci testování softwaru se nejčastěji využívá prostředí DEV, SYS a INT. V rámci prostředí DEV se nejčastěji provádí jednotkové testy, v prostředích SYS a INT pak systémové, integrační a akceptační testy. [19]

4 Vlastní práce

Po souhrnu teoretických východisek se bude praktická část diplomové práce zabývat testováním reálného projektu, který je ve fázi vývoje nových funkcionalit. Po implementaci funkcionalit a následném nasazení na testovací prostředí dochází k ověření funkčnosti. Při zohlednění metodiky testování je využito V modelu, charakterizovaného v teoretické části, s přechodem na akceptační testování.

V rámci vlastní práce bude provedena analýza časové a finanční náročnosti manuálního a automatizovaného testování s doporučením v podobě kvantitativního vodítka.

4.1 Popis systému

Obrázek 7: Logo projektu Animal Breath [Vlastní zpracování autora]

Webová aplikace, která je v rámci diplomové práce testována, se nazývá AnimalBreath. Jedná se o novou sociální síť, kde mohou uživatelé vytvářet profily svých domácích mazlíčků a přihlašovat je na různé soutěže. Majitelem a zároveň jednatelem aplikace je Jakub Zajíček, jehož motivací k vytvoření projektu je rostoucí zájem o sdílení informací a fotografií svých domácích mazlíčků na sociálních sítích. Aplikace AnimalBreath slouží výhradně pro domácí mazlíčky s cílem pomáhat, ať v populárním směru mazlíčků v podobě soutěží, či v podobě podpory při propojení s útulky zvířat či informování o nalezených a ztracených zvířatech.

Pro vytvoření profilu svého domácího mazlíčka je nezbytná registrace a přihlášení uživatele do systému za účelem vytvoření účtu, pod kterým dále budou podúčty

jednotlivých mazlíčků. Přihlášený uživatel může vytvářet profily svých mazlíčků, přidávat jejich fotografie, sledovat ostatní mazlíčky, či mazlíčky svých přátel a využívat dalších výhod aplikace.

4.2 Funkcionality systému

Funkcionality systému dle zadavatele projektu:

- Registrace
- Přihlášení
- Zeď
- Notifikace
- Chat
- Soutěže
- Útulky
- Ztracené zvíře

Vzhledem k obsahové náročnosti diplomové práce budou testovány pouze funkcionality registrace a přihlášení, které budou více rozepsané v kapitole 4.4.

4.3 Plán testování

V této kapitole bude vypracován návrh testovacího plánu omezeného pouze na potřeby v rozmezí diplomové práce. Testovací plán je podstatný pro stanovení strategie a organizaci procesů týkajících se testování implementovaných funkcionalit webové aplikace AnimalBreath.

4.3.1 Přehled plánovaných testů

Obsahem testování bude pouze proces registrace a proces přihlášení uživatele do systému. Testovány budou základní scénáře již zmíněných funkcionalit. Alternativní

a negativní scénáře testovány nebudou. Testovat se bude na grafickém uživatelském rozhraní portálu. Testování bude zprostředkováno manuálně a automatizovaně se zaměřením pouze na front-endovou část aplikace.

4.3.2 Cíl testování

Testování webové aplikace v rámci diplomové práce má 2 hlavní cíle. Prvním a obecným cílem testování je verifikace funkcionalit. Funkcionality musí splňovat specifikované funkční požadavky definované v dokumentaci pro případy užití v kapitole 4.4.

Další cíl testování vychází ze zaměření diplomové práce, tedy odhalení vhodnosti použití manuálního a automatického testování, včetně jejich komparace na základě finančního a časového faktoru.

4.3.3 Vstupní předpoklady

Vstupní předpoklady pro zahájení činnosti testování softwaru jsou následující:

- Dostupná dokumentace potřebná k procesu testování
- Dostupnost potřebných nástrojů k procesu testování
- Vytvořené a připravené testovací scénáře pro exekuci
- Nakonfigurované testovací prostředí

4.3.4 Výstupní předpoklady

Výstupní předpoklady pro ukončení činnosti testování softwaru jsou následující:

- Veškeré testovací scénáře jsou provedeny
- Veškeré závažné defekty jsou vyřešeny

4.3.5 Role a odpovědnosti

Role v projektu, jenž je předmětem diplomové práce, jsou zvoleny 4 základní a to: Test manažer, analytik, vývojář a tester. Jednotlivé odpovědnosti zvolených rolí lze vyčíst z následující tabulky:

Role	Odpovědnost
Test manažer	Zajištění a nastavení veškerých prostředků a nástrojů potřebných pro funkčnost projektu, zajištění komunikačních kanálů, komunikace se členy projektu.
Analytik	Návrh řešení včetně vytvoření a aktualizování dokumentace, komunikace se členy projektu.
Vývojář	Vývoj, implementace a oprava zadaných defektů, nastavení prostředí pro testování, komunikace se členy projektu.
Tester	Tvorba testovacího scénáře a skriptu, exekuce testovacího scénáře a skriptu, zaznamenávání nalezených defektů, přetestování opravených defektů, tvorba testovacích dat, komunikace se členy projektu.

Tabulka 2: Role a odpovědnosti v projektu [vlastní zpracování autora]

Záznam „Tester“ je zvýrazněn, jelikož činnosti této role jsou podstatné pro zpracování cílů diplomové práce.

4.3.6 Strategie testování

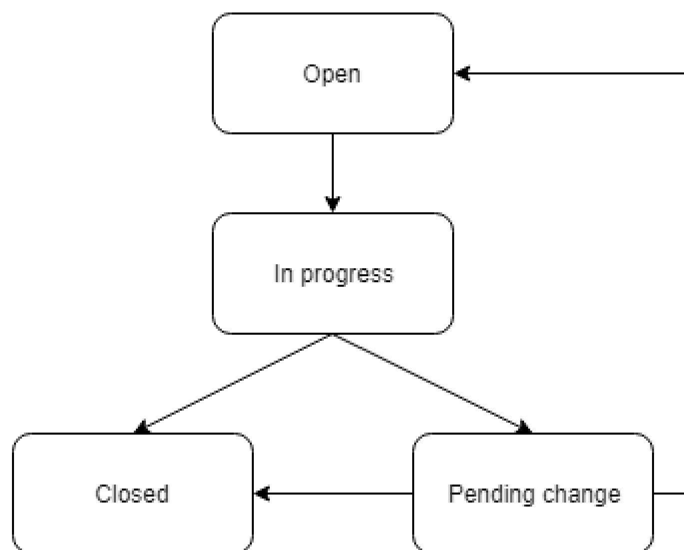
Výchozím bodem pro návrh, tvorbu a exekuci testovacích scénářů a skriptů jsou dokumentace v kapitole 4.4 rozvíjející případy užití, pomocí nichž se autor práce může seznámit s aplikací jako takovou, či jejími jednotlivými funkcionalitami. Manuální testovací scénář bude označen jedním z následujících stavů:

- Passed – úspěšný test
- Failed – neúspěšný test
- Blocked – blokovaný test
- N/A – test není možné provést

Automatizovaný testovací skript bude nabývat pouze stavy „passed“ a „failed“. Případné nalezené defekty budou zaznamenávány ve volně dostupné bug-trackingové aplikaci s názvem „bugzilla“, kterou využívá vývojářský tým pracující na aplikaci a bude se řídit životním cyklem defektu obsahujícím následující stavy:

- Open – otevřený defekt
- In progress – defekt v řešení
- Pending change – defekt připravený k retestu
- Closed – uzavřený defekt

Propojení jednotlivých stavů defektu je zobrazeno na obrázku níže:



Obrázek 8: Životní cyklus defektu pro testovanou aplikaci [vlastní zpracování autora]

Nově založený defekt se dostává do stavu „Open“ jakožto otevřený defekt. Po přidělení vývojářského týmu a počátku jeho řešení, se defekt dostává do stavu „In progres“. Ve chvíli, kdy je defekt vyřešen, přesouvá se do stavu „Pending change“. Znamená to, že je defekt připraven k přetestování testerem. V případě, že chování defektu je shodné s očekávaným chováním, pak může být defekt uzavřen. Pokud však chování defektu není korektní, defekt se vrací zpět do otevřeného stavu, kdy celý proces prochází znovu.

Zaznamenané defekty je třeba dále sumarizovat dle priority, jejichž stupeň ovlivňuje rychlost opravy. Stupně priority defektu jsou zobrazeny v následující tabulce:

Stupeň	Priorita	Význam
1	Vysoká	Kritický defekt systému zamezující v činnosti testování či vývoje, nutná okamžitá náprava.
2	Střední	Částečné či minimální dopady na proces testování, oprava v rámci běžného pořadí.
3	Nízká	Bez dopadu na proces testování, kosmetické chyby, oprava po vyřešení defektů vyšší priority.

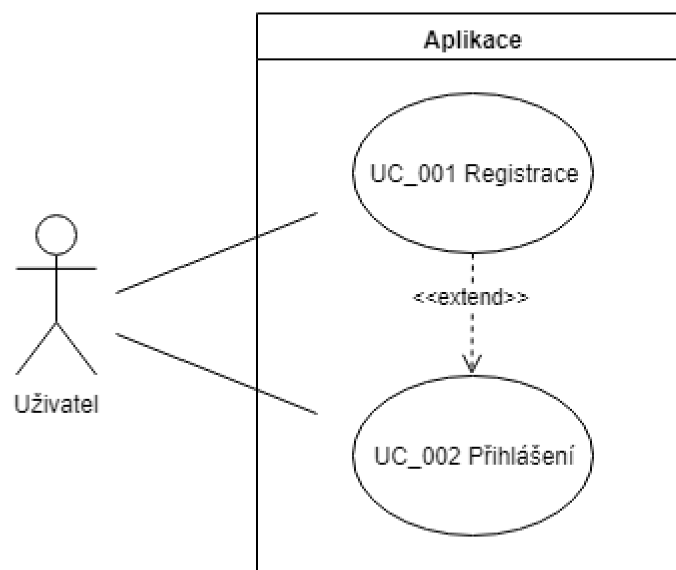
Tabulka 3: Priorita defektů [vlastní zpracování autora]

4.4 Případy užití

Kapitola „Případy užití" představuje dokumentaci, z níž autor vychází pro návrh, tvorbu a exekuci testovacích scénářů a testovacích skriptů. Obsahem jsou především diagramy (use case diagram, diagram aktivit), validační pravidla vstupních hodnot a chybové zprávy, které systém zobrazuje uživateli.

4.4.1 Use case diagram

Na obrázku číslo 9 je zobrazen use case diagram (diagram případu užití), který představuje chování systému z pohledu uživatele. Zaměřuje se pouze na testované funkcionality, tedy registraci a přihlášení. Za předpokladu zpracování celého systému by bylo vytvořeno o mnoho více diagramů s vyšší složitostí. Účelem diagramu je charakterizovat očekávané funkcionality systému, které budou testovány. Slouží jako podklad pro odhad rozsahu. Diagram není zaměřen na vnitřní logiku systému, tzn. že popisuje pouze to, co má systém umožňovat, nikoliv jak toho bude docíleno.

