Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Automatizované funkční testování webových aplikací

Bakalářská práce

Autor: Tomáš Nechanický
Studijní obor: Informační management

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové Srpen 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne Tomáš Nechanický

Poděkování:

Děkuji své vedoucí bakalářské práce paní Mgr. Daniela Ponce, Ph.D. za metodické vedení práce, čas a připomínky. Také bych chtěl poděkovat přátelskému kolektivu ŠKODA AUTO a.s. za vřelé přijetí a v průběhu psaní této bakalářské práce. V neposlední řadě bych také rád poděkoval mé rodině a přátelům za pomoc a podporu během studia.

Anotace

Bakalářská práce je zacílena na funkční testování webových aplikací ve vybraných nástrojích pro automatické testování. Hlavní úkolem této bakalářské práce je zavedení automatizace v oblasti funkčního testování do společnosti ŠKODA AUTO a.s.. Pro tento účel byly vybrány nástroje Selenium WebDriver a IBM Rational Functional Tester. V teoretické části jsou popsány obecné metodiky a druhy testů, které jsou blíže popsány a rozpracovány do jednotlivých kroků testovacích procesů. V rámci obecných metodiky testování jsou jsou rozepsány také fáze testování webových aplikací. Práce obsahuje také podrobný popis obou vybraných nástrojů s vysvětlenými principy jejich fungování. V praktické části je rozepsána vlastní tvorba automatických testovacích skriptů dle platných testovacích scénářů v obou nástrojích. V poslední části je hodnocena práce s nástroji a celkové výsledky porovnány. Na základě práce s těmito nástroji jsou sepsány doporučené postupy pro testování webových aplikací.

Annotation

Title: Automated functional testing of web applications

The bachelor thesis is focused on functional testing of web applications in selected tools for automatic testing. The main task of this bachelor thesis is the introduction of automation in the field of functional testing in the company ŠKODA AUTO as. For this purpose were chosen tools Selenium WebDriver and IBM Rational Functional Tester. The theoretical part describes general methodologies and types of tests, which are described in more detail and elaborated into individual steps of testing processes. As part of the general testing methodology, the stages of web application testing are also described. The thesis also contains a detailed description of both selected tools with explained principles of their functioning. The practical part describes the creation of automatic test scripts according to valid test scenarios in both tools. The last part evaluates work with tools and overall results are compared. Based on the work with these tools are written best practices for testing web applications.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

.Net	Dotnet
AJAX	Asynchronous JavaScript and XML
ANSI/IEEE	American National Standards Institude / Institute of Electrical and Electronics Engineers
API	Application Programming Interface
B2C	Bussiness to Customer
CSRF	Cross-site Request Forgery
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ID	Identificator
IDE	Integrated Development Environment
JEE	Java Enterprise Edition
JRE	Java Runtime Enviroment
RC	Remote Control
SAP	Systems, Applications, Products in data processing
SQL	Structured Query Language
SSL	Secure Sockets Layer
TC	Test Case
VB.NET	Visual Basic .Net (dotnet)
VV & T	Verification, Validation and Testing
XML	Extensible Markup Language
XPath	XML Path Language
XSS	Cross-Site Scripting

1 Úvod

Po samotném vývoji je testování nejdůležitějším aspektem v průběhu životního cyklu vývoje softwaru. Testování je cílené hledaní bugů, tedy chyb. V této bakalářské práci je popsán proces automatizace funkčního testování webové aplikace.

Úvodem teoretické části se nachází literární rešerše spolu s články autorů píšících na toto téma. V následující kapitole je stručně popsán aktuální vývoj testování webových aplikací. Ačkoliv jsou webové aplikace specifickou softwarovou skupinou, platí pro jejich testování stejná pravidla a principy jako pro jiné softwary. Dále jsou zpracovány obecné metodiky pro testování. Velmi častou chybou bývá nedostatečná znalost jednotlivých fází testovacího procesu nebo jeho podcenění, vynechání či označení za nedůležitý. Ve čtvrté kapitole jsou popsány funkce vybraných komerčních a open source nástrojů určených k testování webových aplikací. Pro tuto práci byly vybrány nástroje z rodiny Selenia a nástroj IBM Rational Functional Tester. Výhody a nevýhody obou nástrojů jsou vypsány a porovnány.

V praktické části práce je zpracována příprava a tvorba testovacích scénářů a jejich životní cyklus při jejich následné automatizaci. Na jejich základě jsou následně vytvořeny automatické testovací skripty pro oba vybrané testovací nástroje, které jsou aplikovány na B2C e-shopu ŠKODA AUTA a.s.. V závěru práce jsou výsledky shrnuty a na jejich základě byla vytvořena obecná doporučení pro budoucí automatické funkční testování na ŠKODA e-shopu.

1.1 Cíl práce

Cílem práce je popsat metody funkčního testování webových aplikací, použít vybrané nástroje testování pro ukázkovou webovou aplikaci a následně nástroje porovnat a vyhodnotit.

Dalším cílem bakalářské práce je aktualizace a vytvoření testovacích scénářů pro webové stránky. Pro tuto práci byl vybrán B2C (business to customer) internetový obchod společnosti ŠKODA AUTO a.s. Společnost ŠKODA AUTO a.s. spravuje svůj vlastní vývoj internetového obchodu. Tento e-shop má za úkol prodej originálního příslušenství a propagačních materiálů zákazníkům po celé České republice

Z plynoucí velké časové náročnosti na jednotlivá opakující se testování bylo vhodné navržení a realizace automatického testování. Přepsáním testovacích scénářů do formy automatických testovacích skriptů v nástrojích Selenium WebDriver a IBM Rational Functional Tester. Dále je popsán postup realizace testů pro jednotlivé vybrané nástroje a v konečné kapitole jsou výsledky vyhodnoceny, shrnuty a nástroje jsou mezi sebou porovnány.

2 Metodika zpracování

V následující kapitole nachází literární rešerše spolu s články autorů, která uvádí pohled na téma testování softwaru zpět do minulosti. V další kapitole jsou vypsány nejdůležitější trendy současného vývoje testování webových aplikací.

Metodikou zpracování této práce je vlastní vytvoření testovacích scénářů pro testování a na jejich základě zhotovení automatických testovacích skriptů pro vybrané testovací nástroje. Na základě řízených testů, tedy experimentů, jsou vyhodnoceny výsledky a oba pozorované programy jsou srovnány. V závěru práce jsou výsledky shrnuty a následně zanalyzovány.

Hlavním cílem této práce je sice zavedení automatizovaného testování, ale také nalezení odpovědi na otázku, který testovací nástroj je pro automatizaci lepší a výhodnější.

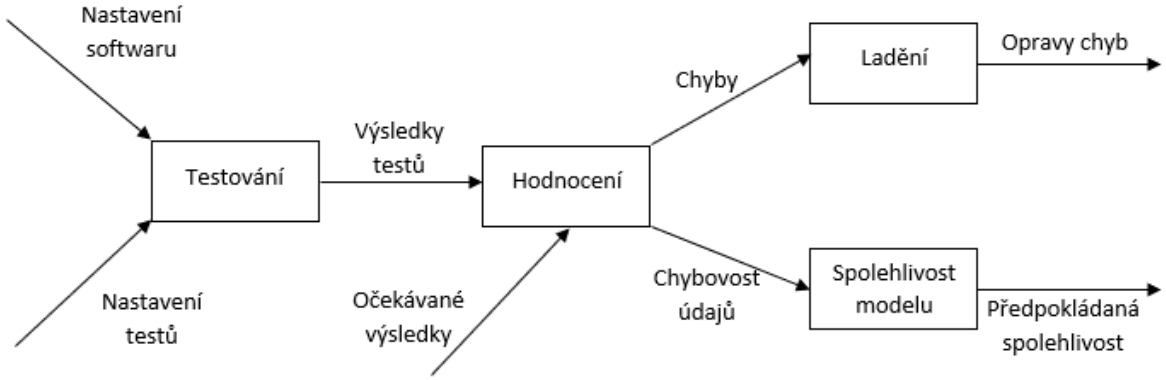
2.1 Rešerše z oblasti historie testovacích technik

Se samotným počátkem vývoje softwaru přichází i začátek testování. Ačkoliv se koncept testování v průběhu času mění, tak samotný vývoj testovacích cílů a definic se podílel až výzkum testovacích technik. Obsahem této rešerše je vývoj koncept testovacího procesu navrženého Gelperinem a Hetzelem [1] v roce 1988.

Počátky testování se vrací zpět v čase do roku 1950, kdy slavný Alan Turing napsal článek, který je zcela nepochybně prvním programovým testem. Článek se snaží odpovědět na otázku: Jak můžeme určit, že program vykazuje inteligentní chování? Jinak řečeno se ptá, zda je chování programu takové, jak bylo původně specifikováno. Dalším zvláštní případem této otázky je: Jak můžeme určit, že program vyhovuje požadavkům? Alan Turing dále definoval provozní test, ve kterém je vymezeno chování programu podle požadavků a člověk (referenční systém), tak aby je nebylo možné od sebe odlišit pro třetí osobu, která prováděla test (tester). Jedná se o první opravdový funkční test. [2]

Od roku 1957 bylo testování odlišeno a vyčleněno od pouhého termínu „zkoušení programu“. Do té doby převládalo samotné ladění programu nad testováním. Charles Baker stanovil dva hlavní úkoly testování: Zjištění, zda program běží a ověřit, zda chování programu řeší zadaný problém. Dalším cílem bylo zaměření na testování a zjištění, do jaké míry aplikace uspokojuje zadané požadavky. Je nutné rozlišovat ladění a testování, jelikož se jedná o dva docela rozdílné procesy, více je možné vidět na obrázku č. 1.

Rozdíl mezi laděním a testováním spočívá v definici úspěchu. V období 60. let 20. století byl vyvíjen tlak na prokázání správnosti testování: Ideálním testem může být pouze test na naprosto bezchybném programu. [3]



Obrázek 1 – Modelové schéma testování

Zdroj: vlastní tvorba

V roce 1979 napsal Glenford J. Myers knihu Umění testování softwaru, která byla základem pro efektivnější návrhy technik testování. Testování softwaru bylo poprvé popsáno jako záměrný proces nalézání chyb v průběhu provádění programu. Počet stanovených testovacích případů bývá větší, pokud je nalezená chyba. Některé chyby s nízkou pravděpodobností bývají nalezeny spontánně a často nevědomky a způsobit selhání testované aplikace. Pokud je účelové testování efektivní pro hledání chyb v programu bude mít logicky i mnohem větší pravděpodobnost.

Národní Institut Standardů a Technologii v roce 1983 vydal Zásady pro ověřování životního cyklu, verifikace a testování počítačového softwaru, které byly popsány jako metody posuzování, zkoušení a analýzy ke správnému zhodnocení stavu softwaru v průběhu jeho životního cyklu. Snaží se o přesvědčení, že pečlivě vybraná sada VV & T[1] technik může být nápomocna k zajištění kvalitního softwaru.

Se snahou o předvídání chyb již z požadavků návrhu a požadavků se setkáváme ji od roku 1988. V knize Boris Beizer je uveden nejvíce obsáhlý katalog zkušebních metod testování softwaru [4]. V průběhu vývoje známe tyto modely jako hodnotící model. Boris Beizer tvrdí, že nejefektivnější metodou prevence před chybami je navrhování testů.

[1] Verification, validation and testing

William C. Hetzel definoval testování v roce 1991 jako projektování, sestavování, plánování, udržování a provádění testů na testovacích prostředích. V roce 1990 vyjádřil čtyři myšlenky ohledně testovacího procesu:

1. Zajistit funkční software
2. Donutit software selhat
3. Snížit softwarová rizika
4. Stav mysli, tj. celkový životní cyklus obav z testovatelnosti

Tato doporučení vedla ke zdůraznění významu návrhu testování v co nejranějším vývoji softwaru, ideálně již na počátku životního cyklu. Celkovým cílem bylo zabránění implementace chyb již při zadávání požadavků anebo specifikace designu. Hodnotící model je závislý na analýze chyb a eventuelně na jiných metodách testování. Oproti tomu v preventivním modelu více kladem důraz na plánování, test a následnou analýzu. S příchodem webu začalo i jeho testování, ke kterému stačí pouhý webový prohlížeč s možností zobrazení zdrojového kódu, jelikož zpočátku byl web převážně statický.

