Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Infomačních technologií

Diplomová práce

Testování softwaru se zaměřením na automatizované testy

Andrea Hořavová

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Automatizované testování softwaru“ jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne _____

Poděkování

Ráda bych touto cestou poděkovala svému vedoucímu Ing. Václavu Lohrovi, Ph. D. za vedení a inspiraci při tvorbě této diplomové práce. Dále bych chtěla poděkovat panu Bc. Oldřichu Bitnerovi za ochotu a poskytnutí softwaru potřebného pro testování v rámci této diplomové práce.

Testování softwaru se zaměřením na automatizované testy

Software testing focused on automated tests

Souhrn

Diplomová práce se zaměřuje na oblast testování softwaru, která je velmi důležitou složkou při vývoji nového softwaru. Testováním je zjišťována elementární funkčnost výsledného softwaru v jeho přesnosti a správnosti, a tím je zjišťována také jeho kvalita. V diplomové práci je zkoumáno zejména automatizované testování na konkrétním softwaru. Na firmou poskytnutém testovací prostředí pro tuto diplomovou práci jsou prezentovány automatizované testy, které jsou konstruovány v nástroji Selenium IDE. Výstupem diplomové práce je tvorba testovacích případů pro přípravu k testování, konstrukce testů v příslušném nástroji a samotná exekuce testů, které informují o výsledcích testování daného softwaru.

Summary

Master´s thesis is focusing on testing software area, which is very important component in development a new software. With software testing is detecting elementary functionality of the resulting software, especially in his accuracy and correctness, and thereby is detecting his qualities. In this master´s thesis is examined especially automated testing on the software. On software environment, which was provided on testing by firm, is presented autometed tests, which is construated in testing tool named Selenium IDE. The output this master´s thesis is creation test case for software testing in applicable tool and executing tests, which is informated about results of the software testing.

Klíčová slova: testování softwaru, test analýza, test design, testovací případy, Selenium IDE, exekuce testů, automatizované testy

Keywords: software testing, test analysis, test desing, test case, Selenium IDE, executing, automated testing

OBSAH

1. Úvod

Diplomová práce pojednává o tématu testování softwaru. Analyzuje pojem testování obecně v oblasti softwarového inženýrství, zabývá se metodami uplatňovanými pro detekci a report chyb a popisuje celkový proces testování softwaru. Testování je činnost patřící mezi vývoj a produkci. Pod pojmem testování si lze představit nástroje, lidi, stanovené metody a postupy. Každý software jako produkt, který přichází na trh a následně k zákazníkům, by měl být patřičně otestován. Nedostatečné otestování nebo dokonce neotestování softwaru před produkcí, by mohlo znamenat nekvalitu nabízeného softwaru potencionálním zákazníkům, kteří mohou ztratit zájem o produkt.

Téma testování softwaru je stejně důležité jako vývoj softwaru. Poměrně velká část světa přijde do styku s počítačem nebo jinou výpočetní technikou a téměř nikomu není pojem software neznámý. S rozmachem osobních počítačů, notebooků, chytrých telefonů a dalších výpočetních zařízení, je dnes software na každém rohu a je mnohem sofistikovanější než na svém počátku. Uživatelé požadují specifické funkčnosti a s tím i zmíněnou kvalitu. Proto jsou nároky na kvalitní software důležité a testování se stalo nepostradatelnou součástí tvorby softwaru.

V této diplomové práci se věnuji procesu testování, protože působím jako tester v korporátní společnosti, kde je testování neodmyslitelnou položkou. Každá změna v systému v takové společnosti může mít nedozírné následky, jak ve finanční ztrátě, tak i ve ztrátě osobního charakteru. Tato skutečnost demonstruje, jak moc je testování softwaru podstatné. Má vlastní zkušenost je, že testování není jen jednoduché a jednoznačné plnění očekávání od softwaru. Je potřeba znát celou architekturu jednotlivých procesů, které jsou spojeny až několika aplikacemi, databázemi a servery. Jako tester musím znát komplexní funkčnost softwaru, abych mohla otestovat veškeré aspekty, které mohou při jeho používání nastat. Tedy nejen pozitivní průchody, ale i simulace výpadků, nefunkčnosti jiných závislých systémů a podobných negativních jevů. Jedině tímto mohu zjistit a odhalit většinu „much“ na softwaru. Moje práce testera zatím spočívá pouze v manuálním testování. Veškeré exekuce testů tedy záleží na mém úsudku a spolehlivosti, na rozdíl od automatizovaného testování. Proto jsem se pro tuto diplomovou práci rozhodla věnovat automatizovaným testům jako součásti testovacího procesu.

Automatizované testování již požaduje potřebnou úroveň znalostí programování. Automatizované testy slouží pro zjednodušení a úsporu času při testování. Některé testy jsou z pohledu časové náročnosti složité pro testera, a proto jsou vhodné pro automatizaci. V

praktické části diplomové práci bude zaměřeno přímo na automatizované testy, které budou navrženy a realizovány na konkrétním software eHelpdesk.cz. Výstupem diplomové práce je vytvoření automatizovaných testů pro daný software v programu Selenium IDE a zhodnocení procesu testování.

2. Testování softwaru

Testování ve své podstatě skrývá velmi komplexní proces zajišťování kvality. Testováním je zjišťována funkčnost, správnost, přesnost a mnoho dalších aspektů, které popisují vlastnosti a chování daného výrobku či produktu. Obecně je testování považováno jako hledání chyb. Samotné testování není otázkou jednoduchého pohledu a odvození chyby na první pohled, protože chyby nejsou samy o sobě přímo zjevné. Obzvláště pak hledání chyb v softwaru je velmi nejednoznačný problém. Software v sobě zahrnuje mnoho proměnných, které mohou zapříčinit nefunkčnost celého produktu. Testování softwaru se dá přirovnat k hledání jehly v kupce sena. Naštěstí existují metodiky, pravidla a zavedené postupy k docílení minimální chybovosti, a tím i kvality. A v převážné většině případů je chyb dostatek na to, aby byly odhaleny. [1][2]

Testování není jen o spuštění nějakého programu, který najde přímo chyby v kódu. Testování vyžaduje i specifický přístup testera, který má za úkol chyby hledat. Správné uvažovaní testera nebo týmu testerů je nedílnou součástí procesu testování. Správný postup, nástroje, prostředí nezaručí komplexnost testování, pokud je přístup testera nedostatečný. [1][2]

Aby byl software otestovatelný, musí tester dobře znát jeho očekávanou funkčnost, podle které jej testuje. Neodpovídá-li jakýmkoliv způsobem chování softwaru zadání, je to vyhodnoceno jako chyba softwaru. Avšak někdy je možné se dostat do fáze, kdy je špatné zadání a je nutné zadání přepracovat. Z tohoto vyplývá také další vlastnost testera při testování softwaru, a to, že tester by měl úzce spolupracovat s vývojáři a zadavateli. [4][5]

3. Quality Assurance

Testování je činnost související se zajišťováním kvality produktu. Pojem Quality Assurance, přeloženo jako zajišťování kvality, zahrnuje celou oblast procesů vývoje softwaru na rozdíl od samotného testování.

Zajišťování kvality spočívá ve zvolení vhodné metodiky vývoje softwaru, viz kapitola 8. Tato metodika má velký vliv na to, jakým způsobem se bude software vyvíjet a tím i jaký výsledek daná metoda přinese. Další aspekty zajišťování kvality jsou nákladovost na celý proces vývoje. Zvolením nevhodné metodiky hrozí vysoká nákladovost například nekompletní organizací řízení lidí, komunikace, testování, plánování, atd.

Quality Assurance se stará o:

- Zvolení vhodné metodiky vývoje softwaru
- Plánování cílů
- Definování organizační struktury
- Definovaní rolí a kompetencí
- Koordinace a komunikace
- Analýza a řešení rizik
- Snižování nákladů
- Efektivita při vývoji

[6][7]

4. Standardy ISO/IEEE/IEC

Quality assurance používá pro svou činnost standardů, které byly vytvořeny organizacemi a které pomáhají udržovat vztah mezi požadavky zákazníka a kvalitou produkovaného softwaru.

Standardy popisují, jaké požadavky jsou potřeba za jednotlivé oblasti quality assurance, tak aby výstup byl co nejkvalitnější. Quality assurance zahrnuje oblasti plánování, sběr a analýzu požadavků, návrh softwaru, integrace do prostředí, testování, akceptace, atd.

Následující seznam zobrazuje standardy, které řídí zajišťování kvality softwaru:

Název standardu	**Popis**
ISO 9000:2000	QMS Fundamentals and Vocabulary
ISO 9001:2000	QMS Requirements
ISO 9004:2000	QMS Guidelines for performance

	improvements
ISO 10005:1995	Quality management. Guidelines for quality plan
ISO 10006:2003	Quality management system. Guidelines for quality management in projects
ISO 10007:2003	Quality management system. Guidelines for configuration management
ISO 10012:2003	Measurement management systems. Requirements for measurement processes and measuring equipment
ISO 12119:1994	Information technology. Software packages. Quality requirements and testing
ISO 12182:1998	Information technology. Categorization of software
ISO 12207:2002	– International standard on software life cycle processes (common framework for software life-cycle processes, with well-defined terminology, that can be referenced by the software industry
ISO 15288:2002	System engineering. System life cycle processes
ISO 15504:1998	Information technology. Software process assessment (SPICE)
ISO 15846:1998	System engineering. System life cycle processes / Configuration management

ISO 15910:1999	Information technology. Software user documentation process
ISO 19011:2002	Guidelines for Quality and Environmental Management Systems Auditing
ISO/IEC/IEEE 29119-1	Concepts & Definitions
ISO/IEC/IEEE 29119-2	Test Processes
ISO/IEC/IEEE 29119-3	Test Documentation
ISO/IEC/IEEE 29119-4:	Test Techniques
ISO/IEC/IEEE 29119-5	Keyword-Driven Testing
ISO/IEC 33063	Process Assessment Model for Software Testing

Tabulka č. 1. – Standardy softwarového testování

[8][9]

5. Pojmy v testování

Pod slovem testování si lze představit mnoho obecných pojmů, které můžou a nemusí přímo s testováním souviset. Následující pojmy jsou používané v oboru testování a jejich stručným popisem bude lépe porozuměno v dalších kapitolách diplomové práce.

5.1 Chyba

Chybou se rozumí jakékoliv nestandardní chování. Mezi chybu může být zařazeno selhání, defekt, vada, odchylka, nepřesnost, závada, anomálie atd. Jedná se o obecný pojem vyjadřující nesourodost, neočekávanost, nekonzistentnost při testování softwaru. Více viz kapitola **Chyba! Nenalezen zdroj odkazů.**.[1]

5.2 Specifikace produktu

Dokument, na kterém se dohodli zákazník a vývojář. V tomto případě je zákazník jako zadavatel zakázky. Specifikace určuje, jak by měl daný softwarový produkt ve finální podobě vypadat a jak by se měl chovat při běžném používání koncovým uživatelem. Specifikace obsahuje všechny náležitosti, které pomáhají testerovi pochopit funkčnost softwaru, tak aby byl schopný po test analýze vytvořit testovací případy a následně tyto testy exekuovat. [1][5]

5.3 Testovací prostředí

Testovací prostředí je možné chápat jako simulace nějakého produkčního/uživatelského prostředí, které hardwarově i softwarově odpovídají tomu produkčnímu. Zásahy v testovacím prostředí nemají vliv na funkčnost hlavního produkčního prostředí a neovlivňují tak chod softwaru používaný uživatelem. [1][5][3]

5.4 Testovací případ

Neboli „Test case“ je testerem vytvoření scénář, který má své formální náležitosti a popisuje proces testování nějaké části programu. Jde o dílčí část z množiny všech testů, které jsou potřeba k otestování celého softwaru. [1][5][3]

5.5 Testovací nástroje

Jsou pomocné programy, ve kterých tester připravuje analýzu, tvoří test case, exekuuje testy, zakládá defekty, komunikuje s developery či zadavateli. Výstupem těchto programů jsou analýzy stavů, rozdělení práce, stanovení testovacího času, kontroly plánů, apod. Příklad testovacího nástroje testera může být program HP ALM nebo JIRA. [1][3]

5.6 Testovací data

V případě, že software potřebuje data pro vykonávání své funkce, je potřeba vytvořit umělá data. Tato uměle vytvořená data však musejí odpovídat parametrům těm reálným, aby

byl výsledek relevantní. Testovacími daty mohou být například uživatelé, kteří se přihlašují do systému. K udržitelnosti a dostupnosti testovacích dat složí databázové systémy, se kterými pracuje nejen tester. [1][5][3]

5.7 Test analýza

Test analýza je proces studování specifikace pro daný testovací software a zjišťování další informací, které pomůžou testerovi lépe pochopit funkčnost softwaru, který testuje. Je potřeba, aby tester pochopil zadání specifikace, oproti které testuje, stejně jako její zadavatel. Po test analýze následuje vytváření testovacích případů, tedy test design. [1][5][3]

5.8 Test design

Pojmem test design se rozumí samotné vytváření testů včetně všech náležitostí, jako jsou popis testu a jednotlivé kroky scénáře, stanovení vhodných testovacích dat, výběr vhodného testovacího prostředí. Design testů následuje vždy po důkladné test analýze. A výstupem je soubor všech testů, které důkladně pověří všechny části softwaru podle zadané specifikace. [1][5][3]

5.9 Testovací scénáře

Testovací scénáře jsou součástí test designu. Testerem je tvořena chronologická posloupnost kroků, které na sebe navazují. Tato návaznost by měla být co nejkonkrétnější, tak aby se dala bezprostředně detekovat existence chyb. Některé testy se používají i pro opakované testování jako udržitelnost základní funkce softwaru při dalších změnách, a proto je potřeba testovací scénáře udržovat aktualizované. [1][5][3]

5.10 Test result

Výsledek testů, které prošli exekucí, se porovnává s očekávaným/ plánovaným výsledkem. Pokud je testovací výsledek shodný s očekávaným výsledem, tak test proběhl

úspěšně. V případě, že je mezi výsledky rozdíl, tester zadává defekt na opravu softwaru. A v dalším kole jej retestuje. [1][5][3]

5.11 Exekuce testů

Jsou-li připravené veškeré aspekty k testování: test analýza, test design, testovací data a prostředí. Může se tester pustit do provádění testů, které se nazývá exekuce. Tester provádí úkony přesně podle napsaných scénářů a vyhodnocuje výsledek (test result).

