VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Automatizace testování softwaru nástrojem Selenium

BAKALÁŘSKÁ PRÁCE

Student: Vojtěch Votlučka

Vedoucí: doc. Ing. Alena Buchalcevová, Ph. D.

Oponent: Mgr. Ladislav Kolář

2015

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne .....

Vojtěch Votlučka

Poděkování

Na tomto místě bych rád poděkoval vedoucí mé bakalářské práce, doc. Ing. Aleně Buchalcevové, Ph.D. za její vstřícnost při vedení práce a vždy konstruktivní připomínky ke správnému směrování práce. Také patří díky mým kolegům, kteří mi poskytli cenné vstupy a rady pro tvorbu práce. V neposlední řadě bych rád poděkoval svým rodičům, kteří mi byli oporou po celou dobu mého studia i zpracovávání této práce.

Abstrakt

Hlavním cílem bakalářské práce je popis automatizace testování softwaru pomocí nástroje Selenium WebDriver. Konkrétně se jedná o automatizaci funkčního testování webových aplikací v prostředí elektronického bankovnictví. Mimo popis postupu tvorby testů se jedná také o vytvoření reálně aplikovatelných praktických příkladů napsaných v programovacím jazyce Java.

Od teoretického představení problematiky testování softwaru se práce dostává k představení nástroje Selenium a poté jeho využití v oblasti elektronického bankovnictví. Následuje popis postupu automatizace, jejího spouštění i krátké srovnání kapacit z hlediska nákladů mezi manuálním a automatizovaným testováním.

Klíčová slova

Selenium WebDriver, software, elektronické bankovnictví, metodika, automatizace testování softwaru, webové aplikace, druhy testování, Java

Abstract

The main goal of this bachelor thesis is a complete description of the process of software testing automation using Selenium WebDriver. The work is focusing on functional testing of web applications in internet banking. Apart from the description of this process, the paper also creates applicable examples of test scripts written in Java.

Starting with theoretical introduction to the area of software testing, the work continues by introducing the Selenium WebDriver testing tool and its use in the environment of internet banking. Then follows the description of the process of automation, its execution and a brief cost comparison between manual and automated testing.

Keywords

Selenium WebDriver, software, internet banking, methodology, software testing automation, web applications, types of software testing, Java

Obsah

1. Úvod

V dnešní době je kvalita softwaru velmi důležitá a představuje často rozhodující faktor při volbě zákazníka. Software se díky rozmachu informačních technologií nalézá téměř všude, a to od jednoduchých aplikací jako jsou kalkulačky až po složitější software, jakým jsou například operační systémy. Proto, aby softwarové firmy nalákaly a udržely zákazníky, je kvalita softwaru velmi důležitá. K zajištění takové kvality ovšem nestačí jen dobrý nápad a naprogramování té či oné aplikace.

Obzvláště mezi velkými hráči či v odvětvích, kde je velká konkurence, je kvalita softwaru velmi důležitá. Je potřeba především k zisku konkurenční výhody. Zajištění kvality se dá nejlépe dosáhnout pomocí testování softwaru, tedy zkoušení jeho funkčnosti, aby se zamezilo tomu, že bude zákazník nespokojen. Toho se dá docílit pomocí testovacích scénářů, které provádí možné kroky uživatelů a kontrolují reakce softwaru v těchto situacích.

Testování softwaru ovšem již dávno není jen manuální „klikání“ testerů, ale i stále populárnější a mnohdy finančně výhodnější automatizované řešení těchto testů. Nástrojů k automatizaci testování je velká řada a i mezi nimi panuje velká konkurence. Konkrétní řešení záleží také na typu softwaru, jehož testování se firma rozhodla automatizovat.

1.1. Zaměření práce a důvod výběru tématu

Tato bakalářská práce se zabývá možnostmi testování softwaru, a to jak z pohledu manuálního testování, tak z pohledu automatizovaného. Mimo představení těchto možností se práce jmenovitě zaměřuje na automatizační nástroj k testování, Selenium WebDriver, jeho teoretické i praktické představení. Praktická ukázka je z oblasti elektronického bankovnictví. Jsou rozebrány také výhody a nevýhody Selenia oproti manuálnímu provádění testů.

Výběr tématu pramení především z mého zájmu o tuto oblast IT. Právě díky praktické zkušenosti s testováním softwaru pomocí nástroje Selenium WebDriver mohu práci směřovat i mimo teoretickou oblast. Promítám v ní znalosti získané na základě reálných užití a výzev v současném byznysu, konkrétně tedy v oblasti elektronického bankovnictví.

1.2. Cíl práce

Cílem bakalářské práce je popis automatizace testování softwaru pomocí nástroje Selenium WebDriver, vytvoření praktického příkladu a popis postupu vytváření testů.

Dílčími cíli je představení nástroje Selenium, shrnutí možností testování a možností jeho aplikace v elektronickém bankovnictví.

1.3. Metoda dosažení cíle

K dosažení cíle jsou použity rešerše podkladů z odborné literatury zaměřené na tématiku testování softwaru a nástroj Selenium. Mimo odborné literatury jsou analyzované i již zpracované vysokoškolské práce s podobným zaměřením.

Součástí metody je především analýza webových zdrojů, které jsou znatelně aktuálnější. Analyzované zdroje jsou jak z českého prostředí, tak ze zahraničních serverů. Jedním z hlavních zdrojů je oficiální dokumentace zabývající se nástrojem Selenium.

Pro dosažení cíle je použita také syntéza dostupných zdrojů zabývajících se tématikou testování softwaru a automatizace testování pomocí nástroje Selenium WebDriver. Syntézou dostupných zdrojů je dosaženo získání aktuálních a zevrubných informací pro tuto práci.

1.4. Předpoklady a omezení práce

Práce předpokládá základní orientaci ve světě informačních technologií, a ačkoliv k jejímu lepšímu pochopení mohou pomoci znalosti o programování či testování softwaru, nejsou k jejímu pochopení nezbytně nutné.

Oblast testování softwaru je široká a já jsem se vzhledem k mým praktickým zkušenostem zaměřil především na oblast testování softwaru pro elektronické bankovnictví a jeho automatizaci pomocí nástroje Selenium. Toto zúžení tématu umožní lépe analyzovat a zpracovat informace k přiblížení této problematiky.

1.5. Struktura práce

Práce je strukturována do kapitol a podkapitol. Po úvodu představujícím detaily práce a jeho zaměření, je přiblížena problematika testování softwaru obecně.

V prvních kapitolách práce se jedná především o teoretické představení problematiky testování softwaru a nástroje Selenium, kterými se práce zabývá.

V dalších kapitolách se práce již zabývá praktickým využitím automatizace a popisem postupu, kterým vznikají jednotlivé automatizované testovací scénáře. Tato část zahrnuje i závěry z těchto kroků a finální podoby zpracovaných skriptů.

Závěrem jsou shrnuty použité zdroje a samotný závěr práce vycházející z předchozích kapitol.

1.6. Výstupy a přínosy práce

Výstupem práce je představení nástroje Selenium a praktická ukázka jeho využití v reálném byznysu. Součástí jsou konkrétní realizované příklady a popisy testů v prostředí Selenium v oblasti elektronického bankovnictví, reporty jejich výsledků a výstupů.

Přínosem práce je představení možností automatizovaného testování pomocí nástroje Selenium, jmenovitě v oblasti elektronického bankovnictví. Dalším přínosem je možnost použití popsaných postupů automatizace i k automatizaci jiných oblastí testování softwaru.

2. Rešerše zdrojů na dané téma

Tato kapitola se zabývá dostupnými zdroji na téma testování softwaru a jeho automatizaci. Je v ní zmapováno množství dostupných zdrojů zabývajících se touto či podobnou problematikou.

2.1. Odborné publikace

Tématu testování softwaru a automatizaci právě pomocí nástroje Selenium se věnuje řada odborných publikací. Z důvodu aktuálnosti a mnohem většího výběru jsem ve většině případů volil ze zahraničních publikací.

Česká publikace, která se zabírá procesy testování a řízení kvality softwaru je od autorů Roudenského a Havlíčkové. Kniha ***Řízení kvality softwaru: průvodce testováním*** (Roudenský & Havlíčková, 2013) obsahuje informace o úrovních a typech testování, návrhu testovacích případů, tvorbě testů dle specifikace, reportech defektů i automatizaci. Kniha poskytuje obecný přehled problematiky.

Zahraniční publikace, která se zabývá detailněji nástrojem Selenium je od autora Avasarala. Tato kniha, ***Selenium WebDriver Practical Guide*** (Avasarala, 2014), představuje nástroj a jeho elementy, pokročilé možnosti nástroje s podrobným popisem jednotlivých prvků i s přesahem do jiných oblastí jako je kupříkladu testování mobilních aplikací.

Kniha, která obsahuje podrobné návody jak pracovat s nástrojem Selenium, jak tvořit, udržovat a zlepšovat automatizaci testování pomocí Selenia pochází od autora Gundecha. ***Selenium Testing Tools Cookbook*** (Gundecha, 2012) je zahraniční publikace, která obsahuje návody i na komplexnější úkoly s nástrojem Selenium WebDriver a pojednává o nejčastějších problémech, které se během automatizace testování moderních webových aplikací, mohou objevit. Ať už se jedná o pop-up okna, více otevřených oken, čekání prohlížeče či o mnoho dalších aspektů.

2.2. Akademické práce

V oblasti automatizace testování softwaru byly již některé vysokoškolské práce vydány, ovšem žádné se nezaobíraly specificky oblastí elektronického bankovnictví či pouze nástrojem Selenium WebDriver. Podobné zaměření se dá najít v následujících vybraných pracích.

Jednou takovou prací je diplomová práce ***Automation of regression testing of web applications*** (Chmurčiak, 2013)***.*** Autor se v této práci zabývá všemi nástroji Selenium a automatizací regresního testování vybrané webové aplikace za využití těchto nástrojů.

Další akademickou prací je ***Automatizované testování webových aplikací*** (Pietrik, 2012)***.*** Tato práce se zaobírá testováním webových aplikací s důrazem na jednu konkrétní. Jsou zde popsány veškeré nástroje Selenium.

Práce, která se zabývá tvorbou vlastního rozhraní k testování webových aplikací, ale obsahuje i příklady automatizace pomocí nástroje Selenium WebDriver, je akademická práce s názvem ***Automated Testing of the Component-based Web Application User Interfaces*** (Húska, 2012)**.**

Další prací, která se zabývá automatizací testování webových aplikací pomocí nástroje Selenium a jeho propojením s dalšími nástroji jako je např. TestNG je ***Funkční testování webových aplikací*** (Mazoch, 2012)**.** Tato publikace zmiňuje také všechny nástroje Selenium.