Obecně je testování rozděleno na testování bílé, šedé nebo černé skříňky. Dále také pak na testování u uživatele, na které se zaměřuje ve svém díle Testování Softwaru [5] Ron Patton. V publikaci Lydia Ash [6] je možné se dovědět o konkrétních druzích a typech testování od zátěžového testování až po jednotlivá bezpečnostní rizika. Například zabezpečení za pomocí SSL, HTTPS či proxy serverů v síťové komunikaci.

2.2 Současný vývoj testování webových aplikací

V současné době dochází k progresivnímu vývoji webových aplikací za pomocí využití nových verzí jazyku HTML, PHP nebo SQL. Většina webů je dnes již optimalizována, jak na zobrazení v běžném prohlížeči, tak i na mobilních zařízeních. Je proto nutné zohlednit v průběhu testování různá rozlišení, aby byla funkce ověřena a byla dostupná i na mobilních telefonech a tabletech.

Nevýhodou v automatizaci je nutnost psaní programového kódu pro každou validaci.
Jedním ze současných trendů testování je vizuální testování aneb pokud není žádoucí na všechny psát testovací skripty. Tímto druhem testování nelze ověřit úplně vše, nicméně není nutný žádný kód pro validaci, případně ani žádné selektory. Nejdříve je proveden scénář testu, ze kterého se na webové aplikaci udělá snímek obrazovky, který se následně porovnává s předpřipraveným očekáváným snímkem obrazovky. Oba obrázky jsou porovnány a v případě neshod jsou reportovány jako chyby. Výhodou je možnost najití chyb, které jsme vůbec nehledali. Mohou doplnit část funkčních testů. Je také možné je automatizovat. [20]

Další možností jsou testovací služby, tzn. cloud testing services, jež představující novinku v testování webových aplikací. Ty poskytuje třetí strana v podobě kompletního řešení. Například se jedná o vzdálený přístup na linuxový server, ve kterém běží testovací systém s mnoha prohlížeči a testing bot. Vše běží na cloudovém serveru se vzdáleným přístupem. Velkou výhodou této služby je velké množství prohlížečů a žádné nároky na vlastní infrastrukturu. Bohužel nevýhodou je cena takovéhoto testování, proto je vhodné ho použít jako doplněk k ostatním druhů testování. [19]

3 Obecné metodiky testování

Testováním se rozumí cílené hledání chyb. Na počátku je nezbytné stanovení, co je chybou a co chybou není. Každé neočekávané chování softwaru je chybou až po posouzení testera, zda je výsledné chování správné nebo není. Tímto chováním může být cokoliv od chybových hlášek až samotný pád aplikace. Z pohledu uživatele je to cokoliv, co by se mu nelíbilo nebo nějaká část softwaru, která mu jeho práci s aplikací nezlehčuje (je přebytečná). Chybou aplikace může být i stav, se kterým se aplikace nedokáže vypořádat. Jednou z nejvíce uznávaných definic pro chybu byla popsána Ronem Pattonem v 5 bodech:

1. *Software nedělá něco, co by podle specifikace produktu dělat měl.*
2. *Software dělá něco, co by podle údajů specifikace produktu dělat neměl.*
3. *Software dělá něco, o čem se produktová specifikace nezmiňuje.*
4. *Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.*
5. *Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo - podle názoru testera softwaru - jej koncový uživatel nebude považovat za správný.*

[5 str. 14]

Proces testování je možné definovat jako ověření reálných vlastností softwaru, zda oproti jejich očekávaným a požadovaným vlastnostem. Smyslem testování je porovnání softwarových kladů a záporů za účelem zjištění a zhodnocení jeho kvality. Zhodnocení kvality je možné docílit vícero způsoby. Pro správné testování je nutné stanovení druhů testů, míru a čas testování. Pro automatické testy je dále ještě nutné stanovení vhodného testovacího nástroje v závislosti na konkrétním druhu testu. Druh a způsob testování je odvislí od konkrétní aplikace či produktu. Správný výběr je vždy důležitý, neboť chybný výběr by mohl zapříčinit neodhalení velkého množství chyb, jejíchž následné odstranění v průběhu dalšího vývoje, by byly mnohem větší než na počátku testování.

3.1 Druhy testování

Jedním z nejdůležitějších aspektů v počátku vývoje softwaru je životní cyklus vývoje softwaru. Podobné je to i s výběrem správného druhu testování.

3.1.1 Testování bílé, černé a šedé skříňky

Testování bílé, šedé a černé skříňky je jedno ze základních typů rozdělení testování. Tato metoda je odvislá od znalostí a zkušeností testera.

Testování bílé skříňky (white box testing) je strukturované testování samotného programového a aplikačního kódu. Vyžaduje znalosti v programování a struktury dané testovaného kódu, ve kterém je aplikace implementována. Toto testování je náročnější na zkušenosti a znalosti testera, a proto bývá dražší. Výhodou tohoto testování je odhalení chyb na úrovní samotného kódu a opravení tohoto konkrétního nežádoucího kódu dochází ke zkvalitnění celé aplikace. Nevýhodou tohoto druhu testování, že tester nevidí aplikaci z pohledu, kterým ji vidí běžný uživatel.

Přímým opakem je testování černé skříňky (black box testing, které se také obecně nazývá funkční testování), při které tester nemá k dispozici přístup do zdrojového kódu aplikace, ale pouze základní nebo žádnou dokumentaci. Testování je možné za pomoci předpřipravených testovacích scénářů (test suit), které popisují postup, jakým má být test následně testerem proveden. Testovací scénáře bývají unifikované do logických celků dle aktuálně testované konkrétní funkce. Delší testovací scénáře bývají rozdělené do více částí (test case). Velkou výhodou tohoto druhu testování je, že test je prováděn z pozice běžného uživatele. Mezi další výhody patří rychlost a jednoduchost. Zároveň nejsou kladeny tak vysoké požadavky na zkušenosti testera jako v případě testování bílé skříňky.

Ve středu mezi testováním bílé a černé skříňky leží testování skříňky šedé. U tohoto druhu testování jsou předpokládány znalosti o implementaci a základní struktuře testované aplikace. Zároveň má tester k dispozici kód nebo jeho část, ale v menší míře než v případě testování bílé skříňky. Mezi dobré příklady tohoto testování je webová stránka, u které máme k dispozici hotový HTML kód a výsledně zobrazenou stránku.
[5 stránky 169 - 177]

3.1.2 Automatické / Manuální testování

Rozdělení testování na automatické nebo manuální je určeno podle toho, kdo jej vykonává –software nebo člověk. Pakliže musí být výsledek vyhodnocen člověkem, kvůli lidskému úsudku nebo zda je daná část testu nezautomatizovatelná je vhodné zvolit manuální testování. Naopak v případě mnohačetného testování nebo testování velkého objemu dat je vhodnější využití automatického testování. Výběr konkrétního testovacího nástroje je odvislí od druhu testů. [7]

3.1.3 Statické a dynamické testování

V případě testování funkční aplikací, která musí být v průběhu testu spuštěná anebo naopak spuštěná být nemusí se rozlišuje testování na dynamické a statické. Testy statické nevyžadují mít aplikací spuštěnou v průběhu testu. Proto lze statické testy uplatnit i na ne zcela spustitelné verzi aplikace. Statické testy jsou vykonávány nad částmi kódy a výstupy jsou vyhodnocovány na základě vstupů a výstupů po provedeném testu. Výsledek je ověřován dle platné dokumentace. Nicméně dynamické testy vyžadují již spustitelnou verzi aplikace, které jsou pak ověřovány v podobě zadávání neplatných a platných hodnot. Tento druh testování je využíván v pozdějším stadiu vývoje softwaru. [7] [8]

3.1.4 Bezpečnostní testování

Bezpečnostní testování je procesem ověřování schopnosti softwaru odolat vůči různým vlivům o jeho zneužití. Tento druh testování je nejnáročnější, jelikož je nutné odhalit slabá místa v softwaru, které má každá aplikace jiné a specifické. Pro webové aplikace platí, že bývají náchylné proti následujícím typům útoků:

- XSS[2] - Metoda vložení škodlivého JavaScript kódu a narušení stránky. Vstupy nejsou dostatečně zabezpečené.
- SQL injection – V databázové vrstvě se jedná o napadení škodlivým SQL skriptem, který přes nedostatečně ošetřený vstup vykoná útočníkův SQL dotaz.

[2] Cross-site scripting

- CSRF[3] - Potvrzení formuláře. V případě, že útočník zná nebo dokáže předpokládat část URL adresy nebo posílané hodnoty parametrů proměnných, změní jejich hodnoty, zobrazované objekty nebo obsah stránky.
- Phishing útoky, například záměna přihlašování přihlašovací stránky aplikace
- Odchytávání nezašifrované datové komunikace.
- Sdělení informací, které útočník dokáže využít pro následný útok, například z chybně definované hlášky chybového stavu aplikace.

[9]

3.2 Fáze testování

Samotný testování proces obsahuje ve svém cyklu několik jednotlivých fází. S testovacími fázemi jsou spjaty dva velmi důležité pojmy, kterými jsou výstupní a vstupní fázová kritéria. [9]

Bez jasně a explicitně stanovených kritérií pro vstup a výstup z jednotlivých fází by se práce na testování rozpustily do beztvaré a nikým a ničím neřízené činnosti. [5 str. 265]

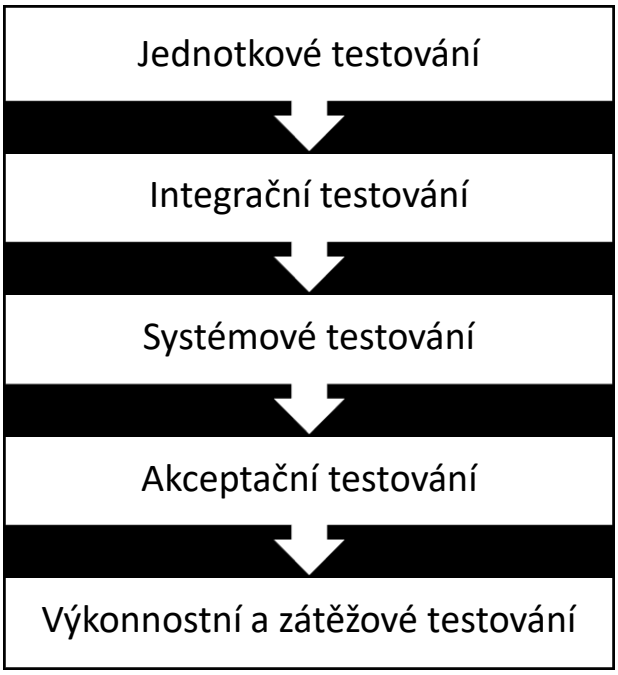


Schéma 1 - Znázornění schématu posloupnosti testovacího procesu
Zdroj: vlastní tvorba

[3] Cross site request forgery

3.2.1 Jednotkové „unit“ testování (vývojářské testování)

Během jednotkového testování jsou předmětem testování tzn. units, které představují dílčí části kódu. Toto testování je používané samotnými vývojáři již při vývoji softwaru. Při dalším vývoji aplikace jsou pak následně použity již otestované a funkční části kódu. Při testování je izolována testovaná část kódu od zbývající části a tyto části se kontrolují v předpřipravených situacích, které by mohly po spuštění aplikace nastat. Jednotlivé příklady testů jsou rozpracovány v kapitolách 6.1.1 a 6.2.1. [7] [10]

3.2.2 Integrační testování

Úkolem integračního testování je verifikace a kontrola propojení, funkcionality, komunikačních průchodů a přenos informací do okolí. Integrační testování bývá nazýváno „vnitřní testování integrace“. Dalším typem je „vnější testování integrace“, které má za úkol ověřit správnou průchodnost dat a chování aplikace v návaznosti na použité prostředí, rozhraní, konfiguraci hardwaru anebo operačním systému. Jako příklad lze uvést rozhraní mezi dvěma programy jež využívají rozhraní webových služeb pro účel komunikace anebo implementace webové aplikace do systému SAP. [7] [9]

3.2.3 Systémové testování

V pořadí další fází je systémové testování, které má za účel obecný test celé aplikace. Postupuje se standartně dle testovacích scénářů, ve kterých jsou popsány jednotlivé situace chování uživatele během práce s aplikací. Po nalezení chyb jsou následně odstraněny a poté je test opakován. [9]

3.2.4 Akceptační testování

Akceptační testování slouží jako výstupní kontrola webové aplikace pro překontrolování všech naspecifikovaných požadavků. Slouží jako kontrola pro zadavatele požadavků, který po následném překontrolování výsledku zhodnotí, zda může být aplikace akceptována. Kritickým měřítkem jsou akceptační kritéria, která musí být splněna. Například maximální odezva aplikace při určitém počtu uživatelů anebo rychlost startu aplikace v sekundách. [9]

3.2.5 Zátěžové a výkonnostní testování

V poslední vývojové fázi jsou výkonnostní a zátěžové testy, které ověří chování aplikace na jejích samotných hraničních možnostech a zároveň ověří celkovou stabilitu aplikace. Poskytnuté výsledky testu dají vývojářům představu o zatížení serveru v případě přístupu velkého množství uživatelů. Předpokladem je, že všichni uživatelé budou plně využívat testovanou aplikaci. Pro příklad lze uvést časový interval pro zpracování objednávka internetového obchodu. Ty většinou bývají dlouhé a při velkém zatížení může docházet k zobrazení chybových hlášek. Během výkonnostního testování je objektem sledování více parametrů, které jsou zaznamenány a porovnávány. Tímto způsobem lze odhalit chyby s pamětí po spuštění a v průběhu chodu aplikace. Jedna z možných příčin je využití příliš složitých algoritmů a neoptimalizovanými databázovými dotazy.