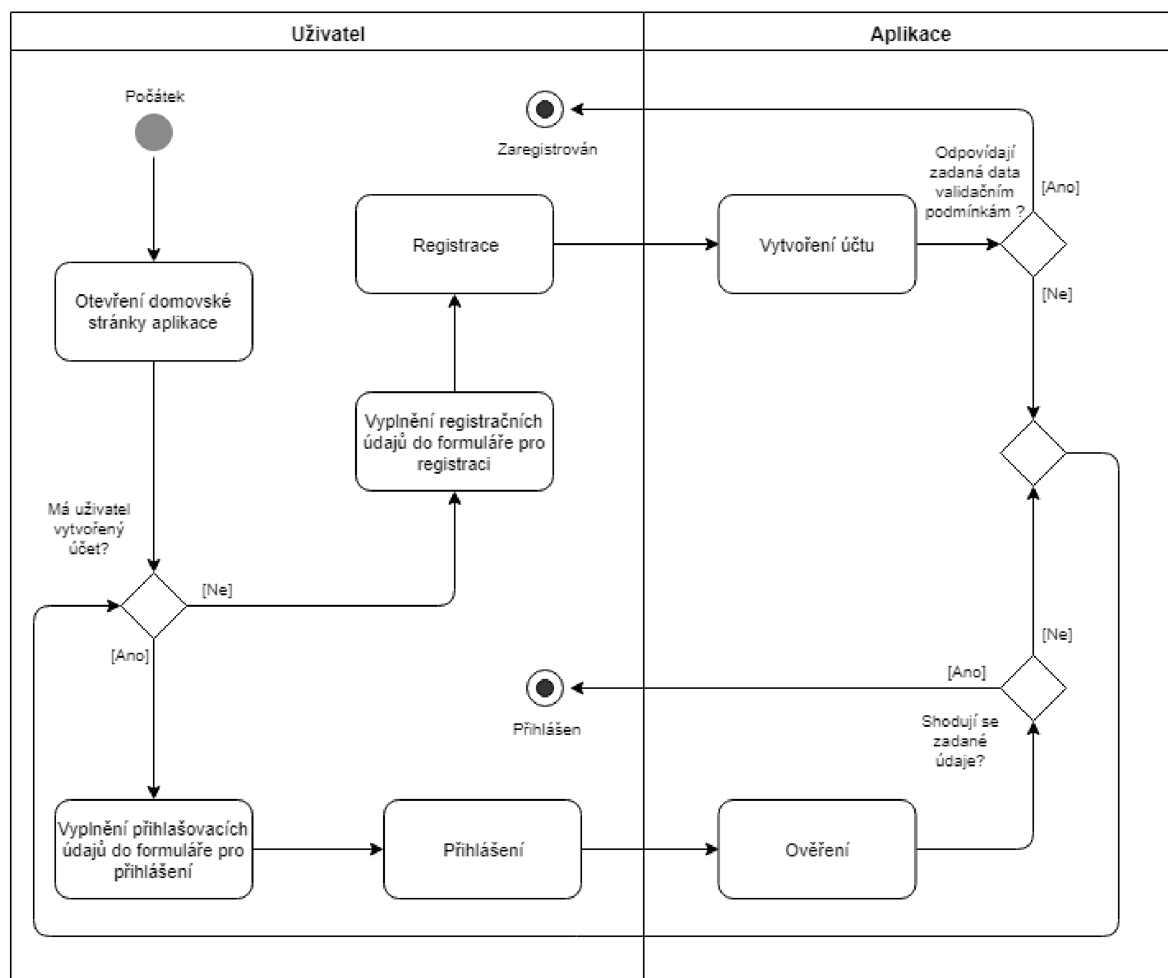


Obrázek 9: Use case diagram [vlastní zpracování autora]

V diagramu vystupuje takzvaný „účastník“ jímž je uživatel, který přistupuje k systému. Uživateli, jenž má přístup k aplikaci, se nabízí možnost případu užití registrace označené jako „UC_001 Registrace“ a případu užití přihlášení označené jako „UC_002 Přihlášení“. Mezi případem užití registrace a případem užití přihlášení je vztah „extend“, který značí rozšíření jednoho případu užití jiným.

4.4.2 Diagram aktivit

Diagram aktivit má za úkol charakterizovat chování systému, především procedurální logiky, procesů a workflow[3]. Na obrázku níže se nachází diagram aktivit, který popisuje daný sled aktivit vedoucích k procesům přihlášení a registrace.



Obrázek 10: Diagram aktivit [vlastní zpracování autora]

Uživatel si v prohlížeči otevře domovskou stránku aplikace Animal breath, na které jsou zobrazeny 2 formuláře. Jeden formulář je pro přihlášení, druhý formulář pro registraci. Jestliže má uživatel již vytvořený účet, pak do formuláře pro přihlášení zadá

[3] Workflow – schéma provádění komplexnější činnosti rozepsané na jednodušší činnosti a jejich vazby

přihlašovací údaje a zvolí přihlášení. Systém ověří, zda se zadané údaje uživatelem shodují s údaji v databázi. Za předpokladu, že údaje jsou shodné, pak je uživatel přihlášen do systému. Pokud uživatel nemá vytvořený účet, vyplní registrační formulář a odešle. Systém vytvoří účet na základě kontroly zadaných dat. V případě, že jsou zadaná data korektní, jinými slovy odpovídají validačním podmínkám, pak je uživatel úspěšně zaregistrovaný.

4.4.3 Validační pravidla

Validační pravidla jsou nesmírně důležitá, především pro funkcionalitu registrace. Tvorba nových uživatelských účtů, které by nepodléhali validačním podmínkám, by mohla způsobit nekonzistentnost databáze, včetně uložení nekorektních dat. Validační pravidla jsou omezení uživatele systémem na jeho vstupní data. V tabulce č. 4 jsou zpracována validační pravidla webové aplikace.

Atribut	Validační pravidlo
Jméno	Alespoň 1 znak
Příjmení	Alespoň 1 znak
Email	Emailová adresa ve tvaru "jmeno_emailu@server.cz"
Telefon	Alespoň 9 numerických znaků s výjimkou znaku "+" v předvolbě
Heslo	Alespoň 6 znaků
Heslo pro kontrolu	Shodné s řetězcem znaků v atributu "Heslo"

Tabulka 4: Validační pravidla v registračním formuláři [vlastní zpracování]

V registračním formuláři se dále vyplňuje pohlaví, avšak ve formě combo boxu, tudíž hodnota zadaná v tomto atributu nepodléhá žádnému validačnímu pravidlu.

4.4.4 Chybové zprávy

Chybové zprávy částečně vycházejí z validačních pravidel. Jedná se o zprávy, které jsou zobrazeny systémem uživateli. Zprávy se mohou zobrazit v různých případech, nejčastěji však ve chvíli, kdy uživatel zadá hodnoty atributů neodpovídajícím validačním

pravidlům. Takové zprávy jsou označovány jako negativní. Příklad pozitivní zprávy je zpráva o úspěšné registraci.

V tabulce č. 5 jsou zpracovány zprávy, jejich identifikátor a obsah, které může webová aplikaci při procesu registrace a přihlášení zobrazit uživateli.

ID zprávy	Text zprávy
MSG001	Jméno je povinné
MSG002	Příjmení je povinné
MSG003	Email je povinný
MSG004	Email je neplatný
MSG005	Telefon je neplatný
MSG006	Heslo je povinné
MSG007	Heslo musí obsahovat minimálně 6 znaků
MSG008	Kontrolní heslo je povinné
MSG009	Hesla se neshodují
MSG010	Přihlášení nebylo úspěšné
MSG011	Email je již zabraný
MSG012	Registrace byla úspěšná. Na váš email byl zaslán odkaz k ověření vlastnictví vaší emailové schránky.
MSG013	Verifikace emailu byla úspěšně dokončena. Můžete se přihlásit.

Tabulka 5: Chybové zprávy [vlastní zpracování autora]

4.5 Specifikace případů užití

V následujících podkapitolách budou blíže specifikované jednotlivé případy užití, které budou v rámci diplomové práce testovány. Jedná se tedy o funkcionality „UC_001 Registrace“ a „UC_002 Přihlášení“. Specifikace případů užití vychází především z use case diagramu, diagramu aktivit, validačních podmínek a chybových zpráv. Na jejich základě pak bude proveden návrh testovacích scénářů a testovacích skriptů. Pro případy užití registrace a přihlášení budou vytvořeny pouze základní scénáře (tzv. sunny day testy), v praxi by byly vytvořeny také alternativní scénáře.

4.5.1 Analýza případu užití pro registraci

Název

UC_001 Registrace

Základní popis

Případ užití registrace má za cíl zaregistrovat uživatele do systému.

Aktéři

- Uživatel
- Systém

Vstupní podmínky

- Uživatel se nachází na domovské stránce aplikace
- Uživatel není na svůj email dosud zaregistrován do systému

Výstupní podmínky

- Uživatel je zaregistrován (má vytvořený účet)

Základní scénář

1. Systém zobrazí uživateli formulář pro registraci
2. Uživatel vyplní všechna povinná pole validními daty, nepovinná pole vyplní libovolně
3. Systém provede validaci vstupních dat
4. Uživatel odešle registrační formulář
5. Systém zobrazí MSG012
6. Uživatel potvrdí svůj účet otevřením odkazu v příchozím emailu
7. Uživatel je přesměrován zpět do systému a je zobrazena MSG013

4.5.2 Analýza případu užití pro přihlášení

Název

UC_002 Přihlášení

Základní popis

Případ užití přihlášení má za cíl přihlásit uživatele do systému.

Aktéři

- Uživatel
- Systém

Vstupní podmínky

- Uživatel se nachází na domovské stránce aplikace
- Uživatel je na svůj email již zaregistrován do systému (dokončený UC_001 Registrace)
- Uživatel zná své přihlašovací údaje
- Systém není zaklopen či jiným způsobem omezen

Výstupní podmínky

- Uživatel je přihlášen do systému

Základní scénář

1. Systém zobrazí uživateli formulář pro přihlášení
2. Uživatel vyplní všechna povinná pole (email a heslo) validními daty
3. Systém provede validaci vstupních dat
4. Uživatel odešle požadavek k přihlášení do systému
5. Systém ověří shodu zadaných hodnot s hodnotami uloženými v databázi (autentizace)
6. Systém přihlásí uživatele do systému

4.6 Testovací data

Jak lze logicky odvodit, testovací data slouží jako prostředek k otestování daného softwaru, či jeho části. Data by měla být vytvořena pečlivě a korektně, jelikož mají značný vliv na efektivitu testovacího procesu. Určují počáteční stav systému, při kterém se test vykonává. Testovací data jsou podstatná mimo jiné i z několika dalších důvodů. Jedním z nich je úspora času na zavedení systému do stavu, ve kterém začíná testovací scénář. Dalším důvodem je zachování stejného stavu pro pozdější přetestování opraveného defektu.

V ideálním případě jsou data vytvářena před zahájením testování test analytiky a testery. Evidence těchto dat je buď sdílena prostřednictvím různých nástrojů (např. MS Excel) napříč projektem, či individuální pro daného testera. [4]

4.6.1 Způsoby vytvoření testovacích dat

Nezávisle na způsobu vytvoření testovacích dat je vždy vhodné vycházet ze sběru požadavků. Jinými slovy je potřeba znát specifikaci dat, které mají být vytvořeny. Požadavky na testovací data lze získat z následujících zdrojů:

- Požadavky z testovacích scénářů
- Specifické potřeby business testerů pro UAT
- Informace o distribuci dat v produkční verzi systému

Samotné vytvoření testovacích dat je pak možné provést několika různými způsoby. Mezi ty úplně nejčastější patří:

Ruční typování dat

Tester vytváří manuálně testovací data prostřednictvím konkrétních systémů. Výhodou tohoto způsobu je, že není nutná žádná zvláštní infrastruktura a výchozí investice. Vytváření testovacích dat manuálním způsobem je navíc často nekomplikované, tudíž tuto činnost může provádět i nekvalifikovaný pracovník, který byl pouze zaškolen.