2.3. Internetové zdroje

Na internetu je k dispozici nesčetné množství informací o testování obecně, i konkrétně o nástroji Selenium WebDriver. Vyzdvihnu tedy ty nejpodstatnější.

Nejdůležitějším zdrojem pro nástroj Selenium jsou oficiální stránky tohoto nástroje, tj. ***SeleniumHQ*** (SeleniumHQ, 2015). Stránky projektu obsahují návody na všechny nástroje Selenium a jejich základní popisy.

Dalším důležitým zdrojem jsou stránky věnované automatizaci tímto nástrojem v jazyce Java, které se nachází na ***GoogleCode.com*** (Google Code, 2015). Zde jsou popsány jednotlivé balíčky, třídy apod.

Užitečným odkazem může být i oddíl věnovaný problematice Selenia na stránkách ***ToolsQA.com*** (ToolsQA, 2015). Zde jsou zmíněny jak základní, tak i pokročilejší řešení běžných problémů s nástrojem a jeho rozšířením.

Velké množství zdrojů se dá nalézt i na fóru, kde se řeší problémy v mnoha oblastech IT, Selenium nevyjímaje, a tím je **Stack OverFlow** (Stack OverFlow, 2015)**.**

Konkrétněji oblasti automatizace testování elektronického bankovnictví se nevěnují žádné profesionální weby, ale na některých webových stránkách se přece jen základní informace k této problematice najít dají.

K základnímu pochopení problematiky testování internetového bankovnictví lze nalézt informace například v prezentaci od C. Ramireze na **SlideShare** (SlideShare, 2013). Dalším zdrojem může být například web **Software Testing Help** (Software Testing Help, 2015).

3. Základní principy testování softwaru

Tato kapitola pojednává o základních principech testování softwaru. Je zde představeno manuální testování softwaru i možnosti automatizace. V neposlední řadě je zde i zmíněno specifické užití testování v oblasti elektronického bankovnictví.

Vzhledem k tématu práce je zapotřebí do problematiky uvést i obecnější přehled problematiky testování softwaru, aby bylo vidět, kam právě automatizované testování softwaru zapadá. Proto je to i dílčím cílem této práce.

3.1. Úrovně testů

Testování softwaru se dělí na několik úrovní na základě toho, v jaké fázi vývoje se provádí. Posloupnost testování je logická a dělí se na 5 základních úrovní. Na základě velikosti projektu i finančních možností se některé testovací fáze mohou přeskočit, ale obecně se to nedoporučuje, jelikož se mohou defekty odhalit o to později, což s sebou nese zvýšené finanční náklady. Níže ukázané členění je vysvětleno na základě informací z portálu, který se specializuje na problematiku ověřování kvality softwaru (Testování softwaru, 2015).

Nejdříve se provádí testování programátorem a v této fázi se většinou provádí tzv. "test čtyř očí". Programátor tedy kontroluje část kódu napsanou jiným programátorem. Jiný programátor tam může odhalit chyby, kterých si programátor této části kódu při vývoji nevšiml. Tato fáze bývá podceňovaná, ale chyby objevené v ní mají minimální náklady.

Dalším krokem je jednotkové testování, které také probíhá na úrovni programového kódu a bývá tedy prováděno vývojáři. Toto je poslední fáze, která probíhá v procesu samotného vývoje softwaru. Tyto testy se píší v programovém kódu a je třeba je připravovat průběžně, jelikož zpětně se tvoří velmi komplikovaně s velkými zásahy do kódu.

Dále přichází na řadu funkční testy, které testují funkcionalitu aplikace. Tyto jsou prováděny dodavatelem. Je v ní ověřováno, zda funkce fungují tak jak mají, a jak požadoval zákazník.

Po tomto otestování se provádí integrační testy, které ověřují bezchybnou komunikaci mezi jednotlivými komponenty. V této fázi je ověřena bezproblémová komunikace mezi komponenty. Komponenty jsou myšlené jak hardwarové, tak softwarové.

Další fází je systémové testování, kdy se již aplikace testuje jako celek a ověřuje se již z pohledu zákazníka, jestli vše funguje jak má dle testovacích scénářů. Tyto scénáře by měly být uzpůsobeny předpokládanému užití zákazníka a krokům, které v praxi mohou nastat. Jeho součástí jsou funkční i

nefunkční testy. Tato fáze je prováděna v několika kolech, ve kterých se odhalují a opravují defekty. Jedná se o výstupní kontrolu kvality softwaru před předáním zákazníkovi.

Poslední fází je akceptační testování, které se již provádí na straně zákazníka. V této fázi by již mělo být minimum defektů a jejich oprava by měla být poměrně rychlá. Testování v této fázi probíhá obvykle pomocí testovacích scénářů, které zpracuje dodavatel se zákazníkem. Pokud byly předchozí fáze otestovány důkladně, v softwaru by se nemělo již objevit tolik chyb, i když se málokdy stane, že by se neobjevila žádná.

3.2. Členění dle vlastností testů

Testování softwaru se ovšem může členit i na základě vlastností a zaměření. Nejzákladnější dělení dle tohoto kritéria je na testy funkční a nefunkční. Níže uvedené detaily ohledně těchto kritérií jsou čerpány z webu, který se věnuje problematice testování softwaru (SW Testování, 2015).

Funkční testy jsou testy, které se zaměřují na požadavky týkající se funkčnosti aplikace. Tyto požadavky definují užívání aplikace. Toho se většinou dociluje za pomoci tzv. způsobů užití neboli *UseCase*. Právě na základě těchto způsobů užití se testuje chování aplikace v situacích, které s nimi mohou souviset. Tyto testy mohou být prováděny manuálně i automatizovaně.

Další kategorií jsou nefunkční testy, které se zaměřují na zbylé vlastnosti aplikace. Jedná se tedy o vše, co přímo nesouvisí s funkcionalitou aplikace jako takové. Jsou to především výkonové testy, zátěžové testy, stress testy či penetrační testy.

Výkonové testy zjišťují, zda je aplikace schopna zvládnout očekávanou zátěž více současně připojených uživatelů. Kontroluje se například doba odezvy, která by neměla překročit předem definované hodnoty. Simulace takové zátěže může být prováděna specializovanými nástroji k tomu určenými.

Dalším typem nefunkčních testů jsou zátěžové testy, které jsou obvykle prováděny současně s testy výkonovými. Tyto testy ověřují stabilitu aplikace například při narůstání objemu dat.

Stress testy již neodhalují nedostatky v době odezvy, ale spíše chyby, které se mohou objevit až při velké zátěži. Hledají se tedy takové scénáře, které by mohly vést k pádu aplikace. Někdy jsou definovány společně s výkonovými a zátěžovými do jedné kategorie.

Poslední kategorií jsou testy penetrační. Tyto testy ověřují bezpečnost aplikace. Toho se dá docílit prováděním simulovaných útoků na aplikaci. To lze provádět buď ručně, nebo opět pomocí specializovaných nástrojů. Tyto testy by měly odhalit slabé stránky aplikace, které mohou ohrozit bezpečnost aplikace, dat či uživatelů.

Kategorií je samozřejmě více, ale výše zmíněné byly ty hlavní, které se běžně vyskytují v praxi a tester s nimi přichází často do styku.

3.3. Regresní testy

Regresní testování je proces, při kterém se testuje stávající funkcionalita po implementaci jakýchkoliv změn či nových funkcí do aplikace. Po přidání nové funkcionality se totiž může stát, že přestane korektně fungovat nějaká již dříve implementovaná funkcionalita. Cílem efektivního regresního testování je to, aby byla ověřena veškerá důležitá funkcionalita aplikace za co nejmenšího množství redundantních testů. (Microsoft, 2015)

Cílem regresního testování není tedy pokrýt veškeré do té doby vytvořené testovací scénáře, ale vybrat z nich vzorek, který ověří důležité funkčnosti aplikace. Jedná se tedy o typ funkčních testů. Vzhledem k tomu, že regresní testování se provádí tak často, je to ideální kandidát k automatizaci.

3.4. Manuální testování softwaru

Manuální testování softwaru je proces, při kterém se hledají defekty v aplikaci. Defekt je chyba v aplikaci, zpravidla způsobená chybami v kódu. Cílem testování je, aby se takových defektů vyskytovalo ve finálním produktu co nejméně a naopak co nejvíce existujících bylo odhaleno před předáním produktu zákazníkovi.

Tester při tomto testování zaujímá roli koncového zákazníka a ověřuje funkčnost veškerých funkcionalit. Je to ten nejzákladnější typ testování a je potřeba provést vždy i před zavedením případné automatizace, jelikož ověřuje její vhodnost. Tester v tomto případě nepotřebuje k testování žádný testovací nástroj a postupuje dle testovacího plánu. V současné době je automatizace stále testování stále populárnější a i když se často jeví především u větších projektů jako nejvhodnější řešení, 100 procentní automatizace není možná. Z tohoto důvodu je manuální testování nedílnou součástí testovacího procesu. (Software Testing Class, 2015)

Cílem manuálního testování je zajištění toho, aby byla aplikace bez defektů a odpovídala požadavkům a specifikacím zákazníka. Tester se musí vžít do role koncového klienta a odhalovat neočekávané chování aplikace a defekty. Nalezené defekty se hlásí vývojářům a tester má poté za úkol re-testovat opravené defekty. Zjednodušeně řečeno, cílem je dobrá kvalita produktu a spokojený zákazník.

Pro řádné otestování aplikace je potřeba mít detailní testovací plán a testovací scénáře. Bez těchto předpokladů se spoléhá na vysoké analytické schopnosti testera a nezaručuje splnění kvalitativních požadavků. K dosažení cíle je ideální pokrýt testy 100 procent funkcionalit aplikace. I přes detailní testovací proces nelze nikdy zaručit odhalení 100 procent defektů. (Guru99, 2015)

3.5. Možnosti a důvody automatizace testování softwaru

Automatizace testování softwaru se stává stále více populární, jelikož umožňuje rychlé ověření klíčových funkcionalit a provádění velkého množství testovacích případů v krátkém čase. Na druhou stranu je vždy spojeno s počátečními náklady na jeho zavedení a prozatím není příliš vhodné pro testování grafického zobrazení, což je potřeba testovat manuálně. Možný přechod k automatizaci je tedy na zvážení každé firmy a vstupuje do něj mnoho faktorů, jako jsou již zmíněné počáteční náklady, návratnost tohoto zavedení či četnost testování. Pokud se firma rozhodne, že se jí automatizace vyplatí, stále to povětšinou neznamená automatizaci stoprocentní. Většina projektů i nadále vyžaduje manuální testování u případů, které se automatizovat nedají, jako jsou kupříkladu zmíněné pokročilejší kontroly grafického uživatelského rozhraní.