[9] [11]

4 Typy testovacích nástrojů

Ve funkčních testech webových aplikací s velkým objemem dat a nutností jednotlivé testy opakovat je velmi příhodné test automatizovat za pomocí nějakého z dostupných nástrojů pro automatizaci. Existuje velké množství nástrojů, které vybíráme podle zvoleného druhu testu – nástroje pro zátěžové, bezpečnostní nebo funkční testování).

Efektivita těchto nástrojů je velice veliká a každým rokem se zvyšuje a užití nástrojů je stále více uživatelsky přívětivější. Dříve byla znalost programovacího jazyka naprostou nutností pro psaní testovacích skriptů jako zdrojového kódu. Aktuální nástroje se snaží částečně tyto podmínky eliminovat a nabízí zjednodušené možnosti nahrávání a tvorby zjednodušených skriptů, případně jednoduchá grafická rozhraní včetně podrobného zobrazení výsledků testů.

Bohužel nelze zautomatizovat úplně vše, jelikož to není možné. Některé testy vyžadují lidský úsudek anebo zásah člověka, a proto musí být testovány manuálně. Je velice pravděpodobné, že se v blízké budoucnosti tato skutečnost nezmění. Tato práce je zaměřena na funkční testování z pohledu běžného uživatele. K tomu je nutné nezbytné vývojové prostředí a webový prohlížeč.

Příznivé vlastnosti testovacích nástrojů:

- **Rychlost** – nástroje na testování jsou v porovnání s člověkem vícenásobně rychlejší a lze je spouště i paralelně.
- **Efektivita –** v průběhu testování může tester připravovat ji další testy či vykonávat jiné činnosti, jelikož spuštěný test není na testerovi závislý.
- **Přesnost a správnost**– automatické testy jsou velmi přesné v případě, že jsou dobře naprogramované. Na rozdíl od člověka nedělají chyby a pracují vždy se stejnou přesností.
- **Neúnavnost** – automatické skripty na testování mohou být spuštěny po neomezenou dobu.

[9]

Obecným pravidlem pro automatické testování je, že nelze vše automatizovat, a proto je nutná úvaha testera nebo tvůrce automatizace nad tím, kterou funkci je možné zautomatizovat, a pro kterou je příhodnější využití manuálního testování. [9] Stejně to ve svém citátu uvádí Elbert Hubbard:

One machine can do the work of fifty ordinary men. No machine can do the work of one extraordinary man.

V překladu:

Jeden stroj dokáže dělat práci padesáti normálních lidí. Žádný stroj nedokáže dělat práci za někoho mimořádného.[6]

Pro praktickou část této bakalářské práce byly vybrány nástroje z komerční oblasti a open source tak, aby pak následně mohli být mezi sebou porovnány. Z open source[4] byly vybrány nástroje z rodiny Selenia, které jsou jednoznačně nejrozšířenější a nejlepší pro testování webových aplikací. Na druhé straně byl vybrán nástroj od společnosti IBM, Rational Functional Tester.

4.1 Nekomerční (Open Source) a komerční nástroje

V následujících podkapitolách jsou vypsány nástroje, u kterých jsou porovnány výhody a nevýhody. Vybrány byly z řad Open Source softwaru nástroje Selenium a z komerčních nástrojů IBM Rational Functional Tester. V úvahu je brána i uživatelská podpora, například při řešení problémů, a dostupnost návodů a specifikace. Selenium je hojně rozšířen, díky své obrovské popularitě, a proto jsou návody, rady a triky dostupné na řadě diskuzních fórech. Na druhé straně v případě nástroje od IBM je tomu naopak. Nicméně je dodáván s velmi podrobnou dokumentací, která bohužel neobsahuje konkrétní příklady a řešení.

[4] Software s otevřeným zdrojovým kódem, který lze po dodržení určitých podmínek prohlížet, upravovat a využívat.

4.1.1 Nástroje Selenium

Jedná se o celou rodinu nástrojů pro webové testování, které využívají stejné jádro – Selenium. Tyto nástroje jsou open source a jsou vyvíjeny již od roku 2004. Jsou běžně dostupné a relativně uživatelsky přívětivé, proto jsou dosti hojně využívané.

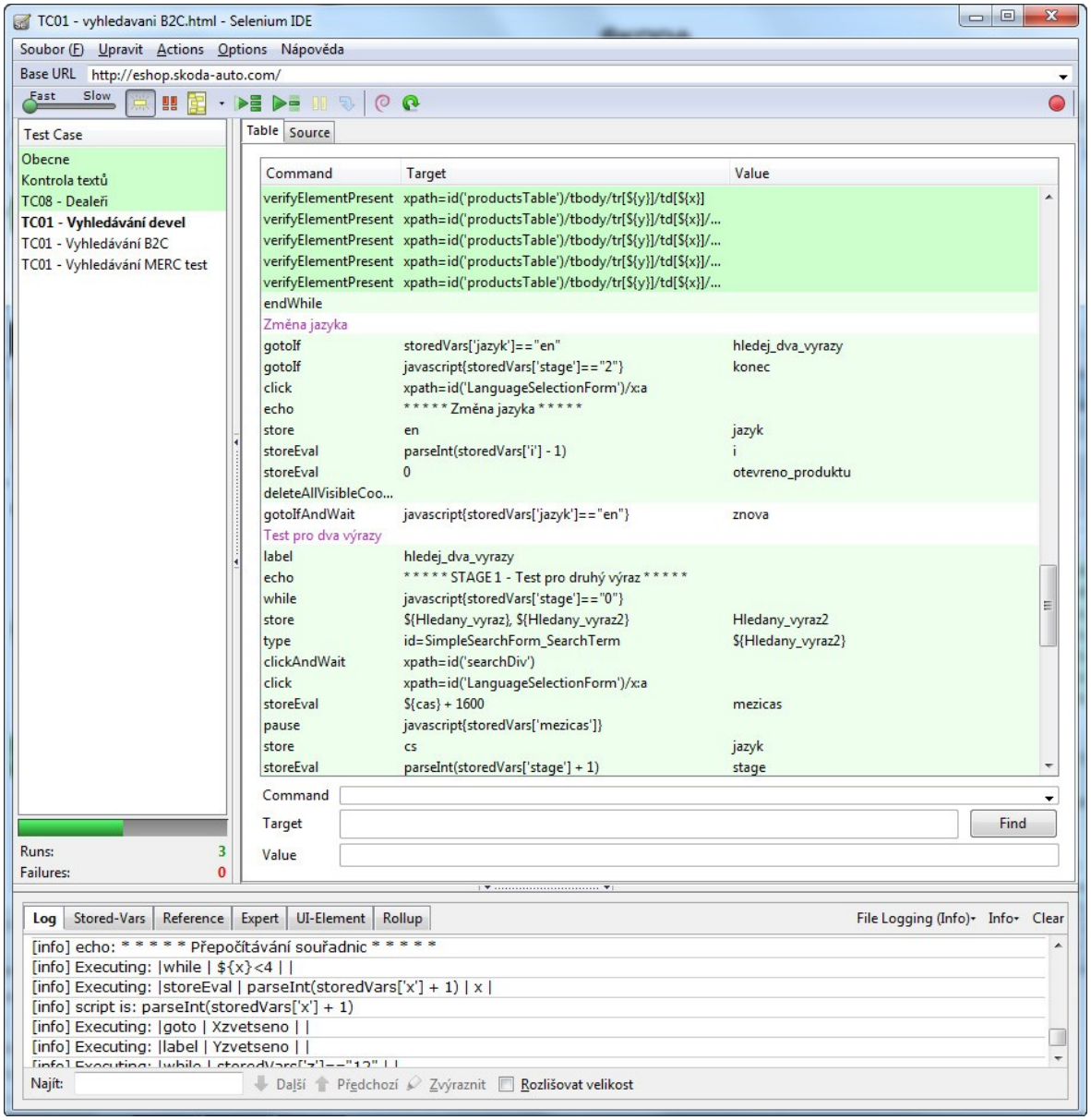
Princip a většina funkcí je společná pro všechny nástroje z rodiny Selenia. Selektory se odkazují na objekty za pomoci DOM[5], CSS[6] XPath[7] nebo ID[8] elementů na stránce. Nejprve je zjištěna, zda elementy na stránce existují a dále jsou s nimi vykonány nadefinované akce a hodnoty jsou ověřeny dle testovacích scénářů. Konkrétní způsob je závislý na konkrétně zvoleném nástroji Selenia.

4.1.2 Selenium IDE

Základním a nejvíce rozšířeným z nástrojů Selenia je Selenium IDE, které vyniká svojí jednoduchostí. Jedná o jednoduché rozšíření webového prohlížeče, např. Firefox nebo Google Chrome. Selenium IDE je velmi jednoduché na ovládání ale přesto dosahuje svými funkcemi úrovně základního vývojového prostředí (IDE) nezbytného pro testování.

Testovací skripty lze jednoduše vytvářet pouhým nahráváním akcí a pak následným klikáním na stránce dle definovaného testovacího scénáře. Hotové skripty se zapisují v jazyce Selenese, který je pro Selenium specifický.