Ruční typování dat je vhodné především v situaci, kdy není vyžadována jeho dlouhodobá opakovatelnost. Právě potřeba opakovaného zakládání těchto dat může vyústit v nevýhodu tohoto způsobu, jelikož oproti ostatním způsobům vytváření dat se může stát poměrně drahou variantou. Další nevýhodou je nutnost funkční testovací aplikace na testovacím prostředí. [4]

Automatická výroba dat

Ruční typování dat může být nahrazeno jiným způsobem, a to automatickou výrobou dat. Podmínkou pro tento způsob je konzistentní definice, tedy přesné hodnoty samostatných položek datových entit. Způsobů, jak automatizovat výrobu testovacích dat, je několik. Mezi nejčastější však patří:

- Automatizovaný front-end test
- SQL skript na úrovni databáze

Obrovskou výhodou tohoto způsobu je především samotná automatizace, která v případě, že je dobře zpracovaná, nese mizivé riziko chybovosti dílčích dat. Nevýhoda pak nastává, jeli zapotřebí vytvořit složitá data. Taková data pak automatická výroba nezvládá, či nese riziko vysoké chybovosti. Stejný problém nastává při požadavku na variabilitu dat. Vytvářet variabilní data za pomoci automatizace je nadměrně obtížný proces, jenž nemusí vždy skončit úspěšně. [4]

Kopie produkčních dat beze změny dat

Způsob vytvoření testovacích dat prostřednictvím zkopírování celé produkční databáze do testovacího prostředí je dalším ze způsobů, které lze v praxi využít. V rámci testování jsou pořízená data nejlepší. Data jsou konzistentní, variabilní, kvalitní, a především rychle dostupná. Avšak z pohledu bezpečnosti je tento způsob velice rizikový, poněvadž hrozí únik citlivých dat, jelikož data, na kterých se testuje, jsou převzaté z produkčního prostředí. Často dochází také k problémům, jako je nedostatečná kapacita databázového serveru na testovacím prostředí, či přepsání původních testovacích dat vytvořených testery. [4]

Částečná kopie produkčních dat

Způsob částečné kopie produkčních dat lze definovat jako upravený způsob kopie produkčních dat beze změny dat, jelikož se provádí kopie pouze selektované části produkční databáze. Tím se odstraní problémy s rozdílnou kapacitou databázových serverů mezi testovacím a produkčním prostředí. Riziko úniku citlivých dat zůstává, avšak jeho rozsah je zmenšen pouze na selektovanou část kopírovaných dat. [4]

4.6.2 Tvorba testovacích dat

Vytvoření testovacích dat pro otestování vybraných funkcionalit testované aplikace bude provedeno v závislosti odpovídajícím potřebám. V potaz se bude brát způsob testování a typ testovacího scénáře.

Testovací data pro manuální testování

Pro manuální testování v rámci základního scénáře registrace budou testovací data vytvářena přímo samotnou exekucí testu v podobě ručního typování dat. Pro základní scénář přihlášení budou data vytvořena předem a také prostřednictvím ručního typování dat.

Testovací data pro automatizované testování

Pro automatizované testování základního scénáře registrace budou testovací data vytvořena v rámci exekuce automatizovaného testu. Je možné konstatovat, že pro tento případ bude zvolen způsob „automatická výroba dat“, jelikož se data vytvoří pomocí automatizovaného front-end testu. Pro základní scénář přihlášení budou data vytvořena ručním typováním. Ačkoliv by při testování základního scénáře pro přihlášení manuálním i automatický způsobem mohla být data vytvořena pomocí automatizace, je to pro budoucí průběh práce irelevantní, tudíž byla zvolena metoda ručního typování dat.

Testovací data

V následujících tabulkách jsou zobrazena testovací data potřebná pro otestování základních scénářů vybraných funkcionalit:

Jméno	Příjmení	Email	Heslo	Heslo pro kontrolu	Typ testu
Ab	Mtester	RTMan1@mailinator.com	Testovani123	Testovani123	Manuální
Ab	Atester	RTAut1@mailinator.com	Testovani123	Testovani123	Automatický

Tabulka 6: Testovací data pro proces registrace [vlastní zpracování autora]

Registrace obsahuje navíc pole „pohlaví“ a „telefonní číslo“, které jsou však nepovinné, tudíž nejsou zahrnuty v tabulce jako relevantní parametry. V rámci testu základního scénáře mohou být vyplněny libovolně.

Email	Heslo	Typ testu
LTMan1@mailinator.com	Testovani123	Manuální
LTAut1@mailinator.com	Testovani123	Automatický

Tabulka 7: Testovací data pro proces přihlášení [vlastní zpracování autora]

4.7 Manuální testování

Na základě analýz a specifikace případu užití je možné vytvořit samotné testovací případy. Jak již bylo výše zmíněno, testy jsou zaměřeny pouze na základní scénáře vybraných funkcionalit.

K procesu manuálního testování bude využit nástroj TestRail, který zprostředkovává správu životního cyklu aplikace. Nástroj byl vybrán na základě autorovo předešlé zkušenosti a možnosti využití jeho trial[4] verze. V rámci diplomové práce umožní nástroj TestRail do vytvořeného projektu přidat nové testovací případy a následně zajistit jejich spuštění a vyhodnocení. Nástroj TestRail je dostupný z portálu gurock.com a jedná se o třicetidenní zkušební verzi zdarma.

V případě zachycení nesprávného chování systému, budou defekty zadávány taktéž do nástroje TestRail.

[4] Trial – zkušební verze softwaru

4.7.1 Tvorba testovacího případu registrace

V nástroji TestRail je možné vytvořit nový testovací případ v záložce „Test Cases“. Pro každý nový testovací případ je nutné zadat název testu, prioritu, předpoklady, kroky a očekávané chování. Vytvořený testovací případ je zobrazen na obrázku níže. Tvorba testu trvala 8 minut a 28 vteřin.

T1 TC_001 Registrace – základní scénář

Type	Priority	Estimate	References
Other	High	None	None

Automation Type
None

Preconditions

- Uživatel se nachází na domovské stránce aplikace
- Email není v systému zaregistrovaný

Steps

1 Ověř, že se na domovské stránce aplikace zobrazuje formulář pro registraci.

Expected Result

Na domovské stránce aplikace se zobrazuje registrační formulář.

2 V registračním formuláři vyplň korektními daty povinné atributy: jméno, příjmení, email, heslo, heslo pro kontrolu. Nepovinné údaje vyplň libovolně či nevyplňuj.

Expected Result

Data je možné zadat. Systém provede validaci zadaných hodnot a nezobrazí žádnou chybovou hlášku.

3 Odešli vyplněný registrační formulář.

Expected Result

Systém zobrazí MSG012 a do emailové schránky přijde email s odkazem k ověření.

4 Ověř emailovou adresu otevřením odkazu v emailu.

Expected Result

Systém otevřel nové okno s aplikací a zobrazil MSG013.

Obrázek 11: Tvorba testovacího případu registrace v nástroji TestRail [vlastní zpracování autora]

4.7.2 Tvorba testovacího případu přihlášení

Stejný způsobem byl vytvořen testovací případ pro přihlášení. Doba vytvoření testu byla 5 minut a 16 vteřin. Testovací případ přihlášení je zobrazen na obrázku níže.

T2 TC_002 Přihlášení – základní scénář

Type	Priority	Estimate	References
Other	High	None	None

Automation Type
None

Preconditions

- Uživatel se nachází na domovské stránce aplikace
- Je identifikovaný uživatel se zaregistrovaným a ověřeným emailem

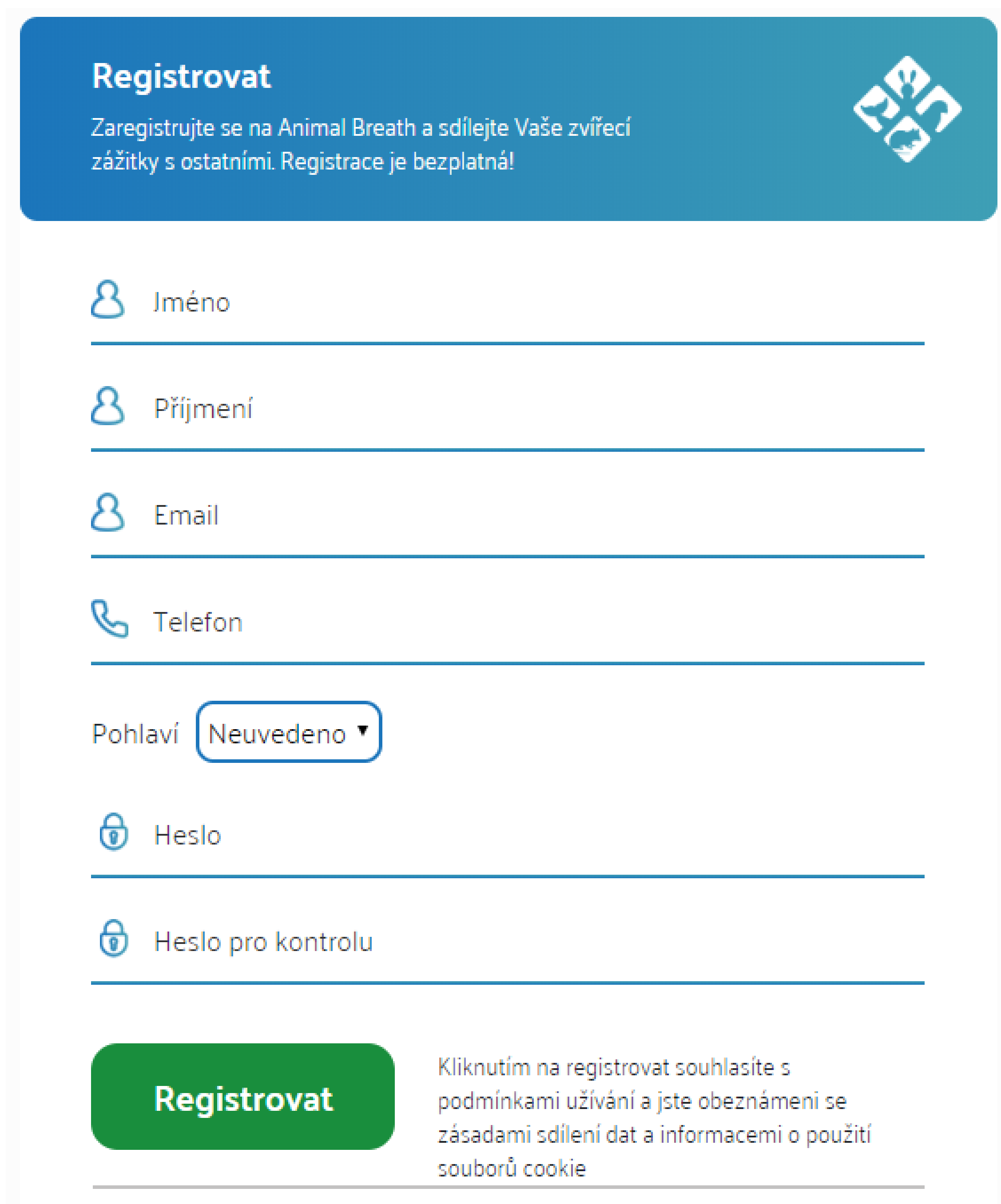
Steps

1. Ověř, že se v záhlaví domovské stránky aplikace zobrazuje formulář pro přihlášení.
   Expected Result
   Přihlašovací formulář se zobrazuje v záhlaví domovské stránky aplikace.
2. Ve formuláři pro přihlášení vyplň korektními daty povinné atributy: email, heslo.
   Expected Result
   Systém provede validaci zadaných hodnot a nezobrazí žádnou chybovou hlášku.
3. Odešli vyplněný přihlašovací formulář.
   Expected Result
   Systém přihlásil uživatele do systému.

Obrázek 12: Tvorba testovacího případu přihlášení v nástroji TestRail [vlastní zpracování autora]

4.7.3 Exekuce testovacího případu registrace

Základní scénář testovacího případu registrace představuje takový případ, kdy nenastává žádná jiná alternativa a cílem je dokončit úspěšně proces registrace. Na následujícím obrázku je zobrazen formulář určený pro registraci.



Obrázek 13: Registrační formulář [vlastní zpracování autora]

Předpokladem pro vykonání testu je přítomnost uživatele na domovské testovací stránce aplikace na adrese animalbreath.test.com. Emailová adresa, kterou uživatel vyplní do registračního formuláře musí být unikátní, nesmí být v systému již zaregistrována.

Úplně prvním krokem v testovacím scénáři je ověření existence registračního formuláře na domovské stránce aplikace, bez kterého by nebylo možné proces registrace vykonat.

Ve druhém kroku se vyplňují uživatelem všechny povinné atributy (jméno, příjmení, email, heslo, heslo pro kontrolu) a je systémem provedena validace daného pole po jeho opuštění. Formulář je vyplněn připravenými testovacími daty v tabulce č. 6. Jelikož vstupní hodnoty podléhají validačním pravidlům, není zobrazena u žádného pole chybová hláška. Vyplněný registrační formulář je na obrázku níže.