[1][5][3]

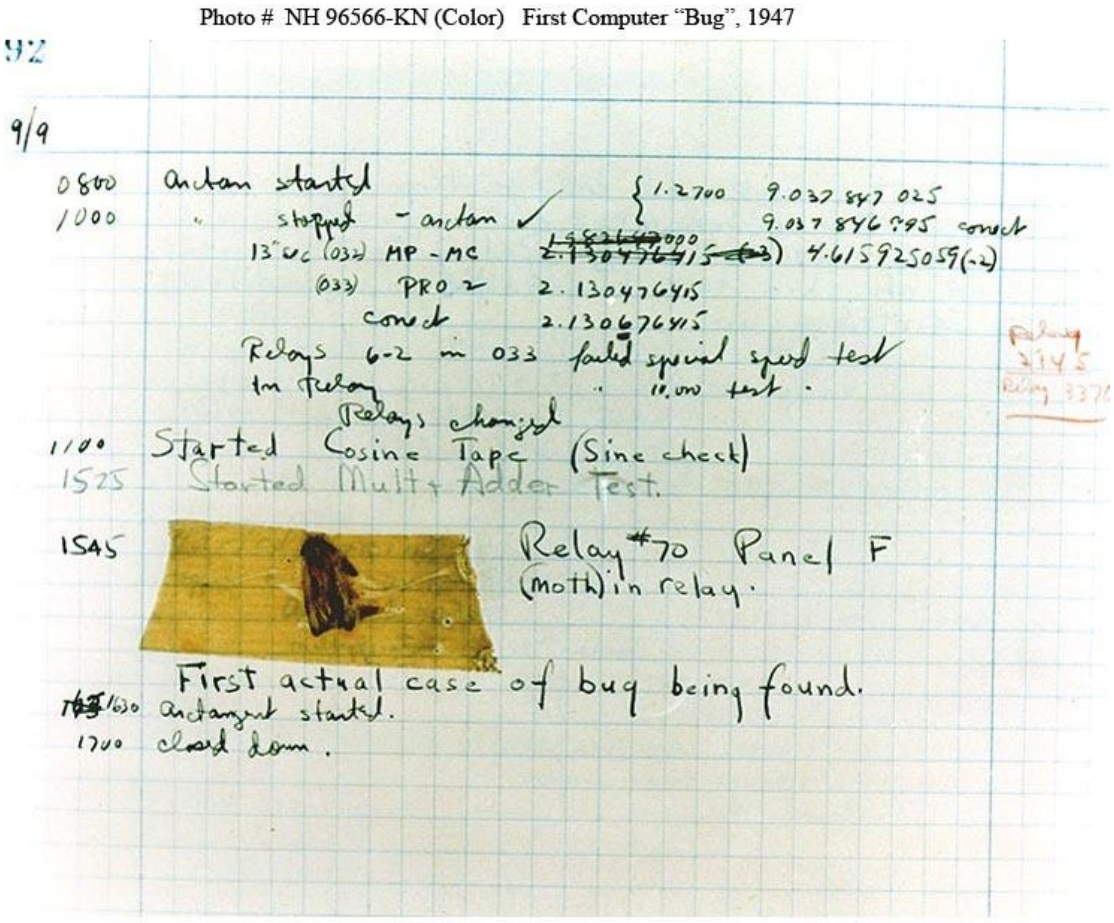
5.12 Logy

Logy slouží testerovi jako pomůcka při detekování chyb v softwaru. Chyba nemusí být na první pohled zřejmá a odpovědností testera je, developerovi přiblížit její nález. Proto má tester přístup do různých logů, kde se snaží hledat podle indicií známky selhání. Tím může být například volaný čas nějaké služby nebo jiný další parametr konkretizující oblast selhání. [1][5][3]

6. Softwarová chyba

První zmínky o nalezené softwarové chybě, jsou zaznamenány již v roce 1947 na Harvardské univerzitě, kde pracovala matematička, počítačová vědkyně a důstojnice Grace Murray Hopperová. Při práci na elektromagnetickém superpočítači Mark II G. M. Hopperová a její kolegové narazili na nefunkčnost superpočítače, který byl hardwarového původu. Paradox nefunkčnosti superpočítače, který G. M. Hopperová a její tým řešili, souvisí s dnešním označením pro chybu, tj. „Bug“, což v překladu znamená štěnice nebo brouk. V hardwaru superpočítače byla totiž nalezena můra, která zapříčinila selhání celého superpočítače. Informace o chybě s tímto následkem byla zaznamenána v logu s následujícím výrokem: „First actual case of bug being found.“

Přeloženo jako: „První skutečný případ chyby byl nalezen.“ [10]



Obráze č.1 – První bug

6.1 Definice chyby

Úkolem testování je detekování neočekávaných chování software, ale zároveň je důležité specifikovat podmínky očekávaného chování softwaru. Záleží na upřesnění funkčnosti softwaru, aby bylo možné tvrdit, že chování softwaru neodpovídá očekávání. Pokud toto není domluveno, tak tester nemůže poznat, zdali se jedná o chybu nebo o funkčnost.

Různé zdroje uvádějí co je a co není chybou. Neexistuje však standardizovaná definice, co je softwarová chyba.

Lydia Ash se zmiňuje o chybě jako o neočekávaném chování softwaru, kdy není testerovi známo, zdali je toto chování správné. Může se jednat chyby ve formátu textu, chybové hlášky či výpadky softwaru. Chyba je podle Lydie Ash vše, co uživatel software neočekává při jeho používání. [2]

Konkrétněji popisuje chybu Ron Patton ve své knize, kde za softwarovou chybu považuje, když:

- Software nedělá něco, co by podle specifikace dělat měl,
- Software dělá něco, co by podle specifikace dělat neměl,
- Software dělá něco, o čem se specifikace nezmiňuje,
- Software nedělá něco, o čem se specifikace nezmiňuje a zmiňovat by se měla,
- Software je obtížně srozumitelný, těžko se s ním pracuje, nebo jej – podle názoru testera – koncový uživatel nebude považovat za správný. [1]

Určuje tedy chybu podle toho, jak je napsána funkční (softwarová) specifikace produktu. Ta určuje vlastnosti, chování výsledného produktu a je vytvářena ve spolupráci vývojového týmu a zadavatele. Tester má oproti čemu testovat, a tak může detekovat chyby při testování, protože má poklady popisující funkčnost softwaru.

Nejen v publikacích, ale v i testerské praxi mají chyby různé termíny, zejména podle svého vzniku či významu. Obecně je možné setkat s těmito pojmy, označující nesprávnou funkčnost nebo úplnou nefunkčnost:

- Error
- Faults
- Failure
- Defekt

Ačkoliv všechny tyto pojmy mají ve svém významu chybu, tak se vzájemně od sebe liší. Pojem error či faults je označován jako konstrukční chyba nebo odchylka od požadovaného nebo zamýšleného stavu ve zdrojovém kódu. Error, jako chyba sama o sobě, nezpůsobuje selhání (Failure) bez příčiny, která ji spouští. Failure neboli selhání je nekorektní nebo chybějící chování programu, které je spouštěno nějakou chybou (error/faults). S tím souvisí i pojem defect (závada). Defektem se rozumí chyba (error) nebo selhání (Failure), které je odhaleno při testování. [1] [2] [11]

6.2 Severita a priorita chyby

Jelikož při testování je možno nalézt velké množství chyb s různou vážností a oprava nalezených a reportovaných chyb může mít odlišnou dobu náročnosti, která je vynaložena na tuto opravu. Používá se jednoznačné rozlišení vážnosti zadávaných chyb. Toto rozdělení pomáhá vývojářům naplánovat čas na opravu, aby tato oprava byla co nejefektivnější. Chyby jsou rozlišovány podle priority a severity. Severitu a prioritu chyb určuje tester při detekování chyby v průběhu testování.

Severita neboli závažnost chyby udává, do jaké míry může chyba ovlivnit funkčnost celého softwaru. Například pokud při testování webové aplikace, tester zjistí, že se s uživatelem nemůže přihlásit do této aplikace. Nebo v případě, že nefunguje jedna z hlavních činností systému. Tyto chyby lze považovat za velmi závažné, protože ovlivňují chod celého systému. Méně závažné je možné považovat chybu, která nenarušuje příliš chod systému. Tím může být například pravopisná chyba, anebo málo používaná funkčnost systému.

Severitu tedy lze rozlišovat dále podle míry závažnosti.

- High
- Medium
- Low

Označovaná míra se může lišit podle stanov, definic firem. Je možné se setkat i s abecedním označením závažnosti. Kdy A je vysoká závažnost chyby, B je střední závažnost a C je nízká závažnost.

Priorita je informace pro vývoj, která udává jak rychle je potřeba reportovanou chybu opravit. Priorita je popsána v požadavcích zadavatele. Například je-li název webové stránky chybný, tak priorita chyby je vysoká, avšak severita je nízká. Nejedená se o závažnou chybu ohrožující chod systému. Priorita se také stupňuje na:

- High
- Medium
- Low

Pro lepší orientaci může být priorita značena číslicemi 1, 2, 3. Priorita 1 značí vysokou urgentnost, priorita 2 střední a priorita 3 nízkou urgentnost.

Hlavní kombinace severity a priority vypadají následovně:

- High severity x Low priority (A3) – chyba, která narušuje jednu z hlavních funkčností softwaru, ale její oprava není podle požadavku tolik důležitá pro testování
- High severity x Hight priority (A1) – chyba, která zapříčiňuje nefunkčnost hlavní činnosti softwaru a nelze dále software testovat, blokuje všechny další testy, tzv. blocker
- Low severity x Hight priority (C1) – chyba, jejíž funkčnost neovlivňuje celý software, může se jednat o důležitou část softwaru, kterou je třeba co nejrychleji opravit
- Low severity x Low priority (C3) – chyba, která nijak neovlivňuje testování, může se jednat o tzv. kosmetickou vadu [6]

Zpravidla jsou veškeré detekované a reportované chyby s „áčkovou“ severitou opravovány prioritně. Pokud nastane situace, kdy je vyreportováno více „áčkových“ chyb, opravuje se chyba, která nejvíce brání v testování, a jejímž odstraněním se „odemknou“ další testy. Další hledisko může být časového charakteru, tzn. například se opraví chyba s předpokládanou delší časovou náročností. [11]

6.3 Nejznámější světové chyby

Nejprokazatelnější důvod proč kontrolovat systém, předtím než jej začnou používat koncoví uživatelé, prezentují následující historické chyby, které byly odhaleny až po jejich užívání.

Mariner 1

Roku 1962 v Americe NASA vypustila sondu Mariner 1 do vesmíru. Tato sonda měla proletět okolo Venuše, ale po necelých pěti minutách letu sonda změnila kurz a musela být obsluhou zničena. Po prošetření NASA zjistila, že chyba byla způsobena systémem, který špatně vyhodnocoval rychlost a vzdálenost. Konkrétněji bylo zjištěno, že celý neúspěch měla

na svědomí jedna pomlčka ve strojovém kódu. Tato chyba stála americkou vládu přes půl milionu dolarů. Nehledě na zmařený čas při výzkumu a vývoji sondy. [13]

Arien-5

Dalším podobným selháním se uvedli světu konstruktéři ESA, jejichž bezpilotní kosmická sonda Arien předčila i Mariner I. Arien-5 byl vypuštěn v roce 1996 a jeho let trval celých 40 vteřin. Hned po startu změnil směr své původní dráhy a začal se rozpadat, pak zahájil svou sebe destrukci. Chyba v tomto případě byla opět v konstrukčním kódu, tentokrát problém nastal v převodu dat. Arien pracoval s 64-bitovými čísly s pohyblivou řádovou čárkou a převáděl jej na 16-ti bitová celá čísla. Tato konverze dat byla pro 16-ti bitová celá čísla příliš velká a způsobila tak hardwarový kolaps, načež Arien havaroval. [14] [15]

Therac-25

Horor z medicínského prostředí, při němž zemřelo i několik lidí. Přístroj Therac-25 využívaný v letech 1985 – 1987 v USA a Kanadě, pracoval s radiací pro ozařování pacientů trpící rakovinovým onemocněním. Pacienti, kteří podstoupili léčbu v těchto letech, byly ozáření více než potřebnou dávkou a na následky ozáření zemřeli. Ačkoliv byla chyba v nedostatečném otestování hardwaru a softwaru, kdy ochrana proti předávkování nepracovala správně, lékaři zařízení plně důvěřovali, než předali problém k řešení. [16][17]

Y2K

Jedna z nejznámějších programátorských chyb, která se odehrála v roce 2000, kdy se kvůli úspoře místa použila zkrácená čísla pro rok, tzn. místo roku 1999 bylo použito označení „99“. Tento systém číslování však na přelomu 19. a 20. století ohrozil téměř celý svět. Programátoři nevzali v potaz, že rok 1900 a 2000 bude brán jako jeden a ten samý. Okolo této kauzy vznikly různé teorie chování softwaru na celé zemi. Mnozí se připravovali dokonce na konec světa. A jelikož se navíc tento problém začal řešit teprve až v druhé polovině 19. století, nikdo s přesností nedokázal říci, jak moc bude problém rozšířen a co všechno zasáhne.

Hrozilo odstavení pitné vody, nefunkční mhd, bankovní spojení atd. Lidé měli strach z kompletního selhání celého systému. Naštěstí tento problém neměl široké působení. Sice několik článků nefungovalo a padaly komplexní sítě různých dodavatelů, nejednalo se však o katastrofu v širokém spektru. Některé firmy s tímto problémem dokonce počítali a pracovali na jeho zabezpečení. U nás tento problém nebyl až tak zaregistrován, epicentrum bylo směrováno spíše na Spojené státy. [18]

6.4 Příčiny vzniku chyb

Nejčastější důvodem vzniků chyb při tvorbě softwaru je jednoznačně lidský faktor. Tak jak je software navržen, popsán a naprogramován je ovlivněno lidmi, kteří se na něm podílí. Každá přenášená informace může být chápána s procentuální odlišností v rámci sdílení mezi lidmi. Software sám o sobě funguje podle toho, jak je programován.

Zákonitě čím více se na tvorbě softwaru účastní lidí, tím více se dá očekávat problém, tzv. „komunikačního šumu“.

Zárodek softwaru vzniká při tvorbě specifikace, která popisuje vlastnosti a chování výsledného softwaru zadavatelem. Již zde se dá predikovat vznik prvních potencionálních chyb, které se mohou později projevit na výsledku. Zadavatel vyjádří svůj návrh a zanese jen do specifikace, kterou obdrží následně developer a podle této specifikace začne software vyvíjet. Neobsahuje-li specifikace podrobné informace, tak vývojář nebo tým vývojářů pracuje s údaji, která má k dispozici. Zároveň nedojde-li ke společné komunikaci zadavatele a vývoje, návrh softwaru se v tomto případně může výrazně lišit od předpokladů, které byly očekávány. Takto navržený a implementovaný software na testovací prostředí přichází k testerovi, který podle svých znalostí a zkušeností může chybu odhalit. Horší variantou je, že ani tester chybu neodhalí a software ponese svou chybu až do konečné fáze, kde chybějící funkcionalitu nebo jiný nesoulad rozpozná až uživatel v lepším případě ještě zadavatel. Tento problém přímo souvisí i s nákladovostí na takovouto chybu. Náklady na detekci chyb logaritmicky stoupají v rámci času jejího odhalení. Čím později je chyba nalezena, tím větší je množství vynaložených peněz na její opravu.

Příčinou chyb, které tester odhalí při testování, nemusí přímo pocházet pouze z nedostatečné specifikace. Jsou-li podklady k návrhu softwaru dostatečné, ještě to není zárukou zcela bezchybně naprogramovaného softwaru. Vývojář je také jen člověk. Vzniklá chyba může být způsobena jeho nedostatečnými znalostmi a zkušenostmi či pouhou nedbalostí, stejně jako u testera. Ačkoliv odpovědností testera je, aby software byl před vyprodukováním schopný vyhovět požadavkům uživatele, tudíž aby obsahoval co nejméně chyb a dalších nesrovnalostí.

Pomineme-li hlavní důležité články, kde s největší pravděpodobností narazíme na vznik chyby, tak i nekompletní koordinace lidí, času a prostředků přispívá svým dílem k chybovosti v procesu tvorby softwaru.[1][3][5]

7. Práce testera

Po návrhu softwarové specifikace a vývoje přichází práce pro testování. Ve všech fázích procesu probíhají testy na různých úrovních, viz kapitola 6. Úkolem testera za danou oblast je akceptace a posunutí softwaru do další fáze. Tuto akceptaci je třeba dokázat na úrovni stanovených požadavků na software. [1]

Tester musí znát jednotlivé funkčnosti softwaru, které testuje. Informace se dozví ze zadání softwarové specifikace a dalších doprovodních dokumentací. Má stanovené požadavky, nad kterými provádí své testy a které mají být splněny. Dále porovnává své výsledky s těmi očekávanými. Rozhodne-li v nesoulad mezi těmito výsledky, trackuje chybu k jeho ohnisku a veškeré informace o této chybě vrací odpovědné osobě. Odpovědná osoba může být například zpět vývojář, administrátor nebo vlastník.