[5] Document Object Model. Objektově orientovaná representace XML nebo HTML dokumentu, umožňující přístup či modifikaci obsahu, stylu nebo struktury.
[6] Cascading Style Sheets neboli kaskádové styly, kolekce metod pro grafickou úpravu webových stránek.
[7] XML Path Language, počítačový jazyk pomocí, kterého lze vybírat jednotlivé elementy stránky a pracovat s jejich hodnotami a atributy.
[8] Unikátní identifikátor tagu ve zdrojovém kódu stránky



Obrázek 2 - Selenium IDE

Zdroj: Snímek obrazovky Selenia IDE

4.1.3 Selenium WebDriver

Tento nástroj je nejdůležitějším nástrojem z rodiny Selenia. Je to nástupce Selenia RC, a proto mají oba nástroje mnoho společných rysů, například práce s prohlížečem. Prohlížeč po spuštění načte JavaScriptové funkce, které jsou pak využity k jeho ovládání. Nicméně tento způsob nefungoval vždy a docházelo k problémům během testování dynamicky měnících se stránek. Proto se o této staré metody upustila a WebDriver ovládá

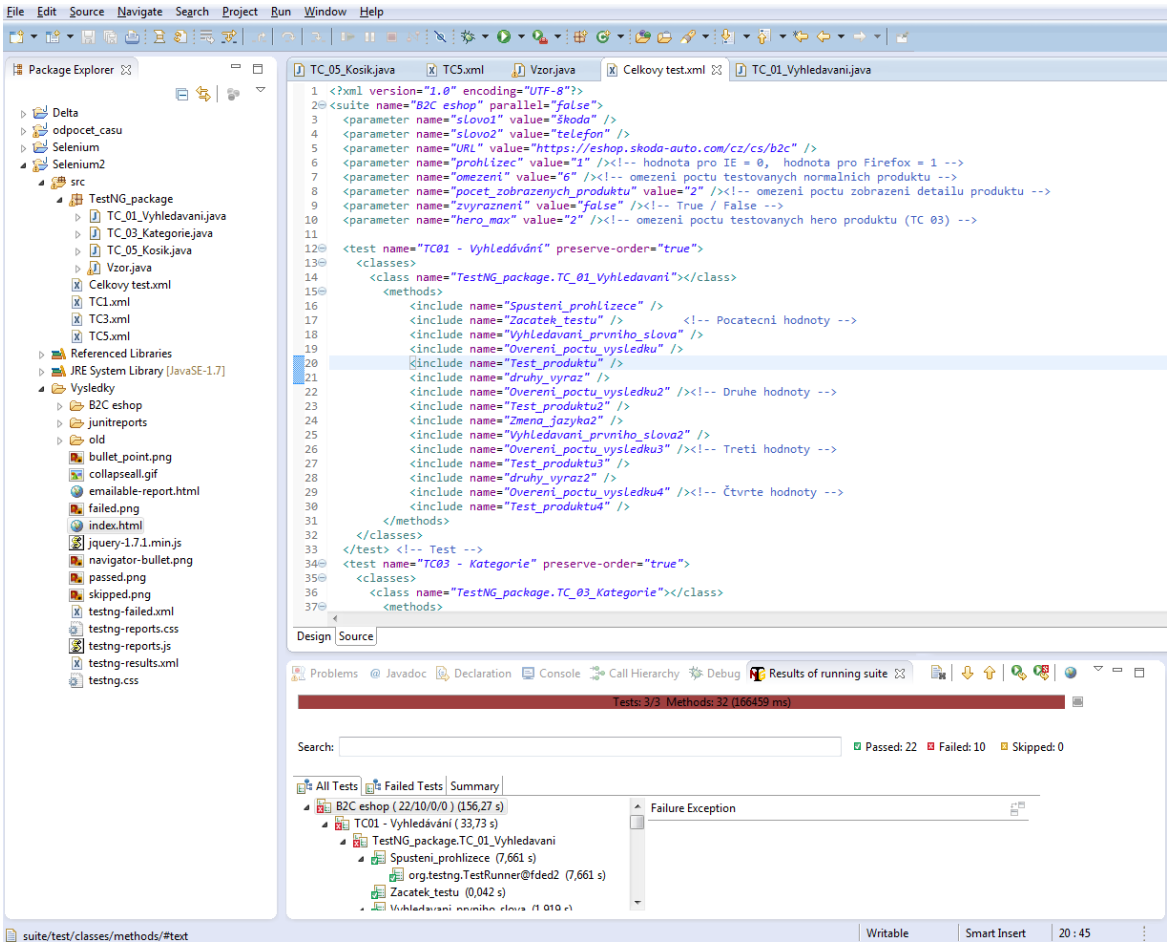
prohlížeč přímo pomocí API[9], které mají dnešní prohlížeče již implementované pro automatické ovládání právě pro případ automatického testování. Jednotlivá API se pro různé prohlížeče liší.

Pseudo-prohlížeč HtmlUnit je zajímavostí, kterou ocení testeři, kteří chtějí vyzkoušet test bez reálného prohlížeče. Test je následně spuštěn ve virtuálním prostředí. Bohužel se tato funkce se nechová jako v reálném prohlížeči, proto se objevují chyby v případě dynamicky měnících se stránek. [12]

Selenium WebDriver je tvořen mnoha Java archivy, které jsou v rámci vývojového prostředích použity pro vytvoření nového projektu, např. Eclipse. Testovací scénář je přespán do programovacího kódu, například v jazyce Java. Velkou výhodou je možnost využití jiných jazyků jako Python, Ruby anebo C#.

Oproti Seleniu IDE je podporováno spouštění více prohlížečů najednou. Nicméně je tu absence funkce nahrávání, a proto musí být testy psány manuálně. K vyhodnocení výsledků se dají použít frameworky TestNG, ReportNG, JUnit4.

[9] Application Programming Interface je označení rozhraní pro programování aplikací. Obsahuje sbírku procedur, funkcí, tříd a knihoven, které může programátor využít



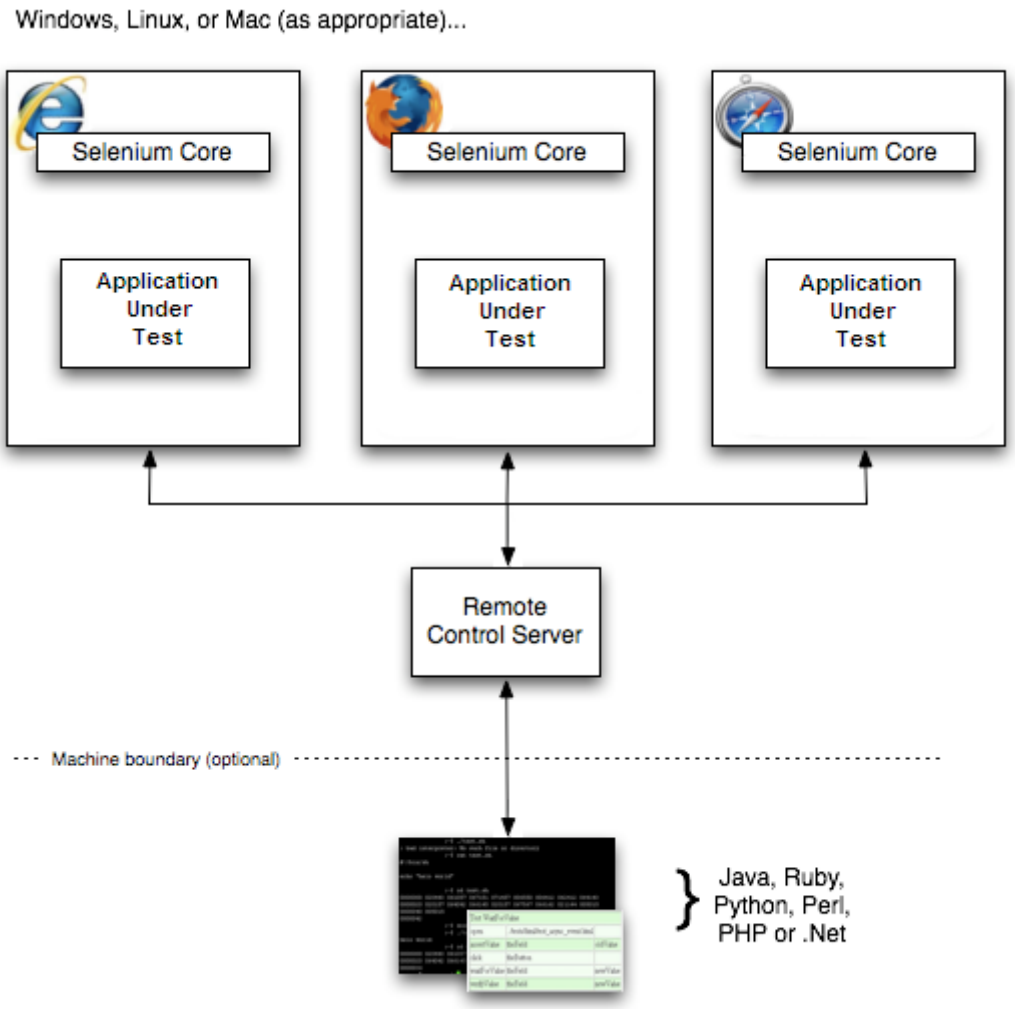
Obrázek 3 – Prostředí Eclipse se Selenium WebDriver

Zdroj: Vlastní tvorba – snímek vývojového prostředí Eclipse

4.1.4 Selenium Remote Control

Jedná se o původní Selenium, ze kterého se vyvinuly ostatní nástroje. Bývá také označován jako Selenium 1. Nástroj Selenium RC se složen ze dvou částí: serveru a klienta. Serverovou část řídí Java servlet, který hostí spuštěné relace testování. Ten může běžet na jakémkoliv JEE[10] aplikačním serveru. Na klientské části je pak standartně WebDriver. Servlet musí běžet na počítači s prohlížečem. Podpora ze strany vývojářů a uživatelů je stále velmi vysoká.

[10] Java Enterprise Edition



Obrázek 4 – Schéma diagramu architektury Selenia

Zdroj: https://www.seleniumhq.org/docs/05_selenium_rc.jsp#how-selenium-rc-works
Platnost odkazu byla naposledy ověřena v 11. 8. 2019.

Výše uvedený diagram znázorňuje komunikaci serveru a klienta. Nejprve jsou příkazy poslání ke zpracování na server Selenia RC, který předá informace jednotlivým běžícím prohlížečům pomocí JavaScriptů, API anebo Selenium Core, v závislosti na konkrétním prohlížeči. Naskriptované akce jsou vykonány a ověřeny, zda byly vykonány správně.

Selenium RC Server spouští a ukončuje okna prohlížeče, překládá a spouští Selenese příkazy, a především vystupuje jako http proxy server, který zachycuje a ověřuje data procházející mezi serverem a testovanou aplikací. Poté odešle výsledky zpět k dalšímu

zpracování. Hlavním cílem RC serveru je spuštění Selenium Core v prohlížeči. Selenium-Core je množina JavaScriptových funkcí, které jsou interpretovány a vykonávají Selenese příkazy. [12]

4.1.5 Selenium Grid

Selenium Grid je část z nástrojů Selenia nabízející možnost spustit více souběžně běžících testů ve více oknech webového prohlížeče. Je možné testování provádět na aktuálně běžícím počítači nebo více počítačích v různých prostředích. Tyto funkce jsou již součástí Selenium RC a Selenium Webdriveru. [12]

4.1.6 Cypress

Tento nástroj je poměrně nový a není založený na Seleniu. Snahou tohoto nástroje je učinit testování, co nejjednodušší a nejvíce komplexní. Snaží se využívat co nejpokročilejší metody, postupy pro testování a reportování výsledků. Například: Při každém testu je udělá screenshot nebo video z celého testu. Během testování lze skript ladit a případné problémy s automatickým skriptem snáze odstranit. Velkou výhodou je automatické čekání po akcích a krocích v testu. Hodnoty parametrů jsou ukládány a po celou doby testu pečlivě sledovány. Je sledována komunikace se serverem pro případné odladění. Tento nástroj je vhodný pro end-to-end testy, integrační testy anebo pro jednotkové testování. Bohužel velkou nevýhodou tohoto nástroje je jeho cena a malá podpora komunity na rozdíl od nástrojů z rodiny Selenia. [22]

4.1.7 IBM Rational Functional Tester

Z řady komerčních nástrojů byl vybrán IBM Rational Functional Tester, který umí tvořit objektově orientované testovací skripty. Mimo standartního funkčního testování je nástroj určen také pro testování Siebel, Flex, Java aplikací, .Net, AJAX a SAP aplikací, což je důvodem, proč byl vybrán tento nástroj. Stejně jako Selenium IDE obsahuje funkci nahrávání, ale má i navíc možnost ladění a vytváření kontrolního bodu (místo, kde se test zastaví a čeká na potvrzení testera).

Nástroj je k dispozici ve dvou distribucích. První je založená na vývojovém prostředí od Microsoftu Visual Studio .NET s podporovaným jazykem VisualBasic. Druhá distribuce je určena pro platformu Eclipse. V této práci byla využita distribuce na platformě Eclipse s jazykem Java.

Během procesu nahrávání tvoří IBM Functional Tester mapu použitých testovacích elementů spolu s použitými vlastnostmi a hodnotami atributů daných objektů. Ty jsou během testu sledovány, ale také je lze následně rychle znovu použít při manuální úpravě nahraného skriptu. Tuto mapu lze tvořit i manuálně a objekty jsou dostupné i po vypnutí funkce nahrávání. Poté je třeba je pouze změnit či aktualizovat. V případě změny vlastnosti se dané změny projeví ve všech testovacích skriptech, kde je element použitý.

V průběhu nahrávání jsou do testovacího skriptu vloženy kontrolní elementy, které zachycují informace o objektu (v závislosti na typu objektu). Ty jsou následně uloženy do souboru se základními údaji o objektu. Ty pak slouží k verifikaci hodnot testované aplikace oproti očekávaným hodnotám.