K automatizaci existuje velké množství nástrojů a možnosti automatizace nedílně záleží na typu aplikace, která se testuje. Zatímco testování desktopových aplikací je náročnější a převážně parketou komerčních aplikací, testování webových aplikací, které se těší poměrně velké oblibě, nabízí i možnosti využití bezplatných aplikací.

V oblasti webových aplikací stojí za zmínku zejména Selenium, které je populárním testovacím nástrojem a umožňuje testování ve většině populárních prohlížečů. Dalšími nástroji, taktéž bezplatnými jsou kupříkladu SoapUI, Watir či Windmill. Mezi komerčními testovacími nástroji se jedná především o Ranorex, který umožňuje tvořit testy webových aplikací i grafického uživatelského rozhraní desktopových aplikací. (Testing Tools, 2015)

Primárním důvodem k automatizaci testování softwaru je finanční a časová úspora. Automatizace umožňuje provádět stejné operace precizně a identicky při každém opakování. K provádění automatizovaných testů je potřeba znatelně méně testerů, což snižuje náklady. Testovací scénáře se mohou také provádět znatelně rychleji než při přítomnosti lidského faktoru. Stále není možno automatizovat veškeré testy uživatelského rozhraní, kde se manuální testování nedá nahradit.

Vzhledem k tomu, že tvorba automatizovaných skriptů s sebou nese větší finanční investice než manuální testování, vyplatí se to především u často opakovaných testů či komplexních transakcí, kde se klade důraz na preciznost. Nejvhodnějšími kandidáty k automatizaci jsou regresní testy, které ověřují, zda nové funkcionality neovlivnily funkčnost aplikace. Tyto testy se provádějí opakovaně a často, finanční návratnost je u nich tedy poměrně rychlá. Mimo regresní testy, které jsou nejvhodnějším kandidátem na automatizaci, je také vhodné automatizovat tzv. Smoke testy, které zjednodušeně kontrolují, zda aplikace funguje a může postoupit do dalších fází testování. Tyto testy s sebou nesou také velké úspory, jelikož kontrolují hlavní oblasti a to, zda byla aplikace nasazena správně a prostředí se dá dále testovat. (Software Testing Class, 2015)

3.6. Specifika automatizace testů v oblasti elektronického bankovnictví

Elektronické bankovnictví, což je webová aplikace umožňující přístup k bankovním službám, je dnes nedílnou součástí bankovního styku. Jelikož přináší časové i finanční úspory jak bankám, tak klientům, je na tuto oblast kladen velký důraz. Klienti přes ně mohou uskutečňovat transakce od jednoduchých plateb až po žádosti o hypotéky. Banky na druhé straně nepotřebují tolik zaměstnanců, kteří by jinak tyto služby poskytovali pouze osobně.

V oblasti bankovnictví, a tedy i souvisejícího elektronického bankovnictví, dochází neustále ke změnám. Inovace v tomto odvětví přicházejí velice často a v této vysoce konkurenční oblasti trhu, je potřeba je co nejrychleji a nejlépe implementovat. Jakékoliv výpadky ve funkčnosti této webové aplikace mohou znamenat ztrátu klientů. Klienti jsou samozřejmě to, co banky živí, a funkčnost webových stránek a elektronického bankovnictví jsou to, s čím klienti přicházejí do styku nejčastěji.

V oblasti elektronického bankovnictví je automatizace testování velmi vhodná, jelikož vzhledem k tomu, že se jedná o webovou aplikaci, je poměrně jednoduchá a z pohledu softwaru i finančně nenáročná. V dnešní době je elektronické bankovnictví většinou poskytováno i skrze mobilní aplikace. Testování těchto aplikací je ovšem poněkud náročnější a vydalo by na samostatnou práci. Proto se v této práci omezím na elektronické bankovnictví ve formě webových aplikací. Stejně jako u jiných aplikací její největší výhoda je v oblasti regresních a smoke testů. V mnoha ohledech je elektronické bankovnictví aplikace jako každá jiná, ale vyskytují se zde jistá specifika, obzvláště v oblasti bezpečnosti či komplexity.

Jedním ze specifik elektronického bankovnictví jsou vysoké požadavky na zajištění bezpečnosti. Jelikož v produkčním prostředí pracuje elektronické bankovnictví s penězi, jsou vysoké bezpečnostní standardy naprostou nutností. Pro přihlášení se dají používat různé certifikáty, čipové karty či sms kódy. I tyto kroky se musí během testování, a tedy i následné automatizace testování, zahrnout do testovacího plánu. (Bezpečný internet.cz, 2015)

Dalším specifikem je rozsah elektronického bankovnictví. Systém elektronického bankovnictví je velice rozsáhlý a mimo dlouhých komplexních transakcí se často testují kupříkladu platby přes internet, kterých se mohou provádět velké počty denně jen se změnou parametrů. I tento rozsah lze samozřejmě velmi dobře pokrýt testy. Zde se dá ušetřit velké množství času a zajistit i ověřit funkčnost elektronického bankovnictví, která je vzhledem k propojení s reálnými financemi, pro klienty velmi důležitá.

Stejně jako i v jiných aplikacích, i v elektronickém bankovnictví dochází ke změnám v aplikaci. Vzhledem k tomu, že se služby poměrně často obměňují či aktualizují, musí být automatizace přizpůsobena právě jednoduchým úpravám, aby se celý proces nemusel dělat znovu. Velmi často jsou požadavky byznysu jen prosté změny textu, popisků tlačítek či právních formulací v reakci na změny zákona. Logika aplikace zůstává v drtivé většině případů stejná. Právě na tyto "povrchní" změny lze automatizaci uzpůsobit. Automatizace většinou funguje pomocí identifikace prvků na stránce dle parametrů. I takové jednoduché pravidlo, jako je nedefinování prvků na základě často obměňovaných textů, může poté v údržbě testů přinést velké úspory.

Posledním hlavním specifikem testování této oblasti je generování dokumentů. Ať se již jedná o potvrzení plateb či výpisy z účtu, elektronické bankovnictví často pracuje s dokumenty, nejčastěji ve formátu PDF. Toto je jedno z úskalí většiny nástrojů, jako je právě například Selenium. Nejčastěji se buď nechá tento krok ověření na manuálním testerovi či pomocí vhodných doplňků pracujících na bázi screenshotů, které se dají s testovacím nástrojem propojit. Selenium jako takové to oficiálně neumožňuje, ovšem lze i pomocí WebDriveru obsah některých dokumentů ověřit. (Shah, 2015)

3.7. Možnosti využití automatizovaných testů v oblasti elektronického bankovnictví

V této kapitole se zaměřuji na některé konkrétní oblasti, ve kterých se dá právě automatizace využít a její zavedení je v nich tedy poměrně častým jevem. Samozřejmě je možné využití i v jiných oblastech, ovšem není již natolik typické.

3.7.1. Regresní testy rozsáhlých aplikačních systémů

Jak již bylo zmíněno v podkapitole 3.3, regresní testy ověřují, zda implementace nových funkcionalit nezměnila stávající funkčnost aplikace. Jelikož elektronické bankovnictví je velmi rozsáhlý aplikační systém, toto riziko ovlivnění stávajících funkcionalit se právě touto komplexitou zvyšuje. Po každém release by tedy mělo následovat i důkladné regresní testování. Vzhledem k tomu, že regresní testování je z velké části neměnné a vzhledem k neustálým inovacím i byznysovým požadavkům časté, je to vhodný kandidát pro automatizaci.

Regresní testování elektronického bankovnictví a jeho automatizace umožňují i úpravu parametrů při spouštění testů. Při automatizaci se tvoří co nejfunkčnější, ale zároveň nejuniverzálnější testovací případy, které parametrizaci v rámci testování umožňují. Jednotlivé testovací scénáře se neustále opakují, ovšem data jako jsou čísla bankovních účtů a podobné se mohou v závislosti na konkrétním použití měnit. V automatizovaných testech je samozřejmě možno nastavit tyto parametry jako konfigurovatelné a není třeba kvůli jednomu parametru měnit celý testovací scénář. Příkladem může být jednorázová tuzemská platba, což je esenciální funkčnost a vždy testovaná funkcionalita. Zde mezi konfigurovatelné parametry patří např. číslo účtu, částka či datum. Důvodů, proč je potřeba

změn parametrů, může být mnoho, ale jedním z nich je například nedostatek financí na testovacím bankovním účtu či potřeba vyzkoušet platbu mimo pracovní dny.

3.7.2. Smoke testy v rámci podpory nasazení změn

Spuštěním smoke testů před zahájením samotného testování se dá zjistit, zda webová aplikace a testovací prostředí jako celek funguje a má tedy cenu testovat dle jednotlivých testovacích scénářů. Může se totiž stát, že při nasazení nové verze se něco pokazí. To by měly právě zjistit tyto smoke testy, které ověří běžné funkcionality. Jelikož běžné funkcionality se obvykle nemění, je to jeden z ideálních kandidátů k automatizaci. Spuštěním těchto testů, které nezaberou z důvodu malého rozsahu tolik času, se dá zamezit zbytečnému investování do testerů, kteří by poté museli čekat na nové nasazení a zjištění problému.

Právě při nasazení změn se stává, že nastane nějaký problém, který zamezí možnosti dalšího testování, jako je například špatná implementace nových funkcí. Tato oblast je tedy u tak velkých aplikací, jako je elektronické bankovnictví, velice důležitá a může s sebou nést nemalé úspory. Smoke testy v automatizované formě se mohou spustit přes noc, aby se již před zahájením práce druhý den mohlo zjistit, zda bude moci probíhat testování.

3.7.3. Opakované provádění komplexních transakcí

Mimo výše zmíněné oblasti je automatizace velmi výhodná i u komplexních transakcí, které nejsou v elektronickém bankovnictví a jeho testování výjimkou. Může se jednat například o rozsáhlé žádosti. Některé žádosti, které vyžadují velké množství informací, a musí se pro ověření funkčnosti projít celé, mají velké množství kroků. Právě z tohoto důvodu se u nich vyplatí automatizace. Právě vyplňování formulářů je nenáročné na automatizaci a dokáže ušetřit velké množství času. Příkladem mohou být například žádosti o půjčky, kde banka vyžaduje velké množství informací od osobních údajů po údaje o zaměstnání či jiných závazcích.