Dobrý tester je přemýšlivý a simuluje situace, za kterých má software fungovat podle zadání či nemá fungovat a s jakými podmínkami. Objeví-li chybu, musí ji umět i najít a dostatečně prezentovat ostatním článkům, tak aby byla chyba opravitelná. Po opravě dokazuje retestováním akceptaci opravy. [1][2]

7.1 Vlastnosti testera

Předpoklady kladené na práci testera by měly být pečlivě promyšlené. V závislosti na různé oblasti v testování se můžou vlastnosti na testera mírně odlišovat.

Obecně vzato pro všechny oblasti v testování platí stejné pravidlo, jaký by měl tester být. Jelikož je výsledkem kvalita, tak neodmyslitelnou vlastností je pečlivost. Jak již bylo zmíněno, tester analyzuje software, tedy je očekáváno analytické myšlení, které tester uplatní při detekci chyb a případné prevenci. Verifikuje-li tester software, na kterém pracoval, přejímá za něj svou odpovědnost a v závislosti na jeho kvalitách ručí svou prací za spolehlivost svého výkonu. Dobré je zmínit i časovou spolehlivost, tedy dochvilnost. Jelikož software prochází cyklem od vývoje až po vydání do produkce, jsou veškeré činnosti rozloženy do časového harmonogramu. Splnění a nesplnění stanového plánu ovlivňují všichni účastníci tohoto cyklu. Naivním tvrzením je, že vše se stihne podle plánu, ačkoliv jsou stanovené i časové rezervy. Každopádně správné načasování, organizování a řízení může značně přispět k plánovaným pozitivním výsledkům. [2]

7.2 Znalosti testera

Požadavky na znalosti testera se liší v rámci pohledu na software. Pro ověřování strojového kódu programu je důležitá znalost programovacího jazyka/ů, v němž je software napsán. Takovýto tester pracuje se softwarem na úrovni strojového kódu, kde ověřuje třídy, jednotlivé objekty, komponenty, metody, moduly apod. Ke své práci využívá dalších nástrojů, simulátorů či ovladačů. Znalosti jsou na této pozici velmi technické. Nejčastěji jsou takovými testeři sami programátoři.

Při testování implementace softwaru do simulovaného prostředí je potřeba, aby tester dobře znal funkčnost systémů, se kterými bude software spolupracovat a uměl tak detekovat chyby při nasazení a spolupráci těchto systémů. Tato znalost je na bázi zkušenosti. Jedná se o znalosti například spolupracujících databází, serverů, případně dalších specifických systémů, jejichž funkčnost je z pohledu testera uživatelská až administrátorská. [1][2]

7.3 Kompetence

Tester jako jedna součást celého projektu má své určité kompetence dané stanovy své firmy. Zodpovědností testera je zejména verifikace, detekce, akceptace a prevence. Konkrétněji se jedná o zodpovědnost při ověřování softwaru, čímž je myšlena jeho funkčnost, s tím souvisí i správné odhalování chyb v softwaru. Ověří-li tester software a uzná jej za akceptovatelný, bere na sebe zodpovědnost v podobě prokázané funkčnosti. A nakonec prevence, čímž se snaží zabezpečit nežádoucí chování softwaru.

Různorodé kompetence může mít software v závislosti na dané firmě. Tester může mít své kompetence omezené, což se může být například u velkých firem, kde je tester součástí testingové oblasti zaměřující se na určitý projekt. Nebo naopak může být tester v jiné firmě více alokován do různých projektů nebo oblastí.

Tyto kompetence se liší ve spoustě faktorů. Jednak ve zkušenostech a znalostech testera. Například tester na seniorní pozici bude mít, díky své delší praxi, vyšší kompetence než takový junior tester, který teprve začíná. Nebo v rámci firmy, kde korporátní gigant bude mít vlastní oblast testování a testeři zde jsou rozděleni do daných projektů, nad nimiž pracují test koordinátoři. Ale naopak malá firma s menším počtem zaměstnanců může mít na jedné pozici testera, vývojáře a koordinátora. Jelikož obsah práce není tak markantní jako u gigantu či větší firmy.[1][3]

8. Životní cyklus vývoje softwaru

Software jako každý produkt, byť nehmotného charakteru, má svůj životní cyklus. Na počátku stojí nějaká idea, která je postupně realizována do konečné podoby. To, jakým způsobem bude tato idea zrealizována, může mít několik podob, které jsou popsány v následujících modelech. Jednotlivé modely popisují metodiku tvorby vývoje softwaru z pohledu historického vývoje. Uplatnění těchto metodik je individuální, ačkoliv některé metodiky jsou ne příliš vhodným modelem pro výsledný kvalitní produkt. [1]

8.1 Model velkého třesku

Tento model popisuje životní cyklus vývoje softwaru jako teorii velkého třesku. Veškerá energie se soustředí pouze na vývoj softwaru za co nejkratší dobu. V tomto modelu neexistuje žádná specifikace produktu ani žádné plánování, software je specifikací sám o sobě. Principem je zadání, které obdrží vývojář a podle toho vytvoří software, který pak předá zadavateli. Teprve až v okamžiku, kdy zadavatel obdrží objednaný produkt, zjišťuje jeho nedostatky. Předmětem toho modelu totiž nebývá zahrnuta oblast testování, maximálně jen v základní verzi, a to, že software alespoň funguje. Zadavatel, který v tomto případě může být rovnou uživatel, teprve testuje funkčnost softwaru. Nejsou zde zahrnuty ani oblasti analýzy návrhu, schvalovací procesy nebo dopadové analýzy.

Výhodou se dá tedy o tomto modelu říci, že je nenáročný z hlediska organizace. Nevýhod však je více, ačkoliv je proces životního cyklu produktu jednoduchý, výsledek, za který musí zadavatel zaplatit, už nemusí být příliš valný. Budou-li vývojáři sebelepší, neexistuje stoprocentní jistoty bezchybného produktu. Tím hůře, když zákazník objevuje chyby sám a do poslední chvíle není seznámen s funkčností softwaru. Tento model je asi vhodný pro dobrodružné a hlavně bohaté zákazníky, kterým tolik nezáleží na výsledku.

[1]

8.2 Model programuj a opravuj

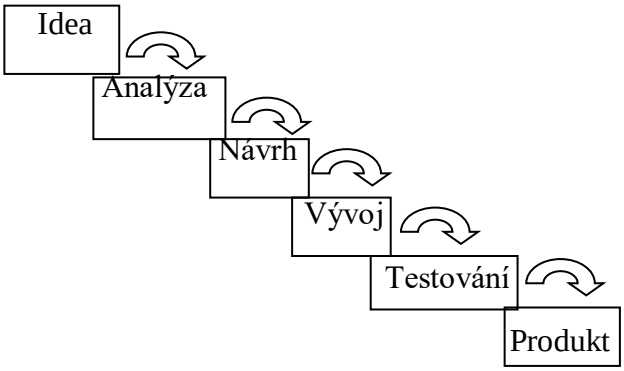
Model „programuj a opravuj“ zahrnuje do své koncepce i testování přímo spojenou s vývojem. To znamená, že programovaný software je určité fázi svého vývoje předáván testerovi, který ověřuje dosavadní funkčnost. Nalezené chyby posílá zpět do vývoje na opravu, avšak nezávisle na testování programátoři stále pracují na vývoji softwaru, takže tester obdrží novou verzi softwaru bez ohledu na předchozí opravy. Nové verze mohou nést

sebou různé modifikace či nové funkce oproti předchozí testovací verzi. Dochází zde ke značnému chaosu, kdy tester ani neví, co má testovat, respektive začíná stále od začátku.

V rámci malých projektů nebo prototypních ukázek může být tento model dostačující, protože zde není velká náročnost na software. Model „programuj a opravuj“ používá částečně organizační strukturu jako je dokumentace a plánování. [1]

8.3 Vodopádový model

Vodopádový model je jedním z nejpoužívanějších modelů nejen u programování. Software prochází úrovněmi, které plynule přecházejí až do úplně realizace. Představa vodopádového modelu je následující:



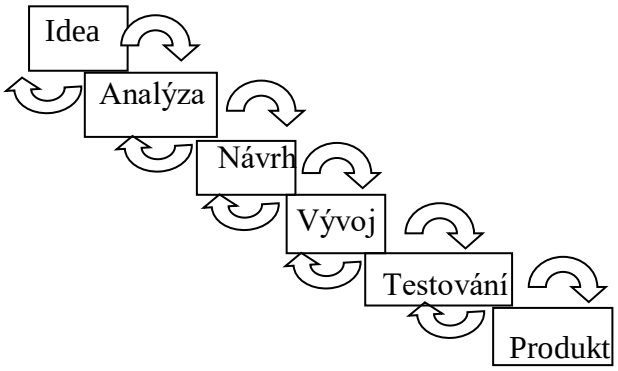
Obrázek č.2 – Vodopádový model

Na začátku je idea, jak by měl software vypadat. Z idey je vypracována analýza, která se po schválení mění na návrh. Navržený software je předán k vývoji a po dokončení vývoje přechází rovnou k testování, kde poslední fází je hotový produkt.

Proces vodopádu klade důraz na pečlivost, tak aby jednotlivé úrovně mohly pracovat samostatně. To znamená, že software se z jedné úrovně nedostane do druhé, dokud vše není pečlivě zpracováno. Software prochází jednotlivými úrovněmi postupně a nedochází zde ke zmatečnému testování a vracení nebo nedostatku informací, který by mohly velmi ovlivnit výsledek. Výhoda spočívá ve velmi propracovaném postupu, který je podrobně dokumentován a hlídán, tak aby výsledek odpovídal ideji co nejpřesněji. Nevýhodou u tohoto modelu je chybějící zpětná vazba. Vodopádový postup je striktní a drží stejný směr, nikdy zpět. Problém může vyvstat v případě, že dojde k nekompletnímu předání informací a software pak pokračuje do dalších úrovní s neočekávanou chybou, která se odhalí až při předání do produkce. Tímto i vzniká nákladovost celého projektu, který se tak může stát méně výnosným, než se předpokládalo a bylo navrženo na začátku.[1]

8.4 SSADM (Structured Systems Analysis design and methodolgy)

Tato metoda procesu vývoje softwaru vznikla Anglii a postupně se rozšířila i do Evropy. SSADM model popisuje vodopádový model, ale s praktickým vylepšením tohoto modelu. Model počítá i se zpětnou vazbou mezi úrovněmi, která ve vodopádovém modelu chybí. Náročnost na řízení celého projektu u vodopádového modelu je v tomto případě odstraněna lepší komunikací a lepší koordinací mezi oblastmi. Jednodušší přístup a kontrola zajišťuje lepší výsledky v dalších fázích a pozitivně tak ovlivňuje finální výsledek. Vzniklé nedostatky během vývoje jsou rychleji detekovány, čímž se snižuje nákladovost na celý projekt a tím i samotná kvalita produktu.



Obrázek č.3 – SSADM model

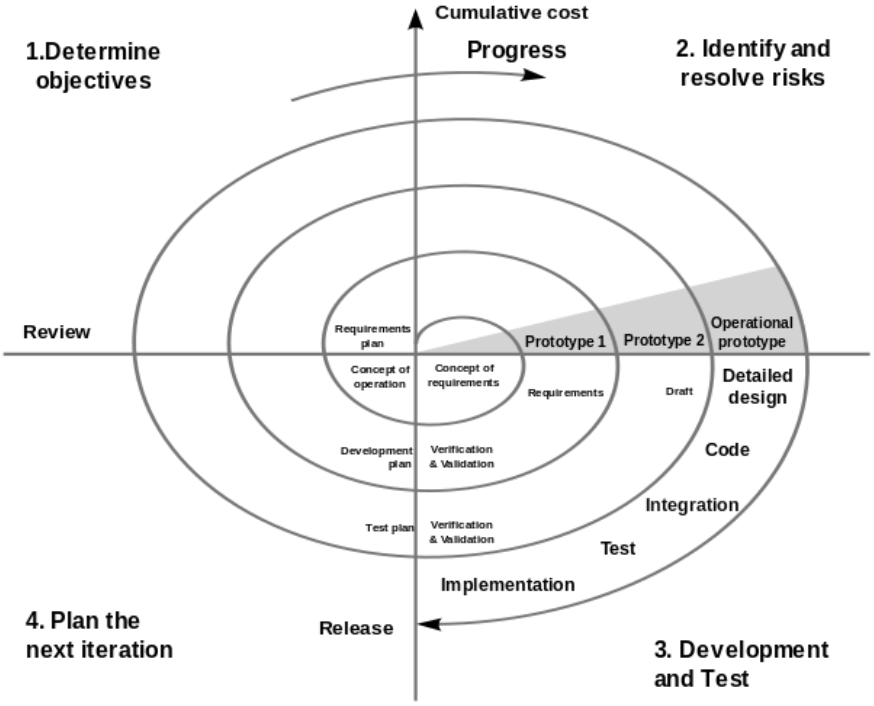
[19][20]

8.5 Spirálový model

Barry Boehm přišel s návrhem nazvaný „A spiral model of software Development and Enhancement“. Přeloženo jako „Spirálový model vývoje a zdokonalování softwaru“.

Model je založen na těchto 6 bodech:

1. Určení cílů, alternativních řešení a omezení
2. Rozpoznávání a řešení rizik
3. Vývoj a testování aktuální úrovně
4. Plánování další úrovně
5. Rozhodování o přechodu na další úroveň



Obrázek č.4 – A spiral model of software Development and Enhancement

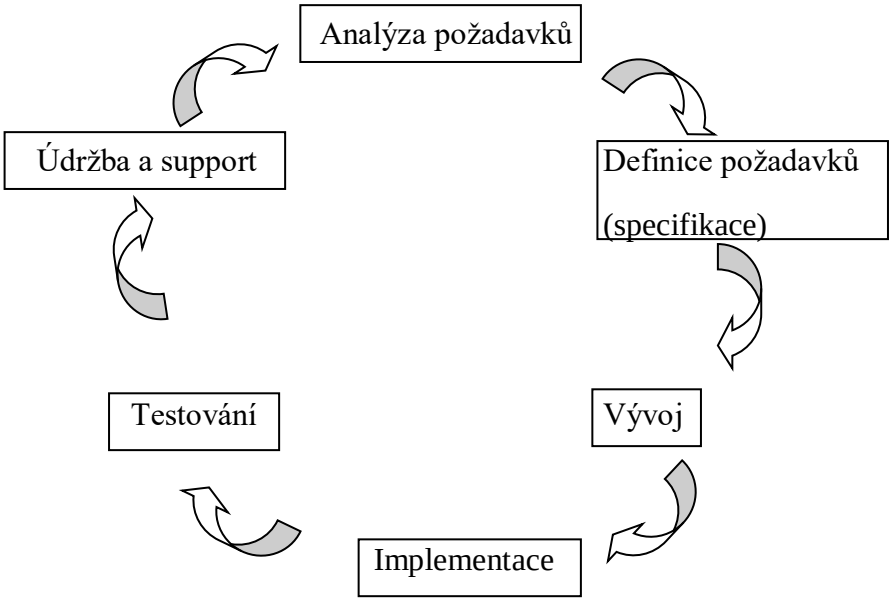
Tento spirálový model pracuje na základě dílčích úloh, které procházejí celým cyklem a teprve po otestování a případných opravách je akceptován. Jak je zobrazeno na obrázku, začátek projektu vzniká ve středu grafu, kdy je vytvořen koncept požadavků, potřebných operací a je vytvořen plán. S nabývající spirálou je vytvořen prototyp, který je validován a verifikován podle stanovených požadavků. Na vývoj je prototyp dodán ke zpracování a je požadován plán vývoje. Vývoj vytvoří plán, který opět projde verifikací a k již vytvořenému návrhu je připraven testovací plán. V poslední fázi jsou doplněny k prototypu návrhy detailu a software se začne vývojem programovat. Následuje integrace do prostředí a proces testování, implementace a „vyreleasování“, tedy akceptace a nasazení do produkce.