Velkou výhodou je nezávislost na prohlížeči a testovacím prostředí. Například můžete nahrát skript na Windows ve Google Chrome a následně ho spustit na systému Linux v prohlížeči Opera. Na konci testu je automaticky vytvořen report ve formátu HTML, který mimo informací o výsledků jednotlivých testů také obsahuje screenshoty.
Zdroj: [13]

5 Přípravy a spouštění testů

Se stále zvyšujícím se celkového počtu funkcí, webových aplikací a funkcionalit vznikají nové anebo se více rozšiřují současné testovací scénáře. Ve společnosti ŠKODA AUTO a.s. jsou testovací scénáře ve formě tabulek v Microsoft Excel. Aktuální testovací scénáře odpovídají současné verzi webové aplikace a jsou předlohou pro psaní automatických testovacích skriptů. V následujících podkapitolách bude rozpracován vlastní proces tvorby testu počínajíc tvorbou testovacích scénářů.

5.1 Testování ve ŠKODA AUTO a.s.

Vypracování této práce předcházelo velmi četné funkční manuální testování e-shopů ŠKODA AUTO a.s. na vývojových, produkčních a testovacích prostředích. Velká časová náročnost manuálního repetitivního funkčního testování byla podmětem pro realizaci této bakalářské práce. B2C a B2B Internetový obchod je vyvíjen a spravován v rámci společnosti ŠKODA AUTO a.s. vlastním k tomu určeném oddělením. Během vývoje jsou spouštěny pravidelně v cyklu tři druhy testů – zátěžové, funkční a akceptační.

- **Funkční testování** je prováděno již v průběhu vývoje aplikace vývojáři. V případě, že je část aplikace plně funkční je odzkoušena na vývojovém prostředí a vystavena intenzivnímu funkčnímu testování testery a vývojáři.
- **Zátěžové testování** je prováděno nástrojem IBM Rational Performance Tester.
- **Akceptační testování** je prováděno na konci vývoje aplikace. Zadavatelé ověřují správné chování aplikace vůči požadovanému stavu v dokumentaci.

V první fázi je zadáním požadavků na změnu nebo nový vývoj. Po vytvoření a akceptování příslušné dokumentace je dalším krokem samotný vývoj změny aplikace programátory. Poté následuje fáze testování, které je prováděno vývojáři. Nová funkce je nejdříve otestována na lokálním testovacím prostředí. Pokud je vše v pořádku tak následuje testování na vývojovém prostředí, na které je umístěna nová verze internetového obchodu. Poslední fází testování je celkový kompletní test. V případě, že je aplikace plně funkční je ještě otestována její komunikace a provázanost s ostatními systémy na testovacím prostředí. Zde je provedeno i akceptační testování. Pokud aplikace projde celým testovacím

procesem bez chyb je umístěna na produkční prostředí. V opačném případě se musí opravit chyby v aplikaci a poté je celý proces opět opakován.

5.2 Internetový obchod ŠKODA AUTO a.s.

B2C e-shop běží na platformě WebSphere Commerce od IBM. Skládá se ze tří částí. První částí je samotný internetový obchod. Druhým je nástroj pro jeho správu – IBM Management center a poslední částí je WebSphere Commerce Accelerator, který má za účel správu objednávek. Oba nástroje jsou dostupné pouze s patřičným administrátorským oprávněním. Na e-shopu se po příchodu zákazníka zobrazí úvodní stránka s výběrem kategorií produktů.Produkty může zákazník filtrovat nebo vyhledávat. U každé produktu je možné zobrazit si ho v detailu s více variantami produktu či dostupností na skladu. Zboží, které si uživatel přeje koupit jsou nejdříve vloženy do košíku. Posledním krokem je dokončení objednávky ve třech krocích, které musí zákazník projít a následně objednávku potvrdit. Všechny možné případy chování uživatele jsou pokryty testovacími scénáři na základě platné dokumentace požadavků e-shopu.

5.3 Životní cyklus testování

V následujících podkapitolách jsou rozepsány jednotlivé kroky testovacího cyklu spolu s praktickými příklady. Pro plán testování jsou využity testovací scénáře spolu se specifikacemi. Jednotlivé části životního cyklu jsou podrobněji zobrazeny na schématu 2 a vysvětleny pro oba nástroje separátně ve svých vlastních kapitolách.

Schéma 2 - Životního cyklus testování

Zdroj: vlastní tvorba

5.3.1 Plán testování

Oba první dva body z životního cyklu testovaní se váží ke testovacím scénářům. Počáteční fází je plán testování, který je vytvořen na základě plánu implementace a přiložené dokumentace. Proces plánování testů je obsahem následujících kapitol.

Dle softwarového testovacího standardu ANSI/IEEE číslo 829/1983 je plán testování definován takto:

„Plán testování je vysvětlen jako popis rozsahu, přístupu, prostředků a časového plánu testovacích činností. Identifikovat testované položky a funkce, které mají být testovány. Dále identifikace osob, které budou plnit jednotlivé testovací úkoly a všechna rizika spojená s testovacím plánem. Plán testu je pouze vedlejším produktem podrobného procesu plánování, který při jeho vytváření probíhá. Důležitý je samotný proces plánování, nikoliv výsledný dokument.“ [5]

K manuálnímu testování jsou využity testovací scénáře, které zároveň slouží i jako předloha pro tvorbu automatických skriptů. Jejich tvorba spočívá v logickém pochopení fungování aplikace na základě specifikace. Následně tvůrce testů sestaví jednotlivé standardizované body takovým způsobem, aby plně pokryly všechny možné varianty a stavy aplikace, které mohou, ale také nemohou nastat.

5.3.2 Specifikace testu

U testspecifikace není striktně daná forma, resp. je určena člověkem zodpovědným za její tvorbu. Obecně platí, že testspecifikace musí být jasně napsána, pochopitelná a bez chyb. Testy by měli být přizpůsobeny pro mnohačetná opakování a měli by být jasně definované způsoby verifikace. Důvodem pro psaní testspecifikace je řád, organizovanost a přehled nad jednotlivými testy. [7]

Před tvorbou nové testspecifikace je žádoucí rozdělit webovou aplikaci na jednotlivé kategorické funkce a vlastnosti. Poté tvorba testspecifikace probíhá pro každou separovanou funkci zvlášť. Obecně se dá testspecifikace rozdělit do tří částí:

- Návrh testovacího scénáře
- Případy testovacího užití – testcasy
- Metody testování

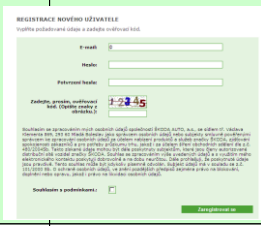
5.3.3 Popis testovacího scénáře

Testovací scénář obsahuje popis stav aplikace před testem, jednotlivé kroky a akce, přehled očekávaných stavů, vstupní a výstupní hodnoty. Po projití všech kroků ve všech testech by si měl být tester jistý, že je daná funkce správně otestována.

Projekt	E-shop B2C	**Tester**	
Subsystém		**Datum provedení testu**	
Test case	TC02 - Přihlášení	**Testovaná verze**	
Vypracováno		**Poznámka**	

Statistika						
Celkový počet kroků / verifikací		**Otestováno**	**Prošlo**	**Nekompletní**	**Neprošlo**	**Počet chyb**
48	kroků	Netestováno 0	0	0	0	

Cíl testu
Ověření funkce přihlášení do Eshop, změna hesla, zapomenuté heslo a registrace

Scénář				**Výsledek**				
Číslo kroku	**K/V**	**Popis**	**Vstupní data**	**Prošlo**	**Nekompletní**	**Neprošlo**	**Číslo chyby**	**Komentář**
1	V	Spusťte hlavní stránku E-shopu B2C						
		Registrace						
2	K	klikněte na menu Registrace						
3	V	otevře se formulář ŠA pro registraci v B2C, viz obr?						
4	K	Vyplňte tyto údaje: e-mail, heslo, ověření hesla, opsání kódu, odsouhlasení podmínek => Klikněte na "zaregistrovat se"	Informace o založení uživatelského účtu a odeslání e-mailu					
5	V	Přišlo potvrzení na zadaný mail?						
6	K	Klikněte na odkaz a dokončete registraci						

Obrázek 5 - Ukázka testovacího scénáře

Zdroj: Vlastní tvorba – snímek testovacího scénáře

V hlavičce testovacího scénáře je název scénáře, jméno testera, datum provedení testu, číslo testované verze a případné poznámky. V sloupečku *popis* jsou pod sebe zapsány testovací kroky a akce. Sloupec pro *vstupní data* slouží specifikaci vstupů před zahájením testu. Vedlejších tři sloupce slouží testerovi pro hodnocení funkčnosti aplikace v právě testovaném kroku. Sloupec *číslo chyby* slouží k verifikaci či zapsání nové chyby. V posledním sloupci je zapsán případný komentář. Test je opakován pro ostatní prohlížeče a jazyky. Provedené výsledky jsou sečteny pod hlavičkou v tabulce statistik. Report je následně zaslán k verifikaci chyb a následně jsou chyby předány vývojářům k opravení.

6 Popis a užití nástrojů

V následujících kapitolách jsou popsány postupy při vytváření a konfigurace testů a následně jsou testy spuštěny. Výsledky testů jsou podrobně rozebírány v kapitole 7.

6.1 Tvorba sady WebDriver testů

Internetový obchod B2C podporuje především Microsoft Edge, ale jsou také plně podporovány i jiné prohlížeče jako například: Chrome, Firefox a další. V této práci jsou popsány postupy tvorby skriptů, které lze analogicky přizpůsobit pro všechny dostupné prohlížeče. Testy pro tuto práci jsou vytvořené v platformě Eclipse spolu s reportovacím frameworkem TestNG, pro tvorbu výsledků testů ve formátu HTML. Pro vyhotovení testu byla použita verze WebDriveru 3.14.0, která byla aktuálně poslední verzí v době psaní této práce.

6.1.1 Příprava testů

V prvním kroku je vytvoření nového projektu v Eclipse. Pro tuto práci byl vybrán programovací jazyk Java, ale je možné si zvolit i jiný programovací jazyk například Python C#, Perl, C++ nebo Ruby. Je také možné využít jiné vývojové prostředí kromě Eclipse, pokud je podporován ze strany Selenia.

Dalším krokem je přidání nezbytných nástrojových knihoven:

- *selenium-java-3.14.0-srcs.jar, selenium-java-3.14.0.jar, selenium-3.14.0\libs* – knihovny WebDriveru,
- *testng.jar* – knihovny TestNG, lze stáhnout i přes Eclipse marketplace.

Ve vývojovém prostředí Eclipse je automatická kontrola chybějících knihoven. V případě, že některá knihovna chybí nebo je poškozená je zobrazeno uživateli automatické řešení v podobě nového stažení potřebné knihovny, která je následně naimportována mezi ostatní třídy. Pro framework TestNG jsou specifické anotace – *BeforeSuite*, *AfterSuite* a *Test*.

```
1  @BeforeSuite
2  public void Spusteni_prohlizece (ITestContext parametr) throws Exception{}
```

```
3  @AfterSuite
4  public void Ukonceni_prohlizece (ITestContext parametr)
   throws Exception {}
```

Metody se specifickými anotacemi BeforeSuite a AfterSuite se vykonávají před nebo po spuštěním testu. ITestContext je předávaným parametrem, které je definován konfiguračním XML souboru. To vše má za následek, že nejdříve se spustí okno prohlížeč, a až poté je vykonáván test. Na konci je okno prohlížeče ukončeno. Základem testovací sady v nástroji Selenium WebDriver je spuštění prohlížeče. Pro prohlížeč Google Chrome je to samé proveditelné níže uvedeným zdrojovým kódem:

```
1  driver = new ChromeDriver();
2  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
3  driver.get(baseURL);
```

Na počátku spuštění testu je vytvořen driver, kterým je ovládáno API prohlížeče s přednastavenou výchozí dobou čekání na zobrazení načtení objektů na stránce. Bez této časové hodnoty bude test vyhodnocen jako chybný, jelikož driver nebude čekat a započne testování na nenačtené stránce. Po načtení stránky započne samotný test.