Obrázek 14: Vyplněný registrační formulář [vlastní zpracování autora]

Ve třetím kroku je registrační formulář uživatelem odeslán, systém zobrazí informační hlášku o úspěšné registraci (MSG012) a zašle na zadanou emailovou adresu odkaz, jímž se ověřuje vlastnictví emailové adresy.

Obsahem čtvrtého kroku je ověření emailové adresy a tím dokončení procesu registrace. Uživatel otevře svou emailovou schránku, kterou zadal do registračního formuláře a otevře příchozí email, ve kterém se nachází odkaz s názvem „Potvrdit svůj účet“. Po otevření výše zmíněného odkazu je na následujícím obrázku zobrazena hláška, která představuje úspěšný proces registrace, tedy naplnění očekávaného chování systému.

Verifikace emailu byla úspěšně dokončena. Můžete se přihlásit.

Obrázek 15: Dokončení procesu registrace [vlastní zpracování autora]

Proces registrace proběhl v pořádku. Uživateli se při vyplňování vstupních hodnot nezobrazila žádná hláška z chybových zpráv, ověřovací email přišel na zadanou adresu a úspěšně se vytvořil uživatelský účet. Testovací případ může být v nástroji TestRail označen stavem „Passed“.

Passed

2/18/2019 2:14 PM Martin C.
Edit

Elapsed
1m 59s

This test was marked as 'Passed'.

Steps

1. Ověř, že se na domovské stránce aplikace zobrazuje formulář pro registraci.
   Expected Result
   Na domovské stránce aplikace se zobrazuje registrační formulář.
2. V registračním formuláři vyplň korektními daty povinné atributy: jméno, příjmení, email, heslo, heslo pro kontrolu. Nepovinné údaje vyplň libovolně či nevyplňuj.
   Expected Result
   Data je možné zadat. Systém provede validaci zadaných hodnot a nezobrazí žádnou chybovou hlášku.
3. Odešli vyplněný registrační formulář.
   Expected Result
   Systém zobrazí MSG012 a do emailové schránky přijde email s odkazem k ověření.
4. Ověř emailovou adresu otevřením odkazu v emailu.
   Expected Result
   Systém otevřel nové okno s aplikací a zobrazil MSG013.

Obrázek 16: Vyhodnocení testu registrace v nástroji TestRail [vlastní zpracování autora]

Na obrázku č. 16 je podstatná informace pod názvem „Elapsed“ (v překladu do češtiny „uplynulý“), která reprezentuje čas exekuce. Testovací případ registrace byl

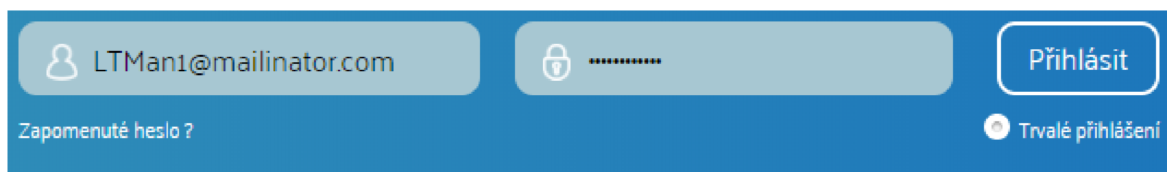
testován po dobu 1 minuty a 59 sekund. Doba exekuce testu může být ovlivněna rychlostí internetu, či opoždění potvrzovacího emailu.

4.7.4 Exekuce testovacího případu přihlášení

Předpokladem pro otestování případu přihlášení je přítomnost uživatele na domovské testovací stránce aplikace na adrese animalbreath.test.com. Uživatel, s nímž se exekuce testu bude provádět, je zaregistrován do systému a jeho emailová adresa byla ověřena.

Taktéž jako při testování registrace, je prvním krokem v testovacím scénáři ověření existence formuláře. Přihlašovací formulář se nachází v záhlaví domovské stránky aplikace na pravé straně.

Druhý krok testu je zaměřen na vyplnění přihlašovacího formuláře, jeho atributů, konkrétně emailové adresy a hesla. Formulář je vyplněn připravenými testovacími daty z tabulky č.7. Po vyplnění validních hodnot a opuštění daného pole je systémem provedena validace. Vyplněný přihlašovací formulář je zobrazen na obrázku níže.

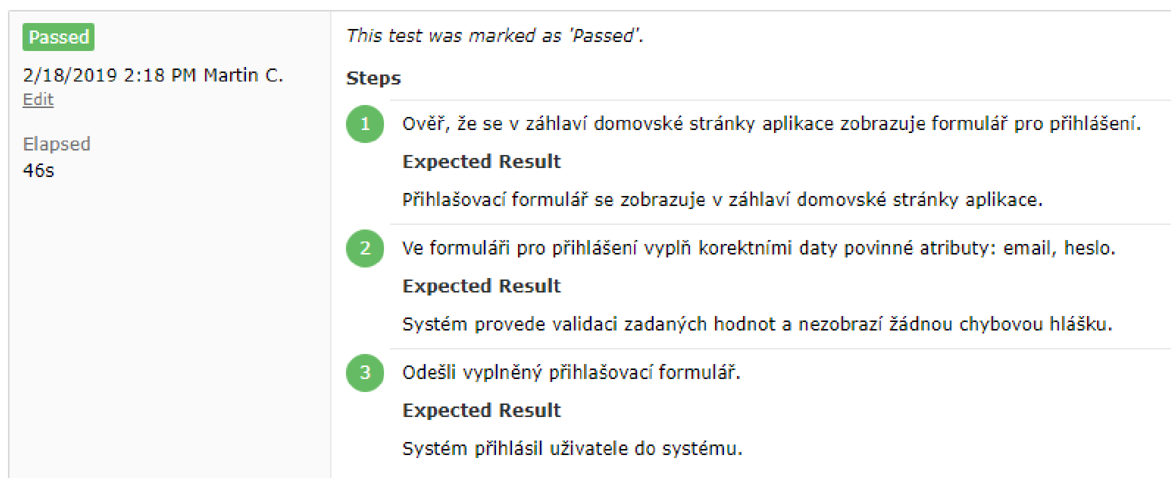


Obrázek 17: Vyplněný přihlašovací formulář [vlastní zpracování autora]

V rámci třetího kroku uživatel odešle vyplněný přihlašovací formulář, systém provede autentizaci a uživatel je přihlášen do systému. Hlášku o úspěšném přihlášení systém nezobrazuje, jelikož nebyla součástí požadavku zadavatele. Po přihlášení uživatele do systému je zobrazena „zed“ a jsou umožněny další funkcionality systému (viz obrázek níže).

Obrázek 18: Horní panel po přihlášení do systému [vlastní zpracování autora]

Jelikož proces přihlášení proběhl úspěšně, nezobrazila se žádná hláška z chybových zpráv a uživatel byl do systému přihlášen, je možné označit testovací případ v nástroji TestRail stavem „Passed“.



Obrázek 19: Vyhodnocení testu přihlášení v nástroji TestRail [vlastní zpracování autora]

Testovací případ přihlášení byl proveden v čase 46 sekund. Doba exekuce je kratší nežli při testování procesu registrace, jelikož samotný proces není časově náročný. Při testování přihlášení navíc není potřeba čekat na příchozí email a pracuje se jen a pouze s testovanou aplikací.

4.8 Automatizované testování

K vytvoření a exekuci automatizovaného testu byl využit nástroj Eclipse IDE dostupný na adrese www.eclipse.org/ide. Autor zvolil tento nástroj z důvodu předešlých zkušeností. Do vývojového prostředí Eclipse byly při vytváření projektu naimportovány knihovny pro podporu Selenium WebDriver pro jazyk Java. Knihovny jsou volně dostupné na adrese www.seleniumhq.org/download a jsou nezbytné pro tvorbu automatizovaného testu. Do projektu byl dále naimportován Google Chrome Driver, dostupný na stejné adrese jako využité knihovny, jelikož prohlížeč Google Chrome byl autorem zvolen výchozím prohlížečem pro exekuci testů. V praxi by bylo vhodné provést testování na všech nejvyužívanějších prohlížečích. V rámci knihoven je naimportován také framework JUnit, který slouží pro psaní testů v selenium.

Pro účel diplomové práce není podstatné, jakým způsoben je automatický test napsán. Pro každý test byla vytvořena jedna třída. Název třídy pro test registrace byl zvolen RegistrationUser, pro test přihlášení pak LoginUser. Každá třída pak obsahuje výše zmíněný import zobrazený jako text níže.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.junit.*;
```

Pro každý test byly přes anotace @BeforeClass a @AfterClass vytvořeny metody, jenž definují, co se má před zahájením a po dokončení testu stát. Samotný test se pak nachází v anotaci @Test. Oba testy také obsahují postupný výpis do konzole popisující aktuální činnost systému. Výpis do konzole je proveden prostřednictvím kódu níže. Veškeré využití výpisu do konzole v testu nebude nadále v práci komentováno.

```
System.out.println("výpis do konzole");
```

Metoda beforeClass() obsahuje inicializaci driveru a vytvoření její nové instance. Jelikož nová instance driveru není maximalizována, obsahuje metoda maximalizaci inicializovaného driveru.

```
@BeforeClass
public static void beforeClass() {
	System.out.println("Pouštím beforeClass: ");
	System.out.println("Inicializace chrome driveru: ");

	System.setProperty("webdriver.chrome.driver", "src\\chromedriver.exe");

	driver = new ChromeDriver();
	System.out.println("Driver inicializovan - OK");
	driver.manage().window().maximize();

}
```

Metoda afterClass() v testech obstarává ukončení instance driveru, poté co test proběhl do svého konce. Kód metody je k dispozici níže.

```
@AfterClass
	public static void afterClass() {
		System.out.println("Poustím afterClass: ");
		System.out.println("Zavírám chrome driver: ");

	 for(String handle : driver.getWindowHandles()) {
		driver.switchTo().window(handle);
		driver.close();

	  }
	}
```

4.8.1 Tvorba testovacího skriptu registrace

Jak bylo zmíněno výše, každý testovací případ je v samostatné třídě. Testovací skript pro registraci se nachází v metodě _RegisterUser_test v anotaci @Test.

```
@Test
	public void _RegisterUser_test() {
```

Na začátku celého skriptu je vhodné definovat textové řetězce, které se budou vyplňovat v registračním formuláři testované aplikace a na portálu mailinator.com.

Takovéto provedení je vhodné především pro případnou úpravu řetězců. Pokud by tester potřeboval změnit vstupní data, musel by je hledat v celém kódu. Takto se nacházejí na samém začátku skriptu.

```
String jmeno ="Ab";
String prijmeni ="Atester";
String email = "RTAut1@mailinator.com";
String heslo = "Testovani123";
```

V tuto chvíli je možné otevřít v testovacím okně testovací webovou aplikaci, na které se bude daný test provádět. K tomu poslouží následující metoda WebDriveru.

```
driver.get("https://www.animalbreath.test.com");
```

Po načtení domovské stránky aplikace je nutné vyplnit registrační formulář. Z kapitol 4.4 a 4.5 je známo, že se jedná o tyto povinné pole: jméno, příjmení, email, heslo, heslo pro kontrolu. K vyplnění těchto atributů je za potřebí vyselektovat jednotlivé elementy. To je provedeno prostřednictvím metody findElement(By);. Selektování se může provést různými způsoby. Ideální je selektovat dle identifikátoru, jelikož se jedná o unikátní hodnotu. Další možností je selektování dle třídy, to však může způsobit problémy, jelikož jednu třídu lze přiřadit vícero elementům. Selektování prostřednictvím Xpath je další přijatelnou metodou. Xpath je jazyk, který automaticky získá z dokumentu daný element, tudíž jeho použití je jednoduché, někdy však při tvorbě automatizovaného testu nemusí být funkční. Elementy polí v registračním formuláři obsahují identifikátor id. Ve chvíli, kdy jsou elementy vyselektovány, je možné jej naplnit testovacími daty z tabulky č. 6 pomocí metody sendKeys();. Zápis kódu je k dispozici níže.