S narůstající spirálou narůstají i náklady na software, které ale můžou být včas podchyceny v případě náhlého nárůstu, a to operativním detekováním problémů a stanovením rizik. [21]

8.6 Iterační model (RUP, SCRUM)

Iterační model je podobný spirálovému modelu, s tím rozdílem, že jednotlivé úrovně mezi sebou zpětně komunikují a vývoj softwaru nekončí nasazením do produkce. I model

tedy pracuje převážně se zpětnými vazbami, které vytvářejí další požadavky od zákazníků. Proces vývoje se takto cyklí v závislosti na požadavcích.



Obrázek č.5 – Iterační model

Podle obrázku je viditelný postup vývoje softwaru. Na začátku se analyzují požadavky od zákazníků, které jsou zapracovány do konceptu specifikace. Vypracovaná specifikace softwaru přechází k návrhu softwaru do vývoje, kde je software naprogramován. Vyhotovený software je dále implementován do prostředí a následně testován. Poslední fáze údržba zahrnuje podporu softwaru po nasazení do produkce a užívání. Může se stát, že nebyly vyhotoveny všechny plánované funkčnosti nebo se, na základě požadavků zákazníků, navrhne další funkčnost softwaru. K tomuto slouží fáze údržby neboli podpory. Jejímž úkolem je sběr těchto požadavků jako podklad pro další cyklus vývoje. [22]

8.7 Extrémní programování

Hlavním předmětem extrémního programování je kompletní analýza požadavků, která je precizně zpracována a její vyhotovení se dále nemění. V hotovém stavu je pokládána za zdroj pro další úrovně v cyklu produktu.

Jednoduchost v modelu značí pracovat rychle a efektivně bez jakýchkoliv omezení, po analýze požadavků je na vývoji sestavovat kód přímo odpovídající požadavkům, tak aby každý naprogramovaný kód softwaru mohl být předán k testování a pak k akceptaci. Během

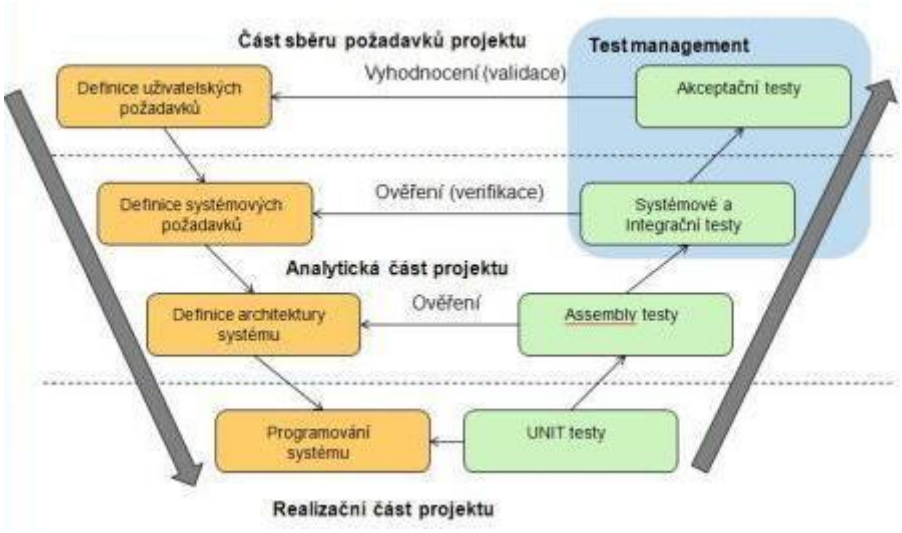
akceptace již tým vývojářů pracuje na dalším části kódování softwaru. Těmto dílčím naprogramovaným částem, které jsou otestovány a akceptovány, se říká balíčky a jedná se o tzv. „sprinty“. Vývoj neustále pracuje na programování těchto balíčku a ihned po dokončení jej posílají do další fáze. Integrování jednotlivých „sprintů“ do systému může u extrémního programování probíhat i několikrát denně. Výsledkem tohoto modelu je efektivnost, kdy se nečeká na žádné kompletní otestování softwaru. Testování malých částí také napomáhá k rychlejšímu odhalování chyb a také rychlejší opravě, takže celý cyklus životního modelu softwaru se zrychlí. Důležitá je však správná komunikace a zpětná vazba, to vše může mít v opačném případě za následek nechtěný výstup. Velkou roli hraje i rozhodování, v tomto modelu je třeba jednat operativně. Například pokud má vývojář problém s kódováním a již celý den se mu nedaří kód rozchodit, je důležité se rozhodnout, zdali má smysl dále pátrat či začít od začátku a jinak.

V extrémním programování se používá párové programování. Na kódování softwaru se podílí dva vývojáři, kteří mají rozdělený postup práce. Každý se zaměřuje na něco jiného, ale programují společně. Tím vzniká dvojí kontrola. U extrémního programování se lze setkat s dalšími procesy při kódování, jako je například přepis kódu (refaktorizace) nebo testy před zahájením kódování.

Výhodou extrémního programování je efektivnost, rychlost a snižování nákladů na software, ale je kladen velký důraz na programování a tím na velkou zátěž programátorů a součinnost testerů, kteří pracují ve velkém presu s přesčasy. Dlouhodobé udržení takového pracovního tempa je pro člověka velmi náročné. [23][24]

8.8 V-model

Testování pomocí V-modelu zahrnuje testování oproti jednotlivým fázím vývoje. Tento postup zaručuje eliminaci chyb již v raných fázích vývoje softwaru a jejich rychlou nápravu. Cílem modelu je zaručit co nejmenší chybovost na výstupu softwaru.



Obrázek č.6 – V-model

Z obrázku je viditelný postup testování a vývoje. Z levé strany vede šipka směrem dolů a popisuje stav vývoje softwaru. Na pravé straně je rostoucí šipka, která zobrazuje testovací fáze. Vrcholem modelu je naprogramovaný software, který je testován Unit testy, viz kapitola 9.1. Unit testy zajišťují korektnost naprogramovaného softwaru. Další úroveň směrem nahoru, popisuje ověřování správné navržení architektury softwaru. Tato úroveň analyzuje požadavky na systém. V této úrovni se taktéž nachází systémové a integrační testování, které ověřuje definované požadavky. Poslední úrovní jsou akceptační testy, které ověřují, že stanovené uživatelské požadavky jsou splněny a je možné software produkovat. [25][26]

9. Druhy testování

Existují různé druhy testování softwaru z různých druhů úhlů pohledu na software. Tyto testy jsou odrazem prostupu softwaru testováním z pohledu určitých hledisek, která jsou na testovaný software kladeny.

9.1 Jednotkové testování (Unit testy)

Prvotní testování softwaru po dokončení vývoje. Software se v tom stádiu nachází ve stavu, kdy byl prvotní záměr o tvorbu softwaru splněn a nyní je na čase zjistit, zdali funguje

na úrovni strojového kódu. Jednotkové testování neboli unit testování, jak z názvu vyplývá, se zaobírá testováním jednotek. Tedy zde se provádí testy na úrovni programového kódu a ověřuje se správnost a přesnost naprogramovaného softwaru za jednotlivé dílčí části. Tyto části se testují nejlépe izolovaně od zbytku programu, tak aby bylo prokazatelné správné fungování bez návaznosti na další část. Tímto systematickým testováním lze odhalit chybu již v prvopočátku. Chyba je zde detekovatelná a ihned opravitelná. Unit testeři používají pro takovéto testování doprovodných programů, které slouží jako simulátory. Za unit testera se dá považovat samotný vývojář, coby dodavatel softwaru, jehož úkol je dodat funkční produkt pro další testování. [1][5]

9.2 Integrační testování

Integrační testy jsou používány pro ověření funkčnosti celého softwaru jako celku. Integrační testy jsou prováděny již pro kompletní software, který byl otestován za jednotlivé oblasti v unit testech, a nyní pracuje komplexně. V této úrovni testování je software stále izolován od ostatních systémů, které by jej mohli ovlivňovat ve funkčnosti. Po akceptování integračních testů, včetně oprav detekovaných chyb, přechází software k systémovému testování. [1][5]

9.3 Systémové testování

Systémové testování napovídá, že kompletně otestovaný software jako celek je v této fázi připojován k ostatním systémům, tak aby mohl fungovat plnohodnotně. Jedná se tedy o testování implementace softwaru do infrastruktury. Systémové testování je jedno z nejsložitějších fází, kde může často docházet v problému komunikace v rámci ostatních systémů. Software zde musí umět komunikovat s více programy psanými i v jiných programovacích jazycích. Systémové testování je také velmi náročné na časový harmonogram, protože nikdo předem nedokáže odhalit případnou kolizi. Dají se předpovídat rizikové oblasti, kterým se věnuje vyšší priorita při implementaci. Systémoví testeři zde mají nejvíce práce a musejí velmi pečlivě otestovat veškeré spoje komunikace, tak aby ve výsledku fungovala celá infrastruktura, která bude produkována. [1][5]

9.4 Akceptační testování

Akceptačním testováním se potvrzuje dodávka softwaru dodavatelem, kdy software je již implementován a funkční na testovacím prostředí. Předmětem akceptačních testů je akceptace softwaru zadavatelem, který kontroluje splnění zakázky podle stanovených požadavků. Testeři v této fázi mohou ještě nalézat nedostatky, které mohly vyplynout v rámci špatné dokumentace, informovanosti nebo nedbalosti. Může se také jednat o detekci méně prioritních chyb, se kterými zadavatel nesouhlasí. Například grafické zobrazování. Uzná-li tester, že je vše splněno a schválí stav softwaru, přechází se do fáze nasazování do produkce k užívání reálnými uživateli. [5]

9.5 Regresní testování

Regresní testování probíhá na již zaběhnutých systémem, které jsou odrazem systémů na produkci. Každá modifikace daného softwaru je potřeba otestovat, zdali neovlivnila chování již zavedeného a funkčního systému. Regresní testování probíhá na testovacím prostředí, kde je testována funkčnost všech stávající systémů. Neprobíhají zde již unit testování, pouze systémové testy. Většinu regresních testů je možné automatizovat, protože sada testů pro regrese, v závislosti na velikosti celé infrastruktury a možnosti zasažení, může být velmi obsáhlá a mohou vzniknout nechtěné náklady, v rámci časové náročnosti, na výkon testera/testerů. [5]

9.6 Post deployment testování

Finální fáze nasazení softwaru do produkčního prostředí. Software otestovaný ve všech úrovních testování (Unit, Integrační, Systémové, Akceptační) je implementován do produkčního prostředí a je systémově testován stejně jako na testovacím prostředí. Důvodem může být nesourodost mezi testovacím a produkčním prostředí, například v hardwarovém vybavení. Testování nasazení do produkce probíhá většinou v nejméně využívaný čas uživateli, tak aby nebyla příliš narušena běžná funkčnost. Po schválení je software jako produkt využívatelný. [5]

9.7 Testování výkonnosti

Jelikož software může být používán určitým množstvím lidí, je potřeba otestovat hraniční výkonnost, při které hrozí selhání. Výkonnostní testy informují kolik je software schopný zvládnout úkonů za určitý časový interval, například přihlášení do systému. Posouzení výsledků odezvy odhaluje zvládnutelný výkon při zátěži. Zdali je odezva velmi pomalá či dokonce hrozí selhání. Jaké jsou kapacity do budoucna nebo jestli je potřeba nějaké navýšení kapacit. [5]

9.8 Smoke testy

Smoke testy se vztahují na takový software, který prošel všemi fázemi testování a akceptace. Tento software již je v oběhu, ale může se stát, že některé jeho funkce nejsou plně využívány a je potřeba ověřovat, nejlépe před novou implementací, zda tyto části fungují. Smoke testy tedy zjišťují aktuální funkčnost používaného softwaru. To že se software vyvine, otestuje a produkuje, neznamená, že nemůže být ovlivněn působením svým okolím (výpadky, uživatelé, změna nastavení, …). Pravidelnými smoke testy lze ověřit a potvrdit dosavadní funkčnost softwaru. [1]

10. Organizační struktura

V metodologii procesu vývoje softwaru tento cyklus doprovází lidé s určitými kompetencemi. Podle rozsahu vyvíjeného softwaru je potřeba koordinace a řízení celého procesu. K tomu slouží organizační struktura, která v rámci quality assurance pomáhá urychlovat a zefektivňovat práci při vývoji. [1]

10.1 Zákazník/Zadavatel

Zákazník nebo zadavatel je vlastníkem softwaru. Sbírá a zadává požadavky na software. Jeho kompetence jsou nejvyšší a veškerý vývoj se řídí jeho instrukcemi.

Jak tvrdí citát T. Bati: „Náš zákazník, náš pán.“ Zadavatel je přítomný v celém procesu vývoje a má možnost zasahovat do všech fází. Bohužel ne vždy je zadavatel technicky zkušený, a vznikají tak nesrovnalosti, co vlastně zadavatel žádá. [1]

10.2 IT Analytik

Analytik má na starosti rozbor zadaných požadavků a sepsání podrobné specifikace, která informuje další články o funkčních požadavcích na software. Navrhuje schématické části funkčností softwaru. Kontroluje možnosti vývoje v rámci podniku, definuje rozhraní, apod. Výsledkem práce IT analytika je kompletní dokumentace potřeba pro vývoj a testing. [1][27]

10.3 IT Architekt

Po sestavení dokumentace je na řadě IT architekt, který navrhuje a řídí plánování integrace softwaru do systému. Sestavuje schémata fungování softwaru po integraci do prostředí a navrhuje jeho nejefektivnější nasazování. Z tohoto návrhu čerpají další oblasti, jako je developer, administrátor, tester nebo project manager. Předpokládá se velmi dobrá znalost systémového prostředí, tak aby návrh dával smysl a byl reálný. [28]

10.4 Project manager

Projektový manažer sestavuje schéma a sestavení týmu lidí, podílející se na projektu. Rozpracovává daný projekt na určité části, které zaplánovává do časového harmonogramu. Dohlíží na tento harmonogram a řídí práci na projektu. Podílí se na finanční náročnosti celého projektu. Informuje zadavatele o stavech, rizicích a plánech vývoje softwaru. [29]

1.110.5 Test Manager

Manažer oblasti testování má na starosti řízení testerů a celého testovacího prostředí. Komunikuje se všemi ostatními články a je odpovědný za veškerou činnost a rozhodnutí při

testování. Jeho úkolem je dohlížení a dodržování metodiky procesu vývoje softwaru. Stejně tak i dodržování stanovených standardů. Jako každý manažer i v testování musím být takový člověk odpovědný, pečlivý a mít komunikaci na velmi dobré úrovni. Managování lidí je náročná disciplína a test manager byl měl umět dobře vést lidi ve svém týmu, aby dosahovali očekávaných výsledků. [30]

2.110.6 Test koordinátor

Ve fázi testování je potřeba řídit progresy testů. Test koordinátor sestavuje časový harmonogram a plánuje začátek a konec testování. Jelikož se na testování může podílet spoustu lidí, je nemožné a nezodpovědné spoléhat na neřízenost. Koordinátor zjišťuje stavy testů a pomáhá při řešení vzniklých problémů, tak aby byl splněn plán. Je to obdobná práce jako project manager avšak s nižšími kompetencemi a je využívána spíše u méně rozsáhlejších projektů, kde není tak velký nárok na řízení. Výstupem test koordinátora jsou statistiky testování, které slouží jako podklady pro další testovací období a informovanost pro ostatní zainteresované.[1]

3.110.7 Test Engineer

Pod názvem test engineer se skrývá pozice testera, který podle specifikací vytváří testovací případy a v období testování provádí exekuci. Ověřuje tedy funkčnosti softwaru a nalézá chyby, které reportuje administrátorovi nebo vývojáři. Spadá pod test managera nebo pod projektového managera. Jeho zodpovědností je pečlivě prověřit software a akceptovat funkčnost. [1][2]

10.8 Developer

Developer podává návrh softwaru, který programuje na základě požadavků zadavatele a řídí se časovým harmonogramem a metodikou projektovým nebo test managerem. Úzce s těmito oddělením spolupracuje a je odpovědný za technickou stránku softwaru. Na pozici developera je žádána vysoká technická znalost a zkušenost.[1]

11. Automatizované testování

Součástí testování jsou i automatizace testů. Většina testů používaných v jednotlivých úrovních ověřování softwaru je prováděna manuálně pomocí vhodných nástrojů. Unit testování provádějí testeři pomocí nástrojů a simulátoru na testování dílčích částí kódů, které částečně provádí program, ale částečně tester, který zejména vyhodnocuje výsledky a opravuje chyby. U systémového a integračního testování tester používá prostředí a software, tak jak jej vidí uživatel a testuje jeho funkčnost v rámci celého systému. Samotný program však neotestuje nové nasazení podle očekávání. Existuje mnoho aspektů a změn, se kterými během testování program pro automatizace nepočítá a nedokáže test správně vyhodnotit. V tomto případě jen slepě ověřuje, byť s pozitivním výsledkem. Zkušený tester dokáže poznat nesoulad lépe, případně dokáže zhodnotit závažnost chyby.