Dalšími komplexními transakcemi jsou kupříkladu několikafázové transakce, které vyžadují časovou prodlevu a pozdější ověření. Příkladem může být scénář, který ověřuje propsání transakce do výpisů, cílového účtu či provedení v konkrétní čas. Namísto testera, který se bude ke scénáři opakovaně vracet a toto ověřovat, se to dá také automatizovat. Právě zde je vidět i další výhoda automatizace, a to precizní provádění testů. Při takto rozsáhlých transakcích by manuální tester mohl v některém kroku udělat chybu, a celý tento časově náročný scénář by se musel provádět znovu.

4. Představení nástroje Selenium

V této kapitole je představen nástroj Selenium, který umožňuje automatizaci funkčního testování webových aplikací. Tento bezplatný nástroj je stále vyvíjen a díky podpoře drtivé většiny prohlížečů, v kterých dokáže testovat a množství programovacích jazyků, kterými lze kód psát, je poměrně univerzálním testovacím nástrojem.

4.1. Popis nástroje Selenium

Selenium je jedním z nejpopulárnějších bezplatných testovacích nástrojů sloužícím k automatizaci funkčního testování webových aplikací. Primárně byl tvořen pro prohlížeč Mozilla Firefox, ale nyní již podporuje i veškeré ostatní populární prohlížeče, jako jsou Internet Explorer, Google Chrome či Safari.

Původně se Selenium dělilo na 4 produkty, Selenium WebDriver, Selenium Remote Control, Selenium IDE a Selenium Grid. Po aktualizaci produktu na Selenium 2.0, se ovšem většina těchto produktů sloučila. Nyní existují tedy 2 primární produkty, a to Selenium WebDriver a Selenium IDE. (SeleniumHQ, 2015)

Selenium IDE je jednodušší nástroj, který umožňuje jednoduché vytváření automatizovaných skriptů bez jakýchkoliv znalostí programovacích jazyků. Jedná se o plug-in do prohlížeče Firefox, pomocí kterého se nahrává, co uživatel dělá v prohlížeči a poté se to dá ukládat a přehrávat. Toto řešení je vhodné především pro malé projekty a jednotlivé uživatele. Ovšem dá se využít i k jednodušší tvorbě pokročilejších skriptů v nástroji Selenium WebDriver, jelikož může pomoci identifikovat prvky.

Selenium WebDriver je již pokročilý nástroj, který umožňuje tvorbu robustních regresních testů v různých testovacích prostředích/prohlížečích. Za pomoci tohoto testovacího nástroje a znalosti programovacího jazyka, jako je např. Java, lze vytvořit velmi komplexní automatizované testovací scénáře. Testy se poté vytváří ve vhodném vývojovém programu, jako je např. IntelliJ IDEA či Eclipse.

4.2. Konkrétní oblasti využití nástroje Selenium WebDriver

Selenium WebDriver je klíčové rozhraní, ve kterém je možné psát a provádět automatizované testování webových aplikací za pomocí rozličných programovacích jazyků, jako je např. Java. Pomocí knihoven ve vybraném programovacím jazyku Selenium WebDriver poté ovládá jednotlivé prohlížeče. Toto rozhraní provádí kroky v prohlížeči na základě napsaných testovacích scénářů přesně tak, jako by uživatel manuálně „klikal“. Po provedení kroků v prohlížeči zasílá zpět výsledky testu do testovacího scénáře k analýze testerovi. (Avasarala, 2014, pp. 12-13)

Nástroj Selenium WebDriver může být použit napříč spektrem webových aplikací. Vhodným automatizačním nástrojem může být mimo oblast elektronického bankovnictví i například na webových portálech mobilních operátorů. Uplatnění zajisté najde i u menších projektů, jako mohou být i osobní webové stránky, ale v malém rozsahu by postačoval i nástroj Selenium IDE. Jelikož s sebou automatizace v tomto nástroji většinou nese poměrně velké počáteční finanční investice v případě plošného zavedení, vyplatí se především u větších projektů, kde je testování prováděno pravidelně a investice se tedy v delším časovém horizontu navrátí. Následné návratnosti pomůže samozřejmě i nutnost menšího počtu testerů, které by jinak produkty museli testovat ručně.

Mimo zjevných oblastí, jako jsou velké projekty, které jsou finančně výnosné, by se automatizace měla dělat i v případě esenciálních služeb, jejichž výpadek by mohl způsobit velké komplikace. Jmenovitě tedy například ve veřejném sektoru, kde kupříkladu u elektronických registrů je soustavná funkčnost portálu nezbytná, by se mělo investovat do automatizace.

4.3. Postup automatizace testování webových aplikací pomocí Selenia WebDriver

Automatizace v nástroji Selenium WebDriver není příliš komplikovaná, ale i přesto je potřeba na začátku celého procesu několik věcí nastavit. Nástroj sám o sobě je bezplatný, takže není potřeba zakupovat žádné licence a lze přejít rovnou k instalaci. K WebDriveru je možné získat i Selenium-Server, který ovšem není potřeba, pokud bude testování probíhat na stejném počítači, na kterém bude běžet i webový prohlížeč. Selenium-Server je potřeba pouze v případě, že se prohlížeč spouští na vzdáleném počítači.

4.3.1. Instalace Selenia WebDriver

Instalace Selenia se mírně liší na základě toho, jaký programovací jazyk bude použit k tvorbě testovacích scénářů. Selenium podporuje jazyky, jako jsou Java, C#, Python, Ruby, PHP, Perl či Javascript. Rozdíly nejsou nikterak veliké a veškeré detaily se dají najít v oficiální dokumentaci Selenia (SeleniumHQ, 2015). Jelikož jednoznačně nejpopulárnějším programovacím jazykem pro tvorbu testů v Seleniu je Java, zaměřím se tedy právě na ni.

Projekt lze nejjednodušeji založit pomocí Apache Maven, což je nástroj ke správě projektů. Tento nástroj dokáže spravovat sestavení projektu, reporting i dokumentaci z jednoho centrálního zdroje informací. V tomto případě se jedná o soubor pom.xml, který je rozebrán níže. (The Apache Software Foundation, 2015)

Po vytvoření potřebných složek a souborů se pokračuje stažením potřebných knihoven Selenium a následným importem projektu do vývojového prostředí.

4.3.1.1. Tvorba složky k uchovávání souborů projektu Selenia

Nejdříve je potřeba si vytvořit složku, kde se budou uchovávat veškeré soubory k projektu v Seleniu.

4.3.1.2. Tvorba souboru pom.xml

K použití Apache Maven je potřeba vytvořit soubor *pom.xml.* To lze udělat například v textovém editoru. Základní verze tohoto souboru vypadá takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>MySel20Proj</groupId>
        <artifactId>MySel20Proj</artifactId>
        <version>1.0</version>
        <dependencies>
            <dependency>
                <groupId>org.seleniumhq.selenium</groupId>
                <artifactId>selenium-java</artifactId>
                <version>2.45.0</version>
            </dependency>
        </dependencies>
</project>
```

Obrázek 1 - vzorový pom.xml (SeleniumHQ, 2015)

Tento soubor musí být umístěn v adresáři, který byl vytvořen v prvním kroku právě za účelem uchovávání souborů projektu Selenia. V textovém editoru je zde potřeba změnit verzi na aktuální. Tu lze zjistit přes odkaz dostupný přímo na oficiálních stránkách Selenia, Maven Information (SeleniumHQ, 2015). Tam lze již aktuální vytvořený soubor pom.xml i stáhnout.

4.3.1.3. Stažení knihoven Selenia

Dalším krokem je spuštění příkazové řádky, navigace do adresáře se soubory projektu a spuštění Apache Maven pomocí následujícího příkazu:

mvn clean install

Maven si stáhne potřebné knihovny Selenia a přidá je k projektu. (SeleniumHQ, 2015)

4.3.1.4. Import projektu

Nyní již stačí projekt importovat do zvoleného vývojového prostředí, kupříkladu Eclipse. Pro Eclipse dokáže Maven vygenerovat potřebné projektové soubory pomocí tohoto příkazu:

mvn eclipse:eclipse

Po otevření programu Eclipse a nového pracovního prostředí je třeba jít do menu *File – Import – General – Existing Projects into Workspace – Select root Directory – Browse – Project Folder – OK.*

Pro integraci Eclipse s Maven je ještě užitečné stáhnout a nainstalovat M2Eclipse plugin, a pak kliknout pravým tlačítkem na projekt a vybrat *Maven – Enable Dependency Management*. (SeleniumHQ, 2015)

Po importu projektu je již vše připraveno k automatizaci. Veškeré níže ukázané vzory kódu v této kapitole jsou vytvořeny na základě oficiální dokumentace Selenia (SeleniumHQ, 2015).

4.3.2. Příkazy k tvorbě testovacího scénáře

Nyní budou ukázány základní příkazy, které je možné použít k automatizaci testovacího scénáře. Příkazů je samozřejmě více, než je zde ukázáno, ale jedná se o ty hlavní, s kterými se tester může při automatizaci setkat.

4.3.2.1. Inicializace WebDriveru

Pravděpodobně prvním příkazem, který bude tester potřebovat pro tvorbu testovacího případu je inicializace WebDriveru. Konkrétní inicializace na příkladu Firefoxu vypadá takto:

```
WebDriver driver = new FirefoxDriver();
```

4.3.2.2. Navigace na stránky

Pro navigaci na stránky obsahuje WebDriver příkaz *get*, na konkrétním příkladu je vidět zde:

```
driver.get(http://www.zvolenyodkaz.com)
```

4.3.2.3. Nalezení prvků na stránce

Dalšími nezbytnými příkazy pro automatizaci je vyhledávání tzv. WebElements, což jsou zjednodušeně jakékoliv prvky na webové stránce. Vyhledávání probíhá užitím metody, ve které se dále určuje, na základě jakého parametru se prvek hledá. Tyto parametry se vyhledávají v kódu webové aplikace, nejčastěji tedy v HTML či CSS.

4.3.2.3.1. Lokace dle ID

Ideálním lokátorem je ID, které je preferované jako nejefektivnější možnost. Častým problémem v tomto případě je ovšem, že vývojáři webové aplikace definovali neunikátní ID či automaticky generované. Ačkoliv se tyto praktiky silně nedoporučují, je běžnou praxí, že se automaticky generované ID stále vyskytují ve velkém množství webových aplikací. V ideálním případě, kdy jsou webové aplikace napsané se zachováním unikátních definovaných ID, se k nalezení elementu dá použít tato definice:

