Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Testování software

Jan Kubiš

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Kubiš

Systémové inženýrství

Název práce

Testování softwaru

Název anglicky

Software testing

Cíle práce

Hlavním cílem bakalářské práce je porovnat nástroje pro testování software.
Dílčími cíli bakalářské práce jsou:
- Vytvořit kritický přehled literatury a vymezit elementární teoretické poznatky z oblasti testování software.
- Analyzovat metody testování v rámci celého životního cyklu softwaru.
- Provést vícekriteriální analýzu variant vhodného nástroje pro testování SW.
- Syntetizovat výsledky a formulovat závěry bakalářské práce.

Metodika

Metodika problematiky bakalářské práce je založena hlavně na studiu a analýze odborné literatury. Selekce vhodných nástrojů pro testování software je provedena prostřednictvím vícekriteriální analýzy variant. Vlastní práce je založena na konkrétních příkladech testování a následnou analýzou výsledků. Na základě odborných informačních zdrojů a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30-40 stran

Klíčová slova

Testování software, Data, Selenium, JUnit, Regrese

Doporučené zdroje informací

AMMANN, Paul; OFFUTT, Jeff. Introduction to software testing. Cambridge University Press, 2008.
HASS, Anne Mette Jonassen. Guide to Advanced Software Testing, Artech House, 2008, ISBN 9781596932869.
MATHUR, Aditya P. Foundations of Software Testing, 2/e. Pearson Education India, 2008.
MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. The art of software testing. John Wiley & Sons, 2011.
PUSULURI, Nageshwar Rao. Software Testing Concepts And Tools. Dreamtech Press, 2006.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 28. 10. 2015

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 11. 2015

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2017

Čestné prohlášení

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

V Praze dne 15.3.2017 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Janu Tyrychtrovi, Ph.D. za odborné vedení, pomoc a rady při zpracování této práce.

Testování software

Abstrakt

Tématem bakalářské práce je problematika testování softwaru, v dnešní době se jedná o velmi důležitou součást vývoje nového softwaru. Testováním je zajišťována funkčnost, kvalita a především kontrola, zdali byl software vyvinutý podle zadané specifikace. Práce se věnuje vymezení teoretických pojmů v oblasti testování, popisuje úrovně testování od počátku po výsledný produkt. Praktická část práce je zaměřená na automatizaci testovacích procesů, kde byla provedena analýza a porovnání nástrojů pro automatizované testování pomocí vícekriteriální analýzy.

Klíčová slova: testování software, defekt, automatizace testování, integrační testování, akceptační testování, SilkTest, Selenium, TestComplete

Software testing

Abstract

The subject of bachelor thesis is software testing area, which is currently very important part in software development. Testing takes care about functionality, quality and especially check whether the software has been developed according to the desired specifications. The thesis provides theoretical concepts in the field of testing, describes the levels of testing from the beginning to the final product. The practical part is focused on automatization of the testing process and comparison of tools for automated testing was done with multi-criterion analysis.

Keywords: software testing, defect, test automation, integration testing, acceptance testing, SilkTest, Selenium, TestComplete

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V současné době je kvalita a funkčnost dodaného softwaru jedním z rozhodujících faktorů při výběru dodavatele. Software se díky rozmachu informačních technologií nalézá v mnoha oblastech lidské činnosti, od primitivních mobilních aplikací až po velký a komplexní software, do kterého můžeme zařadit například systémy pro řízení výroby, řízeni budov nebo analýzu rozsáhlých datových souborů. Cílem výrobce není jen naprogramovat a dodat softwarové řešení, ale také si udržet a případně získat nové zákazníky, pro které kvalita dodaného produktu hraje stěžejní roli právě při výběru dodavatele. Jako téma bakalářské práce jsem si proto vybral testování software, tato oblast je nepostradatelnou součástí vývoje veškerého softwaru. Neustále rostoucí složitost a nároky na funkcionalitu vedou ke stále větší chybovosti a tím rostoucí potřebě těmto problémům předcházet již v počátcích vývoje. Problematiku testování softwaru jsem si jako téma bakalářské práce vybral z několika důvodů. Jedním důvodem je má několikaletá praxe v oblasti testování. V minulosti jsem pracoval na pozici tester v Československé obchodní bance, a.s., kde jsem byl součástí týmu na testování internetového bankovnictví, dále jsem působil jako test analytik v České spořitelně, a.s., kde jsem nabral praktické zkušenosti v oblastní návrhu testovacích scénářů. V současné době působím jako senior tester ve společnost Qest Automation, s.r.o., kde je mou odpovědností exekuce integračních testů a mým hlavním úkolem je návrh a vývoj automatizovaných testů pro webové a mobilní aplikace.

2 Cíl práce a metodika

2.1 Cíl práce

Tématem bakalářské práce je problematika testování softwaru. Hlavním úkolem v teoretické části je vymezit teoretická východiska v testování, popsat průběh testování od počátku vývoje až po výsledný produkt, popsat role v testovacím procesu a co každá role obnáší. Praktická část je zaměřena na automatizované testování, kde je provedena analýza a porovnání nástrojů pro automatizované testování softwaru. Následně je doporučen výběr vhodného nástroje pro zavedení automatizace testování.

2.2 Metodika

Metodika bakalářské práce v teoretické části je založena především na studiu a analýze odborné literatury. V praktické části je provedena analýza a následně porovnání nástrojů prostřednictvím vícekriteriální analýzy variant. Na základě odborných informačních zdrojů a výsledků vlastního šetření jsou formulovány závěry bakalářské práce.

2.2.1 Vícekriteriální analýza variant

Modely vícekriteriální analýzy variant se zabývají problémy, jak zvolit jednu nebo více variant z množiny přípustných řešení, které je možné následně doporučit. Od rozhodovatele se vyžaduje, aby při výběru variant postupoval maximálně objektivně. Rozhodovatel je jednotlivec nebo skupina, jejímž úkolem je učinit výsledné rozhodnutí. V praxi se setkáváme s případy, kde je od sebe oddělen zadavatel úlohy a její řešitel (analytik). Výhoda tohoto oddělení spočívá v tom, že analytik nebývá zainteresován na výsledku rozhodnutí, a proto postupuje maximálně objektivně. Nevýhodou může být fakt, že analytik nemusí být dobře seznámen se všemi detaily úlohy, které nemusely být zachyceny při počáteční definici zadání. Toto může vést k situaci, kdy se výsledek úlohy jeví jako objektivní, ale prakticky by byla vhodnější jiná varianta. [8][9]