Automatizované testování je vhodné na regresní testování. Tyto systémy jsou již zaběhnuté, a pokud se netestuje jejich nová funkčnost nebo změna, mají standardní chování a nepředpokládá se u nich změna v tomto chování. Zde je tedy vhodné použít automatizaci, která bude mít stanovené ověřené vstupy a hodnocení výstupu má smysl. Velkou výhodou automatizací je, že jsou sníženy nároky na pracnost testera a tím i náklady na celé testování. V rámci těchto automatizací můžou testy probíhat v jakýkoliv čas, nejsou omezené lidskými potřebami. Tak jak jsou testy pro automatizace napsány, tak s takovou přesností a správností budou zpracovávány. Je tu tedy lidský zásah a je potřeba spoléhat na zkušenou a dobrou práci automatizéra.

Automatizované testy jsou vhodné na provádění zátěžových nebo výkonnostních testů. Díky své nenáročnosti na čas a výkon jsou nástroje pro automatizaci velmi výkonné při testování oproti manuálnímu testerovi, kterému by stejný počet testů trval několikrát déle. Automatizované testy se hodí i pro ověřování funkčnosti stávajícího produktu, a to na základě smoke testů. Tedy použití základní sady testů pro ověření hlavní funkčnosti softwaru. [4]

11.1 Nástroje pro automatizované testování

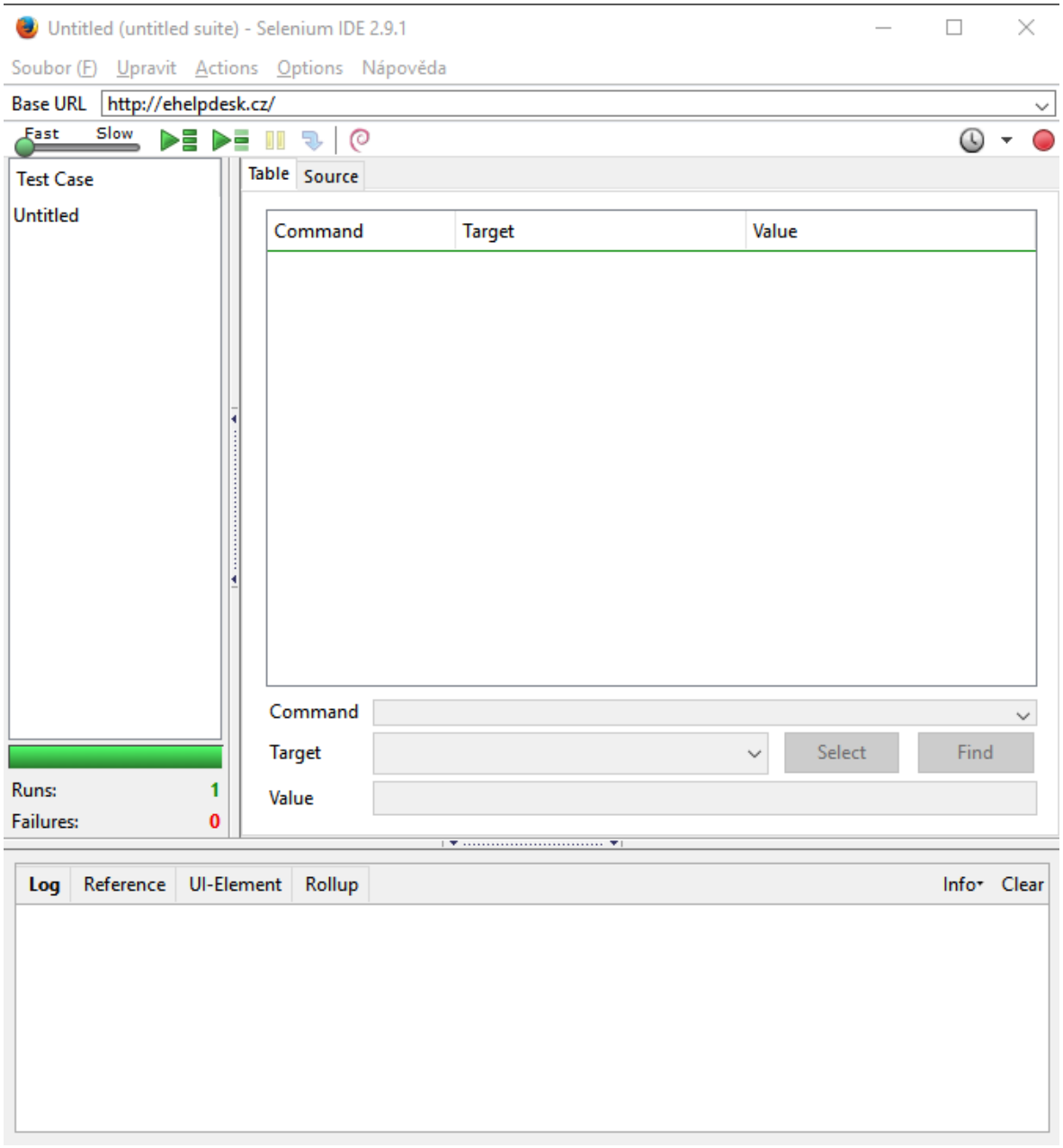
Pro automatizaci testů, které jsou prováděny manuálně je potřeba vhodného nástroje. Automatizace jsou prováděny pomocí programovacího jazyka a existuje několik programů, které jsou na automatizace testů navrženy.

11.1.1 Selenium IDE

Selenium IDE (Integrated Development Enviroment) je prototypový nástroj pro vytváření testovacích případů pro webové aplikace. Jedná se o plugin internetového prohlížeče Firefox, který poskytuje snadno použitelné rozhraní pro psaní skriptů. Funguje na bázi nahrávání uživatelských akcí, které uživatel vykonává. Tento skript lze dále upravovat podle potřeb. Nevýhodou tohoto nástroje je, že se zde nadají používat iterace či podmíněné příkazy. Pro praktickou část však postačí činnost tohoto nástroje pro demonstraci automatizovaných testů oproti manuálním.

Selenium IDE obsahuje možnost programovat skripty v několika jazycích, kterýmiž jsou:

- C#
- Python
- Java
- Javascript
- Ruby



Obrázek č.7 – Selenium IDE

Po instalaci Selenium IDE, který byl doplněn přes rozšíření Firefoxu, je nástroj spuštěn z horní lišty prohlížeče. Jeho grafická podoba je vidět na obrázku. Nástroj obsahuje hlavní menu, z něhož lze vytvářet testovací případy (test case) nebo celé spoustitelné sety. Dále je možné pouště různé akce jako jsou spustětí test case, celého setu nebo jen konkrétního kroku.

V levé části je pole, kde se shromažďují všechny test casy podle názvu. Je možné mezi nimi překlikávat a editovat je.

Uprostřed se nachází hlavní pole s příkazy. Ty je možné zadávat buď pomocí přímo nahráváním (record) v podobě malé červené tečky vpravém rohu nebo ručně pomocí našeptávače. Vždy je nutné zadat příkaz (command), který určuje, co se v tomto kroku bude provádět. Dále target (cíl), čímž je myšlen vstupní zdroj, kde má nástroj příkaz vykonat. Cíl je může být označen pomocí id, třídy, odkazu, atd. Pole hodnota je již nepovinné v rámci příkazu, pokud jej příkaz vyžaduje, nesmí být hodnota prázdná. Například jednáli se o příkaz click, není potřeba uvádět hodnotu. Nástroj pouze klikne na danou oblast. V případě, že bude příkaz obsahovat select, zde již musí být přiřazena hodnota, kterou má vybrat.

Je možné si zkontrolovat před spuštěním, že zadaný příkaz je na očekávaném místě, k tomu slouží tlačítko find. To označí místo, které bylo definováno.

Před prvním spuštěním celého test casu je možné si také ověřit funkčnost jednolivých kroků, a to kliknutím na daný krok pravým tlačítkem a příkazem „execute this command“. Nástroj provede exekuci a ihned je vidět, zdali je krok validní.

Další pomůckou při exekuci testů je log, který se nachází dolní části. Zde jsou evidovány všechny kroky a případně i chyby, který vznikly při exekuci. [31]

11.1.2 Selenium WebDriver

Selenium WebDriver je z rodiny vývojářů, kteří vytvořili i Selenium IDE. Na rozdíl od svého kolegy je mnohem sofistikovanější a nabízí více využití v programování. Selenium IDE bylo vytvořeno pro jednoduché a rychlé navrhování testů. WebDriver nabízí integrované API (Application Programming Interface) a lepší podporu pro dynamické webové stránky. Cílem vývojářů bylo podporovat dobře designované objektové orientované prostředí, které řeší problémy při rozšiřujícím se sortimentu webových aplikací.

Selenium WebDriver pracuje na přímém volání prohlížeče a využívá jeho nativní podporu pro automatizace, tyto funkce jsou v závislosti na používaném prohlížeči. Stejně jako v Seleniu IDE je možné pracovat s několika typy programovacích jazyků:

- Java
- C#
- Python
- Ruby

- Php
- Perl
- Javascript

Selenium WebDriver umožňuje programovat v těchto jazycích po stažení potřebných knihoven a doplňků. Nejsou tedy obsahem samotného programu. Všechny tyto moduly jsou dostupné na webových stránkách poskytovatel selenia http://docs.seleniumhq.org/.

Pro založení testovacího případu (test case) nebo celé sady (test suit) je potřeba vytvořit a pojmenovat projekt. K tomuto projektu se vytváření samotné testy, které pak tvoří stromovou strukturu. Obsahuje okno pro psaní skriptů i logovací okno. [31

11.1.3 SoapUI

Software SoapUI je volně dostpný nástroj pro automatizované testování. Je vhodný pro funkční a zátěžové testy. Zaměřuje se na testování webových služeb nebo rest služeb. SoapUI nabízí také přívětivé a nenáročné uživatelsky grafické prostředí pro své uživatele. Software podporuje všechny standardní protokoly a webové technologie.

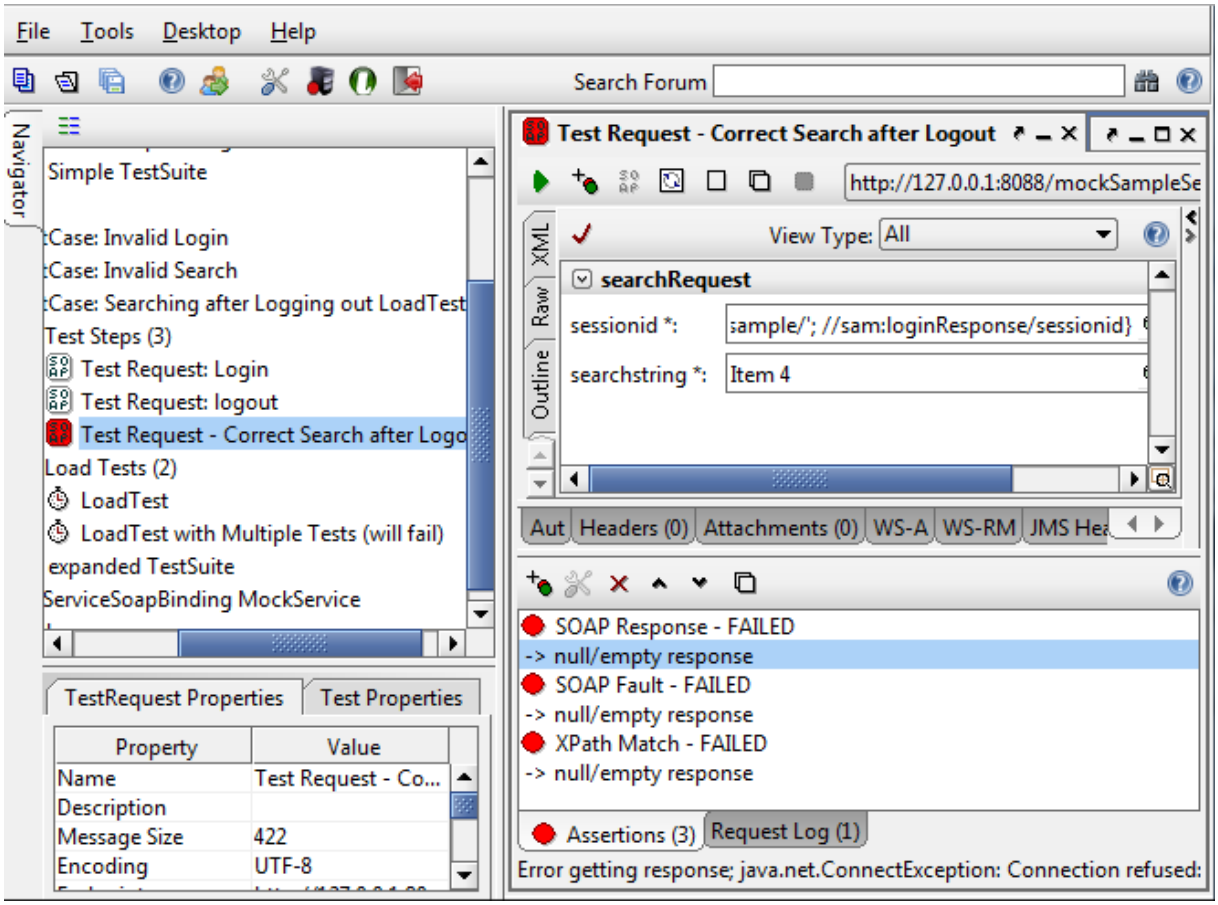
Stejně jako Selenium IDE je možné v SoapUI vytvářet sady test (test suit), testovací případy (test case) nebo pracovat s jednotlivými kroky (step). SoaupUI pracuje se SOAP předpisem (Simple Object Access Protocol), což je předpis řídící činnost mezi klientem a serverem pomocí HTTP. Výměna dat probíhá v xml formátu a má určitý tvar. Dále SoapUI pracuje s WDSL.

Pro tvorbu testů v Soap UI je zapotřebí založit nový projekt. Dále je potřeba definovat strukturu a metody. SoapUI umí ze zadaného WDSL vyčíst jednotlivé operace, které požadovaná webová služba používá. Tyto informace jsou použity pro vytvoření požadavků. Tyto požadavky jsou uloženy a je možné z nich vybírat a sestavovat tak jednotlivé kroky pro editaci testovacích případů.

Nástroj Soap UI je tedy vhodný pro testování funkčnosti webového rozhraní, pomocí sestavení požadavků vyčtené z WDSL, je dále potřeba doplnit data, na kterých má být testováno. Získané požadavky jsou ve formátu XML a lze je volně editovat. Po sestavení

testovacího případů a spuštění, má SoapUI k dispozici log, který informuje o stavu exekuce. Tedy informace o zaslaných požadavcích na webovou stránku a odpovědích ze serveru.