```
WebElement regFNInput =
driver.findElement(By.cssSelector("#RegisterModel_FirstName"));
	regFNInput.sendKeys(jmeno);
	System.out.println("Jméno bylo vloženo - OK");

WebElement regLNInput =
driver.findElement(By.cssSelector("#RegisterModel_LastName"));
	regLNInput.sendKeys(prijmeni);
	System.out.println("Príjmení bylo vloženo - OK");
```

```
WebElement regEInput =
driver.findElement(By.cssSelector("#RegisterModel_Email"));
       regEInput.sendKeys(email);
       System.out.println("Email byl vložen - OK");

WebElement regPInput =
driver.findElement(By.cssSelector("#RegisterModel_Password"));
       regPInput.sendKeys(heslo);
       System.out.println("Heslo bylo zadáno - OK");

WebElement regPCInput =
driver.findElement(By.cssSelector("#RegisterModel_PasswordConfirmation"));
       regPCInput.sendKeys(heslo);
       System.out.println("Heslo pro kontrolu bylo zadáno - OK");
```

Pokud plnění polí daty funguje korektně, je potřeba formulář odeslat tlačítkem „Registrovat“. Je nevyhnutelné opět provést selekci elementu, tentokrát potvrzovacího tlačítka. Jelikož pro tlačítko neexistuje žádný jiný unikátní identifikátor, rozhodl se autor provést selekci pomocí Xpath. Na daný WebElement je následně volána metoda click();, která způsobí kliknutí na daný element. Kód je zobrazen níže.

```
WebElement regButton =
driver.findElement(By.xpath("/html/body/div[3]/div[4]/form/input"));
regButton.click();
```

Odeslání registračního formuláře zapříčiní příchod emailu. K testování byl využit portál mailinator.com, který umožňuje vytvářet emailové schránky na jedno použití. Nyní je nezbytné otevřít portál, do pole pro otevření emailu zadat hodnotu emailu použitou v registraci a potvrdit příslušným tlačítkem. Jestliže je kód v pořádku, otevře se emailová schránka. Před tímto procesem bylo přidáno opoždění systému z důvodu, že ověřovací email může dostát jistě prodlevy. Pokud by byla schránka otevřena před příchodem emailu, test by skončil neúspěšně. Uspání vlákna musí být opatřeno pomocí „try“ a „catch“.

```
try {
       Thread.sleep(7000);
}
catch (InterruptedException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

```
driver.get("https://www.mailinator.com");
System.out.println("portál mailinator.com byl otevřen");

WebElement mailLog = driver.findElement(By.cssSelector("#inboxfield"));
mailLog.sendKeys("RTAut1@mailinator.com");

WebElement mailButton =
driver.findElement(By.xpath("/html/body/section[1]/div/div[3]/div[2]/div[2]/div
[1]/span/button"));
mailButton.click();
```

Ve schránce se nachází příchozí email, který musí být rozkliknut. K selekci tohoto elementu bude opět využit Xpath.

```
WebElement mailOpen =
driver.findElement(By.xpath("//div[@class='ng-binding']/following::td[3]"));
mailOpen.click();
```

Po otevření příchozího emailu bylo zjištěno, že se driver nachází v jiném framu, nežli je potvrzovací odkaz, na který se musí kliknout. Přesměrování na potřebný frame vyřešil autor následovně.

```
driver.switchTo().frame("msg_body");
```

Poté je možné rozkliknout odkaz pro ověření emailové adresy. Pro správné chování musel autor přidat do skriptu další uspání systému včetně výjimek.

```
try {
	Thread.sleep(2000);
}
catch (InterruptedException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}

WebElement mailLinkOpen =
driver.findElement(By.xpath("//a[@href[contains(.,'mailinator')]]"));
mailLinkOpen.click();
System.out.println("Potvrzovací odkaz byl rozkliknut - OK");

try {
	Thread.sleep(1000);
```

```
}
catch (InterruptedException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Ve chvíli, kdy test doběhl, nastává další část testu, takzvaná ověřovací. Je nezbytné ověřit, že test proběhl úspěšně. K tomu se využívají příkazy assert. Možností, jak ověřit úspěšnost pomocí assertů je mnoho. Autor práce se v tomto případě rozhodl provést ověření na základě názvu záložky (title). Nejprve je definován očekávaný název záložky po dokončení registrace a ta je poté porovnána se skutečným názvem. Vše je k dispozici níže.

```
String expectedTitle = "EmailVerification - Animal Breath";
String abHandle = driver.getWindowHandles().toArray()[1].toString();
driver.switchTo().window(abHandle);

try {
	Thread.sleep(2000);
}
catch (InterruptedException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}

String actualTitle = driver.getTitle();
Assert.assertEquals(actualTitle, expectedTitle);

}
```

Doba tvorby automatizovaného skriptu pro proces registrace byla 51 minut a 45 vteřin.

4.8.2 Tvorba testovacího skriptu přihlášení

Testovací skript pro přihlášení se nachází ve třídě LoginUser_test. Také je zde využito metod beforeClass() a afterClass(). V samotné anotaci @Test se pak nachází celý test pro přihlášení.

```
@Test
	public void _LoginUser_test() {
```

Na úplném začátku testu je opět vhodné definovat textové řetězce, které budou zadávány do textových polí ve formuláři pro přihlášení a otevřít testovací stránku aplikace. Pro zajištění správného procesu je nastavené opoždění na 1 vteřinu.

```
String email = "LTAut1@mailinator.com";
String heslo = "Testovani123";

driver.get("https://www.animalbreath.test.com");
System.out.println("Stranka AnimalBreath.test.com byla uspesne otevrena");
```

Po otevření domovské stránky aplikace je před zadáním hodnot do povinných polí pro přihlášení vhodné počkat několik sekund z důvodu korektního načtení stránky. Autorovi práce bez opoždění testovací skript nefungoval. Opoždění bylo nastaveno na 1000 ms (1 vteřina).

```
try {
	Thread.sleep(1000);
}
catch (InterruptedException e1) {
	// TODO Auto-generated catch block
	e1.printStackTrace();
}
```

Ve chvíli, kdy je otevřená domovská stránka aplikace a dojde ke vteřinovému opoždění, lze zadat do přihlašovacích polí (email a heslo) připravená testovací data. K identifikaci pole pro zadání emailu a hesla poslouží WebElement s jeho metodou findElement(By). Inputu pro email je přiřazen identifikátor „UserName“. Ve chvíli, kdy je prvek identifikovaný, vyplní se testovacími daty z tabulky č. 7 prostřednictvím metody sendKeys();. K dispozici v bloku kódu níže.

```
WebElement unInput = driver.findElement(By.cssSelector("#UserName"));
unInput.sendKeys(email);
System.out.println("Email byl úspešne zadán");
```

Stejný postup byl proveden také pro pole heslo.

```
WebElement psInput = driver.findElement(By.cssSelector("#Password"));
psInput.sendKeys(heslo);
System.out.println("heslo bylo úspešne zadáno");
```

Po vyplnění přihlašovacího formuláře je zapotřebí formulář odeslat tlačítkem „Přihlásit“. Jelikož tlačítko nemá identifikátor „id“, měl autor možnost selektovat element prostřednictvím třídy či xpath. Vzhledem k tomu, že třída „transparent-rbw“ byla použita jen v daném elementu, bylo použito selektování na základě třídy. Aktivace prvku se následně provede prostřednictvím metody click();.

```
WebElement loginBtn = driver.findElement(By.cssSelector(".transparent-rbw"));
loginBtn.click();
System.out.println("Uživatel byl prihlášen do systému");
```

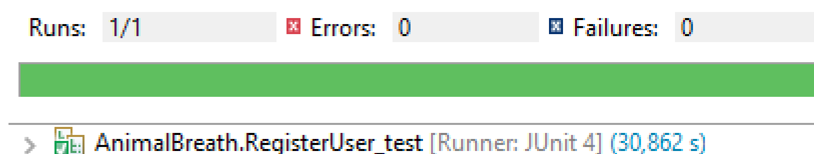
Po přihlášení je vhodné opět počkat několik vteřin pro korektní načtení stránky. Následně je nutné provést ověření prostřednictvím assertu. V tomto případě byl vyselektován element na hledání uživatelů v systému, jenž se zobrazuje po přihlášení. Jestliže systém tento prvek zobrazoval, pak pomocí assertu bylo ověřeno, že test proběhl úspěšně.