V modelech vícekriteriální analýzy variant je stanovena diskrétní množina variant. Každá varianta je ohodnocena podle stanovených kritérií. Cílem je najít takové řešení,

které je podle daných kritérií ohodnoceno nejlépe, a následně seřadit varianty od nejlepší po nejhorší. Taková varianta se nazývá optimální, či kompromisní. Konečné hodnocení variant je závislé na důležitosti stanovených kritérií. Pro stanovení důležitosti kritérií se používají váhy, případně aspirační úrovně. Aspirační úrovně dělí kritéria na akceptovatelné a neakceptovatelné. Dále rozlišujeme ordinální informace, které slouží k uspořádání kritérií podle důležitosti a podle toho, jak jsou ohodnoceny. Další formou je kardinální informace, která má kvantitativní charakter a v případě preferencí se jedná o váhy. K uspořádání dat slouží kriteriální matice Y, kde prvek yij vyjadřuje hodnocení i-té varianty podle j-tého kritéria. Kritéria se dále rozdělují na minimalizační a maximalizační. Výsledkem je pouze některá nedominovaná varianta, ke které neexistuje varianta, která by byla podle všech kritérií lepší nebo s ní rovnocenná. Varianta, která dosahuje ze všech kritérií nejlepší hodnoty se nazývá ideální varianta. Opakem ideální varianty je varianta bazální, která naopak dosahuje nejhorších hodnot. [8][9]

2.3 Metoda bodovací

Metoda bodovací spočívá v tom, že každé z variant podle daného kritéria přiřadíme počet bodů v rámci předem stanovené bodovací stupnice. Bodovací stupnice může mít v podstatě jakýkoliv číselný rozsah, čím více bodů, tím je kritérium více preferované. Je přípustné používat i desetinná místa a více kritériím lze přiřadit stejnou bodovou hodnotu. Celkové hodnocení je poté rovno součtu hodnot bodů. Optimální varianta je následně vybrána ta, které dosáhla v součtu bodů nejvyšší hodnoty. [8]

2.3.1 Návrh kritérií pro porovnání nástrojů pro automatizované testování

Na základě analýzy odborné literatury a konzultace s odborníky v oblasti automatizovaného testování jsem vymezil kritéria vhodná pro porovnání nástrojů pro automatizované testování. Následně byly stanoveny váhy pro jednotlivá kritéria. Stěžejním kritériem pro výběr vhodného nástroje je jeho funkčnost. Do funkčnosti jsem zařadil tři podkritéria, a to podporované programovací jazyky, tvorba testovacích scénářů a hlášení výsledků. Další oblastí kritérií je použitelnost nástroje, které popisuje testovací nástroj z pohledu uživatele, které s nástrojem pracuje. Do použitelnosti jsem zařadil kvalitu uživatelského rozhraní a správa testovacích případů. Další skupinou kritérií je podpora ze strany. Do této skupiny jsem zařadil kvalitu technické dokumentace.

2.3.2 Způsob bodování kritérií

Každý nástroj může být ohodnocen v rozmezí 0-1 bod, v níže uvedených tabulkách je uvedeno bodování jednotlivých kritérií a váhy kritérií přesněji popsáno. Váhy pro jednotlivá kritéria byly stanoveny na základě analýzy požadavků na testovací nástroje od firem zaměřených na automatizaci testování.

Podporované programovací jazyky – spektrum programovacích jazyků, které daný nástroj podporuje. Čím více programovacích jazyků nástroj podporuje, tím širší okruh firem může nástroj využívat, bez toho, aniž by museli najímat specialisty pouze pro daný programovací jazyk.

Tabulka 1 Kritérium Podporované programovací jazyky

Popis	Hodnocení
Není podporován žádný programovací jazyk	0
Podpora pouze jednoho programovacího jazyka	0.5
Podpora více než jednoho programovacího jazyka	1
Váha kritéria	0,124481328

Zdroj: Autor

Tvorba testovacích scénářů – dalším kritickým kritériem pro volbu nástroje pro automatizaci testování je způsob, jakým se samotné testovací scénáře v daném nástroji tvoří. Nejefektivnější způsob tvorby je pomocí záznamu operací uživatele v aplikaci s možností editace, naopak nejméně efektivní způsob, je potřeba všechny testovací scénáře naprogramovat ručně.

Tabulka 2 Kritérium Tvorba testovacích scénářů

Popis	Hodnocení
Testovací scénář lze vytvořit pouze ručním naprogramováním.	0.2
Testovací scénář je tvořen záznamem operací uživatele v testované aplikaci bez možností editace.	0.4
Testovací scénář je tvořen záznamem operací uživatele v testované aplikaci s možností editace.	1

Váha kritéria	0,165975104

Zdroj: Autor

Hlášení výsledků – klíčovou vlastností nástroje pro automatizované testování je hlášení výsledků po dokončení testování. Optimální testovací nástroj by měl mít vlastní funkci na vygenerování hlášení, v nejlepším případě možnost generovat hlášení automaticky po skončení testů.

Tabulka 3 Kritérium Hlášení výsledků

Popis	Hodnocení
Není možnost generovat hlášení výsledků, pro tuto funkcionalitu je potřeba instalace nástroje třetí strany	0.1
Hlášení výsledků je možno vygenerovat po ukončení testů pouze v příkazovém řádku	0.3
Hlášení výsledků je možné vygenerovat v tabulkovém, či jiném textovém formátu.	0.7
Hlášení výsledků lze vygenerovat automaticky po ukončení testů a obsahuje i grafické prvky	1
Váha kritéria	0,124481328

Zdroj: Autor

Uživatelské rozhraní – možnost ovládání testovací nástroje může být buď přes příkazovou řádku, to vyžaduje určité znalosti v příkazové řádce programovat, nebo může nástroj k ovládání používat vlastní grafické uživatelské rozhraní.

Tabulka 4 Kritérium Uživatelské rozhraní

Popis	Hodnocení
Nástroj využívá pouze textové rozhraní (příkazová řádka)	0.3
Nástroj využívá grafické uživatelské rozhraní	0.7
Váha kritéria	0,107883817

Zdroj: Autor

Správa testovacích případů – nástroj nabízí funkce k ukládání, editaci a správě testovacích scénářů, nabízí také možnost spravovat nalezené defekty v aplikaci.

Tabulka 5 Kritérium Správa testovacích případů

Popis	Hodnocení
Správa testovacích případů není k dispozici	0
Správa testovacích scénářů není k dispozici, ale podporuje nástroje třetích stran	0.5
Správa testovacích scénářů je plně integrovaná v nástroji pro testování	1
Váha kritéria	0,165975104

Zdroj: Autor

Technická dokumentace – pokud technická dokumentace zcela chybí, je nástroj ohodnocen nula body, neboť se jedná o zásadní kritérium. Pokud je technická dokumentace neúplná, je nástroj ohodnocen půl bodem. Pokud je k dispozici plná dokumentace, je nástroj ohodnocen jedním bodem.