Jednotlivé části testů jsou označené pomocí anotací *Test* a jsou ohodnoceny na základě jejich správného vykonání. Součástí testů je i konfigurační soubor se zapsanými metodami ve formátu XML.

Ukázka testovací metody:

```
1  @Test(description = "Zacatek_testu")
2  public void Zacatek_testu(){
3  AssertJUnit.assertEquals(driver.getTitle(), "Vítáme Vás v ŠKODA E-shopu");
4  }
```

Touto testovací metodou je ověřen, název úvodní stránky „*Vítáme Vás v ŠKODA e-shopu*“. V případě shody názvu je test vyhodnocen pozitivně. V opačném případě negativně.

Lokalizace prvků:

Identifikace prvků na stránce probíhá za pomocí selektorů na jednotlivé elementy. Selektorem může být například konkrétní *id* a *name* elementů. Výhodou těchto identifikátorů je jejich vzájemná nezávislost na umístění na stránce. V případě obecně definovaného elementu je možné odkázat na konkrétní tag za pomocí relativní adresy CSS, Xpath nebo DOM. Pro Selenium je doporučené využívat CSS lokátorů vhledem k rychlosti a přehlednosti. [14]

6.1.2 Konfigurace testů

Před spuštěním testů je nutné nadefinovat jednotlivých testů a spustitelných metod a k nim mít připraveny všechny potřebné parametry. Testy jsou rozčleněny do jednotlivých vykonávaných metod. Další výhodou parametrizace je ztráta nutnosti měnit velké množství zdrojového kódu při malé změně v testu, ale stačí modifikovat pouze jeden výskyt parametru v konfiguračním souboru. Testy, resp. metody, jsou zapsány v XML souboru v přesně daném, následně spouštěném, pořadí.

Příklad parametrů v XML souboru:

```
1   <parameter name="baseURL" value="https://eshop.skoda-auto.com/cz/cs/b2c" />
```

Zde je využita metoda parametrizace pomocí ITestContext. Tento parametr je načten na začátku testu z externího XML souboru a v závislosti na názvu parametru vrátí odkazovanou hodnotu.

Příklad načtení adresy z konfiguračního souboru:

```
1 @Test(description = "Spusteni prohlizece")
2 public void Spusteni_prohlizece(ITestContext paramater) throws Exception {
3       String URL = paramater.getCurrentXmlTest().getParameter("baseURL ");
4 }
```

Příklad konfiguračního souboru:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <suite name="B2C eshop" parallel="false">
```