```
Assert.assertEquals(driver.findElement(By.xpath("//div/form[@action='/Home/Sear
ch']")).isDisplayed(), true);
}
```

Autor práce měřil čas tvorby výše uvedeného skriptu nehledě na vytvoření projektu. Čas tvorby skriptu pro přihlášení do aplikace byl 23 minut a 9 vteřin.

4.8.3 Exekuce testovacího skriptu registrace

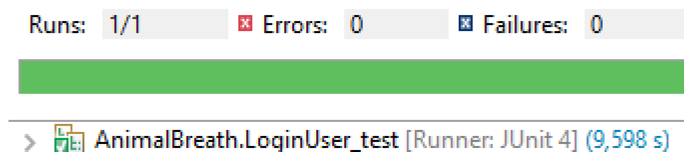
Čas exekuce automatizovaného testu je přímo dostupný z vývojářského studia Eclipse prostřednictvím již zmíněného frameworku JUnit. V levé části studia Eclipse se nachází záložka JUnit, která obsahuje základní informace o exekuovaném testu. Tyto informace jsou pro účel práce dostačující. Jedná se o stav, jakého exekuovaný test nabývá a čas samotné exekuce testu. Doba exekuce automatizovaného testu pro proces registrace byla 30,862 vteřin. Autor hodnotu zaokrouhlil na 31 vteřin.



Obrázek 20: Výsledek exekuce testu pro registraci [vlastní zpracování autora]

4.8.4 Exekuce testovacího skriptu přihlášení

Automatizovaný test pro proces přihlášení byl napsán a spouštěn v totožném vývojářském studiu, tudíž jeho výsledek lze získat stejným způsobem jako u procesu registrace. Doba exekuce automatizovaného testu pro proces přihlášení byla 9,598 vteřin. Autor práce se rozhodl zaokrouhlit čas na celé číslo, tudíž nadále je uváděn výsledek 10 vteřin.



Obrázek 21: Výsledek exekuce testu pro přihlášení [vlastní zpracování autora]

4.9 Výsledek časové náročnosti

Pro účel práce byly zvoleny dva testy, které se v rámci základních scénářů liší svou náročností. Zatím co při procesu registrace je potřeba vyplnit větší množství polí a po potvrzení formuláře musí uživatel potvrdit svůj účet přes email, proces přihlášení vyžaduje pouze vyplnění dvou polí, odeslání formuláře a ověření přihlášení. Náročnost pro tvorbu a exekuci těchto dvou testů se pak logicky liší. Porovnání časové náročnosti mezi manuálním a automatickým testováním bylo pozorováno pro oba testy a výsledné hodnoty jsou k dispozici v tabulkách 8 a 9. Hodnoty jsou uvedeny ve vteřinách.

	Tvorba testu	Exekuce testu	Celkový čas
Manuální testování	508	119	627
Automatizované testování	3105	31	3136

Tabulka 8: Časová náročnost testu pro registrace [vlastní zpracování autora]

	Tvorba testu	Exekuce testu	Celkový čas
Manuální testování	316	46	362
Automatizované testování	1389	10	1399

Tabulka 9: Časová náročnost testu pro přihlášení [vlastní zpracování autora]

Není však vyloučené, že časové hodnoty zobrazené v tabulkách výše odpovídají pouze jednomu vzorku, tedy autoru práce. K tvorbě a exekuci manuálního a automatizovaného testu je zapotřebí lidský faktor, který je unikátní (doba exekuce automatizovaného testu může být ovlivněna způsobem jeho vytvoření). Autor práce získal časové vzorky od kolegů z oboru formou přímé konfrontace, kdy sledoval a měřil rychlost jejich tvorby a exekuce testů. V tabulkách níže jsou k dispozici mimo autorových výsledků také výsledky čtyř participantů. Dohromady se jedná o 5 vzorků a autor bude nadále pracovat s jejich průměrnou hodnotou. Hodnoty jsou uvedeny ve vteřinách.

Testovací subjekt	Návrh - Registrace	Návrh - Přihlášení	Exekuce - Registrace	Exekuce - Přihlášení
Autor	508	316	119	46
Tester 1	445	283	106	38
Tester 2	654	350	143	62
Tester 3	554	385	113	52
Tester 4	518	362	125	35
Průměr	536	339	121	47

Tabulka 10: Časové výsledky všech participantů pro manuální testování [vlastní zpracování autora]

Testovací subjekt	Návrh - Registrace	Návrh - Přihlášení	Exekuce - Registrace	Exekuce - Přihlášení
Autor	3105	1389	31	10
Tester 1	2518	1052	25	8
Tester 2	2338	1281	38	14
Tester 3	2782	1205	22	11
Tester 4	2831	1565	28	7
Průměr	2715	1298	29	10

Tabulka 11: Časové výsledky všech participantů pro automatické testování [vlastní zpracování autora]

Z výsledných tabulek lze konstatovat, že časová náročnost tvorby manuálního testu je nižší nežli časová náročnost tvorby automatizovaného testu. Naopak exekuce manuálního testu oproti automatizovanému je časově náročnější. Tento výsledek byl autorem predikovatelný. V kapitole č. 5 (Výsledky a diskuze) bude autor hledat x-tou exekuci testu, při které se stane efektivnější automatizovaný test z hlediska časové a finanční náročnosti.

5 Výsledky a diskuse

Je obecně známo, že manuální testování je vhodné pro testy, jejichž iterace testování není příliš vysoká. Za předpokladu, že bude test spuštěn pouze jednou či dvakrát, není vhodné vytvářet automatický test, jehož vytvoření může být náročnější nežli vytvoření manuálního scénáře. Naopak v případě regresních testů či smoke testů, jejichž iterace testování bývá vysoká, je využití automatických testů praktičtější. I přes tento fakt, je v mnoha případech pro tento typ testů stále využíváno manuálního testování. Diplomová práce dále řeší vhodnost použití daného typu testování na základě doby tvorby a exekuce testu. Autor na základě časové a finanční náročnosti testu hledá optimální počet exekucí testu, při kterém se stává efektivnější variantou automatizované testování pro oba testovací případy. Výzkum je prováděn na testech dvou rozdílně náročných základních scénářů, u kterých se předpokládá vyšší iterace exekuce.

5.1 Časová analýza náročnosti testu

Následující kapitola řeší časovou analýzu náročnosti testu. Jinými slovy, autor práce hledá počet exekucí, které musí být vykonány, aby se z časového hlediska stal automatizovaný test efektivnějším nežli manuální. Veškeré hodnoty jsou uvedeny ve vteřinách a jsou dosazeny z tabulek č. 10 a č. 11.

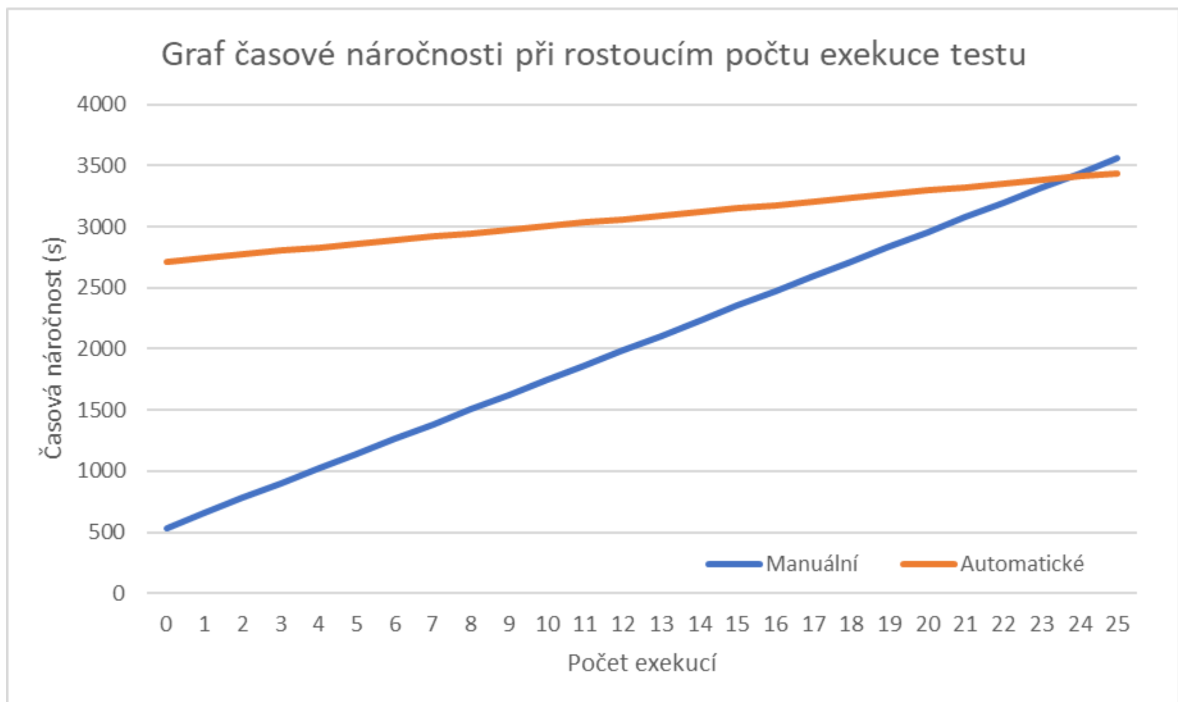
5.1.1 Registrace

Průměrný čas, během kterého byl vytvořen manuální test pro registraci, je 536 vteřin. Exekuce manuálního testu průměrně trvá 121 vteřin. Automatizace onoho testu v průměru zabere 2715 vteřin s exekucí o průměrném čas 29 vteřin. Časová analýza náročnosti testu hledá bod, v kterém se časová náročnost manuálního testu rovná časové náročnosti automatizovaného testu. Nalezení tohoto bodu usnadní tabulka níže, kde je nejdříve uveden čas návrhu, a k němu dále přičítány jednotlivé časy exekucí.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	536	121	121	121	121	121	121
Automatický	2715	29	29	29	29	29	29
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	121	121	121	121	121	121	121
Automatický	29	29	29	29	29	29	29
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce
Manuální	121	121	121	121	121	121	121
Automatický	29	29	29	29	29	29	29
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	Součet		
Manuální	121	121	121	121	3440		
Automatický	29	29	29	29	3411		

Tabulka 12: Nalezení efektivity z časového hlediska pro případ registrace [vlastní zpracování autora]

Součet návrhu a 24 exekucí vyjadřuje situaci, kdy časová náročnost manuálního testu převyšuje časovou náročnost testu automatizovaného. V případě, že by bylo na projektu zamýšleno provádět test registrace čtyřiadvacetkrát a více v období jednoho releasu, z časového hlediska by bylo vhodné, využít automatizovaného testování. K nalezení přesného bodu, kdy se automatizovaný test stává efektivnějším nežli manuální, poslouží následující graf.



Graf 1: Časová náročnost při rostoucím počtu exekuce testu pro registraci [vlastní zpracování autora]

Do grafu byly vloženy přímky manuálního a automatizovaného testování. Na vodorovné ose se nachází počet exekucí daného testu a na ose svislé časová náročnost testu ve vteřinách. Vstupní data přímek jsou pak jednotlivé mezisoučty v daném bodě představující počet exekucí, přičemž nulová hodnota u počtu exekucí představuje počáteční hodnotu, tedy čas, za který byl test vytvořen. Porovnáním rovnic přímek manuálního a automatizovaného testování lze získat přesnou hodnotu exekuce, při níž se časová náročnost manuálního testu rovná časové náročnosti automatizovaného testu. Rovnice přímky manuálního testu je $y = 536 + 121x$ a rovnice přímky automatizovaného testu je $y = 2715 + 29x$. Výsledná hodnota, při níž je časová náročnost obou způsobů testování ekvivalentní, je ***23,6848***. Hodnota byla zaokrouhlena na 4 desetinná místa.

5.1.2 Přihlášení

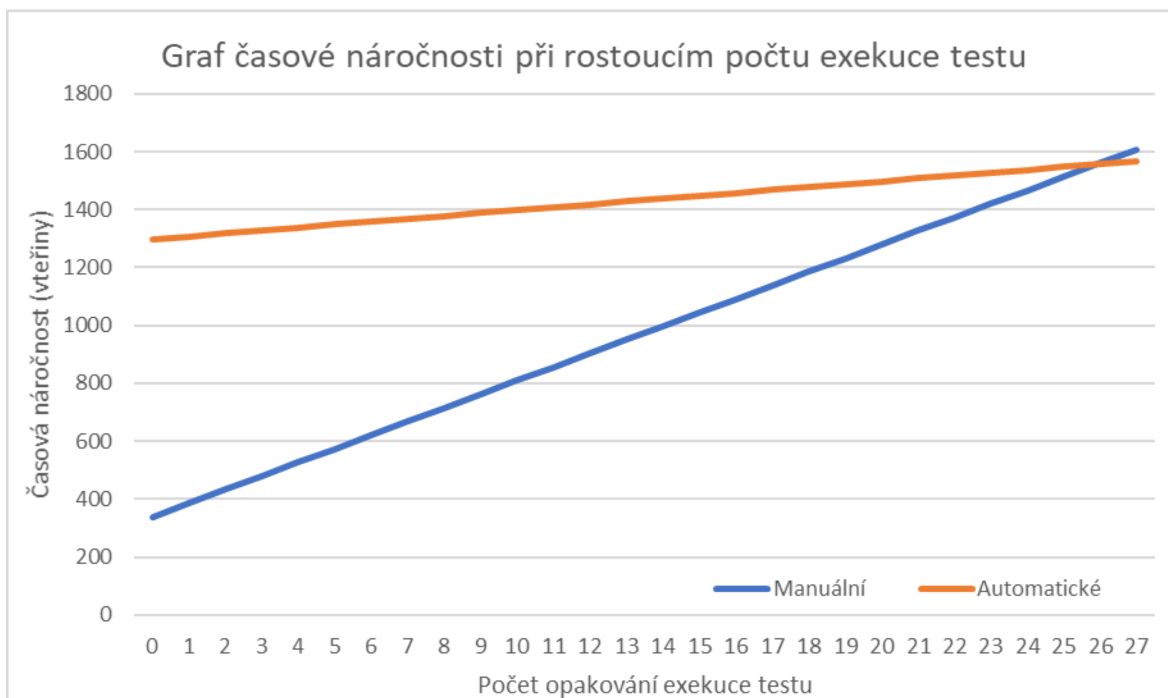
Průměrný čas návrhu manuálního testu pro přihlášení je 339 vteřin s průměrnou časovou délkou exekuce 47 vteřin. Návrh automatizovaného testu průměrně trval 1298

vteřin s průměrnou dobou exekuce 10 vteřin. Nejdříve je uveden čas návrhu, poté je k němu přičítán čas exekuce do doby, než se časy vyrovnají.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	339	47	47	47	47	47	47
Automatický	1298	10	10	10	10	10	10
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	47	47	47	47	47	47	47
Automatický	10	10	10	10	10	10	10
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce
Manuální	47	47	47	47	47	47	47
Automatický	10	10	10	10	10	10	10
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	25. exekuce	26. exekuce	Součet
Manuální	47	47	47	47	47	47	1561
Automatický	10	10	10	10	10	10	1558

Tabulka 13: Nalezení efektivity z časového hlediska pro případ přihlášení [vlastní zpracování autora]

Z tabulky výše lze vyčíst, že při 26. exekuci se stává časově náročnější manuální způsob testování. K nalezení přesné hodnoty, která se bude nacházet mezi 25. a 26. exekucí, vypomůže následující graf.



Graf 2: Časová náročnost při rostoucím počtu exekuce testu pro přihlášení [vlastní zpracování autora]