Tabulka 6 Kritérium Technická dokumentace

Popis	Hodnocení
Technická dokumentace není k dispozici	0
Je k dispozici neúplná technická dokumentace	0.5
K dispozici je plná technická dokumentace	1
Váha kritéria	0,145228216

Zdroj: Autor

Podporované platformy – Podpora operačních systémů hraje stěžejní roli při výběru nástroje pro automatizované testování, neboť je nutné, aby testovací nástroj dokázal pokrýt všechny potřebné platformy, pro které je software vytvářen.

Tabulka 7 Kritérium Podporované platformy

Popis	Hodnocení
Podporuje pouze jeden operační systém	0.1
Podporuje všechny desktopové operační systémy	0.3

Podpora desktopových operačních systémů a webových nebo mobilních aplikací	0.7
Podpora desktopových operačních systému, webových aplikací a mobilních operačních systémů	1
Váha kritéria	0,165975104

Zdroj: Autor

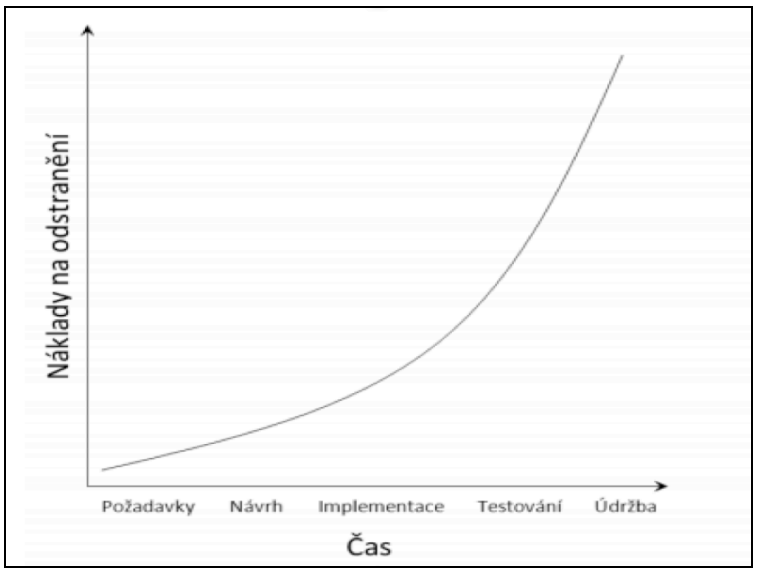
3 Teoretická východiska

3.1 Definice testování softwaru

Testování software je proces kontroly, zda testovaný software splňuje požadavky a specifikaci, podle které byl navržen a vyvíjen. Součástí testovacího procesu je také odhalování funkcionálních či grafických chyb, které vznikly v průběhu vývoje. Testování měří celkovou kvalitu softwaru z hlediska správnosti, výkonosti a použitelnosti. [14]

3.2 Důvody pro testování softwaru

Vývoj softwaru se neustále zrychluje a jeho složitost roste, spolu s tím rostou i požadavky na kvalitu vyvíjeného softwaru. Pokud se uvolní do produkce software, který obsahuje kritické chyby, může dojít k selhání či ztrátě dat uživatelů, které může následně přinést nevratné škody. Potřeba testovat software proto nabývá na stále větším významu, protože čím dříve se chyby odhalí, tím nižší jsou náklady na jejich odstranění a na vývoj produktu jako celku. [1] Obrázek číslo 1. poukazuje na hlavní důvod, proč je testování softwaru kritickou součástí vývoje softwaru.



Obrázek č. 1: Závislost nákladů na odstranění chyby a času

Pokud je proces testování zahájen včas, je možné nalézt defekty již v počáteční fázi vývoje. Pokud se nalezne chyba ve fázi specifikace požadavků, tak náklady na její odstranění jsou téměř nulové. Nalezení té samé chyby po nasazení do produkce může však exponenciálně zvýšit náklady na odstranění škod.

3.3 Cíle testování

Cílem testování je ověření, zda se testovaný software chová správně dle požadované specifikace. Pokud tomu tak není, je třeba upozornit na odlišné chování. Ve většině případů se upraví chování aplikace a následně je povinností testera tuto opravu zkontrolovat. Je třeba si ale uvědomit, že primárním cílem testování není hledat chyby, ale ověření, zda aplikace funguje podle předem stanoveného zadání. [1] [7] [14]

3.4 Definice pojmu „Chyba v softwaru“

V literatuře o testování softwaru se běžně můžeme dočíst, že pojem chyba v softwaru lze rozdělit na několik podkategorií, a to defekt, chyba a selhání, které se na první pohled zdají být podobné, jsou tam však důležité rozdíly. Níže jsou tyto podkategorie popsány. [2][14]

- Defekt je následek pochybení člověka a je příčinou chyby.
- Chyba je stav systému a může vést k selhání.
- Selhání je nesoulad mezi aktuálním a specifikovaným chováním systému.

O chybě v softwaru hovoříme v případě, že je splněna alespoň jedno nebo více z následujících pravidel [20]:

- Aplikace neprovádí něco, co by dle zadané specifikace měla provádět.
- Aplikace provádí, něco, co by podle údajů specifikace neměla provádět.
- Aplikace provádí něco, o čem se specifikace nezmiňuje.
- Aplikace provádí něco, o čem se specifikace nezmiňuje, ale měla by se zmiňovat.
- Aplikace je obtížně srozumitelná, velmi těžko se s ní pracuje nebo je příliš pomalá

3.5 Závažnost a priorita chyb

Jednotná klasifikace závažnosti chyb neexistuje. Většina firem používá svůj vlastní způsob podle svého přístupu k testování. Níže je uvedena obecná, často používaná čtyřstupňová škála závažnosti chyb.

Kritická - Chyba ovlivňuje kritické funkce systému a je nemožné jej správně používat, případně může dojít ke ztrátě dat.

Vysoká – Ovlivňuje základní funkce systému, ale systém je možné nadále v omezené míře používat.

Střední – Nemá vliv na základní funkcionalitu systému, je možné systém používat, nehrozí ztráta dat a všechny stěžejní funkcionality fungují.

Nízká – nemá vliv na jakoukoliv funkčnost systému. Zahrnuje zpravidla kosmetické vady jako například pravopisné chyby či jiné grafické chyby. [7]

Druhým pojmem v této souvislosti je priorita, která určuje jak urgentně je potřeba chybu opravit. Podobně jako u závažnosti ani zde neexistuje jednotná klasifikace. Z pohledu vývoje se v zásadě používají následující **stupně priority**.

Kritická – Nutnost provést opravu okamžitě. Zpravidla není možné pokračovat ve vývoji nebo testování dokud nebude chyba opravena.

Vysoká – Chyba částečně zabraňuje v dalším vývoji či testování. Opravuje je nutno provést, jakmile to bude možné.

Střední – Chyba nemá kritický dopad na funkčnost systému. Oprava bude provedena v rámci běžného pořadí oprav chyb.