```
WebElement element = driver.findElement(By.id("zvoleneID"));
```

4.3.2.3.2. Lokace dle jména třídy

Jelikož se ale velmi často s takto vstřícně napsanými aplikacemi člověk nesetká, musí se přistoupit k použití jiného lokátoru. Dalším velmi vhodným lokátorem je jméno třídy neboli *Class Name*. Jelikož ovšem jméno třídy nebývá v drtivé většině případů jedinečné, dá se použít i k nalezení množiny elementů. Taková množina se dá získat pomocí tohoto příkazu

```
List<WebElement> prvky = driver.findElements(By.className("zvolenaTrida"));
```

4.3.2.3.3. Lokace dle jména značky

Další možností, jak nalézt prvek, je pomocí jména značky neboli *Tag Name*. Ani zde není zaručena unikátnost, jelikož stejných prvků může být samozřejmě na stránce vice. Ukázkou takového příkazu je například tento:

```
WebElement tabulka = driver.findElement(By.tagName("table"));
```

Právě v případě většího množství prvků se dá ten správný nalézt i pomocí jména, které může být na stránce v kódu definováno. Příkladem může být kupříkladu

```
WebElement priklad = driver.findElement(By.name("zvolenejmenoprikladu"));
```

Lokátorem může být také text, který obsahuje hypertextový odkaz. Takovým elementem může být například:

```
<a href="www.seznam.cz">Jdi na seznam</a>
```

V tomto příkladu by se tedy použil příkaz:

```
WebElement seznam = driver.findElement(By.linkText("Jdi na seznam"));
```

Mimo vyhledávání pomocí textu, který někam odkazuje, se dá použít i pouze jeho část. K tomu se dá použít lokátor *Partial Link Text*. Z předchozího příkladu by tedy k nalezení takového prvku stačil tento příkaz:

```
WebElement seznam = driver.findElement(By.partialLinkText("seznam"));
```

V případě použití CSS na stránce, lze prvky najít i právě pomocí tagů CSS. Nechť je příkladem tento příkaz:

```
WebElement obrazek = driver.findElement(By.cssSelector("img[alt='priklad']"));
```

Posledním, ale „nejmocnějším“ lokátorem na webových stránkách je tzv. XPATH. Pomocí tohoto lokátoru lze definovat cesta k prvku a poradí si i s nalezením prvků, které se pomocí předchozích lokátorů nalézt nedají. Příkladem je nalezení textového pole na stránce:

WebElement pole = driver.findElement(By.xpath("//input"));

4.3.2.4. Operace na formulářích

Poté, co jsou nadefinované elementy, s nimi můžeme již manipulovat pomocí jmen, které jsme jim přiřadili. Operací je velké množství, ale za zmínku stojí alespoň ty základní.

Pravděpodobně nejzákladnějším příkazem, který se používá při automatizaci a interakci s elementy na webových stránkách je klikání na prvky. Tento příkaz je jednoduchý, a pokud bude chtít uživatel kliknout na prvek, který si pojmenoval "prvek", bude příkaz následovný:

prvek.click();

Dalším častým příkazem, který zajisté přijde vhod, je vložení nějakého textu do textového pole. Pokud je tedy textové pole pojmenované *priklad*, bude příkaz znít:

priklad.sendKeys("Ahoj!");

Mimo klikání je zajisté vhodnou operací i volba jednotlivých možností na formuláři. K tomu slouží třída *Select*. Pomocí metod, které tato třída poskytuje, lze provést volbu jednotlivých možností. Příkladem definice těchto možností je:

Select moznost = new Select(driver.findElement(By.tagName("select")));

Pro volbu možnosti je vždy vhodné nejdříve odznačit veškeré možné zaškrtnuté možnosti tímto příkazem:

select.deselectAll();

a poté již označit konkrétní zvolenou možnost takto:

select.selectByVisibleText("PrvniVolba");

Pro odeslání formuláře můžeme opět použít příkaz kliknutí na prvek jako kupříkladu: ***driver.findElement(By.id("submit")).click();***

nebo v Seleniu implementovanou metodu *submit* pomocí příkazu:

prvek.submit();

4.3.2.5. Operace s okny

Selenium umí také pracovat s více okny. Na to má metodu *switchTo*. Tato metoda se dá použít, pokud tester zná jméno jednotlivých oken. Příkaz by vypadal následovně: ***driver.switchTo().window("jmenoOkna");***

Pokud jméno okna není známo, lze přepínat mezi všemi otevřenými okny pomocí následujícího příkazu:

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

Častým problémem automatizace je práce s vyskakovacími upozorněními. Od verze Selenia 2.0 je již podpora práce s těmito hláškami integrovaná přímo do Selenia. Přepnout do hlášky, která na okně vyskočila lze pomocí příkazu:

Alert alert = driver.switchTo().alert();

V tomto konkrétním objektu již fungují metody na akceptaci:

alert.accept();

či zrušení:

alert.dismiss();

WebDriver Selenia umí pracovat i s navigací na stránkách, stejně jako umožňuje prohlížeč. Jedná se například o příkazy na posun o stránku zpět či vpřed. Tyto příkazy jsou:

driver.navigate().forward();

či

driver.navigate().back();

Mimo práce s okny umí Selenium pracovat i s různými prohlížeči. Jsou zapotřebí binární soubory jednotlivých ovladačů. Se získanými binárními soubory je již inicializace jednotlivých prohlížečů jednoduchá a může vypadat např. takto:

WebDriver driver = new ChromeDriver();

Při automatizaci přijde někdy vhod i čekání na načtení jednotlivých stránek, což lze provést v Seleniu dvěma způsoby.

Buď explicitně, tedy dokud nenastanou na stránce požadované podmínky, jako je např. možnost kliknutí na prvek na stránce. U tohoto příkazu se nastavuje i časový limit, po kterém vyskočí chyba, pokud událost nenastane. Takový příkaz nejdříve definuje samotnou metodu čekání:

WebDriverWait cekani = new WebDriverWait(driver, 10);

a poté podmínky u prvku, který jsme zvolili:

WebElement prvek=wait.until(ExpectedConditions.elementToBeClickable(By.id("ukazkoveid")));

V tomto případě je časový limit 10 sekund a čeká se, než se na prvek bude dát kliknout.

Druhá možnost je implicitní čekání, tedy daný časový limit bez zvolené podmínky. V tomto případě se nečeká maximálně tento limit jako u explicitního čekání, ale přesně tento limit. Příkaz pro toto čekání je např.:

driver.manage().timeout().implicitlyWait(10, TimeUnit.SECONDS);

5. Popis vzorového příkladu přípravy automatizovaných testů

V této kapitole se řeší postup tvorby automatizovaných testů nástrojem Selenium WebDriver, což je hlavním cílem této práce. Jednotlivé kroky tvorby i exekuce testů zde jsou ukázány a popsány po částech a kompletní vzorové testovací případy jsou níže přiloženy jako příloha.

Každý testovací případ má samozřejmě jiné požadavky, ale základ a některá pravidla by měla být stejná. Jedná se především o samotnou logiku testovacích případů, která bývá u stejného projektu stejná, a to také usnadňuje následné úpravy a aktualizace.

Pokud jsou testovacího případy navrženy správně již od samého počátku, dokáže to ušetřit náklady při jakýchkoliv dalších zásazích do kódu. Správný návrh by měl být vždy konzultován s organizací, pro kterou se to dělá, aby se dalo určit, jaké parametry se budou často měnit. Jak je níže v dalších kapitolách ukázáno, testovací případy jsou tvořeny pomocí práce s prvky na webových stránkách, které si za použití Javy a Selenia definujeme ve třídách testů. Tyto prvky jsou vždy zvoleny na základě určitých parametrů, ať již kódu CSS či HTML značek. Toto je více rozebráno níže.

V ideálním případě se při úpravách někdy změní jen parametry, které jsou zvoleny jako konfigurovatelné. Někdy je samozřejmě zapotřebí změnit i argument prvku, dle kterého ho nacházíme, pokud byly změny ve webové aplikaci. I toto lze ovšem omezit tím, že vybíráme stabilnější parametry prvku a ne kupříkladu dynamické ID či textový popis, který se také může měnit na základě byznysových požadavků dost často.

5.1. Příprava testovacích případů

Příprava testovacích případů je velmi důležitá a důkladná analýza může ovlivnit i délku a náročnost projektu, tedy potažmo náklady vynaložené na testování. Pro co nejlepší automatizaci je zapotřebí si určit přesný postup a posloupnost jednotlivých testovacích kroků.

Určení testovacích kroků by mělo být co nejdůkladnější již v přípravě manuálních scénářů, aby to bylo již pro testera, který automatizuje, jednoznačné. Tento tester totiž řeší především technickou stránku věci, tedy kód a jeho optimalizaci pro co nejefektivnější použití. Dalším aspektem, který se musí řešit již ve fázi analýzy je aktualizovatelnost kódu. Málokdy se stane, že jsou testovací scénáře neměnné, jelikož webové aplikace se neustále vyvíjejí, obzvláště ve vybraném případě elektronického bankovnictví.

Pro to, aby se co nejlépe odhadly budoucí změny, konkrétně oblast těchto změn, je zapotřebí především zkušeností z organizace, ať již na základě informací od dlouhodobých testerů či historickým testovacích případů, kde se dá zjistit, jaké věci se zatím změnily a kde je tedy největší riziko změn. Mnohem lépe se totiž automatizuje a aktualizuje testovací případ s několika parametry

než možná lépe vypadající případ přesně na míru, který se ovšem musí po jakémkoliv zásahu do webové aplikace změnit.

K automatizaci je potřeba také analyzovat, zda je třeba automatizovat veškeré manuální scénáře. Čím více automatizovaných scénářů, tím větší náklady na tvorbu i aktualizaci. V případě, kdy automatizace zcela nahradí regresní testování, je dobré zvážit, zda některé scénáře netestují to samé či zda by stejné potenciální chyby v aplikaci nebyly odhaleny více scénáři. Před automatizací je zajisté dobrá selekce scénářů, které se budou automatizovat, aby nedocházelo k redundantnímu testování, které má samozřejmě spoustu nevýhod, jako je větší časová, finanční i hardwarová zátěž.

5.2. Vzorové skripty automatizace na konkrétním příkladu

Testovacích skriptů, na kterých lze postup automatizace ukázat, je celá řada. Samozřejmě zde nemohou být rozebrány všechny, takže jsem se snažil vybrat nejtypičtější případy, s kterými se lze setkat při testování elektronického bankovnictví. Na níže uvedených skriptech je ukázáno, jak celý proces tvorby těchto scénářů probíhá. V této kapitole jsou analyzovány kroky po jednotlivých částech, na konci dokumentu jsou poté přiloženy již kompletní skripty.

Konkrétně vybrané scénáře jsou *Přihlašování do elektronického bankovnictví*, což je absolutně základní a nezbytná funkcionalita, která se vždy testuje. Dalším vybraným scénářem je založení kontokorentu, což je poměrně komplexní scénář, který obsahuje více formulářů k vyplnění.

Tvorba testovacích skriptů v Seleniu nemá předepsané jmenné konvence, ale bývá zvykem v názvu souboru zmínit funkcionalitu, která se testuje a zakončit slovem *Test*. Takto se postupuje pro jednodušší pochopení scénáře již na první pohled. Kupříkladu *ChangeUsernameTest,* kde je na první pohled vidět, co se testuje. Stejně tak by měly být jednoznačně pojmenovány i prvky v souboru, kupříkladu tlačítko sloužící k přihlášení by mohlo být nazváno *loginButton*. Stejně jako při psaní čehokoliv jiného v programovacích jazycích, volba jazyka užívaného k pojmenování tříd je na vývojáři, ovšem nejuniverzálnějším je angličtina. I proto zmíněné skripty budu ukazovat za použití anglického jazyka v názvech.

Elektronické bankovnictví, stejně jako jiné webové aplikace, se skládá z jednotlivých webových stránek, na kterých se nachází prvky, s kterými pak Selenium a tester dokáží pracovat. Některé stránky se používají kupříkladu jen v jednom testovacím scénáři, ale velké množství, je používáno nejen jedním. Právě z důvodu ušetření kódu i univerzálnosti užití již popsaných prvků, je vhodné pracovat nejen se souborem testu, ale i se souborem, který představuje právě onu webovou stránku. Zde je také vhodné popsat název souboru, tak aby bylo jasné, o jakou stránku se jedná.

Příkladem výše zmíněného může být domovská stránka elektronického bankovnictví, která může být pojmenována *HomePage.* V tomto souboru se definují prvky, s kterými na této stránce lze pracovat a nějaké základní metody. Dalším nutným souborem je již samotný testovací scénář, který se může jmenovat *LoginTest.* Zde již budou rozepisovány jednotlivé kroky scénáře. Tímto je docíleno to, že s prvky webové stránky, může nyní pracovat bez opětovné definice prvků a metod více testovacích scénářů. Kupříkladu pokud bude scénář testovat možnost změny hesla *PasswordChangeTest*, prvním krokem testu bude přihlášení do elektronického bankovnictví. Místo toho, aby se tedy prvky domovské stránky a metoda přihlášení musely znova definovat, dá se již použít stránka *HomePage* a její metody, které byly vytvořeny při tvorbě předchozího testu.

5.2.1. Testovací skript 1 – Přihlášení do elektronického bankovnictví

K představení prvního testovacího skriptu je zde nejdříve rozebrán testovací případ, na základě kterého je tento testovací skript automatizován. Poté již následuje popis automatizace testovacího skriptu na vytvořeném příkladu.

Popis testovacího případu bude prováděn na základě metodiky MMSP, která je dostupná online. Jedná se o metodiku pro menší softwarové projekty. Tato metodika vznikla v rámci diplomové práce na Vysoké škole ekonomické v Praze, a to lokalizací a přizpůsobením metodiky OpenUP. (Rejnková, 2011)

5.2.1.1. Popis testovacího případu

Popis testovacího případu je proveden za pomoci užití šablony testovacího případu z metodiky MMSP dostupné na stránkách projektu. (Rejnková, nedatováno)

Tabulka 1 - testovací případ 1 (Rejnková, nedatováno)

ID		1			
Název testovacího případu		Přihlášení do elektronického bankovnictví			
Popis		Testovací případ testuje přihlašování uživatele do elektronického bankovnictví			
Vstupní podmínky		Jako vstupní podmínka je dostupnost přihlašovacích údajů libovolného uživatele elektronického bankovnictví			
Poznámky		Aplikace nebude nikterak ovlivněna a její funkčnost bude identická před i po testu			
	Odkaz na testovací data	**Akce testera**	**Reakce systému**	**Výsledek (OK/Error)**	**Poznámky/ odkaz na report chyby**

1.		Tester jde v internetovém prohlížeči Firefox na adresu www.internetovebankovnictvi.cz	Načte se přihlašovací obrazovka		
2.	Prihl_udaje.doc	Tester zadá přihlašovací údaje libovolného uživatele	Údaje jsou zadané do textových polí a heslo v podobě teček		
3.		Tester klikne na tlačítko přihlášení	Načte se úvodní obrazovka uživatele		
			Celkový výsledek		

5.2.1.2. Popis automatizovaného testovacího skriptu

Pro testování přihlašování do elektronického bankovnictví jsou tedy potřeba dva soubory. Stránka *LoginPage,* kde jsou definovány prvky, s kterými lze pracovat a základní metody. Dalším je již soubor s testovacím scénářem *LoginTest.* Zde jsou vidět jednotlivé kroky scénáře.

Nejdříve je zde ukázán postup definice nezbytných prvků a metod na přihlašovací stránce.