V grafu se nacházejí přímky manuálního a automatizovaného testování, přičemž nulová hodnota u počtu opakování představuje počáteční hodnotu, tedy čas, za který byl test vytvořen. Rovnice přímky manuálního testování je $\boldsymbol{y = 339 + 47x}$ a rovnice přímky automatizovaného testování je $\boldsymbol{y = 1298 + 10x}$. Porovnáním rovnic lze získat konkrétní hodnotu, ve které se přímky protínají. Tento bod má hodnotu ***25,9189***. Hodnota byla zaokrouhlena na 4 desetinná místa a představuje počet exekucí, při nichž je časová náročnost manuálního testování rovna časové náročnosti automatizovaného testu.

5.2 Finanční analýza náročnosti testu

Finanční analýza náročnosti testu rozšiřuje časovou analýzu náročnosti testu o finanční faktor. Jelikož projektový trojimperativ se skládá mimo kvality a času také z nákladů, je nezbytné zahrnout finanční složku k posouzení vhodnosti použití manuálního a automatizovaného testování. V kapitole 5.1 byly nalezeny počty exekucí, které na základě časové složky upřednostňovali automatizované testování před manuálním. Autor

práce předpokládá odlišný počet exekucí, který upřednostní automatizované testování při zahrnutí finančního faktoru. Práce naopak nezahrnuje náklady na případnou údržbu testů.

Jelikož ze statistických portálů nebylo možné dohledat mzdy pro potřebné pracovní pozice, byli tyto údaje získány z portálu www.jobs.cz, jenž slouží jako inzerční portál oficiálních pracovních nabídek. Práce v IT často nabízí možnost práce na živnostenský list a tyto nabídky bývají ohodnoceny vyšší mzdou. Z tohoto důvodu byly vybrány nabídky pouze pro práci na hlavní pracovní poměr.

Bylo nalezeno 5 nabídek pro test analytika, manuálního testera a automatizovaného testera. Test analytik obstarává tvorbu manuálního testu, u automatizovaného testu provádí tvorbu a exekuci testu tentýž zaměstnanec. Jestliže nabídka neobsahovala konkrétní finanční ohodnocení, ale určitý rozsah, byla brána prostřední hodnota daného ohodnocení. Tabulka hrubých mezd pro manuálního a automatizovaného testera se nachází níže. Autor práce bude nadále pracovat s průměrnou hodnotou mezd.

	1.	2.	3.	4.	5.	Průměr
Test analytik	102 500 Kč	80 000 Kč	65 000 Kč	87 500 Kč	50 000 Kč	77 000 Kč
Manuální tester	28 500 Kč	37 500 Kč	31 500 Kč	40 000 Kč	38 500 Kč	35 200 Kč
Automatizovaný tester	45 000 Kč	50 000 Kč	90 000 Kč	75 000 Kč	80 000 Kč	68 000 Kč

Tabulka 14: Mzdy pracovních pozic z portálu jobs.cz [vlastní zpracování autora]

Mzdy uvedené v tabulce výše představují měsíční ohodnocení. Jelikož časové údaje jsou zachyceny ve vteřinách, je nezbytné průměrné mzdy převést na ohodnocení vteřin. Přepočty mezd jsou zachyceny v tabulce níže.

	Měsíc	Den	Hodina	Minuta	Vteřina
Test analytik	77 000 Kč	3 667 Kč	458 Kč	7,64 Kč	0,1273 Kč
Manuální tester	35 200 Kč	1 676 Kč	210 Kč	3,49 Kč	0,0582 Kč
Automatizovaný tester	68 000 Kč	3 238 Kč	405 Kč	6,75 Kč	0,1124 Kč

Tabulka 15: Přepočet měsíční mzdy na vteřiny [vlastní zpracování autora]

Počet pracovních dnů pro rok 2019 byl v průměru 20,92. K výpočtu byla použita zaokrouhlená hodnota 22 pracovních dnů. Pro výpočet hodinové mzdy se pracovalo s osmi

hodinovou pracovní dobou. Pro zpřesnění výsledků byla mzda na vteřiny zaokrouhlena na 4 desetinná místa.

5.2.1 Registrace

K získání potřebných dat je nezbytné roznásobit časy návrhů a exekucí odpovídajícími hodnotami mzdového ohodnocení z tabulky č. 15. Po roznásobení těchto hodnot lze získat následující tabulku.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	68,2407	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	305,2579	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	Součet		
Manuální	7,0423	7,0423	7,0423	7,0423	237,2566		
Automatický	3,2606	3,2606	3,2606	3,2606	383,5119		

Tabulka 16: Rozšíření o finanční složku testu pro registraci [vlastní zpracování autora]

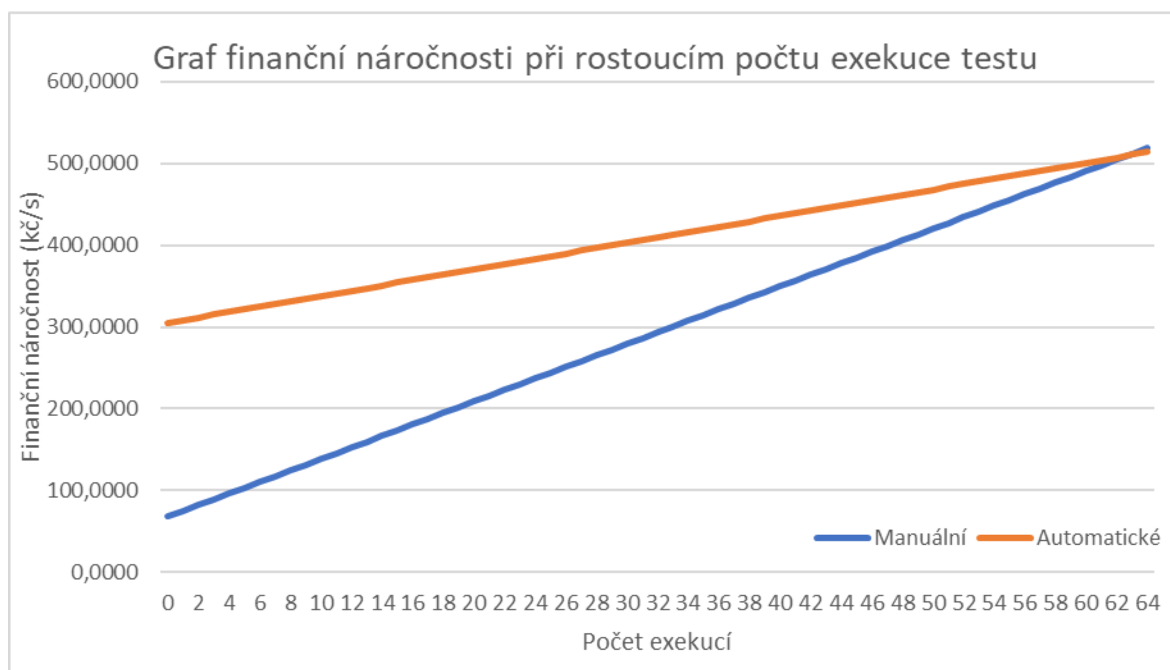
Návrh manuálního testu byl vynásoben mzdou test analytika, exekuce manuálního testu byla roznásobena mzdou manuálního testera a návrh a exekuce automatizovaného testu byl roznásoben mzdou automatizovaného testera. Následně lze vydedukovat, že ačkoliv časová analýza náročnosti testu registrace doporučuje vhodnost použití automatizace při 24. a častější exekuci, z finančního hlediska je při 24. exekuci testu vhodnější způsob testování manuální.

Následující tabulka nachází exekuci, při které se z finančního hlediska stává výhodnější automatizované testování.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	68,2407	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	305,2579	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	25. exekuce	26. exekuce	27. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	28. exekuce	29. exekuce	30. exekuce	31. exekuce	32. exekuce	33. exekuce	34. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	35. exekuce	36. exekuce	37. exekuce	38. exekuce	39. exekuce	40. exekuce	41. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	42. exekuce	43. exekuce	44. exekuce	45. exekuce	46. exekuce	47. exekuce	48. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	49. exekuce	50. exekuce	51. exekuce	52. exekuce	53. exekuce	54. exekuce	55. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	56. exekuce	57. exekuce	58. exekuce	59. exekuce	60. exekuce	61. exekuce	62. exekuce
Manuální	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423	7,0423
Automatický	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606	3,2606
	63. exekuce	Součet					
Manuální	7,0423	511,9074					
Automatický	3,2606	510,6746					

Tabulka 17: Nalezení efektivity z finančního hlediska pro případ registrace [vlastní zpracování autora]

Jak již bylo uvedeno výše, tabulka č. 17 zobrazuje roznásobené časové a k nim přiřazené finanční hodnoty s konečným součtem, kdy finální hodnota manuálního testování převyšuje finální hodnotu automatizovaného testování. Z toho vyplývá fakt, že při 63. exekuci se stává finančně náročnější možností testování manuální. K nalezení bodu, ve kterém se finanční náročnost manuálního testu rovná finanční náročnosti testu automatizovanému, poslouží porovnání rovnic přímek v následujícím grafu.



Graf 3: Finanční náročnost při rostoucím počtu exekuce testu pro registraci [vlastní zpracování autora]

V grafu se nacházejí dvě přímky rovnic. Rovnice přímky manuálního testování je ***y = 68,2407 + 7,0423x*** a rovnice přímky automatizovaného testování je ***y = 305,2579 + 3,2606x***. Porovnáním přímek lze získat konkrétní hodnotu, ve které se přímky protínají. Bod má hodnotu ***62,6748***. Hodnota byla zaokrouhlena na 4 desetinná místa a představuje počet exekucí testu registrace, při nichž je finanční náročnost manuálního testování rovna finanční náročnosti automatizovaného testování.

5.2.2 Přihlášení

Po roznásobení časových údajů hodnotami odpovídajícími finanční mzdě daného pracovníka lze získat následující tabulku.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	43,1597	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	145,9392	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	25. exekuce	26. exekuce	Součet
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	114,2814
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	175,1720

Tabulka 18: Rozšíření o finanční složku testu pro přihlášení [vlastní zpracování autora]

Z tabulky je patrné, že ačkoliv časová analýza náročnosti testu pro přihlášení prokázala, že při 26. exekuci testu se stává automatizované testování efektivnějším, z finančního hlediska je při 26. exekuci testu stále výhodnější testování manuální.

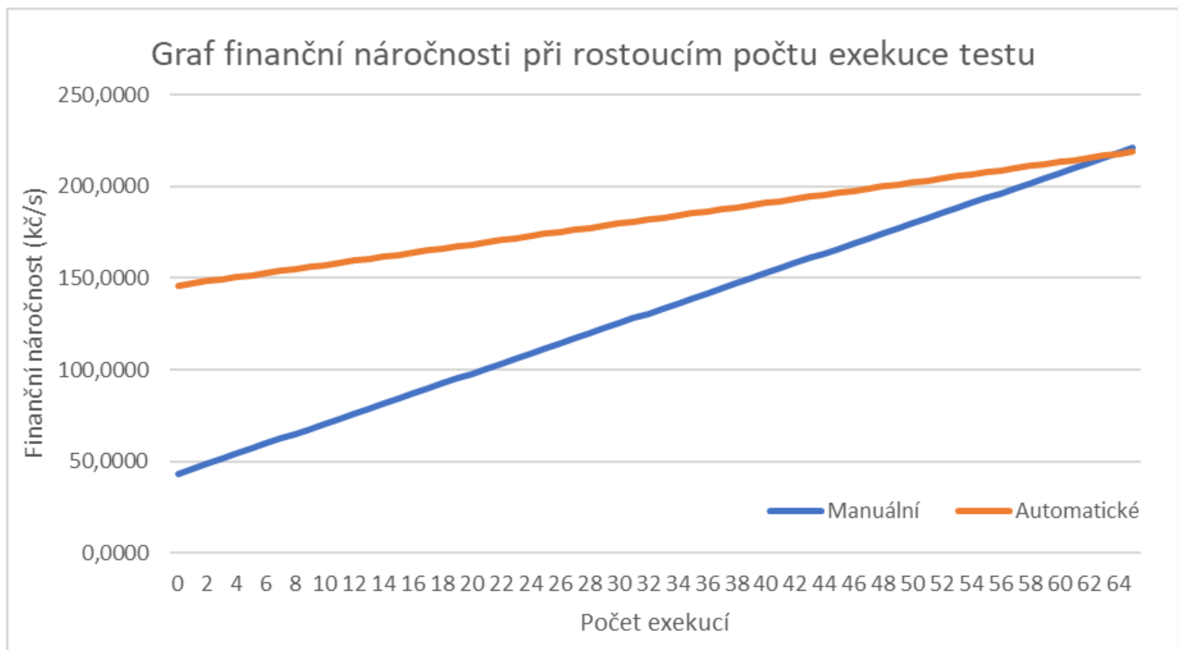
Následující tabulka zobrazuje exekuci, při které se z finančního hlediska stává výhodnější automatizované testování.

	Návrh	1. exekuce	2. exekuce	3. exekuce	4. exekuce	5. exekuce	6. exekuce
Manuální	43,1597	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	145,9392	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	7. exekuce	8. exekuce	9. exekuce	10. exekuce	11. exekuce	12. exekuce	13. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	14. exekuce	15. exekuce	16. exekuce	17. exekuce	18. exekuce	19. exekuce	20. exekuce

Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	21. exekuce	22. exekuce	23. exekuce	24. exekuce	25. exekuce	26. exekuce	27. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	28. exekuce	29. exekuce	30. exekuce	31. exekuce	32. exekuce	33. exekuce	34. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	35. exekuce	36. exekuce	37. exekuce	38. exekuce	39. exekuce	40. exekuce	41. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	42. exekuce	43. exekuce	44. exekuce	45. exekuce	46. exekuce	47. exekuce	48. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	49. exekuce	50. exekuce	51. exekuce	52. exekuce	53. exekuce	54. exekuce	55. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	56. exekuce	57. exekuce	58. exekuce	59. exekuce	60. exekuce	61. exekuce	62. exekuce
Manuální	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354	2,7354
Automatický	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243	1,1243
	63. exekuce	64. exekuce	Součet				
Manuální	2,7354	2,7354	218,2285				
Automatický	1,1243	1,1243	217,8968				

Tabulka 19: Nalezení efektivity z finančního hlediska pro případ přihlášení [vlastní zpracování autora]

Při pohledu na tabulku výše lze konstatovat, že při tvorbě testu a jeho 64 exekucí se z finančního hlediska stává efektivnější možností automatizované testování. Bod zlomu neboli bod, ve kterém se finanční náročnost manuálního testu rovná finanční náročnosti testu automatizovanému, bude vyjádřen porovnáním rovnic přímek v následujícím grafu.