Nízká – Chyba s nízkou prioritou je opravena až po opravení všech předchozích chyb nebo nemusí být opravena vůbec. [7]

3.4 Role v testovacím týmu

3.5.1 Tester

Tester je osoba, která se specializuje na provádění exekuce automatických testů nebo ručně prochází a testuje zadané testovací scénáře. Do nástroje pro správu chyb zaznamenává výsledky a nalezené incidenty či chyby na které během testovaní narazí. [2]

Tester provádí také opětovné přetestování již opravených chyb, které byly nalezeny při předchozím testování. Součástí práce testera je také komunikace s analytiky a vývojáři na analýze a popisu nesprávného chování aplikace nebo posouzení, zda se jedná o defekt nebo chybu ve specifikaci. [7]

3.5.2 Test manažer

Test manažer je stěžejní pozice v procesu testování, kterým nestačí pouze ovládat a dokonale rozumět testování, musí také zvládat komunikaci a umět řídit ostatní pracovníky zapojené do testování. Mimo řízení lidí musí být schopný implementovat a udržovat testovací proces v organizaci pro kterou testovací proces řídí. [7][19]

Test manažer nese zodpovědnost především za definování a implementaci testovacích rolí v rámci společnosti. Definuje rozsah testování v rámci jednotlivých testovacích iterací. Zodpovídá za výběr a implementaci testovacího frameworku. Mezi vedlejší úlohy manažera patří zaškolení testovacího týmu. [19]

3.5.3 Test analytik

Test analytik je osoba, která vytváří samotné testovací scénáře, testovací případy a vytváří dokumentaci k testování. Provádí také analýzu požadavků a je zodpovědná za vytvoření testovacích scénářů, které pokryjí celou aplikaci a analýzou jaká data budou potřeba nastavit nebo získat, aby bylo možné testování provést. [19]

Pro pozici analytika je kritické analytické myšlení, schopnost porozumět zadaní a na základě analýzy vytvořit odpovídající testovací scénáře. Pokud při analýze požadavků

zákazníka nalezne nějaké nejasnosti nebo přímo chyby, je potřeba to urgentně řešit a předat zpět k opravě či přepracování specifikace. [7]

3.5.4 Vedoucí test týmu

Vedoucí test týmu je osoba, která vede samotné testování, vytváří testovací plán a dohlíží na to, aby byl plán dodržován. Vedoucí test týmu má na starost nastavení a správu systému pro hlášení defektu nalezených při testování. Role vedoucího test týmu je velmi podobná roli test manažera, v praxi se můžeme setkat s tím, že obě tyto role jsou spojené do jedné a vykonává je jedna a ta samá osoba. [19]

3.6 Úrovně testování v procesu vývoje software

Během vývoje produktu je testování prováděno v různých úrovních, které mohou pokrýt celý systém nebo pouze jeho jednotlivé části. [1]

3.6.1 Testování jednotek

Testování jednotek je první úroveň testování, jedná se o testy, které provádí sám programátor a testuje jednotlivé malé komponenty software, které lze definovat jako nejmenší testovatelné součásti softwaru. Za jednotky můžeme považovat funkce, procedury nebo třídy. Smyslem je tyto jednotky otestovat izolovaně, nezávisle na ostatních a prokázat, že je jejich chování správné, proto jednotkové testy nesmí používat žádné další třídy, databáze nebo jiné externí zdroje dat, neboť by závislost na nich mohla ovlivnit výsledek testování. [3]

3.6.2 Integrační testování

Integrační testování je druhou úrovní, která navazuje na jednotkové testy. Do této fáze lze vstoupit, pokud máme již naprogramované alespoň dva moduly, které spolu určitým způsobem komunikují nebo si vyměňují data. Cílem integračního testováni je objevit chyby vzniklé spojením a následnou integrací těchto modulu do softwaru. Při provádění systémové integrace lze postupovat několika způsoby. [3]

Velký třesk

Integrace velkým třeskem znamená, že jsou všechny moduly integrovány zároveň v jednom jediném kroku. Takový postup je velmi snadný, ale má jednu velkou nevýhodu, pokud se objeví problém, je obtížné ho nalézt, protože se může nacházet v jakémkoliv modulu. Z toho důvodu je tento postup doporučený spíše pro méně složité systémy. [14]

Přístup shora-dolů

Integrace metodou shora-dolů je prováděna způsobem, kdy z hlediska hierarchické struktury systému začneme integrací největších modulů s nejrozsáhlejší funkcionalitou a postupně přidáváme menší moduly. Velkou výhodou tohoto způsobu integrace je to, že není potřeba čekat na vývoj všech modulů. [14]

Přístup zdola-nahoru

Je opakem přístupů shora-dolů. Začíná se integrací menších modulů a postupně jsou přidávány větší moduly. Z toho vyplývá, že systém jako celek nebude fungovat, dokud nebude integrován největší modul. [14]

Kombinovaný přístup

Kombinovaný přístup kombinuje souběžně přístupy shora-dolu a zdola-nahoru na třech úrovních. Na nejvyšší úrovni jsou integrovány hlavní moduly přístupem shora-dolu, na spodní úrovni jsou naopak integrovány menší moduly, přístupem zdola-nahoru, zbývající moduly jsou integrovány na střední úrovni. [14]

3.6.3 Systémové testování

Výsledkem úspěšného integračního testování je dostatečně stabilní systém, připravený pro fázi systémového testování, tato fáze je klíčová, neboť důkladně ověřuje, zda systém splňuje požadavky specifikované zákazníkem, a jde o poslední možnost, jak odchytit a opravit chyby předtím, než by je zákazník objevil. Systémové testování zahrnuje i různé typy testu k ověření funkčních i mimo funkčních požadavků, tyto typy testů jsou popsány níže. [5][7]

Funkční testy

Funkční testy jsou prováděny za účelem ověření, zda testovaný software splňuje specifikované požadavky, které jsou na jeho funkčnost kladeny. Zaměřuje se na detailní

testování všech funkcí, které byly do systému implementovány, zda fungují správně podle požadavků zákazníka. [11][15]

Bezpečnostní testy

Bezpečnostní testy ověřují zabezpečení testovaného systému. Používá přitom různé simulace možných operací. Testují, zda jsou data chráněna proti neoprávněnému získání. Také se zde testuje autorizace do daného systému, či zda jsou data správně zašifrována. Jakékoliv chyby odhalené při bezpečnostních testech by měli být řešeny okamžitě s nejvyšší prioritou, v opačném případě mohou být například uživatelská data odcizena. [11][15]

Testy použitelnosti

Testy použitelnosti patří mezi stěžejní a nejčastěji používané oblasti systémového testování. Tato metoda sleduje chování uživatelů, tím je možné dodatečně odhalit chyby, které by mohli zůstat neodhalené v předchozích fázích testování. Testy použitelnosti provádíme také, abychom otestovali, jak složité je pro uživatele se naučit ovládat jednotlivé prvky v systému. Testuje se zde také uživatelské rozhraní a přehlednost. [11]