```
public class LoginPage {

    @FindBy(css = ".buttonLogin")
    private WebElement loginButton;

    @FindBy(id = "username")
    private WebElement usernameField;

    @FindBy(id = "password-view")
    private WebElement passField;
```

Kód 1 – LoginPage (autor)

Kód 1 - LoginPage definuje třídu *LoginPage* dle formátování v jazyce Java. Dále jsou zde prvky, s kterými na stránce tester chce pracovat. *@FindBy* definuje na základě čeho je prvek na stránce identifikován. U prvního prvku je použit parametr CSS. Tento je zvolen, jelikož u tohoto prvku je unikátní, zatímco preferované ID unikátní nebylo. U dalších dvou jsou již zvoleny parametry ID, které u těchto prvků byly unikátní.

Prvním je tlačítko k přihlášení. Dalšími prvky jsou textová pole na uživatelské jméno a heslo.

Mnohdy je více unikátních parametrů, na základě kterých se dá prvek definovat. V tomto případě je na uvážení testera, který zvolí. V úvahu je třeba brát, jaké prvky jsou méně náchylné ke změně. Obvykle se spíše mění texty zobrazené na stránkách, než názvy prvků v kódu. Důvodem může být

například změna názvu produktu, popisků apod. Toto se ovšem nemusí vůbec dotknout pojmenování kódu, jako např. CSS prvků, ID apod.

```
    public void setUsername(String username) {
        usernameField.sendKeys(username);
    }
    public void setPassword(String password) {
        passField.sendKeys(password);
    }
```

Kód 2 – LoginPage (autor)

Na kódu 2 jsou již vidět první dvě metody definované na stránce. Jedná se o metody, které vkládají do polí uživatelského jména a hesla text. Tyto metody vyžadují při zavolání parametr ve formě textu. Poté již dříve definované prvky pomocí základní metody Selenia *.sendKeys* zašlou do textového pole parametr, který obdržely při zavolání metody.

```
    public void login() {
        this.setUsername("testovaciuzivatel1");
        this.setPassword("heslo1");
        this.loginButton.click();

    }
    public void login(String user, String pass) {
        this.setUsername(user);
        this.setPassword(pass);
        this.loginButton.click();
    }}
```

Kód 3 – LoginPage (autor)

Poslední dvě metody definované na této stránce, ukázané v kódu 3, slouží obě k přihlášení do účtu elektronického bankovnictví. První metoda nevyžaduje při zavolání parametr a poté při volání metod na poslání textu *setUsername* a *setPassword* používá výchozí parametry. Druhá metoda vyžaduje jak parametr s uživatelským jménem, tak heslem. Tyto parametry poté použije k vypsání do textových polí. Tester má tedy v jednotlivých testovacích scénářích, kde nejdříve přihlašuje uživatele, na výběr, zda použije metodu s výchozím uživatelem, či zvolí nějakého jiného. V obou případech poté ještě pomocí základní metody Selenia *.click()* klikne na tlačítko přihlášení, které bylo výše ve třídě definováno.

Toto je tedy celá stránka, která slouží k přihlašování. Jelikož většina testů elektronického bankovnictví probíhá po přihlášení do samotného uživatelského prostředí, je to stránka velmi používaná a to, že

její metody a prvky nebudeme potřebovat definovat v každém testovacím scénáři definovat znova, ušetří mnoho času i kódu.

Nyní přichází na řadu samotná třída testovacího scénáře, která je v případě prostého přihlášení poměrně jednoduchá.

```
public class LoginTest {

    public LoginTest() {
        super();
    }

    @Test
    public void loginTest() {
      WebDriver driver = new FirefoxDriver();
        driver.get("www.internetovebankovnictvi.cz");
```

Kód 4 – LoginTest (autor)

Kód 4 – LoginTest inicializuje třídu, jak je v Javě obvyklé. Poté je zde již inicializace samotného WebDriveru, který spouští driver prohlížeče Firefox. Dále pomocí příkazu *.get* je driver směřován na webový odkaz s elektronickým bankovnictvím.

```
        LoginPage loginPage = new LoginPage();
        loginPage.login();
        assertTrue(isTextPresent("Vitejte v elektronickem bankovnictvi"));