```
3     <parameter name="URL" value="https://eshop.skoda-auto.com/cz/cs/b2c" />
4     <parameter name="prohlizec" value="0" /> <!-- IE = 0,  Firefox = 1 -->
5     <parameter name="zvyrazneni" value="true" /> <!-- True / False -->
6     <test name="TC05 - Kosik" preserve-order="true">
7       <classes>
8         <class name="TestNG_package.TC_05_Kosik"></class>
9         <methods>
10              <include name="Spusteni_prohlizece" />
11              <include name="Kontrola_hover_kosiku" />
12              <include name="Kontrola_zmena_barvy" />
13              <include name="Vlozeni_produktu_do_kosiku" />
14              <include name="Kontrola_obrazku" />
15              <include name="Kontrola_odkazu_do_kosiku" />
16              <include name="Kontrola_produktu_v_kosiku" />
17              <include name="Zmena_hodnot" />
18        </methods>
19      </classes>
20    </test> <!-- Test -->
21  </suite> <!-- Suite -->
```

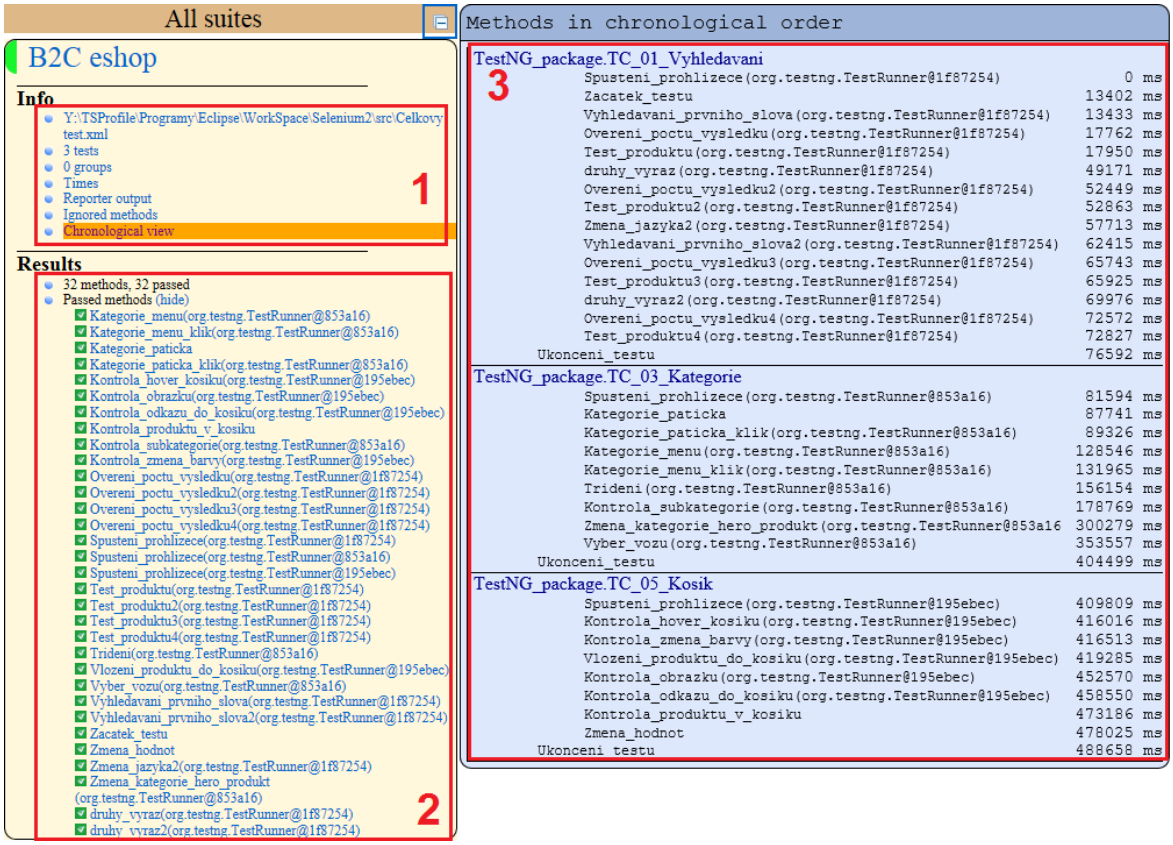
V hlavičce XML souboru je zapsána použitá verze xml syntaxe a typ šifrování. Počátek konfigurace je označen tagem *suite*. Testy mohou být spouštěny paralelně (ve více oknech najednou) nebo sériově chronologicky za sebou.

V této práci jsou spouštěny testy chronologicky za sebou v závislosti na pořadí, ve kterém jsou zapsány. Případné parametry jsou následně předány patřičným testovacím metodám. Test je určen tagem *test*, který je označen názvem (parameter *name)*. Dále následuje definovaná třída, ve které jsou umístěny testovací metody.

6.1.3 Spuštění testů

Testy jsou spouštěny za pomocí kliknutí pravým tlačítkem v prostředí Eclipse na konfigurační XML soubor a zvolení možnosti Run As » TestNG Suite.

Závěrem je výstup v podobě HTML reportu. Metody vyhodnoceny jako neúspěšné jsou zabarveny červeně spolu s vypsáním selhané metody. Součástí reportu jsou i časy trvání testů seřazené vybraného kritéria, například dle časů trvá testů.



Obrázek 6 - Ukázka reportu generovaného zapomocí WebDriver spolu s TestNG

Zdroj: Snímek ukázky reportu vygenerovaného pomocí TestNG

1 Menu filtrování zobrazení. Zobrazen je seznam test suitů, testovacích metod, chybových hlášek a časů.

2 Pole s výpisem testovaných metod. Červeně označené proběhly neúspěšně, zelené označené úspěšně, metody označené jako *ignored* byly z nějakého důvodu přeskočeny.

3 Hlavní obsah reportu v závislosti na zvoleném filtrování.

6.2 Tvorba sady Rational Fuctional testů

Nástroj IBM Rational Functional Tester dokáže testovat ve více prohlížečích – Microsoft Edge, Firefox, Netscape nebo Google Chrome. Pro tuto práci byl vybrán prohlížeč Firefox.

Nejdříve musí být testovací nástroj správně nastaven, než začneme vytvářet testovací skripty. Neprve je nutné nakonfigurovat Java prostředí dle přiložené dokumentace. Dále pak je možné využití vlastního JRE[11]. V případě, že vlastní JRE není plně funkční nebo nesprávně nakonfigurováno lze zvolit integrované, které je součástí nástroje. Stejně jako u WebDriveru je v první řadě nezbytný nejprve výběr prohlížeče. Zároveň je nezbytné předdefinovat spolu s počáteční stránkou prohlížeč, který se bude pro testování používat. Pro tuto práci byla použita poslední verze Rational Functional Testeru od společnosti IBM – verze 10.0.

6.2.1 Příprava testů

Logickým prvním krokem je založení nového projektu. Dále pak následuje samotné vytváření testovacího skriptu. Pro jeho tvorbu má tester na výběr ze dvou možností. Prvním je zjednodušené skriptování Simplified scripting, které je zjednodušeným kódem ve zvoleného programovacího jazyka do jednoduchých příkazů, které lze následně upravovat bez nutnosti zásahu do zdrojového kódu. Bohužel tímto způsobem, nelze dělat velmi podrobné testy, jelikož míra variability příkazů je omezená. Druhým způsobem je Java scripting. Obdobně jako v případě Selenia WebDriveru se jedná o regulérní kód v programovacím jazyce Java. Lze tímto způsobem tvořit i velmi složité algoritmy.

Pro tvorbu skriptu je možné využít funkce nahrávání jež je velkým usnadněním. Pro její použití je nutné otevřít z nástroje IBM okno prohlížeče a zapnout funkci nahrávání. Nástroj následně všechny akce zaznamenává.

[11] Java Runtime Enviroment

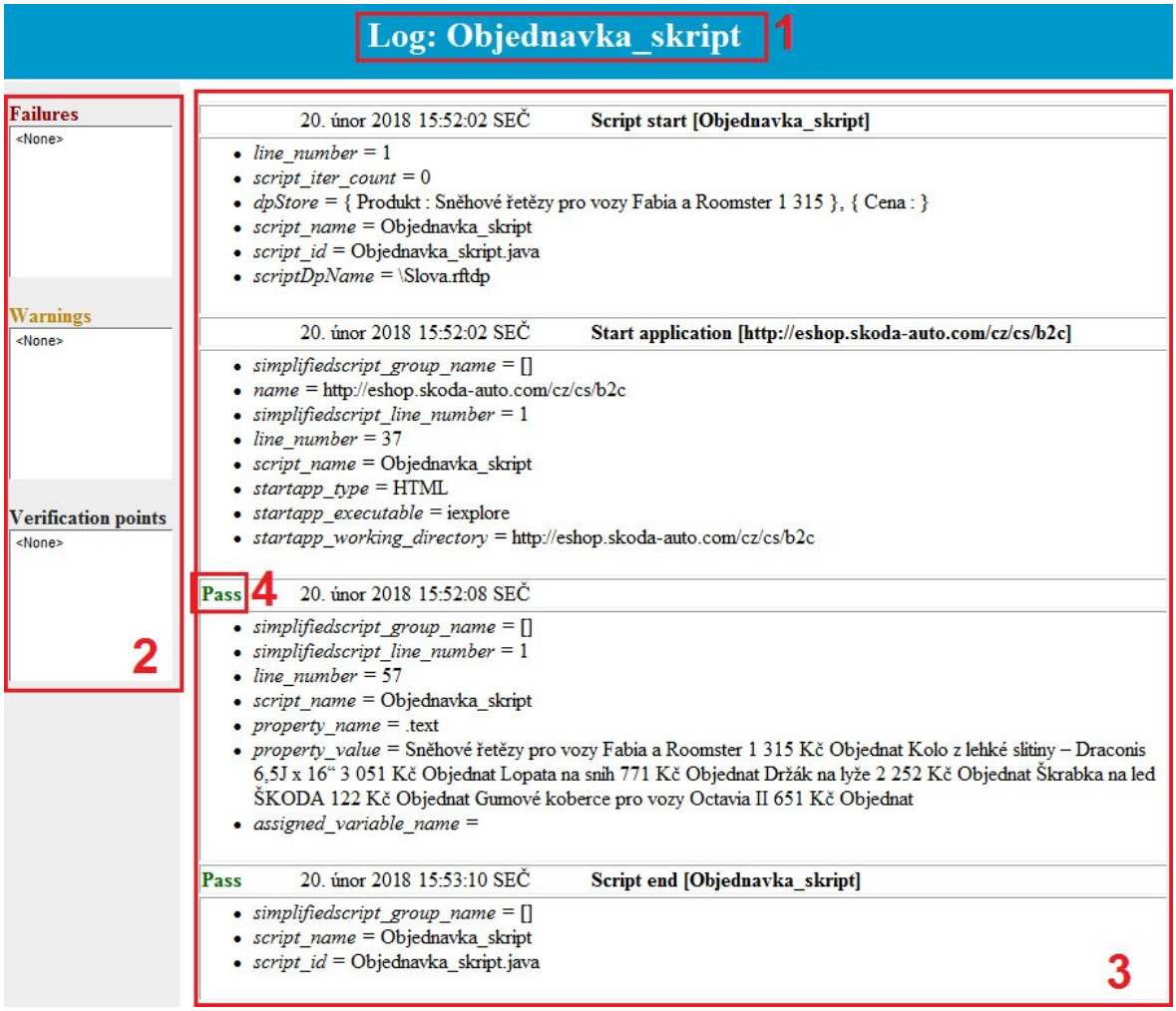
Funkce nahrávání

Kliknutím na ikonku *Start Application* otevřeme okno příslušného webového prohlížeče a dále kliknutím na tlačítko *Record* zahájíme nahrávání. Nejdůležitější je stanovení *Verification points*, které slouží pro ověření správného chování aplikace. Ať se jedná o samotnou existenci daného elementu nebo o porovnání zjištěné hodnoty vlastnosti nebo atributu proměnné s očekávanou hodnotou. Toto je stěžejní princip fungování nástroje. Celková tvorba testovacích skriptů je tím to způsobem velmi snadná a poměrně rychlá. Stanovení verifikačních bodů je docíleno jedním kliknutím a přetáhnutím ikonky nad testovací objekt a následní zvolení hodnot, které mají být testovány.

6.2.2 Spuštění testů

Po zhotovení testovacích skriptů následuje jejich spuštění. Po kliknutí na tlačítko *Run Functional Test Skript* se test spustí. Nejprve je nutné zadat název reportovaného souboru, který bude automaticky vygenerován po skončení testu. Není doporučené mít otevřené jakékoliv okno s prohlížečem v průběhu testování, jelikož je možná shoda názvu testovaných elementů s názvy elementů v potenciálně otevřeném okně prohlížeče. V takovém případě je pak zobrazena chybová hláška, ale test dále pokračuje.

Na závěr jsou vygenerovány výsledky ve formátu HTML. Výsledný report obsahuje všechny důležité aspekty – verifikační body, varování, neúspěšně provedené testy a chyby. U chyb je možné mít ve výsledku přiložený snímek obrazovky.



Obrázek 7 - Ukázka vygenerovaného reportu nástrojem Rational Functional Tester

Zdroj: Vlastní tvorba – snímek generovaného reportu z IBM Rational Functional Tester

Popis reportu nástroje IBM Rational Functional Tester:

1. Název reportu.
2. Základní přehledové menu, kde jsou zobrazené verifikační body, varování a chyby.
3. Obsah reportu spolu se základními informacemi. Chyby jsou zobrazeny spolu s chybovou hláškou a pravděpodobnou příčinou – chyba v konkrétním elementu objektu stránky. Součástí může být i screenshot obrazovky v momentě chyby.
4. Neúspěšné testy jsou označené zeleně s textem *Pass*, a neúspěšné červeně s textem *Fail*.

7 Vlastní testování a vyhodnocení testů

Jedním z cílů této práce je porovnání obou nástrojů. V následujících podkapitolách jsou porovnány oba nástroje a vypsány jejich kladné a záporné stránky do přehledných tabulek pro každý nástroj.

7.1 Porovnání nástrojů

Mezi hlavními stanovenými cíli této bakalářské práce je vybrání a porovnání vhodných nástrojů pro testování webových aplikací. Jednou z omezujících podmínek bylo vybrání nástroje s Open Source licencí a druhý s placenou komerční licencí. Tyto dva nástroje porovnat a zjistit jejich silné a slabé stránky, které budou přehledně zobrazeny. Údaje mají být získané zkušenostmi získané během práce s oběma automatickými nástroji.

Selenium WebDriver + framework TestNG

Framework TestNG a nástroj Selenium WebDriver je výbornou kombinací. Velkou výhodou Selenia WebDriver je možnost využití dostupných modifikací, kterých je poměrně mnoho. Například záměnou frameworku TestNG lze kompletně změnit formu reportování a výstupu. Silnou stránkou je také identifikace elementů pomocí relativních Xpath, DOM, CSS odkazů nebo pomocí konkrétních názvů a id. To má za výhodu nezávislost umístění elementů na stránce. Slabou stránkou je funkce nahrávání, proto musí být testy psány ručně. Díky tomu je čas potřebný na vytvoření automatického skriptu delší než v případě druhého porovnávaného nástroje.

Silné stránky	Slabé stránky
+ Využití DOM, CSS a XPath	- Java (podmínka programátorských zkušeností)
+ Spouštění testů na dálku	- Absence funkce nahrávání (pouze u Selenium IDE)
+ Velké podpora uivatelů	- Rychlost tvorby testu
+ Open Source	
+ Možnost užití vlastních modifikací	

IBM Rational Functional Tester

Nástroj IBM Rational Functional tester byl použit bez jakýchkoliv modifikací. Velkou výhodou tohoto nástroje je bezpochyby rychlá tvorba automatických testů za pomoci funkce nahrávání a možnost využití zjednodušeného skriptování. Tyto funkce ve výsledku ušetří opravdu mnoho testerova času při tvorbě automatických skriptů.

Silné stránky	**Slabé stránky**
+ Zjednodušené skripty	- Omezená podpora ze strany IBM
+ Nahrávací funkce	- Složité testování dynamických prvků
+ Rychlost tvorby testu	- Placený nástroj
+ Screenshoty v rámci chybových hlášení	
+ Uživatelsky jednodušší	

7.2 Porovnání ručního a automatického testování

Díky tomuto porovnání si můžeme reálně udělat představu o tom, jak moc byla práce testera zjednodušena za pomocí automatizovaného testování.

Zajímavým ukazatelem je celkové porovnání časů potřebných k přípravě automatických testů a délky jejich trvání od spuštění, oproti délce času potřebného na manuálního testování, viz. obrázek 8. Testy byly provedeny v prohlížečích Google Chrome a Firefox (FF):

Tabulka 1 – Srovnání manuálního a automatického testování testu TC 03 Kategorie

Název testu	Doba přípravy	Automatické testování		Manuální testování	
		FF	Chrome	FF	Chrome
TC Kategorie	2h 30 min	3 min 10 s	3 min 0 s	11 min 45 s	11 min 30 s

Zdroj: vlastní tvorba

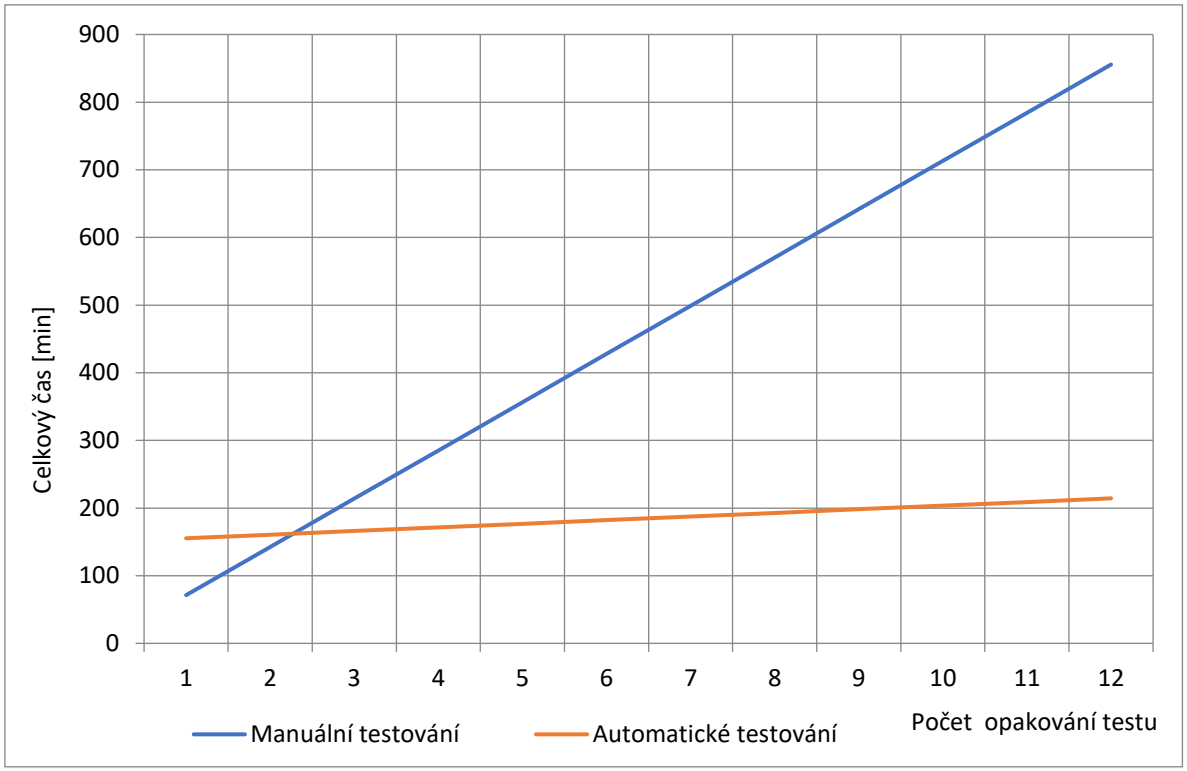
Celková průměrná doba na přípravu automatického testu je 2h 30m. Tento čas nezohledňuje čas potřebný na tvorbu testovacího scénáře, jelikož je v případě automatického i manuálního testování stejný, a proto pro toto porovnání nemá smysl. Čas testu je relativně krátký, ale liší se dle zvoleného prohlížeče, internetové odezvě, rychlosti a případně odezvě cílového serveru. Celkový čas je počítán od počátku testu až po samotný konec testovacího scénáře.
V případě manuálního testování je výhodou nulový čas potřebný k přípravě. Proto testování začíná hned od počátku. Tento druh testování bude vždy lepší v případě krátkých a spontánních testů, pro které nemá smysl vytvářet automatické testy. Pro dlouhé, velmi časté nebo testování velkého množství dat je vždy výhodnější automatické testování.

Z výše uvedené tabulky je možné spočítat časovou úsporu, která vznikla zavedením automatických testů.

$$Celkový\ čas = čas\ manuálního\ testu * 2\ jazyky * 3\ prohlížeče$$

Celkový čas jednoho **manuálního** testu pro prohlížeč Firefox:

$$Celkový\ čas = 11\ \text{min}\ 45\ \text{sec} * 6$$
$$Celkový\ čas = \mathbf{70\ \boldsymbol{min}\ 30\ sec}$$



Obrázek 8 – Srovnání celkových časů automatického a manuálního testování

Zdroj: vlastní tvorba

Výše uvedený graf na obrázku 8 znázorňuje grafický rozdíl mezi manuálním a automatickým testováním. U automatického testování je počítáno s fixním potřebným časem pro zahájení testování – potřebný čas je počítán pro čtyři opakování jednoho spuštěného automatického testu, a navíc doba potřebná na přípravu testovacího skriptu. U manuálního testu čas na přípravu je roven nule, a proto je vyměřený čas roven čtyřnásobku času pro manuální test. Křivka automatického testování je zvýšena hodnotu - 2,5 hodiny, která je potřebná na tvorbu automatického skriptu. Růst křivky manuálního testování je strmější než u křivky automatického testování, jelikož doba potřebná pro jeden test je o poznání větší.