Zátěžové testy

Zátěžové testy představují proces zahlcení systému velkým počtem požadavků. Provádí se za účelem analýzy chování daného systému za normálních i očekáváných pracovních podmínkách při maximálním zatížení. To pomáhá zjistit do jaké míry žádosti bude testovaný systém stabilní a jaké situace mohou nastat při překročení této hranice. [11]

3.6.4 Akceptační testování

Jakmile testovaný produkt úspěšně projde systémovým testováním, tak přichází fáze akceptačního testování koncovým zákazníkem. Ve fázi akceptačního testování se na produkt nahlíží jako na hotový, proto hlavním cílem není nalézt chyby, ale předat produkt koncovému zákazníkovi, který provede testování, zdali produkt splňuje všechny specifikace, podle kterých byl navržen. Pokud produkt neprojde akceptačním testováním, tak zpravidla nebude nasazen do produkce, ale předán zpět na vývoj k přepracování. Součástí akceptačního testovaní je i testování ostatních součástí produktu, mezi tyto součásti patří manuál k produktu a šablony dokumentu. [7][12]

3.6.5 Regresní testování

V průběhu vývoje produktu mohou byt některé jeho součásti modifikovány z důvodu opravy chyby nebo rozšíření funkčnosti. Takové změny mohou způsobit, že v již otestované komponentě vznikne nová chyba a pro je potřeba provádět průběžně regresní testování. Cílem regresního testování je ověřit, zda po změně nebo přidání nové komponenty do produktu zůstávají ostatní části funkce beze změny. Regresní testování využívá již existující testovací případy, aby se mohly porovnávat aktuální výsledky s dříve získanými výsledky. Regresní testování je velice nákladný proces, jelikož je po každé změně nutné celý systém otestovat znovu, proto test analytik vytváří balík regresních testů, který pokrývá nejdůležitější části aplikace, a tyto testy jsou pouštěny po každé změně v produktu a regresní testování celého produktu je prováděno až po dosažení většího počtu změn. [4]

3.6.6 Smoke a Sanity testování

Smoke testování je prováděno za účelem ověřit, zda je výsledný produkt dostatečně stabilní a všechny jeho hlavní komponenty funguji správně. Smoke testy se proto zaměřují hlavně na to, zda hledání vrací zpět výsledky, po spuštění dojde k připojení na server nebo zdali je možno se dostat do všech modulů. V dnešní době jsou smoke testy zpravidla automatizovány. [7][12]

Společně se smoke testy je prováděno také sanity testování. Sanity testy lze považovat za důkladnější provedení smoke testů. Testují nejen, zda hledání vrací správně výsledky podle specifických kritérií, nebo zdali je možno se připojit k serveru pouze pokud projde uživatel úspěšně autentizací. Sanity testování se dnes také provádí převážně automatizovaně. [7][12]

3.7 Typy testů podle velikosti

Malé testy

Do kategorie malých testů patří testování na úrovni jedné funkce nebo modulu. Zaměřují se hlavně na testování správnosti dat na vstupu a výstupu, správnost datových typů a zdali jednotlivé funkce nevrací chybu. Malé testy bývají zpravidla časově

nenáročné, každý jeden test běží v řádu vteřin. Ve většině případů jsou naprogramovány a běží automatiky bez zásahu člověka. [10]

Středně velké testy

Většina testů, která spadá do kategorie středně velkých, bývá zpravidla zautomatizovaná. Výjimku tvoří složitější testovací scénáře, které mohou být náročné na údržbu, v takovém případě se exekuce testů provádí manuálně. Důraz je kladen na otestování integrace zpravidla dvou, výjimečně tří funkcí nebo modulů, které spolu navzájem určitým způsobem komunikují. Důraz je kladen na testování správnosti výstupů, které přijímají vstupní data právě z více než jednoho modulu. [10]

Velké testy

Velké testy pokrývají testování tří a více funkcí či modulů, v praxi se můžeme setkat s počtem modulů a funkcí v řádech desítek. Velké testy simulují skutečné chování uživatele v aplikaci. Automatizace testování jde zde jen velmi obtížně z důvodu velké složitosti operací. Zpravidla vyžadují pro kontrolu lidský faktor. [10]

3.8 Manuální testování

Manuální testování je proces testování prováděný ručně člověkem s cílem nalézt chyby v softwaru bez použití nástrojů pro automatizované testování. Používá se při testování nových funkcionalit, nebo tam, kde nelze provést testy automatizací, například zadávání SMS klíče při simulaci bankovní platby. Výhodou manuálních testů je přítomnost lidského faktoru, který dokáže nad problémem přemýšlet na rozdíl od počítače, který při automatizaci jenom provádí naprogramované příkazy. Díky tomu je manuální testování vysoce flexibilní a adaptivní. Manuální testování je vhodné pro integrační, systémové a akceptační testování.

3.9 Automatizované testování

Automatizované testování je v praxi stále relativně podceňovanou disciplínou, ačkoliv nasazení automatizace do testovacích procesů může přinést úspory a zvýšit kvalitu výsledného produktu a zároveň může značně urychlit proces regresního testování. [7]

3.9.1 Význam automatizace testování

Hlavním cílem automatizace testování softwaru je časová úspora při jejich exekuci. Obecně lze říci, že automatizace testů softwaru má za účel usnadnit a zefektivnit provádění postupu testování softwaru. Pokud jsou testy prováděny zcela automaticky, tj. bez možnosti zásahu lidského faktoru, výrazně se tím snižuje možnost chybného provedení postupu testování softwaru a tím také pravděpodobnost bezporuchového provozu softwaru. [13]

3.9.2 Předpoklady pro zavedení automatizovaného testování

Každá firma, která se snaží zvýšit efektivitu, zpravidla vždy dospěje k zavedení automatizovaného testování. Jedná se však o nákladný a dlouhodobý proces, který vyžaduje rozsáhlou počáteční analýzu, zda je vůbec vhodné a možné automatizaci testovacích procesů zrealizovat. Níže jsou uvedeny předpoklady, které jsou obecně považovány za stěžejní předpoklady pro zavedení automatizace do procesu testování. [1]

- Dostatek času k zavedení do procesu.
- Dostatek finančních prostředků.
- Dostatečné množství zkušených lidí a vhodné nástroje.
- Software ve fázi, kde nedochází k častým změnám. Následná údržba testů po každé změně funckionality by se mohla prodražit.
- Vhodná strategie a plánovaní, zavedení automatizace bez představy o jejich přínosech často vede k neúspěchu a může značně zvýšit finanční náklady. [7]