        driver.quit();
    }}
```

Kód 5 – LoginTest (autor)

V části *Kód 5 - LoginTest* se již inicializuje stránka *LoginPage*, na které je využita metoda *login()* bez parametru, tedy za použití výchozího uživatele definovaného ve třídě stránky. Nyní by se již prohlížeč měl nacházet uvnitř uživatelova účtu. Aby se to ovšem dalo ověřit, je potřeba kupříkladu ukázaný příkaz *assertTrue*, který ověřuje, zda je na stránce přítomen text, který se nachází jen uvnitř přihlášeného uživatele. Úplně na konci je příkaz *driver.quit()*, který zavírá webový prohlížeč.

5.2.2. Testovací skript 2 – Založení kontokorentu

K představení druhého testovacího skriptu je opět nejdříve rozebrán testovací případ, na základě kterého je tento testovací skript automatizován. Poté již následuje popis automatizace testovacího skriptu na vytvořeném příkladu. Popis bude prováděn na základě výše zmíněné metodiky MMSP.

5.2.2.1. Popis testovacího případu

Popis testovacího případu je proveden za pomoci užití šablony testovacího případu z metodiky MMSP dostupné na stránkách projektu. (Rejnková, nedatováno)

Tabulka 2 - testovací případ 2 (Rejnková, nedatováno)

ID		2			
Název testovacího případu		Založení kontokorentu			
Popis		Testovací případ testuje podání žádosti o kontokorent			
Vstupní podmínky		Jako vstupní podmínka je dostupnost přihlašovacích údajů libovolného uživatele elektronického bankovnictví			
Poznámky		U vybraného uživatele bude podaná žádost o kontokorent a před opětovným testem je potřeba v systému zrušit či zvolit jiného uživatele			
	Odkaz na testovací data	**Akce testera**	**Reakce systému**	**Výsledek (OK/Error)**	**Poznámky / odkaz na report chyby**
1.		Tester jde v internetovém prohlížeči Firefox na adresu www.internetovebankovnictvi.cz	Načte se přihlašovací obrazovka		
2.	Prihl_udaje.doc	Tester zadá přihlašovací údaje libovolného uživatele	Údaje jsou zadané do textových polí a heslo v podobě teček		
3.		Tester klikne na tlačítko přihlášení	Načte se úvodní obrazovka uživatele		
4.		Na úvodní stránce tester klikne na produkty a služby	Rozevře se menu		
5.		Tester vybere možnost „půjčka“	Zobrazí se možnosti půjček		
6.		Tester zvolí možnost „kde sjednat“	Zobrazí se možnosti sjednání		
7.		Tester zvolí možnost „online“	Načte se formuláře žádosti o kontokorent		

8.		Tester vyplní povinné údaje libovolnými daty a zvolí pokračovat	Aplikace přejde na druhou záložku žádosti		
9.		Tester opět vyplní libovolně povinné údaje, zaškrtne „souhlasím“ a zvolí pokračovat	Aplikace přejde na třetí záložku žádosti		
10.		Tester zaškrtne „souhlasím“ a zvolí dokončit	Žádost přejde do nové obrazovky		
11.		Tester ověří, zda se zobrazuje „Děkuji za trpělivost“ na konečné obrazovce	Text se nachází na stránce		
			Celkový výsledek		

5.2.2.2. Popis automatizovaného testovacího skriptu

U tohoto testovacího skriptu se již pracuje s prvky na několika různých stránkách webové aplikace, takže zde jsou definovány celkem 3 stránky s prvky.

První stránkou, která je opět definována je stránka *LoginPage,* která je již definována výše. Další definovanou stránkou je *HomePage*, tedy domovská stránka elektronického bankovnictví. Poslední použitou je již samotná stránka kontokorentu nazvaná *OverdraftPage*.

Nyní je nejdříve tedy ukázána stránka *HomePage.*

```
public class HomePage {

    @FindBy(xpath = "//span[contains(text(),'Produkty a sluzby')]")
    private WebElement productsAndServicesButton;

    @FindBy(xpath = "//a[contains(text(),'Pujcky')]")
    private WebElement loanButton;

    @FindBy(xpath = "//img[@alt='Kontokorent']")
    private WebElement kontokorentButton;

    @FindBy(xpath = "//a[contains(text(),'Kde sjednat')]")
    private WebElement whereToMakeButton;

    @FindBy(xpath = "//a[contains(@href,'/klient/forms/kontokorent')]")
    private WebElement makeOnlineButton;
```

Kód 6 – HomePage (autor)

Zde je vidět, jak jsou definovány prvky na této stránce. *Kód 6 – HomePage* ukazuje, že jsou k definici prvků použity rozličné parametry, které ovšem všechny splňují podmínku unikátnosti. Ve všech případech se jedná o práci s parametrem XPATH, který umožňuje kupříkladu i vyhledávání na základě částí textu. Jedná se o ideální parametr, když prvky nemají jednoznačně definovaný unikátní parametr typu ID.

Nejdříve je prvek, jimž je tlačítko produktů a služeb, které je definováno pomocí SPAN, což je jeden ze základních HTML tagů. Konkrétní prvek je určen pomocí metody *contains* s parametrem *text()*, kde je definováno, jaký text má obsahovat tento prvek.

Dalším prvkem je tlačítko půjčky, které je definováno za pomocí HTML tagu A, což je prvek, který někam odkazuje, zpravidla na hypertextový odkaz. V tomto případě je u prvku definován opět text stejně jako v předchozím případě pomocí metody *contains*.

Třetím prvkem na této stránce je již tlačítko kontokorentu, které využívá definice za pomocí tagu IMG, což je tag obrázků. V tomto případě je využíváno parametru *@alt*, který obsahuje alternativní text za obrázek.

Čtvrtý prvek se, co definice týče, nikterak neliší od prvku druhého a taktéž využívá HTML značky A a metody *contains*.

Poslední prvek na této stránce je definován taktéž tagem A a metodou *contains*, ale již nevyužívá parametru text, ale vyhledává na základě parametru *@href*, tedy hypertextového odkazu, kam tlačítko odkazuje.

Na příkladu těchto prvků si lze všimnout jednotného a jednoznačného pojmenování prvků tak, aby bylo na první pohled jasné, o jaké prvky se jedná, např. *loanButon*.

```
public void clickLoanButton() {
    loanButton.click();
}
public void clickOverdraftButton() {
    kontokorentButton.click();
}
public void clickWhereToMakeButton () {
    whereToMakeButton.click();
}
public void clickMakeOnlineButton() {
    makeOnlineButton.click();
}
```

```
    public void clickProductsAndServicesButton() {
        productsAndServicesButton.click();
}
```

Kód 7 – HomePage (autor)

Nyní jsou na kódu 7 ukázány metody, které jsou na stránce použity, ve všech případech se jedná o metody využívající metodu .click(), která, jak z názvu pramení, kliká na prvek.

Poslední stránkou, která je definována v tomto příkladu je stránka *OverdraftPage*. Jelikož se v tomto testovacím scénáři jedná o několika krokový formulář, prvků na této stránce je velké množství. Proto bude ukázka rozdělena po částech.

```
public class OverdraftPage {

    @FindBy(xpath = "//div[@location='amount']//input")
    private WebElement overdraftAmountField;

    @FindBy(xpath = "//div[@location='number']//input")
    private WebElement certificateField;

    @FindBy(xpath = "//div[@location='doesNotExpire']//input")
    private WebElement unlimitedValidityButton;

    @FindBy(xpath = "//span[contains(text(),'Pokracovat')]")
    private WebElement continueButton;

    @FindBy(xpath = "//div[@location='monthlyNetIncome']//input")
    private WebElement monthSalaryField;

    @FindBy(xpath = "//div[@location='monthlyInstallments']//input")
    private WebElement actualPaymentsField;
```

Kód 8 – OverdraftPage (autor)

Na ukázce těchto prvních prvků je vidět, že z důvodu neunikátních parametrů jakou jsou ID či CSS prvky, jsou zvoleny XPATH, které u prvních 3 prvků využívají HTML tagu DIV a jeho parametru *@location* a pomocí dvou lomítek za ním, odkazují na navazující prvek. Zde se jedná o prvek input, tedy jde o nalezení textového vstupu prvku. Pomocí lomítek lze takto znázorňovat posloupnost prvků i v několika krocích, například //div//span//a.

Dalším prvkem je tlačítko pokračovat, které je definováno za pomoci HTML značky SPAN a textu, který obsahuje. Na konci jsou opět prvky s parametrem *@location.*

```
    @FindBy(xpath = "//div[contains(@location, 'employerName')]//input")
    private WebElement employerNameField;

    @FindBy(xpath = "//div[contains(@location, 'employerIC')]//input")
    private WebElement employerICField;

    @FindBy(xpath = "//div[contains(@location, 'employerPhone')]//input")
    private WebElement employerPhoneField;

    @FindBy(xpath = "//div[contains(@location,
'employeePosition')]//div[contains(@class, 'button')]")
    private WebElement employeePositionDroplist;

    @FindBy(xpath = "//td[contains(@class, 'MenuItem')]//span[contains(text(),
'vedouci pracovnik na dobu neurcitou')]")
    private WebElement employeePositionDroplistChef;

    @FindBy(xpath = "//div[contains(@location, 'employedFrom')]//input")
    private WebElement employedFromField;

    @FindBy(xpath = "//div[contains(@location,
'maritalStatus')]//div[contains(@class, 'button')]")
    private WebElement maritalStatusDroplist;

    @FindBy(xpath = "//td")
    private WebElement maritalStatusDroplistFirstItem;
```

Kód 9 – OverdraftPage (autor)

U kódu 9 vidíme opět definované prvky převážně na základě vyhledání textového vstupu. Za zmínku stojí prvky *employeePositionDroplist* a *employeePositionDroplistChef*. U prvního z těchto prvků je pomocí vybrání lokace DIV a třídy následujícího tagu DIV vybráno tlačítko rozevíracího menu. U následujícího prvku je vybrána možnost z tohoto menu se značkou TD a následujícím HTML tagem SPAN obsahujícím určitý text. Tímto je vybrána v rozevíracím menu zvolená pozice pracovníka.

Poté je definován další prvek pomocí parametru *@location* a následné značky INPUT pro nalezení textového vstupu. Posledními prvky jsou opět rozevírací menu a vybraná položka z něj. U posledního

prvku je vidět, že je definována jen značka TD, nikoliv přesný text, který má obsahovat. Toto znamená, že si Selenium vybere první prvek s touto značkou nezávisle na obsahu textu. To lze provést v případě, že je testerovi jedno, jaký prvek bude vybrán. Někdy je potřeba splnit jen povinnost něco vybrat bez ohledu na to co.

```
    @FindBy(xpath = "//div[contains(@location, 'householdMembersNumber')]//input")
    private WebElement householdMembersNumberField;

    @FindBy(xpath = "//div[contains(@location,
'householdMembersWithoutIncome')]//input")
    private WebElement householdMembersWithoutIncomeField;

    @FindBy(xpath = "//div[contains(@location, 'education')]//div[contains(@class,
'button')]")
    private WebElement educationDroplist;

    @FindBy(xpath = "//span[contains(text(), 'VS')]")
    private WebElement educationDroplistUniversity;

    @FindBy(xpath = "//div[contains(@location,
'housingType')]//div[contains(@class, 'button')]")
    private WebElement housingTypeDroplist;

    @FindBy(xpath = "//span[contains(text(), 'dum / byt v os. nebo druzst.
vlastnictvi')]")
    private WebElement housingTypeOwnApartment;

    @FindBy(xpath = "//div[contains(@location,
'stayOnPreferredAddressFrom')]//input")
    private WebElement stayOnPreferredAddressFromField;
```

Kód 10 – OverdraftPage (autor)

Kód 10 opět ukazuje především volbu prvků pomocí lokace DIV tagu a následně značku INPUT. Jedná se tedy především o textová pole. Výjimku tvoří například opět rozevírací menu *educationDroplist,* kde je vybráno tlačítko za pomoci dvou navazujících prvků DIV a jejich parametrů *@location* a *@class*. U dalšího prvku je pomocí zvoleného textu značky SPAN zvoleno VŠ vzdělání jako položka tohoto menu. U dalšího rozevíracího menu *housingTypeDroplist* je situace téměř identická.