Stavy testů jsou označeny zeleně pro úspěšné vykonání testu nebo červeně, nastane-li chyba při exekuci.



Obrázek č. 8 – SoapUI

V levé části se nachází jednotlivé projekty a testovací případy (Navigator), podle stromové struktury. Horní pravé části je okno pro tvorbu testů, tedy pro zadávání příkazů, které po spuštění sběhnout a výsledek je zobrazen v pravém dolním rohu. Zde se nachází log požadavku nebo informace o konkrétních chybách. V levé dolní části se pak nachází nastavené vlastnosti testovaných požadavků a nastavení testů. [32]

12. Praktická část

V praktické části diplomové práce se budu zabývat automatizovaným a manuálním testováním zvoleného softwaru. Pro tuto práci jsem si zvolila webovou aplikaci

eHelpdesk.cz od firmy MyViVo, na kterou budu aplikovat systémové a integrační testování. Výsledkem praktické části bude srovnání a vyhodnocení manuálního a automatizovaného testování, které budou prováděny na základě stejných podmínek, tak aby nedošlo ke zkreslení výsledku.

Testování spočívá v přípravě množiny testovacích případů, které budou aplikovány na software a jejich exekuci budu porovnávat podle stanovených kritérií. Předmětem této části tedy není ověřování kvality softwaru, jelikož tento produkt již je plně funkční a jeho testování by nemuselo nepřinést žádoucí výsledky, které proces testování popisuje, ale ukázat rozdíly mezi automatizovaným a manuálním testováním. Přínos praktické části v tomto případě znamená demonstraci v praxi a hodnocení výsledků mezi těmito způsoby testování.

12.1 eHelpdesk

Webová aplikace eHelpdesk se nachází na adrese www.ehelpdesk.cz [ehelpdesk.cz]a jedná se o software firmy MyViVo. Tato aplikace funguje jako podpora pro klienty firmy, která se zabývá outsourcingem v oblasti IT. eHelpdesk nabízí klientům podporu v podobě integrované znalostní báze, která obsahuje jednotlivé oblasti jako například Microsoft Office, Windows, Linux nebo další výčet programů, se kterými se klient při svém působení setká. V těchto oblastech jsou po rozkliknutí obsaženy jednotlivé podoblasti, definující časté problémy a jejich řešení. V každé této nápovědě je popsán problém, jeho chování, a následný popis řešení formou printscreenů a popisků. Takto si klient vyhledá sám daný problém a může jej pomocí návodu vyřešit. Některé zásahy však nejsou v kompetencích klienta, a proto je také možné přes tuto aplikaci zadat popis problému, který je dále přesměrován na řešitele, tím je zaměstnanec firmy MyViVo. Rozdíl mezi znalostní bází a požadavkem na vyřešení problému je v přístupnosti. Znalostní báze je veřejná a slouží pro všechny klienty i neklienty, kteří mohou využít jejich řešení. Pro zadání požadavku na konkrétní problém může jen klient firmy, který se do aplikace přihlašuje pomocí přihlašovacích údajů. Tyto požadavky bývají již zpoplatněny a jsou v souladu se smlouvou, která je uzavřena mezi klientem a firmou. [33]

12.3 Postup

Pro zahájení testování bude potřeba rozvrhnout testovací data, která budou používána pro testy. Testovací prostředí potřebné pro exekuci testů a vytvoření testovacích případů, a také testovací nástroj pro automatizované testování. Samotná exekuce bude probíhat ve dvou stavech

1. Manuální testování
2. Automatizované testování

Následně budou výsledky z exekucí testů srovnány a hodnoceny formou srovnávací tabulky a závěrem.

Testovací data

Pro účely testování budou použity pouze přihlašovací údaje, jelikož webová aplikace nepotřebuje jiná vstupní data. Přihlašovací údaje v tomto případě zahrnují emailovou adresu a vytvořené heslo.

Testovací prostředí

K testování byl použit již zmíněný software eHelpdesk.cz poskytnutý firmou MyViVo. Na tomto prostředí budou probíhat veškeré exekuce.

Testovací nástroj

Manuální testování bude probíhat formou sepsaných testovacích případů, které budou exekuovány ručně přímo na webové aplikaci.

K automatizovanému testování budou použity stejné testovací případy, vytvořené při manuálním testovaní, ale budou aplikovaný nástrojem pro automatizované testování Seleniem IDE.

12.4 Manuální testování

Manuální testování je rozděleno do fází, které budou postupně prováděny. První fáze obsahuje sestavení testovací případů a design testů. Tedy konkrétní metodologie, jak postupovat pro ověření daného testovacího případu.

Druhá fáze zahrnuje samotnou exekuci těchto testů na webové aplikaci eHelpdesk.cz. Ke každému testu bude doplněn výsledek exekuce.

12.4.1 Testovací případy

Název	1. Proklikání znalostní báze
Cíl	Ověření funkčnosti zobrazení jednotlivých problematik dle oblastí
Popis	Na stránkách ehelpdesk otevřít znalostní bázi a proklikání vybraných oblastní a problematik
Očekávaný výsledek	Webové stránky odpovídají a zobrazuje se požadované informace
Postup testu	
Krok 1	Vložit url adresu ehelpdesk.cz do webového prohlížeče Firefox
Krok 2	Kliknout na znalostní bázi
Krok 3	Kliknout na oblast „Windows“
Krok 4	Otevřít článek „Prohodit obraz na dvou monitorech“
Krok 5	Zkontrolovat načtení požadované stránky s obsahem
Krok 6	Kliknout zpět do znalostní báze
Krok 7	Kliknout na „Internetový prohlížeč
Krok 8	Otevřít článek „Nastavení motivu v Google Chromu“

Krok 9	Zkontrolovat načtení požadované stránky s obsahem
Krok 10	Kliknout zpět do znalostní báze
Krok 11	Klinout na „Office“
Krok 12	Otevřít článek „Nastavit nový mailbox do Outlooku“
Krok 13	Zkontrolovat načtení požadované stránky s obsahem
Krok 14	Kliknout zpět do znalostní báze

Tabulka č. 2 - 1. Proklikání znalostní báze

Název	2. Ohodnocení článku
Cíl	Ověření funkčnosti hodnocení
Popis	Přihlášení a otevřít znalostní bázi a ohodnocení článku
Očekávaný výsledek	Článek je ohodnocen „Up“
Postup testu	
Krok 1	Vložit url adresu ehelpdesk.cz do webového prohlížeče Firefox
Krok 2	Kliknout na znalostní bázi
Krok 3	Kliknout na oblast „Windows“
Krok 4	Otevřít článek „Prohodit obraz na dvou monitorech“
Krok 5	Zkontrolovat načtení požadované stránky s obsahem
Krok 6	Kliknout na „Up“ v oblasti „Rate this article“ v článku
Krok 7	Ověřit nárůst hodnocení v oblasti „Rate this article“ v článku

Tabulka č. 3 - 2. Ohodnocení článku

Název	3. Vyhledávání ve znalostní bázi
Cíl	Ověření funkčnosti vyhledávání
Popis	Na ehlepdesk.cz zobrazit článek pomocí vyhledávaného slova
Očekávaný výsledek	Vložení slova do vyhledávače a kliknutí na zobrazení daného článku
Postup testu	
Krok 1	Vložit url adresu ehelpdesk.cz do webového prohlížeče Firefox
Krok 2	To textového pole vložit slovo „Windows“
Krok 3	Kliknout na vyhledat
Krok 4	Zobrazí se veškeré články s obsahem slova „Windows“
Krok 5	Vybrat článek kliknutím „Nastavení mailu do Windows Phone“
Krok 6	Zkontrolovat správnost jeho zobrazení

Tabulka č.4 - 3. Ohodnocení článku

Název	4. Přihlášení a odhlášení
Cíl	Ověření funkčnosti přihlašování
Popis	Na stránkách ehelpdesk provést přihlášení
Očekávaný výsledek	Uživatel se přihlásí pomocí emailu a hesla
Postup testu	
Krok 1	Vložit url adresu ehelpdesk.cz do webového prohlížeče Firefox
Krok 2	Kliknout na „Přihlásit se“
Krok 3	Otevře se přihlašovací okno

Krok 4	Do prvního pole vyplnit email
Krok 5	Do druhého pole vyplnit heslo
Krok 6	Kliknout na „Přihlásit se“
Krok 7	Zobrazí se obrazovka přihlášeným uživatelem, který má možnost přidávat požadavky
Krok 8	Kliknout na odhlášení

Tabulka č.5 – 4. Přihlášení a dohlášení

Název	5. Založení požadavku
Cíl	Ověření funkčnosti zakládání požadavků
Popis	Přihlásit se a založit nový požadavek
Očekávaný výsledek	Uživatel se přihlásí a založí nový požadavek, který se zobrazí v seznamu požadavků
Postup testu	
Krok 1	Vložit url adresu ehelpdesk.cz do webového prohlížeče Firefox
Krok 2	Kliknout na „Přihlásit se“
Krok 3	Otevře se přihlašovací okno
Krok 4	Do prvního pole vyplnit email
Krok 5	Do druhého pole vyplnit heslo
Krok 6	Kliknout na „Přihlásit se“
Krok 7	Zobrazí se obrazovka přihlášeným uživatelem, který má možnost přidávat požadavky
Krok 8	Kliknout na „Přidat požadavek“
Krok 9	Vyplnit následující pole:

	• Předmět = test_funkcnosti • Požadavek = test • Termín do = „výchozí datum“ • Priorita = „výchozí“ • Kategorie = „výchozí“ • Typ = „vychozí“ • Přidat soubor = prázdné • Přidělit na = „Andrea Hořavová“
Krok 9	Kliknout na „Uložit“

Tabulka č. 6 - 5. Založení požadavku

12.5 Automatizované testování

Pro vytváření automatizovaného testování nebude potřeba vytvářet nové testovací případy. Použijí se testy z manuálního testování, jelikož ověřuji stejnou funkčnost. Bude tedy probíhat pouze exekuce již existující testů pomocí nástroje Selenium IDE.

Proklikání hlavních oblastní a zobrazení vybraných článků:

Proklikání znalostní báze	
open	/
clickAndWait	css=h4
clickAndWait	link=Windows
clickAndWait	link=Prohodit obraz na dvou monitorech
clickAndWait	link=Knowledge Base
clickAndWait	link=Internetový prohlížeč
clickAndWait	link=Nastavení motivu v Google Chromu
clickAndWait	link=Knowledge Base
clickAndWait	link=Office
clickAndWait	link=Nastavit nový mailbox do Outlooku
clickAndWait	link=Knowledge Base
close	

Tabulka č. 7 – Proklikání znalostní báze v Seleniu IDE

V prvním kroku otevře stránku ehelpdesk.cz, dále vyhledá nadpis h4 „Znalostní báze“, kterou otevře. Vybere sekci Windows a zde článek o prohození obrazu na dvou monitorech. Jakmile jej otevře, v dalším kroku se vrátí do znalostní báze a pokračuje stejně pro všechny další kroky. V posledním kroku stránku zavře.

Ohodnocení článku:

Ohodnocení článku	
open	/
clickAndWait	css=div.hf-block.hf-kb-block
clickAndWait	link=Windows
clickAndWait	link=Chyba 691
click	link=Up

Tabulka č. 8 – 2. Ohodnocení článku

Po otevření stránek ehelpdesk.cz, rozklikne nástroj znalostní bázi. V dalším kroku klikne na oblast „Windows“ a zobrazí článek o chybě 691. Na konci tohoto článku je možnost ohodnotit článek „Rate this article“. Nástroj klikne na hodnocení „Up“, čímž ohodnotí kvalitu poskytovaného článku.

Vyhledávání článku pomocí fulltextu:

Vyhledávání ve znalostní bázi		
open	/	
click	id=s	
type	id=s	windows
clickAndWait	id=searchsubmit	
clickAndWait	link=Nastavení e-mailu do Windows phonu	
verifyText	css=h2.entry-title	Nastavení e-mailu do Windows phonu

Tabulka č. 9 – Vyhledávání ve znalostní bázi v Seleniu IDE

Ve znalostní bázi lze vyhledávat i pomocí fulltextového vyhledávače. Nástroj otevře stránky ehlepdesk.cz, klikne na vyhledávání. Dále zadá do vyhledávače slovo „Windows“ a čeká na výsledek vyhledávání ve všech článcích, které jsou v bázi dostupné a obsahují slovo

„Windows“. Následně vybere článek „Nastavení e-mailu do Windows Phonu“ a zkontroluje správnost zobrazení článku podle h2.

Přihlášení a odhlášení:

Přihlášení a odhlášení		
open	/	
clickAndWait	link=| Přihlaste se	
type	name=username	andrea.horavova@myvivo.cz
type	id=frm-signInForm-password	myvivo
clickAndWait	id=frm-signInForm-send	
clickAndWait	link=Odhlásit	

Tabulka č. 10 – Přihlášení a odhlášení v Seleniu IDE

V prvním kroku nástroj otevře stránky ehelpdesk.cz a přihlásí se pomocí odkazu na link „přihlaste se“. Zadá username, což je email, a heslo. Poté odešlé žádost na přihlášení, pokud je zadání správné, uživatel s těmito údaji je přihlášen. Následně test provede odhlášení rozkliknutím linku na odhlášení.

Založení požadavku:

založení požadavku		
open	/	
clickAndWait	link=| Přihlaste se	
selectWindow	null	
type	name=username	andrea.horavova@myvivo.cz
type	id=frm-signInForm-password	myvivo
clickAndWait	id=frm-signInForm-send	
clickAndWait	css=button.button.medium	
type	id=frm-helpdeskTaskForm-subject	test1
type	tinymce	test_funkcnosti
assertTitle	eHelpdesk	
verifyText	id=select2-chosen-4	Vybrat
storeText	id=select2-chosen-4	Vybrat
verifyTable	css=table.6.1	Vybrat Přidělit na:
assertElementPresent	id=select2-chosen-4	
assertText	id=select2-chosen-4	Vybrat
assertTitle	eHelpdesk	
verifyText	id=select2-result-label-46	Andrea Hořavová
assertText	id=select2-result-label-43	Andrea Hořavová
storeText	id=select2-result-label-49	Andrea Hořavová
assertElementPresent	id=select2-result-label-52	
click	id=frm-helpdeskTaskForm-save	

Tabulka č.11 – 5. Založení požadavku v Seleniu IDE

Pro založení požadavku na je potřeba přihlášení. Po přihlášení klikne na tlačítko „nový požadavek“ a zobrazí se obrazovka pro vyplnění požadavku. Na obrazovce vyplní předmět „test_funkcnosti“, požadavek „test“ a vybere řešitele požadavku. Ostatní hodnoty jsou nastavené jako výchozí.