3.9.3 Výhody automatizace testování

Mezi hlavní výhody automatizace testování patří časová úspora díky vysoké rychlosti provedení testů. Pokud se testy spouštějí a vyhodnocují automaticky, ušetří čas pracovníků, kteří tyto testy dříve manuálně prováděli. V optimálním případě lze zautomatizovat nejen samotné testy ale i proces testování. Pak lze testy provádět například přes noc bez jakéhokoliv zásahu člověka. Další významnou výhodou je možnost paralelně spouštět více testů najednou. [16][20]

3.9.4 Nevýhody automatizace testování

Automatizace testování sebou nese i jisté nevýhody, zejména vysokou počáteční finanční investici do vývoje a následné údržby testů. Pokud dojde ke změně v aplikaci je nutné testy aktualizovat. V delším časovém horizontu se investice do automatizace vyplatí. [16][20]

3.9.5 Testovací případy vhodné k automatizaci

Jak již bylo řečeno, automatizovat všechny testovací procesy není tak úplně praktické, přestože neexistuje jednoznačná odpověď na co všechno automatizovat a co testovat manuálně. Při výběru jaké testy automatizovat je vhodné zvážit následující kritéria. [13]

- Stabilita, testovací případy, které se často mění nebo dochází k častým změnám funkcionality testovaného softwaru, není vhodné automatizovat.
- Četnost provádění, automatizovat je vhodné takové testy, které se v rámci testovacího procesu často opakují například regresní testy.
- Časová náročnost, testy, které jsou náročné z časového hlediska, pokud jsou prováděny manuálně je optimální zautomatizovat.
- Složitost automatizace, některé testovací případy lze velmi snadno zautomatizovat (ověření číselné hodnoty), jiné je naopak téměř nemožné zautomatizovat (zobrazení správného grafického prvku). [6][10]

3.9.6 Techniky automatizovaného testování

K exekuci automatizovaného testování můžeme využít několik technik, které se od sebe odlišují v různých aspektech, do kterých můžeme zařadit například jak je snadné vybraný nástroj použít, jaké technické dovednosti jsou potřeba znát a podobně. Tyto techniky jsou postupně popsány níže. [7]

Zachycení a přehrávání aktivity uživatele je pravděpodobně nejznámější formou automatizace, která umožňuje dosažení vynikajících výsledků, a to bez potřebných technických znalostí. Tato technika funguje na principu, kdy testovací nástroj zachytává pohyb a aktivity uživatele v aplikaci a následně umožňuje tyto kroky zpětně přehrát. Ověření, zda test proběhl úspěšně je založeno na porovnání získané a její očekávané

hodnoty po dokončení testu. Předností této techniky je bezpochyby její jednoduchost a nenáročnost provedení, ovšem se zde vyskytuje několik nevýhod. Nejzásadnější nevýhodou lze považovat vysokou citlivost na změny v aplikaci, například pokud bude z aplikace odstraněno tlačítko, které bylo využito v testu, tak test nadále nebude možné dokončit a dosáhnout výsledku. [7]

Modifikace vygenerovaných skriptů je technika automatizovaného testování, která stejně jako zachycení a přehrání aktivity uživatele pracuje na zachytávání pohybu a aktivit uživatele v aplikaci, má ovšem jednu významnou funkčnost navíc. Zaznamenané skripty je možné různě upravit pomocí skriptovacích jazyků a implementovat tak složitější logiku a tím dosáhnout značného rozšíření testů o další funkcionality které je díky tomu možné otestovat. [16]

Testování řízené daty je technika vhodná pro situace, kdy je potřeba opakovaně testy, které využívají stejnou logiku, ale pracují s velkým počtem vstupů a výstupů, které se navíc mohou různě měnit. Vstupy a výstupy jsou odděleny od zdrojového kódu aplikace a jako jejich zdroj můžeme použít jednoduchou tabulku. Testovací skript poté začne načítat jeden řádek po druhém, provede potřebné operace a následně porovná získané výsledky s těmi očekávanými, které stejně jako vstupní data načte z nějakého databázového zdroje. [13]

Testování řízené klíčovými slovy je technika velmi podobná technice testování řízené daty. Lze říci, že funguje na stejném principu, odlišuje se tím, že neobsahuje pouze vstupní data, ale obsahuje také příkazy, ze kterých se stává testovací skript. Tyto příkazy jsou za běhu načítány a prováděny, tím je vytvořena samotná logika procedury testu. Tato technika s sebou přináší mnoho výhod, mezi které můžeme zařadit jednoduchost, velmi dobrou udržovatelnost a odolnost vůči změnám v aplikaci. [7]

4 Vlastní práce

Vlastní práce je zaměřená na porovnání nástrojů pro automatizované testování softwaru. Výběr vhodného nástroje může značně zvýšit efektivitu testování. Před samotným výběrem je nutné si stanovit, co od nástroje vlastně požadujeme. Nejprve je provedena analýza jednotlivých nástrojů a následně je provedeno porovnání vícekriteriální analýzou a postupem uvedeným v metodice bakalářské práce. Do porovnání byly zařazeny celkem čtyři v současné době nejvíce používané nástroje pro automatizované testování, a to jsou HP Unified Functional Testing, TestComplete, SeleniumIDE a SilkTest.

4.1 HP Unified Functional Testing

Je nástroj pro automatizované testování od společnosti Hewllet-Packard určený pro testování různých softwarových aplikací a prostředí. V současnosti zahrnuje vlastnosti starších nástrojů od společnosti Hewllet-Packard jako jsou Quick Test Professional, WinRunner či HP Service.

Lze automatizovat testy zaznamenáváním akce uživatele a následně je přehrát a porovnat skutečné chování aplikace s předpokládaným chováním. Tyto nahrané operace se ukládají ve formě jednoduchého programu známého jako skript. Další jeho klíčovou vlastností je schopnost identifikovat objekty v uživatelském rozhraní na základě jména nebo unikátního identifikátoru. Nástroj lze používat jak v příkazové řádku, tak s grafickým rozhraním. Nástroj nemá vlastní správu defektů a testovacích scénářů, je tedy nutné je spravovat za použití nástroje třetí strany, lze snadno integrovat například HP Alm. Mimo oblast testování lze nástroj použít k automatizaci opakujících se úkolů, které by jinak musely být provedeny lidským uživatelem.