Graf 4: Finanční náročnost při rostoucím počtu exekuce testu pro přihlášení [vlastní zpracování autora]

V grafu se nacházejí dvě přímky rovnic. Rovnice přímky manuálního testování je ***y = 43,1597 + 2,7354x*** a rovnice přímky automatizovaného testování je ***y = 145,9392 + 1,1243x***. Porovnáním přímek lze získat konkrétní hodnotu, ve které se přímky protínají. Tento bod má hodnotu ***63,7946***. Hodnota byla zaokrouhlena na 4 desetinná místa a představuje počet exekucí testu přihlášení, při nichž je finanční náročnost manuálního testování rovna finanční náročnosti automatizovaného testování.

5.3 Doporučení

V kapitolách 5.1 a 5.2 byly provedeny analýzy, jejichž výsledky vedou k doporučení použití způsobu testování pro různé typy testů. Typy testů nebudou v této kapitole detailněji uvedeny, neboť podrobná charakteristika se nachází v kapitole 3.8.2.

Pro doporučení je nezbytná tabulka bodů, při nichž je náročnost manuálních a automatizovaných testů ekvivalentní. Tabulka vychází z kapitol 5.1 a 5.2.

	Časová analýza náročnosti testu	Finanční analýza náročnosti testu
Registrace	23,6848	62,6748
Přihlášení	25,9189	63,7946

Tabulka 20: Souhrn výsledků z analýz náročnosti testů [vlastní zpracování autora]

Jelikož se jedná o exekuce testů, budou hodnoty v tabulce výše zaokrouhleny na horní hranici celého čísla. Zaokrouhlená hodnota pak reprezentuje počet exekucí testu, které je nutno vykonat, aby způsob automatizovaného testování byl efektivnější nežli způsob manuálního testování. Nemá smysl se zabývat pouze částí exekuce daného testu.

Časová analýza náročnosti testu registrace odhalila vhodnost použití automatizovaného testování při ***24.*** a vyšším počtu exekuce. Časová analýza náročnosti testu přihlášení doporučuje využít automatizované testování při ***26.*** a vyšším počtu exekuce testu.

Finanční analýza náročnosti testu registrace doporučuje použít automatizované testování ve chvíli, kdy se zamýšlí během jednoho testovacího období ***63.*** a vyššího počet exekuce. Finanční analýza náročnosti testu přihlášení doporučuje automatizované testování pro případ ***64.*** a vyššího počtu exekuce.

Veškeré výše zmíněné výsledné hodnoty jsou pro časové období jednoho „releasu“. Tento termín je při vývoji softwaru velmi často používán a představuje období, během kterého je dodána část či verze softwaru. Během tohoto období povětšinou nedochází k úpravě a údržbě testů, tudíž nebyly do analýz započítány náklady na údržbu testů.

Ačkoliv jsou tesy registrace a přihlášení odlišně náročné, jejich výsledky, při nichž je doporučeno využít automatizované testování, jsou velmi podobné. Není však vyloučeno,

že jiný test ze základních scénářů by mohl přinést odlišné hodnoty. S vysokou pravděpodobností však budou i ostatní testy základních scénářů obsahovat podobné výsledky. Pro přesné a celkové doporučení by bylo vhodné otestovat stejným způsobem i zbytek testů základního scénáře.

Rozdíl mezi časovou a finanční náročností testu je poměrně veliký. Důvodem je vyšší zaměstnanecká mzda automatizovaného testera oproti manuálnímu. Rozdíl pro test registrace je 39 exekucí a pro test přihlášení je rozdíl 38 exekucí. Na základě časových a finančních zdrojů se musí projektový a testovací tým rozhodnout, při jakém počtu exekucí by bylo vhodné využít automatizované testování.

5.3.1 Doporučení na základě typu testu

Hlavní roli v rozhodnutí o vhodnosti použití automatizovaného testování hraje fakt, jak často budou testy vykonávány. Doporučené počty exekucí testů jsou již známé z tabulky č. 20, avšak potřeby exekucí často závisí na rozhodnutí test manažera. Další bod, jenž značně ovlivňuje použití automatizovaného testování je typ firmy vyvíjející aplikaci. Jestliže se jedná o malou firmu, nepředpokládá se časté nasazování, tudíž smoke testy nebudou tak často vykonávány. Za předpokladu, že se jedná o korporátní firmu, může docházet k nasazování nových verzí téměř denně. V takovém případě je využití automatizovaného testování přímo ideální.

Obecně lze tedy konstatovat, že pro smoke, sanity a konfirmační testy je vhodné využít automatizace, pro regresní testování je využití automatizace vhodné z dlouhodobého hlediska a pro end to end testy není automatizované testování příliš vhodné, jelikož iterace jejich testování není dostatečně vysoká.

6 Závěr

Teoretická část práce byla zaměřena na charakteristiku používaných metodik vývoje a testování softwaru a k nim příbuzným pojmům. Práce představila základy rigorózní a agilní metodiky, kvality softwaru, definovala role testovacího týmu a zaměřila se na způsob testování včetně různých typů testů. Další část teoretické práce charakterizovala základní specifika manuálního a automatizovaného testování, testovací případ, správu defektů a testovací prostředí, na němž se testování vyvíjené aplikace provádí. Za pomoci syntézy teoretických poznatků bylo umožněno zpracovat praktickou část diplomové práce.

V praktické části diplomové práce byla nejdříve popsána samotná aplikace a její základní funkcionality, které byly v rámci práce testovány, což posloužilo pro stručný návrh plánu testování. V práci byly vybrány dva odlišně náročné testy, které posléze posloužily pro analýzu vhodnosti použití automatizovaného a manuálního testování. Na základě strategie testování byly vytvořeny případy užití a jejich samotné specifikace, které byly podnětem pro tvorbu manuálních a automatizovaných testů. Manuální testy byly vytvořeny prostřednictvím volně dostupné aplikace TestRail a automatizované testy byly napsány v jazyku Java za pomoci nástroje Selenium WebDriver ve vývojářském studiu Eclipse. Autor sledoval časovou náročnost tvorby a exekuce vybraných testů. Pro zvýšení správnosti výsledků pracoval autor s průměrnou časovou délkou vlastních výsledných hodnot a výsledných hodnot čtyř participantů. Na základě průměrné časové délky tvorby a exekuce testů byla provedena časová a finanční analýza a vyhodnocena časová a finanční náročnost v podobě počtu doporučených exekucí zvolených testů. Práce poskytuje řešení v podobě kvantifikovaného vodítka při rozhodování o vhodnosti použití způsobu testování, nikoli však hodnocení přístupu řešení.

Autor práce navrhl a vykonal stanovené testy a provedl analýzu vhodnosti použití automatizovaného a manuálního testování na základě časové a finanční náročnosti testu, čímž splnil definované cíle. Přestože se autor věnuje testování softwaru několik let, nabyl při psaní práce a provedení výzkumu mnoha hodnotných informací.

7 Seznam použitých zdrojů

1. ALWAN, Motea. *What is System Development Life Cycle?* [online]. 9.1.2015 [cit. 2018-08-04]. Dostupné z: https://airbrake.io/blog/sdlc/what-is-system-development-life-cycle
2. HLAVA, Tomáš. *Testování softwaru* [online]. [cit. 2018-08-04]. Dostupné z: http://testovanisoftwaru.cz/
3. PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 978-807-2266-364.
4. ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-802-5138-168.
5. *TESTINGFREAK: What is Incremental Model?* [online]. [cit. 2018-08-04]. Dostupné z: http://testingfreak.com/incremental-model-software-testing-advantages-disadvantages-incremental-model/
6. GIMON, Zee. *UNDERSTANDING THE RAPID APPLICATION DEVELOPMENT MODEL* [online]. [cit. 2018-08-04]. Dostupné z: https://theappsolutions.com/blog/development/rad-model/
7. BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.
8. *VÝVOJOVÝ CYKLUS SOFTWARE: DOMINUJÍ AGILNÍ METODIKY TRHU?* [online]. 2015 [cit. 2018-08-04]. Dostupné z: https://www.middleware.cz/projektove-rizeni0/24-pristupy-k-vyvoji-software-dominuje-agilni-vyvoj-trhu
9. *RUP* [online]. [cit. 2018-08-04]. Dostupné z: https://techterms.com/definition/rup
10. *Agilní metodiky řízení vývoje software* [online]. [cit. 2018-08-04]. Dostupné z: https://managementmania.com/cs/agilni-metodiky-rizeni-vyvoje-software
11. KNESL, Jiří. *Agilní vývoj: Scrum* [online]. 2009 [cit. 2018-08-04]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/
12. BECK, Kent. *Extrémní programování*. Praha: Grada, 2002. Moderní programování. ISBN 80-247-0300-9.

13. *Extrémní programování (XP)* [online]. [cit. 2018-08-15]. Dostupné z: https://mbi.vse.cz/public/cs/obj/METHOD-92
14. *What is V-model and W-model in Software Testing* [online]. [cit. 2018-08-15]. Dostupné z: https://www.testbytes.net/blog/v-model-and-w-model-software-testing/
15. *Testování softwaru: Testovací tým* [online]. [cit. 2018-09-14]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/testovaci-tym/
16. HAMBLING, Brian a Peter MORGAN. *Software testing: an ISTQB-ISEB foundation guide*. 2010. ISBN 978-1-906124-76-2.
17. JORGENSEN, Paul. *Software testing: a craftsman's approach*. 3rd ed. Boca Raton: Auerbach Publications, c2008. ISBN 0-8493-7475-8
18. *Software Testing Class: Software Testing Life Cycle STLC* [online]. [cit. 2018-10-04]. Dostupné z: https://www.softwaretestingclass.com/software-testing-life-cycle-stlc/
19. BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
20. PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. *Jak testuje software Microsoft*. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5.
21. *SeleniumHQ* [online]. [cit. 2019-03-11]. Dostupné z: https://www.seleniumhq.org/#selenium-commands-selenese
22. *What is Automated Testing?* [online]. [cit. 2019-03-26]. Dostupné z: https://smartbear.com/learn/automated-testing/what-is-automated-testing/