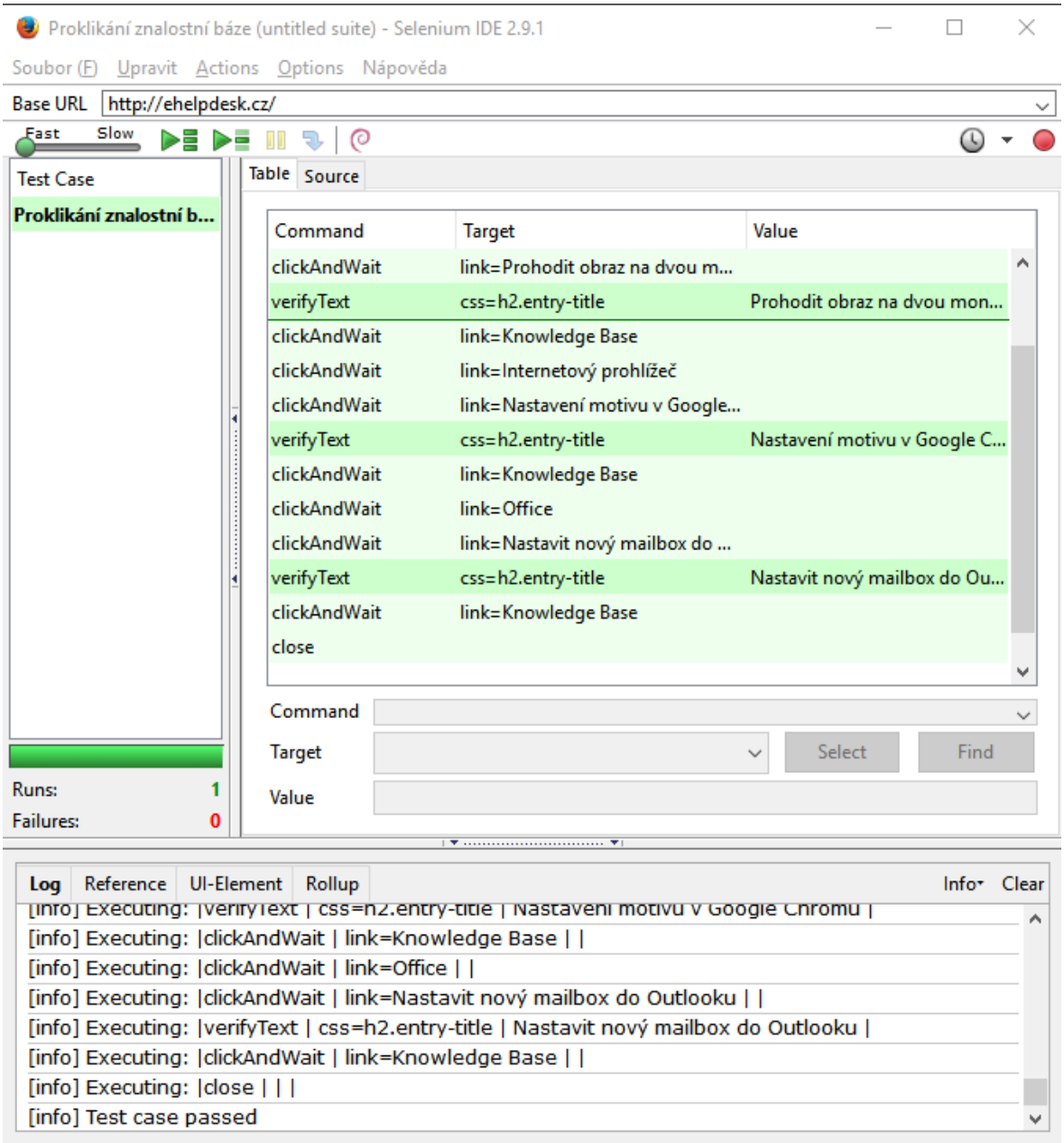
12.1 Výsledky exekuce testů

Výsledky exekuce testů jsou popsány v tabulce, která je rozdělena na jednotlivý test case a dále na manuální testování a testování v Seleniu IDE. Pro každý typ testování jsou vypsány stavy exekucí či vzniklé problémy.

První test na proklikání znalostní báze a zobrazení článku, který má za úkol ověřit funkčnost znalostní báze, proběhl úspěšně.

1. Proklikání znalostní báze	
Mauální testování	V nástroji Selenium IDE
• Mauálně zde nevznikl žádný problem při testování • Rychlost testování se nelišila od automatizovaného, důvodem byla odezva serveru • Nebyla potřeba připravovat žádná data	• Test proběhl úspěšně bez chyb • Doba nutná na vykonání testu se odvíjela od odezvy server, která nebyla překročena • Byla potřeba sestavit úkony, které musí nástroj vykonat, viz obrázek

Tabulka č.12 – Výsledek testu - 1. Proklikání znalostní báze



Obrázek č.9 – Test – proklikání znalostní báze Selenium IDE

Přiložený obrázek prezentuje stav exekuovaného testu, všechna pole jsou označena zeleně, tzn., že všechny kroky byly provedeny úspěšně. Z logu je možné vidět závěr nástroje Selenia, že test dopadl úspěšně.

Parametry:

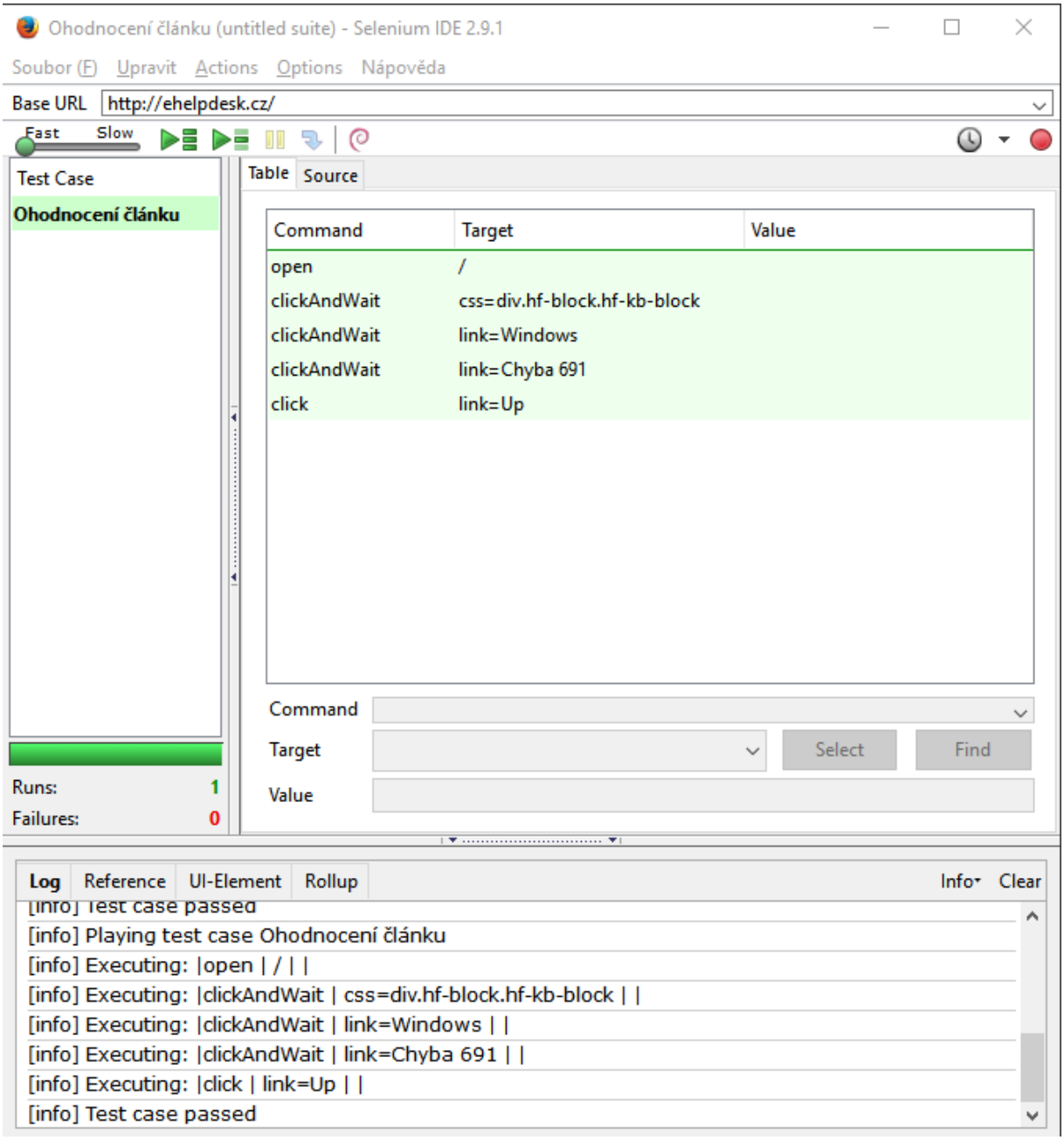
- Open – otevře webovou adresu podle zadané url adresy v horní liště (Base URL)
- ClickAndWait – funkce, která klikne na oblast označenou v cíli a čeká na odpověď
    - Link= „“ – označuje odkaz na danou sekci

- VerifyText – zkontroluje obsah textu podle h2, zdali po kliknutí se načte požadovaný článek
- Close – uzavře webovou stránku
-

2. Ohodnocení článku	
Mauální testování	V nástroji Selenium IDE
• Manuální testování proběhlo uspěšně, článek byl ohodnocen kladně • Pro testování nebyla potřeba data	• Test proběhl úspěšně bez chyb • Rychlost testu se nelišila od manuálního testování, odezva serveru byla stejná • Viz obázek

Tabulka č.13 – Výsledek testu – Ohodnocení článku

Při manuálním testování vše proběhlo vpořádku a nebyl viditelný rozdíl oproti testováním v nástroji Selenium IDE.



Obrázek č.10 – Test – Ohodnocení článku v Selenium IDE

Ohodnocení článku v Seleniu IDE má následující parametry pro exekuci:

- Open – otevře webovou adresu podle zadané url adresy v horní liště (Base URL)
- ClickAndWait – funkce, která klikne na oblast označenou v cíli a čeká na odpověď
    - css=div.hf-block.hf-kb-block - označuje místo, kde se nachází odkaz na znalostní bázi
    - link=Windows – představuje sekce Windows obsaženou ve znalostní bázi

- link=chyba691 – představuje konkrétní článek v sekci Windows

- Clik – funkce, která klikne na oblast, ale již nečeká na odpověď

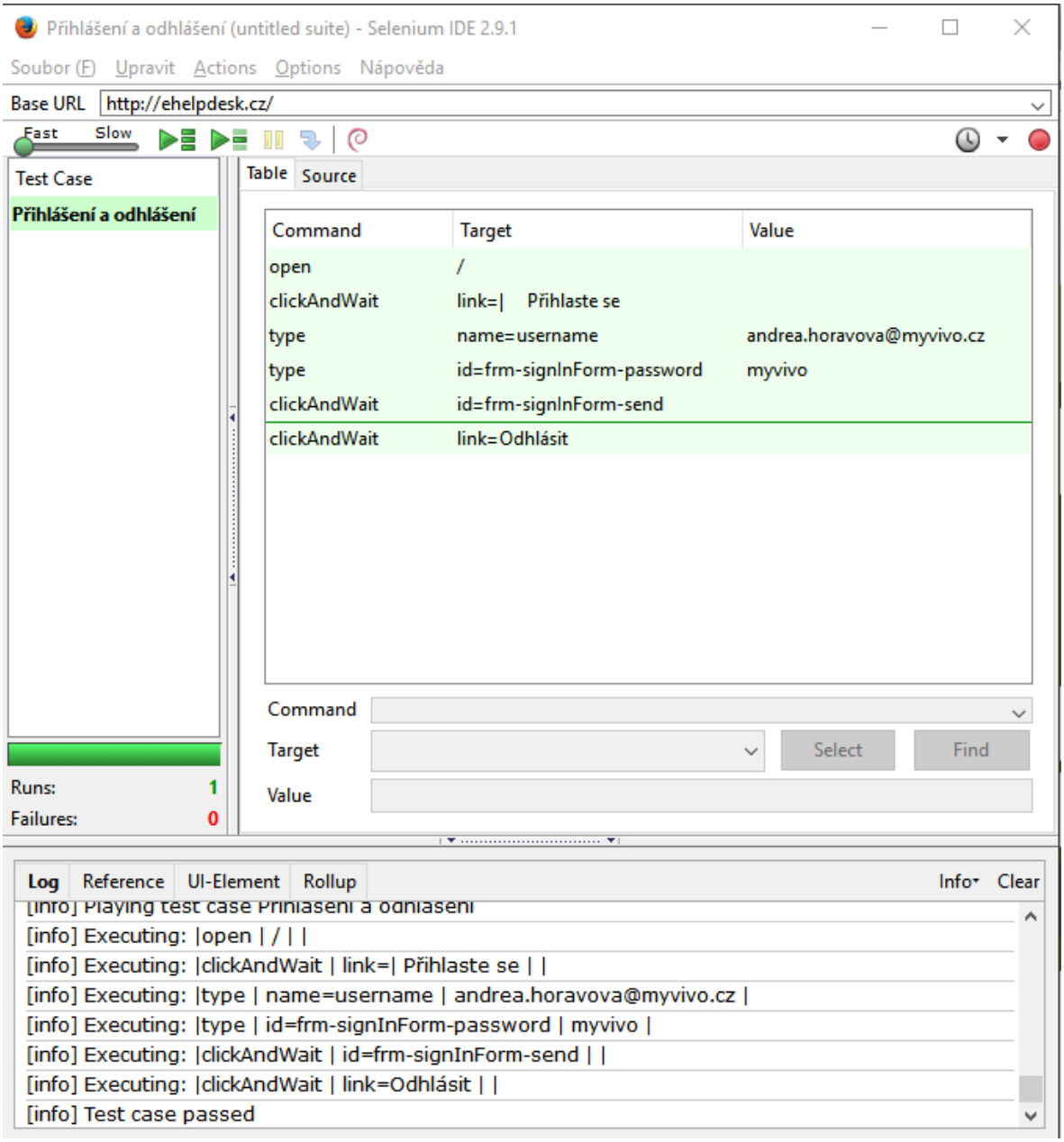
  - Link=Up – představuje odkaz na hodnocení článku (Rate this article)

Test proběhl úspěšně hodnocení článku proběhl úspěšně a nevyskytla se žádná chyba. V logu nástroje Selenium IDE je důkaz o pozitivním testování „Test case passed“.

4. Přihlášení a odhlášení	
Mauální testování	V nástroji Selenium IDE
• Manuálně zde nevznikl žádný problem při testování • Rychlost testování se mírně lišila vyplnění polí pro přihlášení • Byla použita přihlašovací data	• Test proběhl úspěšně bez chyb • Při vyplňování přihlašovacích údajů byl nástroj rychlejší než při standardním ručním psaní • Viz obrázek

Tabulka č.14 – Výsledek testu – Přihlášení a odhlášení

Pro přihlášení a odhlášení byla zapotřebí již data, v podobě přihlašovacích údajů. Rozdíly mezi testování jsou popsány v tabulce.