Klíčové vlastnosti HP Unified Functional Testing:

- podpora všech platforem
- integrace s nástrojem HP Alm
- snadná konfigurace nástroje

Slabé stránky:

- podpora pouze jednoho programovacího jazyka

- nemá vlastní nástroj pro správu testovacích scénářů

Podporované platformy:

- Windows XP a novější
- MacOS X 10.5 Leopard a novější
- Linux
- Android 4.4 a novější
- iOS 8 a novější

Podporované programovací jazyky:

- Visual Basic

Tabulka 8 Shrnutí HP Unified Functional Testing

Kritérium	Body
Podporované programovací jazyky	0.5
Tvorba testovacích scénářů	1
Hlášení výsledků	0.7
Uživatelské rozhraní	0.7
Správa testovacích scénářů	0.5
Technická dokumentace	0.5
Podporované platformy	1

4.2 TestComplete

TestComplete je platforma pro automatizaci testování, vyvinutá společností SmartBear zaměřená na testování aplikací v prostředí Windows a hlavně mobilních operačních systému Android a iOS, ale využití nalezneme i při testování databázových systémů. Testovací scénáře lze vytvářet několika způsoby. Prvním způsobem je zaznamenání operací uživatele, které provede v aplikaci, druhým způsobem je vytvoření scénáře pomocí klíčových slov, poslední možností je naprogramování testovacích skriptů ručně a jejich následná exekuce. TestComplete je rozdělen do několika modulů, modul pro

desktopové aplikace, modul pro webové aplikace a modul pro mobilní aplikace. Každý modul obsahuje funkce pro vytváření automatizovaných testů na zvolené platformě.

Klíčové vlastnosti TestComplete:

- zabudovaný editor pro psaní testovacích scénářů s možností rozšíření o další pomocné moduly do editoru
- možnost paralelně spouštět více testů najednou
- možnost vytvořit si vlastní rozšíření do aplikace podle své potřeby
- kvalitně zpracovaná dokumentace a podpora od výrobce

Slabé stránky:

- určeno pouze pro platformu Windows

Podporované platformy:

- Windows XP a novější

Podporované jazyky:

- Javascript
- Python
- VBScript
- C#
- C++

Tabulka 9 Shrnutí TestComplete

Kritérium	Body
Podporované programovací jazyky	1
Tvorba testovacích scénářů	1
Hlášení výsledků	0.7
Uživatelské rozhraní	0.7
Správa testovacích scénářů	0.5
Technická dokumentace	1
Podporované platformy	0.1

4.3 SeleniumIDE

SeleniumIDE je volně dostupný nástroj pro automatizované testování zaměřené primárně na web a webové aplikace, který byl vyvinutý společností ThoughtWorks. Jedná se o rozšíření do prohlížeče, které nabízí propracovanou možnost zaznamenávání a přehrávání operací uživatele v aplikaci a následné porovnání výsledků. Jedná se o nástroj určený primárně k rychlému testování a díky své jednoduchosti a rychlosti konfigurace je možné, během několika minut začít testovat. Testovací scénáře lze vytvářet bez znalosti programování, proto je tento nástroj velmi populární. SeleniumIDE lze rozšířit o Selenium Webdriver, který nabízí možnost do nahraných testů zasahovat a upravovat je pomocí programování, kde nabízí širokou škálu programovacích jazyků, které lze použít k vývoji a úpravě testovacích scénářů. Velkou nevýhodou je nemožnost pracovat s externími daty. SeleniumIDE taktéž nemá vlastní modul pro správu testovacích scénářů ani hlášení chyb a je nutné implementovat dodatečně od jiného výrobce.

Klíčové vlastnosti:

- snadná konfigurace
- rychlost vývoje testovacích scénářů
- úplná technická dokumentace
- široká podpora programovacích jazyků
- volně dostupný nástroj

Slabé stránky:

- nemožnost práce s externími daty
- chybějící modul pro správu testovacích scénářů a chyb

Podporované platformy:

- Windows XP a novější
- MacOS 10.5 Leopard a novější
- Linux
- Android 4.4 a novější
- iOS 7 a novější

Podporované programovací jazyky:

- Java
- C#
- Python
- Ruby
- Perl
- Php
- Javascript

Tabulka 10 Shrnutí SeleniumIDE

Kritérium	Body
Podporované programovací jazyky	1
Tvorba testovacích scénářů	1
Hlášení výsledků	0.3
Uživatelské rozhraní	0.7
Správa testovacích scénářů	0
Technická dokumentace	1
Podporované platformy	0.7

4.4 SilkTest

Je nástroj pro automatizované testování se zaměřením na podnikové aplikace vyvinutý firmou Segue Software, který dnes vyvíjí a spravuje společnost Micro Focus International. Mimo podnikové aplikace je možné testovat i mobilní a webové aplikace. SilkTest používá mimo jiné specifický 4Test jazyk pro vývoj a programování automatizovaných testů a umožňuje testovat na vizuální úrovni. Ve většině případů používáme SilkTest při testování metodou černé schránky přes grafické rozhraní. SilkTest je má velmi dobře zpracovaný systém pro správu testovacích scénářů a defektů, není tedy nutné implementovat externí nástroj.

Klíčové vlastnosti:

- robustní framework pro testování mobilních operačních systémů
- možnost spouštět více testů najednou
- integrovaný systém pro správu testovacích scénářů
- podpora testování podnikových aplikací
- kvalita uživatelského rozhraní

Slabé stránky:

- neúplná technická dokumentace
- podpora pouze platformy Windows

Podporované Platformy:

- Windows XP a novější

Podporované programovací jazyky:

- Java
- Python
- VBScript

Tabulka 11 Shrnutí SilkTest

Kritérium	Body
Podporované programovací jazyky	1
Tvorba testovacích scénářů	1
Hlášení výsledků	0.3
Uživatelské rozhraní	0.7
Správa testovacích scénářů	0.5
Technická dokumentace	0.5
Podporované platformy	0.1

5 Výsledky a diskuze

Tabulka 12 Porovnání výsledků

Kritérium	HP Unified Functional Testing	Test Complete	Selenium IDE	Silk Test	Váha kritéria
Podporované programovací jazyky	0,062240664	0,124481328	0,124481328	0,124481328	0,124481328
Tvorba testovacích scénářů	0,165975104	0,165975104	0,165975104	0,165975104	0,165975104
Hlášení výsledků	0,087136929	0,087136929	0,037344398	0,037344398	0,124481328
Uživatelské rozhraní	0,075518672	0,075518672	0,075518672	0,075518672	0,107883817
Správa testovacích scénářů	0,082987552	0,082987552	0	0,082987552	0,165975104
Technická dokumentace	0,072614108	0,145228216	0,145228216	0,072614108	0,145228216
Podporované platformy	0,165975104	0,01659751	0,116182573	0,01659751	0,165975104
Součet bodů s ohledem na váhu kritéria	**0,712448133**	**0,697925311**	**0,66473029**	**0,575518672**	

Výsledné pořadí:

1) HP Unified Functional Testing – **0,712448133** bodu (Kompromisní varianta)
2) TestComplete – **0,697925311** bodu
3) SeleniumIDE – **0,66473029** bodu
4) SilkTest – **0,575518672** bodu

V celkovém bodování bylo zjištěno, že mezi zkoumanými nástroji pro automatizované testování není znatelný rozdíl. Univerzální nástroj, který by byl schopný pokrýt všechny potřeby při testování zatím neexistuje. Po celkové stránce vychází nejlépe Nástroj HP Unified Functional Testing, který obsahuje všechny stěžejní funkce, ale testovací scénáře lze programovat pouze v jazyce Visual Basic. To může být pro některé firmy problém, pokud nemají lidské zdroje, které Visual Basic ovládají.