V grafu je patrné, že již po několik opakování je automatický test výhodnější. Například po dvanácti a více testech je časový rozdíl kompletních testů webové aplikace velmi velký – 11 hodin 45 minut.

8 Shrnutí výsledků

V této kapitole je shrnutí celkových výsledků testů a spočtení eventuální možné časové a finanční úspory. Dále pak jsou porovnány jednotlivé časy testů v daných prohlížečích.

Průměrný plat testera je aktuálně 28800 za měsíc. Po zohlednění průměrného počtu pracovních dnů v měsíci - 20 dnů a pracovní doby – 8 hodin, se dá spočítat průměrná hodinová sazba testera – 180 Kč za hodinu. [15]

S předpokladem, že bude testování spouštěno třikrát týdně, tedy dvanáctkrát měsíčně, lze spočítat časovou úsporu – 35,25 hodin čtvrtletně a finanční úsporu – 6345 kč v případě testování jedné webové aplikace. Celkové pokrytí funkčního testování je zaštítěno více jak deseti testovacími scénáři. Celková roční úspora může tedy činit až 25 380 kč v případě kompletního testování jedné webové aplikace.

Porovnání časů automatických testů v závislosti na prohlížečích

V následující tabulce je zobrazen čas trvání vybraných testů v jednotlivých prohlížečích v nástroji Selenium WebDriver, nicméně nemusíme brát na zřetel hardwarovou náročnost testování, jelikož se jedná pouze o automatické testování a hardware je ve všech případech stejný:

Tabulka 2 – Celková doba trvání testů v prohlížečích

Název testu	Firefox	Chrome	Microsoft Edge
TC 01 Vyhledávání	40,37 sec	41,87 sec	45,65 sec
TC 03 Kategorie	234,48 sec	240,30 sec	241,11 sec
TC 05 Košík	73,90 sec	72,09 sec	75,75 sec

Zdroj: vlastní tvorba

Dle výsledků zobrazených v tabulce 2 je jasné, že nejrychlejším prohlížečem je Mozilla Firefox. Na druhém místě se umístil Google Chrome a posledním místě je Microsoft Edge.

Nástroje Selenium byly původně tvořeny pro Firefox (viz kapitola 4.1.2 Selenium IDE), proto byl dříve nástroj lépe optimalizovaný pro tento prohlížeč. Nicméně v průběhu vývoje nástroje se tento rozdíl snížil na téměř zanedbatelný, ačkoliv jsou v něm testy stále nejrychlejší.

Nástroj IBM Rational Functional Tester dokáže zjistit pozici elementu dle konkrétních souřadnic umístění na stránce. Tímto způsobem se velice těžko dají testovat dynamické prvky na stránce, jelikož jejich poloha není stálá. Nevýhodou je také absence relativního odkazování, například za pomocí XPath adresy.

9 Závěry a doporučení

Tato bakalářská práce měla za úkol popsat metody funkčního testování webových aplikací, použít vybrané nástroje testování pro ukázkovou webovou aplikaci a následně nástroje porovnat a vyhodnotit.

V teoretické části byly popsány obecné metodiky pro testování softwaru a fáze testovacího cyklu. Dále jsou popsány jednotlivé druhy testů. Každý druh testů má svůj účel, proto je nutné jej zvážit a zanalyzovat zvlášť, pro určení, zda je daný druh testu vhodný pro další krok v právě plánovaném testování. Zároveň je druh testu odvislý od zvolené testovací metody a nástroje.

K dalším cílům patřilo zautomatizování testování na konkrétní webové aplikaci. Pro tuto práci byl vybrán B2C e-shopu ve společnosti ŠKODA AUTO a.s.. Na základě již vyhotovených testovacích scénářů došlo k jejich aktualizaci a optimalizaci pro automatizované funkční testování. V další fázi byly vybrány vhodné testovací nástroje v řad open source a komerční licencí. Nejvhodnějšími kandidáty byl vybrán nástroj z rodiny Selenia, díky jeho širokému rozšíření a velké uživatelské podpoře, a nástroje od IBM – Rational Functional Tester. Následně byly v obou nástrojích vyhotoveny automatické testovací skripty pro funkční testování. V obou případech byly výsledky ukládány v podobě HTML reportů. Závěrem byly nástroje mezi sebou porovnány.

Během vývoje testovacích skriptů a testování bylo objeveno několik chyb ve webové aplikaci, které byly následně vývojáři opraveny. Jednalo se o spontánní chyby způsobené mnohačetným testováním, které by pravděpodobně nebyly za pomocí manuálního testování odhaleny.

Hlavní cíle této bakalářské práce byly splněny. Automatické testovací skripty vytvořené během psané této práce jsou stále používány. Nicméně je zcela nutná jejich pravidelná aktualizace, tak aby byly skripty dále použitelné. Zautomatizování testování na internetovém obchodu ŠKODA AUTO a.s. vede ke snížení nákladů a úspoře času testera. Již po druhém a každém dalším kompletním testu je automatické testování výhodnější. Manuální testování je vhodné použít na krátké spontánní testování nebo pro případné rychlé ověření chyb.

Výsledky porovnání neukazují na žádného jasného vítěze, jelikož oba nástroje splňují zadané požadavky. V případě Selenia je výhodou velké zastoupení podpory uživatelů na diskuzních fórech a pravidelně stále dochází k vývoji testovacího nástroje, ačkoliv jsou již nyní možnosti relativně rozsáhlé. V poslední řadě je nutné podotknout, že se jedná o open source nástroj, a proto se zdá být vhodnějším kandidátem pro testování na B2C e-shopu ve společnosti ŠKODA AUTO a.s.

I přes rozsáhlé možnosti automatizace, není příhodné a ani možné otestovat všechny objekty, atributy a jejich hodnoty, a proto bude vždy nutný občasný zásah testera.

Seznam použité literatury

Citace

[1] GELPERIN, D a B, HETZEL. *The Growth of Software Testing.* New York : Communications of the ACM, 1988. str. 687-695. ISBN 0001-0782.

[2] Wikipedie. Turingův test. [online] 2019 [vid. 2019-08-10] Přístup z Internetu: http://cs.wikipedia.org/wiki/Turing%C5%AFv_test.

[3] GOODENOUGH, J, B a S, L, GERHART. *Toward a Theory of Test Data Selection.* New York : IEEE Transactions on Software Engineering, 1975. str. 156-173. ISSN 0098-5589.

[4] BEIZER, B. *Software Testing Techniques.* New York : Van Nostrand Reinhold Company Limited, 1990. ISBN 0-442-20672-0.

[5] PATTON, R. *Testování softwaru.* Praha : Computer Press, 2002. ISBN 80-7226-636-5.

[6] LYDIA, A. *The web testing companion : The insider's guide to efficient and effective tests.* Indianapolis : Wiley Publishing, 2003. ISBN 0-471-08112-4.

[7] MAZOCH, B. Funkční testování webových aplikací. [online] 2019 [citace srpen, 11 2019] Přístup z Internetu: is.muni.cz/th/325233/fi_b/bmazoch-bp.pdf.

[8] Testovanisoftwaru.cz. Statické a dynamické testy. [online] 2019 [citace srpen, 10., 2019] Přístup z Internetu: http://testovanisoftwaru.cz/druhy-typy-a-kategorie-testu/staticke-a-dynamicke-testy/.

[9] PIETRIK, M. Automatizované testování webových aplikací. [online] 2019. [citace srpen, 10., 2019] Přístup z Internetu: is.muni.cz/th/172724/fi_m/_DP_Pietrik.pdf.

[10] Wikipedie. Jednotkové testování. [online] 2019 [citace spren, 15., 2019] Přístup z Internetu: http://cs.wikipedia.org/wiki/Unit_testing.

[11] Veluchamy, Thiyagarajan. Software testing Q and A. [online] 2019 [citace srpen, 14., 2019] Přístup z Internetu: http://thiyagarajan.wordpress.com/software-testing-questions-and-answers/.

[12] SeleniumHQ. Domovská stránka projektu. *Selenium Web Browser Automation.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: http://seleniumhq.org.

[13] IBM. Help system. *IBM Rational functional tester documentation.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: https://www.ibm.com/support/knowledgecenter/en/SSJMXE_9.2.0/com.ibm.rational.test.ft.doc/rft_welcome.html

[14] SeleniumHQ. *Selenium documentation*. [online] 2019 [citace spren, 15., 2019] Přístup z Internetu: http://docs.seleniumhq.org/docs/index.jsp.

[15] Srovnání platů. *Srovnání platů a zaměstnání v ČR* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: http://srovnani-platu.blogspot.cz/search/label/IT%20tester

[16] GUNDECHA, U. a MERYK, R. (2017). Selenium testing tools cookbook. Birmingham, UK: Packt Publishing, 2015. ISBN: 978-1-78439-251-2.

[17] AXELROD, A. Complete guide to test automation. Berkeley, California: Apress, 2018. ISBN: 1484238311.

[18] RASMSSON, J. The way of the web tester. Raleigh, North Carolina: The Pragmatic Bookshelf, 2016. ISBN: 1680501836

[19] Trendy v oblasti testování. *Novinky a trendy v oblasti testování.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: https://www.slideshare.net/ondram/trendy-a-novmonosti-test-automation

[20] Codeless Visual Testing. *Automated visual testing in Selenium IDE.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: https://speakerdeck.com/corevo/codeless-visual-testing-using-selenium-ide

[21] Businessit. *Současné trendy v zajištění kvality softwaru.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: http://www.businessit.cz/cz/soucasne-trendy-v-zajisteni-kvality-softwaru.php

[22] Cypress. *Cypress.io documentation.* [online] 2019 [citace srpen, 15., 2019] Přístup z Internetu: https://docs.cypress.io/guides/overview/why-cypress.html#Cypress-ecosystem

Bibliografie

GUNDECHA, U. a MERYK, R. (2017). Selenium testing tools cookbook. Birmingham, UK: Packt Publishing, 2015. ISBN: 978-1-78439-251-2.

AXELROD, A. Complete guide to test automation. Berkeley, California: Apress, 2018. ISBN: 1484238311.

RASMSSON, J. The way of the web tester. Raleigh, North Carolina: The Pragmatic Bookshelf, 2016. ISBN: 1680501836

BEIZER, B. *Software Testing Techniques.* New York : Van Nostrand Reinhold Company Limited, 1990. ISBN 0-442-20672-0.

GELPERIN, D a B, HETZEL. *The Growth of Software Testing.* New York : Communications of the ACM, 1988. ISBN 0001-0782.

LYDIA, A. *The web testing companion : The insider's guide to efficient and effective tests.* Indianapolis : Wiley Publishing, 2003. ISBN 0-471-08112-4.

PATTON, R. *Testování softwaru.* Praha : Computer Press, 2002. ISBN 80-7226-636-5.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Systémové inženýrství a informatika
Forma: Kombinovaná
Obor/komb.: Informační management (im3-k)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Nechanický Tomáš	Vítězná 462, Tanvald	I14029

TÉMA ČESKY:

Automatizované funkční testování webových aplikací

TÉMA ANGLICKY:

Automated functional testing of web applications

VEDOUCÍ PRÁCE:

Mgr. Daniela Ponce, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je popsat metody funkčního testování webových aplikací, použít vybrané nástroje testování pro ukázkovou webovou aplikaci a následně nástroje porovnat a vyhodnotit.

Struktura práce:
1. Úvod
2. Metodika zpracování
3. Obecné metodiky testování
4. Typy testovacích nástrojů
5. Přípravy a spouštění testů
6. Popis a užití nástrojů
7. Vlastní testování a vyhodnocení testů
8. Shrnutí výsledků
9. Závěry a doporučení

SEZNAM DOPORUČENÉ LITERATURY:

Gundecha, U. and Meryk, R. (2017). Selenium testing tools cookbook. Birmingham, UK: Packt Publishing, 2015. ISBN: 978-1-78439-251-2.
Axelrod, A. Complete guide to test automation. Berkeley, California: Apress, 2018. ISBN: 1484238311.
Rasmusson, J. The way of the web tester. Raleigh, North Carolina: The Pragmatic Bookshelf, 2016. ISBN: 1680501836.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....