Obrázek č.11 – Test – Přihlášení a odhlášení Selenium IDE

Test proběhl úspěšně a v logu nebyl zaznamenám problém o chybě. Test byl exekuován s těmito parametry:

- Open – otevře webovou adresu podle zadané url adresy v horní liště (Base URL)
- ClickAndWait – Klikne a čeká na odpověď ze serveru
  - Link=| Přihlaste se – představuje oblast kliknutí pro přihlášení na webových stránkách
  - Id=frm-signInForm-send – pole označené podle div id – zaslání požadavku na přihlášení

- Link=Odhlásit – odkaz na odhlášení

- Type – určuje typ oblasti a kam se má zadat hodnota
  - Name=username – pole pro zadávání přihlašovacího jména s hodnotou andrea.horavova@myvivo.cz
  - Id=frm-signInForm-password – pole označené pod div id s hodnotou myvivo

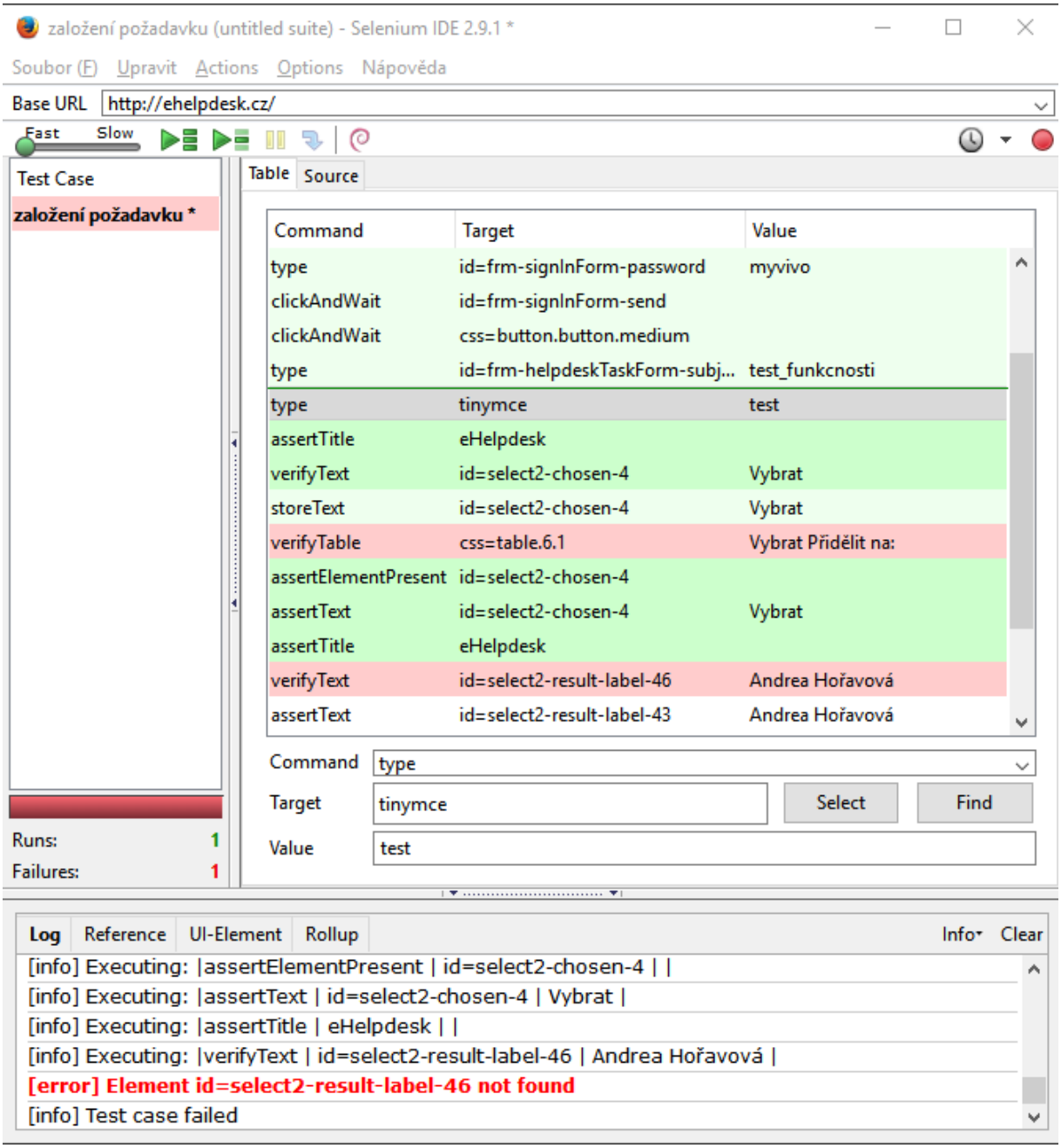
Založení požadavku:

5. Založení požadavku	
Mauální testování	V nástroji Selenium IDE
• Manuální testování proběhlo uspěšně, požadavek byl založen • Pro testování byla použita data pro přihlášení a hodnoty pro vyplnění polí	• Test neproběhl úspěšně • Detaily, viz obázek

Tabulka č.15 – Test - Založení požadavku

Při manuálním testování nevznikl problém při založení požadavku. Požadavek na řešení byl nastaven a přidá do seznamu ostatní požadavku, čekající na vyřešení.

Nástroj Selenium IDE měl problém s identifikací jednotlivých částí požadavku, z důvodu použitých funkcí a definicí webové aplikace. Test tedy neproběhl do konce.



Obrázek č.12 – Založení požadavku v Seleniu IDE

Test začíná přihlášením, které je již otestováné v minulých testech, nasledovala tedy funkce clickandWait, která klikne na tlačítko „Nový požadavek“. Další krok bylo vyplnění pole předmět, kterou zajistila funkce type s hodnout „test_funkcnosti“. A další funkce type s cílem tinymce označuje textové pole, kam se měla zapsat hodnota „test“. V exekuci je vidět, že krok proběhl úspěšně, ale po kontrole na aktuálně testované stránce se text nezobrazil. Ačkoliv při tvorbě testu byla provedena kontrola na označené textové pole, které bylo definované. Důvodem může být zjištěný fakt, že Selenium IDE neumí pracovat s textovými

poli. I přes konkretizaci místa pomocí pluginu Xpath od Firefoxu, se nepodařilo tento problém odstranit.

Dále následovala sekce funkcí assertText, verifyText a storeText. Funkce assertText a verifyText mají podobnou funkci, s tím rozdílem, že kontrola textu v negativním případě při funkci assertText, by měla exekuci testu ukončit se stavem failed. V tomto případě byl problém s ověřením textu (verifyText) a tabulky (verifyTable), kde se nachází combobox s výběrm řešitele. V posledním kroku test skočil stavem failed, i když se jednalo o funkci verifyText, která není omezena pro pokračování exekuci testu. V logu je zmíněno „Element id=select2-result-label46 not found“, tedy že nástroj nenašel cíl. Po důkladném prozkoumání zdrojového kódu stránky, bylo zjištěno, že pro výběr z combokboxu je použito vyhledávání pomocí javascriptové funkce. Tato funkce po několikanásobném retestování nemá stejnou hodnotu labelu a nástroj se na ni nedokáže odvolat, tudíž exekuce skončila stavem failed.

Řešením pro vzniklé problémy je použití sofistikovanějšího nástroje pro automatizované testování, například Selenium WebDriver. V tomto nástroji je umožněno zvolit programovací jazyk a přesně definovat funkce pro vyhledávání v checkboxu či práci s textovým polem mimo jiné.

13. Shrnutí a Závěr

Ve své diplomové práci jsem probírala téma softwarové testování a zaměřila jsem se na automatizaci testů. Softwarové testování neodmyslitelně patří k vývoji softwaru, protože všichni jsme lidi a chybujeme. Navíc v dnešní den téměř vše řídí software a požadavky na jeho kvalitu velice vzrostly než tak, jak tomu bylo na počátku. V kapitole o chybách jsem zmínila několik závažných chyb, jejichž dozvuk měl světovou úroveň. Zjištěným faktorem byl zde již zmíněný člověk. Testování softwaru je dnes tedy velmi důležitou složkou, která by neměla být podceňována.

Zmiňovala jsem také druhy testování podle zvolené metodologie postupu vývoje softwaru, protože otestování neznamená jen vzít si do ruky software a podívat se. Je to kompletní proces zajišťování kvality, který je rozdělen do dílčích částí. Jejich ověření pak znamená akceptaci celého softwaru.

V rámci praktické části jsem zvolila testování manuálním a automatizovaným způsobem. Pro manuální část byla potřeba vytvořit testovací případy, které přesně definují, co je testováno. Stejně tak slouží jako podklad pro automatizované testování, které je podle definovaných kroků vytvořeno v nástroji Selenium IDE. Tento nástroj je velmi přívětivý k uživateli. Má jednoduché rozhraní, ve které se dá lehce orientovat. Práce s tímto nástrojem není nijak náročná a po drobném zaškolení funkcí, které nabízí, se stává tento nástroj ovladatelným a využitelným pro automatizace. Bohužel nestačí úplně na všechno. Při práci s tímto nástrojem jsem se obešla bez již zmíněných možností podmíněných výrazů či iterací, které Selenium IDE neumí. Ale také má tento nástroj problémy se složitějšími úkony jako je textové pole nebo výběr z comboboxu. Pro sestavení kompletní sady testů, které by ověřily veškeré funkčnosti dané webové aplikace, bych tento nástroj nedoporučila, spíše jeho kolegu Selenium WebDriver, který nabízí více programovatelných možností. Pro drobné účely běhu Selenium rozhodně vystačí, například vyhledávání v textu, ověřování textu, vyplňování jednoduchých polí, apod. Práce v Seleniu je vcelku jednoduchá a sám nástroj nabízí uživatelskou podporu při tvorbě automatizovaných testů.

Automatizovat je výhodné a rozhodně šetří náklady a čas, pokud jsou automatizace správně napsané a ověřené. V praktické části jsem se setkala s problémem, kdy manuálním testováním nebyl problém test provést úspěšně. Při automatizace v Selenium IDE jsem narazila na nemožnost zacílit určité oblasti, tak aby jej nástroj nalezl a provedl příkaz definovaný v požadavku testu. Také se mi stalo, že test v nástroji proběhl úspěšně, ale okem testera jsem si všimla, že dopadl špatně, protože nástroj nemá intuitivní myšlení jako člověk. Automatizovat má za mě určitě smysl v rámci prověřených a funkčních testů, které působí jako tzv. smoke testy nebo regresní testy. Tyto testy se provádějí před nebo po integraci do systému a jsou testy k ověření nezměněné funkčnosti celého systému. V tomto smyslu jsou automatizace vhodné ve smyslu efektivity práce.

14. Zdroje

[1] Patton, R.: Testování software. Computer Press, Praha, 2002. ISBN 80-7226-636-5.

[2] LYDIA, Ash. The web testing companion: The insider's guide to efficient and effective tests. Indianapolis: Wiley Publishing, 2003. ISBN 0-471-43021-8.

[3] Black, R.: Managing the Testing process John Wiley & Sons, 2002, ISBN 0471223980

[4] Brown, C. T.; Gheorghe, G.; Huggins, J.: An Introduction To Testing Web Applications With Twill And Selenium. O´Reilly. California. 2007. ISBN 0596527802, 9780596527808.

[5] CEM, Kaner; JAMES, Bach.; BRET, Pettichord. Lessons learned in software testing : A Context-driven approach. New York : John Wiley & Sons, 2001. ISBN 0-471-08112-4.

[6] Testování jako součást procesu aneb co je to QA. *SW Testování* [online]. Praha, 2005 [cit. 2016-03-31]. Dostupné z: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=20:testovani-jako-proces-aneb-co-je-to-qa&catid=3:zaklady&Itemid=11

[7] Quality Assurance (QA). *Http://searchsoftwarequality.techtarget.com/* [online]. Praha, 2007 [cit. 2016-03-31]. Dostupné z: http://searchsoftwarequality.techtarget.com/definition/quality-assurance

[8] The International Software Testing Standard. *ISO/IEC/IEEE 29119 Software Testing* [online]. Praha, 2014 [cit. 2016-03-31]. Dostupné z: http://www.softwaretestingstandard.org/

[9] *ISO - International Organization for Standardization* [online]. 2015 [cit. 2016-03-31]. Dostupné z: http://www.iso.ch/

[10] *First Computer Bug* [online]. US, 2000 [cit. 2016-03-31]. Dostupné z: http://www.jamesshuggins.com/h/tek1/first_computer_bug_large.htm

[11] Software bugs, errors and defects: What's the difference? *SearchSoftwareQuality* [online]. US, 2001 [cit. 2016-03-31]. Dostupné z: http://searchsoftwarequality.techtarget.com/answer/Software-bugs-errors-and-defects-Whats-the-difference

[12] What is the difference between Severity and Priority? *ISTQB EXAM CERTIFICATION* [online]. US, 2016 [cit. 2016-03-31]. Dostupné z: http://istqbexamcertification.com/what-is-the-difference-between-severity-and-priority/

[13] Mariner 1. *NASA - NSSDCA - Spacecraft - Details* [online]. US, 2002 [cit. 2016-03-31]. Dostupné z: http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=MARIN1

[14] The Explosion of the Ariane 5. *Home | Institute for Mathematics and its Applications* [online]. Minneapolis, 2002 [cit. 2016-03-31]. Dostupné z: https://www.ima.umn.edu/~arnold/disasters/ariane.html

[15] Epic failures: 11 infamous software bugs. *Computer World* [online]. San Francisco, USA, 2010 [cit. 2016-03-31]. Dostupné z: http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html?page=3

[16] Deset největších katastrof, za které „vděčíme“ počítačům. *Technet cz* [online]. Praha, 2008 [cit. 2016-03-31]. Dostupné z: http://technet.idnes.cz/deset-nejvetsich-katastrof-za-ktere-vdecime-pocitacum-ppb-/tec_technika.aspx?c=A080129_195559_tec_technika_pka

[17] An Investigation of the Therac-25 Accidents. *Department of computer science* [online]. USA, 1993 [cit. 2016-03-31]. Dostupné z: http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html

[18] Y2K bug. *National Geographics Society* [online]. USA, 2007 [cit. 2016-03-31]. Dostupné z: http://education.nationalgeographic.org/encyclopedia/Y2K-bug

[19] SSADM Diagram. *ConceptDraw Solution Park* [online]. USA, 2009 [cit. 2016-03-31]. Dostupné z: http://www.conceptdraw.com/How-To-Guide/ssadmn-diagram

[20] What is SSADM. *Search Software Quality* [online]. USA, 2008 [cit. 2016-03-31]. Dostupné z: http://searchsoftwarequality.techtarget.com/definition/SSADM

[21] BOEHM,, Barry. *Spiral Development: Experience, Principles, and Refinements* [online]. 2000, , 38 [cit. 2016-03-31]. DOI: CMU/SEI-2000-SR-008. Dostupné z: http://www.sei.cmu.edu/reports/00sr008.pdf

[22] Rational Unified Process (RUP). *Search Quality Software* [online]. USA, 2007 [cit. 2016-03-31]. Dostupné z: http://searchsoftwarequality.techtarget.com/definition/Rational-Unified-Process

[23] Extrémní programování pod drobnohledem. *Živě.cz - O počítačích, IT a internetu* [online]. Praha, 2003 [cit. 2016-03-31]. Dostupné z: http://www.zive.cz/clanky/extremni-programovani-pod-drobnohledem/sc-3-a-111952/default.aspx

[24] Extrémní programování (XP). *MBI - Management Byznysu a Informatiky* [online]. Praha, 2003 [cit. 2016-03-31]. Dostupné z: http://mbi.vse.cz/public/cs/obj/METHOD-92

[25] Druhy testování v procesu vývoje SW. *SW Testování* [online]. Praha, 2010 [cit. 2016-03-31]. Dostupné z: http://www.swtestovani.cz/index.php/uvod-do-testovani/18-druhy-testovani-v-procesu-vyvoje-sw

[26] V-Model (Vee-Model). *Search Software Quality* [online]. USA, 2007 [cit. 2016-03-31]. Dostupné z: http://searchsoftwarequality.techtarget.com/definition/V-Model

[27] Encyklopedie profesí: IT analytik. *Práce.cz* [online]. Praha, 2012 [cit. 2016-03-31]. Dostupné z: http://www.prace.cz/poradna/encyklopedie-profesi/i/it-analytik/

[28] Práce IT architekt. *Profesia.cz* [online]. Praha, 2016 [cit. 2016-03-31]. Dostupné z: http://www.profesia.cz/prace/it-architekt/

[29] Projektový manažer. *Jobs.cz* [online]. Praha, 2016 [cit. 2016-03-31]. Dostupné z: http://www.jobs.cz/poradna/profese/p/projektovy-manazer/

[30] Test Manager. *SmartRecruiters Job Search* [online]. USA, 2016 [cit. 2016-03-31]. Dostupné z: https://www.smartrecruiters.com/PrincipalEngineeringSro/72228329-test-manager

[31] What is Selenium? *Selenium web browser automation* [online]. USA, 2004 [cit. 2016-03-31]. Dostupné z: http://docs.seleniumhq.org/

[32] SoapUI|Functional Testing for SOAP and REST APIs. *About SOAPUI* [online]. USA, 2016 [cit. 2016-03-31]. Dostupné z: https://www.soapui.org

[33] Efektivní software pro HelpDesk s integrovanou znalostní bází. *Helpdesk - Vaše technická podpora*[online]. Praha [cit. 2016-03-31]. Dostupné z: http://ehelpdesk.cz/

15. Seznam obrázků

16. Seznam tabulek