Nástroj TestComplete také obsahuje všechny důležité funkce, podporuje více programovacích jazyků včetně javascriptu, který je v současnosti nejrozšířenějším programovacím jazykem na světě a má velmi kvalitně zpracovanou dokumentaci, ovšem lze použít pouze pro platformu Windows, to lze považovat za jediný problém tohoto nástroje.

SeleniumIDE je výborná alternativa určená spíše pro prototypování testovacích scénářů, díky podpoře mnoha programovacích jazyků a možnosti tvořit testovací scénáře i bez programování. Nemá vlastní nástroj pro správu dat a testovacích scénářů, proto je nutné použít externí nástroj, který nelze ve všech případech plně implementovat.

Nástroj SilkTest je vhodný pro velké podnikové aplikace, navíc disponuje širokou podporou programovacích jazyku hlavně jazykem java, který je v podnikových aplikacích hojně využíván, dále kvalitním uživatelským rozhraním. Každý nástroj má své výhody a nevýhody, celkově bude záležet vždy hlavně na konkrétních preferencích podniku nebo člověka jaký nástroj ve výsledku vybere.

6 Závěr

Teoretická část bakalářské práce měla úkol vymezit elementární teoretické poznatky z oblasti testování softwaru. Byly popsány důvody a cíle proč je důležité software testovat již od počátku vývoje, vymezen pojem chyba v softwaru, lidské role v testovacím procesu a úrovně testování v celém životním cyklu softwaru. V teoretické části se práce také zaměřuje na charakteristiku automatizovaného testování, které je čím dál více populární a mnoho firem jej nově zavádí za účelem urychlení testovacího procesu a jako investici do budoucna. Díky správně zavedené automatizaci lze v delším časovém horizontu očekávat úsporu nákladů za testování. Veškeré teoretické poznatky byly získány na základě analýzy odborné literatury a dostupných zdrojů. Všechny tyto pojmy se mohou v různých literaturách nepatrně lišit, je tedy dobré čerpat z podobných zdrojů, aby se nerozcházely.

Praktická část bakalářské práce se zaměřuje na analýzu nástrojů pro automatizované testování a jejich porovnání. Porovnání bylo provedeno vícekriteriální analýzou za základě zvolených kritérií a stanovených vah. Výsledky byly syntetizovány v závěru vlastní práce. Na základě těchto výsledků bylo možné vhodně vyhodnotit celkové srovnání nástrojů pro automatizované testování software. Na základě analýzy a porovnání bylo zjištěno, že nelze vybrat jeden nejlepší nástroj, který by splňoval všechny požadavky. Každý nástroj má své klady a zápory, případně unikátní vlastnost. Před výběrem je důležité si stanovit požadavky a kritéria, která od požadovaného nástroje očekáváme a na jejich základě postupovat při samotném rozhodování jaký nástroj zvolit.

Jak bylo řečeno v úvodu bakalářské práce, autor se oblasti testování software věnuje několik let, přes to došel při výzkumu a psaní práce k mnoha novým a zajímavým poznatkům. Od samotného výsledku výzkumu až po pohledy odborníků z praxe se kterými měl možnost na výzkumu vlastní práce spolupracovat.

7 Seznam použitých zdrojů

1. PATTON Ron. *Testování software*. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
2. AMMANN, Paul, OFFUTT, Jeff. *Introduction to software testing.* Cambridge: Cambridge University Press, 2008. ISBN 978-0521880381.
3. HASS, Anne Mette Jonassen. *Guide to Advanced Software Testing.* London: Artech House, 2008. ISBN 9781596932869.
4. MATHUR, Aditya P. *Foundations of Software Testing*, 2. vyd. Delhi: Pearson Education India, 2008. 978-8131716601.
5. MYERS, Glenford J., SANDLER, Corey, BADGETT, Tom. *The art of software testing*. Hoboken: John Wiley & Sons, 2011. 978-1118031964.
6. PUSULURI, Nageshwar Rao. *Software Testing Concepts And Tools*. New Delhi: Dreamtech Press, 2006. 978-8177227123.
7. ROUDENSKÝ Petr, HAVLÍČKOVÁ Anna, *Řízení kvality softwaru*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
8. ŠUBRT Tomáš a kolektiv, *Ekonomicko-matematické metody.* Plzeň: nakladatelství Aleš Čeněk, 2001. ISBN 978-80-73803345-2.
9. FIALA, Petr, JABLONSKÝ, Josef, MAŇAS Miroslav. *Vícekriteriální rozhodování*. Praha: VŠE, 1994. ISBN 80-7079-748-7.
10. WHITTAKER James, ARBON Jason, CAROLLO Jeff, *How Google Tests Software*. Bostonu: Addison-Wesley Professional, 2012. ISBN 978-0321803023.
11. PAGE Alan, ROLLISON Bj, JOHNSTON Ken, *Jak testuje software Microsoft*, Brno: Computer press. 2009. ISBN 978-80-251-2869-5.
12. KANER Cem, FALK Jack, NGUYEN QUOC Hung, *Testing Computer Software*. New York: Wiley, 1999. ISBN 978-0471358466.
13. GARG Navneesh, *Test Automation using HP Unified Functional Testing: Explore latest version of QT*. Auckland :AdactIn Group Pty Limited, 2013. ISBN 978-0992293505.
14. NAIK Kshirasagar, TRIPATHY Priyadarshi, *Software Testing and Quality Assurance: Theory and Practice*. Wiley-Spektrum; 2008. ISBN 978-0471789116.

15. PERRY E. William, *Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates*, Hoboken: John Wiley & Sons; 2006. ISBN 978-0764598371.
16. LI Kanglin, WU Mengqi, *Effective Software Test Automation: Developing an Automated Software Testing Tool*. Sybex, 2004. ISBN 978-0782143201.
17. CRISPIN Lisa, *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston: Addison-Wesley Professional, 2009. ISBN 978-0321534460.
18. Selecting Automated Testing Tools. *SmartBear* [online]. Boston: 2015 [cit. 2017-03-06]. Dostupné z: https://smartbear.com/learn/automated-testing/selecting-automated-testing-tools/
19. Software Testing Roles and Responsibilities. *Software Testing Roles and Responsibilities - International Software Test Institute* [online]. Zurich: 2015 [cit. 2017-03-06]. Dostupné z: http://www.test-institute.org/Software_Testing_Roles_And_Responsibilities.php
20. Testování softwaru. *Testování softwaru* [online]. Praha, 2013 [cit. 2017-03-06]. Dostupné z: http://testovanisoftwaru.cz//?s=chyba