```
    @FindBy(xpath = "//div[contains(@location, 'agreementCheck')]//input")
    private WebElement agreementCheckBox;
```

```
@FindBy(xpath = "//div[contains(@location, 'form')]//input")
private WebElement agreementRequestCheckBox;

@FindBy(xpath = "//span[contains(text(),'Souhlasim')]")
private WebElement agreementButton;

@FindBy(xpath = "//span[contains(text(),'Dokoncit')]")
private WebElement finishButton;

@FindBy(xpath = "//span[contains(text(),'Souhlasim s uvedenymi parametry')]")
private WebElement agreementParametersConfirmationButton;
```

Kód 11 – OverdraftPage (autor)

Posledními definovanými prvky na této stránce, viz kód 11, jsou zaškrtávací políčko a dále tlačítka. U zaškrtávacího políčka je definován prvek pomocí lokace tagu DIV a vstupu, kterým nyní není textové pole, ale zaškrtávací políčko. Tlačítka jsou definována pomocí HTML značky SPAN a textu, který obsahují.

Nyní přicházejí na řadu metody definované na této stránce.

```
public void fillOverdraftAmountButton() {
    overdraftAmountField.sendKeys("35000");
}
public void fillCertificateNumber() {
    certificateField.sendKeys("123456789");
}
public void clickUnlimitedValidityButton() {
    unlimitedValidityButton.click();
}
public void clickContinueButton() {
    continueButton.click();
}
```

Kód 12 – OverdraftPage (autor)

Kód 12 ukazuje metody této stránky. U prvních dvou metod o naplnění textových polí textem pomocí metody *.sendKeys().* U dalších se jedná o kliknutí na prvky pomocí metody *.click().*

```
public void fillIncomeOutcome() {
    monthSalaryButton.sendKeys("30000");
    actualPaymentsButton.sendKeys("0");
```

```
        employerNameField.sendKeys("Josef Novak");
        employerICField.sendKeys("29146453");
        employerPhoneField.sendKeys("+420777777777");
        employeePositionDroplist.click();
        employeePositionDroplistChef.click();
        employedFromField.sendKeys("2000/01");
        continueButton.click();
    }
    public void fillHouseholdInfo() {
        maritalStatusDroplist.click();
        maritalStatusDroplistFirstItem.click();
        householdMembersNumberField.sendKeys("2");
        householdMembersWithoutIncomeField.sendKeys("0");
        educationDroplist.click();
        educationDroplistUniversity.click();
        housingTypeDroplist.click();
        housingTypeOwnApartment.click();
        stayOnPreferredAddressFromField.sendKeys("2008");
        continueButton.click();
    }
```

Kód 13 – OverdraftPage (autor)

U těchto metod, jak z názvu pramení, dochází k naplnění částí formuláře informacemi. Jak je na kódu 13 vidět, v případě metody *fillIncomeOutcome()* se jedná o doplnění údajů o příjmech a výdajích. Jsou zde užívány jen metody *.sendKeys()* k vyplnění textových polí a *.click()* ke kliknutí na prvky. U metody *fillHouseholdInfo()*, kde se vyplňují údaje o domácnosti, se jedná o identickou situaci.

```
    public void agreementCheck() {
        agreementCheckBox.click();
        continueButton.click();
    }
    public void clickAgreementButton() {
        agreementButton.click();
    }
    public void clickFinishButton() {
        finishButton.click();
    }}
```

Kód 14 – OverdraftPage (autor)

U posledních metod této třídy se jedná již jen o klikání na prvky, konkrétně na zaškrtávací pole a tlačítko dokončení. Kód 14 tedy definuje velmi jednoduché operace.

Poslední ukázkou kódu u tohoto testovacího případu je samotná testovací třída, která pracuje s výše definovanými metodami a stránkami.

```
public class OverdraftIsFilledAndAuthorisationIsMadeTest {

    public OverdraftIsFilledAndAuthorisationIsMadeTest() {
        super();
    }

    @Test
    public void OverdraftIsFilledAndAuthorisationIsMadeTest() {
        WebDriver driver = new FirefoxDriver();
        driver.get("www.internetovebankovnictvi.cz");

        LoginPage loginPage = new LoginPage();
        String USER = "ukazkovyuzivatel";
        String PASS = "ukazkoveheslo";
        loginPage.login(USER, PASS);
```

Kód 15 – OverdraftIsFilledAndAuthorisationIsMadeTest (autor)

Zde na kódu 15 je definována třída jako taková, inicializovaný WebDriver a jeho odkázání na webovou stránku pomocí metody *.get.* Dále je zde vytvořena třída stránky a dosazeny hodnoty za parametry uživatele a hesla. Poté je zde již metoda k přihlášení pomocí těchto parametrů. Jak je u předchozího skriptu definováno, jedná se tedy o použití konkrétního uživatele na základě nyní definovaných parametrů a ne na základě výchozího uživatele definovaného ve třídě stránky.

```
        HomePage homePage = new HomePage();
        homePage.clickProductsAndServicesButton();
        homePage.clickLoanButton();
        homePage.clickOverdraftButton();
        homePage.clickWhereToMakeButton();
        homePage.clickMakeOnlineButton();
```

Kód 16 – OverdraftIsFilledAndAuthorisationIsMadeTest (autor)

Kód 16 definuje třídu domovské stránky a užívá jejich metod. Konkrétně tedy klikání na tlačítka, kterými je testovací třída navigována k formuláři a stránce kontokorentu.

```
        OverdraftPage overdraftPage = new OverdraftPage();
        overdraftPage.fillOverdraftAmountField();
        overdraftPage.fillCertificateNumber();
        overdraftPage.clickUnlimitedValidityButton();
        overdraftPage.clickContinueButton();
        overdraftPage.fillIncomeOutcome();
        overdraftPage.fillHouseholdInfo();
        overdraftPage.agreementCheck();
        overdraftPage.clickContinueButton();
        overdraftPage.clickAgreementButton();
        overdraftPage.clickFinishButton();
        assertTrue(overdraftPage.contains("Dekujeme za trpelivost"));
```

Kód 17 – OverdraftIsFilledAndAuthorisationIsMadeTest (autor)

Na poslední části kódu jsou vidět již operace na třídě stránky kontokorentu. Jedná se o metody, které vyplňují části formuláře, a poslední metoda ověřuje, zda formulář proběhl v pořádku na základě textu, který by měla aplikace vypsat a jeho ověření pomocí metody *assertTrue.*

5.3. Spouštění testů za pomocí nástroje Selenium

Testy v Seleniu mohou být spouštěny více způsoby. Nejjednodušší možností je spouštění přímo v nástroji, ve kterém je kód napsán. V případě ukázky v této práci tedy v nástroji Eclipse. Tam se testy spouští stejně jako klasický soubor psaný v Javě, tedy příkazem *Run.* Toto spouštění má výhodu v tom, že k němu není potřeba dále již nic nastavovat. Nevýhodou na druhé straně je, že neposkytuje pokročilé reporty výsledků. Toto integrované testovací prostředí se nazývá JUnit. Toto prostředí je ovšem primárně pro jednotkové testy a není tedy nejvhodnějším řešením, pokud se bude spouštění provádět pravidelně. (ToolsQA, 2014)

Pokročilejším testovacím prostředím, které má větší množství funkcí, jako je generování HTML reportů, je TestNG. NG znamená *Next Generation,* tedy další generace. Oproti JUnit jsou zde především jednodušší anotace, které lze užívat při psaní kódu, seskupování testů a možnost paralelního testování. Základní anotace jsou *@BeforeTest*, *@AfterTest* a *@Test*. Jak je ze jmen patrné, udávají testovacímu rozhraní, kdy se části kódu pod nimi provádějí. Lze tedy provádět kód před testem, po testu či jako test samotný. Nezáleží, v jakém pořadí anotace ve třídě jsou. Před testem může být inicializován WebDriver, po testu naopak zavřen. Není to ovšem nutné. Nejzákladnější anotací pro samotné testování je *@Test*. V té může být bez problémů všechen kód. (Guru99, 2015)

5.3.1. Instalace TestNG

K provádění testů v rozhraní TestNG je zapotřebí ho do Eclipse nainstalovat. To lze udělat pomocí kliknutí na položku Menu *Help -> Install New Software.* Tam se již za Name dosadí *TestNG* a za Location *http://beust.com/eclipse.* Poté je již jen potřeba proklikat do konce instalace. (Guru99, 2015)

V Eclipse je dále zapotřebí přidat knihovny TestNG do projektu. Dialog přidání knihovny již nabízí možnost *TestNG,* kterou označíme. Po dokončení tohoto přidání je již vše potřebné v Eclipse nakonfigurováno. Při spouštění scénářů bude nyní již k dispozici možnost spustit pomocí TestNG.

TestNG umí již vytvářet přímo testovací třídy tohoto typu, kde předpřipraví již testovací anotace, které umí používat. Je stejně tak ale možné je ručně doplnit i do existujících scénářů.

5.3.2. Spouštění testů v TestNG

Testovací scénáře se spouští klasicky za pomoci položky v menu *Run -> As TestNG*. Poté se již otevřou dvě okna, jedno s konzolí a druhé s výsledky testů. Obě poskytují podobné informace, jen v případě druhého okna se tak děje za použití grafického rozhraní.

Mimo spouštění z programu je možné taky spouštět testy z příkazové řádky či pomocí plánovače úloh i automaticky každý den. K tomu poslouží *TestNG.xml* soubor, který se umístí do složky projektu.

Tento soubor má níže uvedený tvar, dostupný na stránkách projektu TestNG (TestNG, 2014):

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1" verbose="1" >
  <test name="Regression1"    >
    <packages>
      <package name="test.sample" />
    </packages>
  </test>
</suite>
```

Obrázek 2 - vzorový TestNG.xml (autor)

Nyní se již TestNG může spustit z příkazové řádky a to za pomoci níže uvedeného příkazu, který naviguje do adresáře projektu a k souboru TestNG.xml (TestNG, 2014):

set ProjectPath=C:\Selenium\Selenium_tests\UkazkovyProjekt

*set classpath=%ProjectPath%\bin;%ProjectPath%\Lib**

java org.testng.TestNG %ProjectPath%\TestNG.xml

Pomocí těchto příkazů lze již samozřejmě vytvořit soubor *.bat,* který lze pomocí plánovače úloh spouštět automaticky a pravidelně. Odpadá tedy manuální spouštění testů.

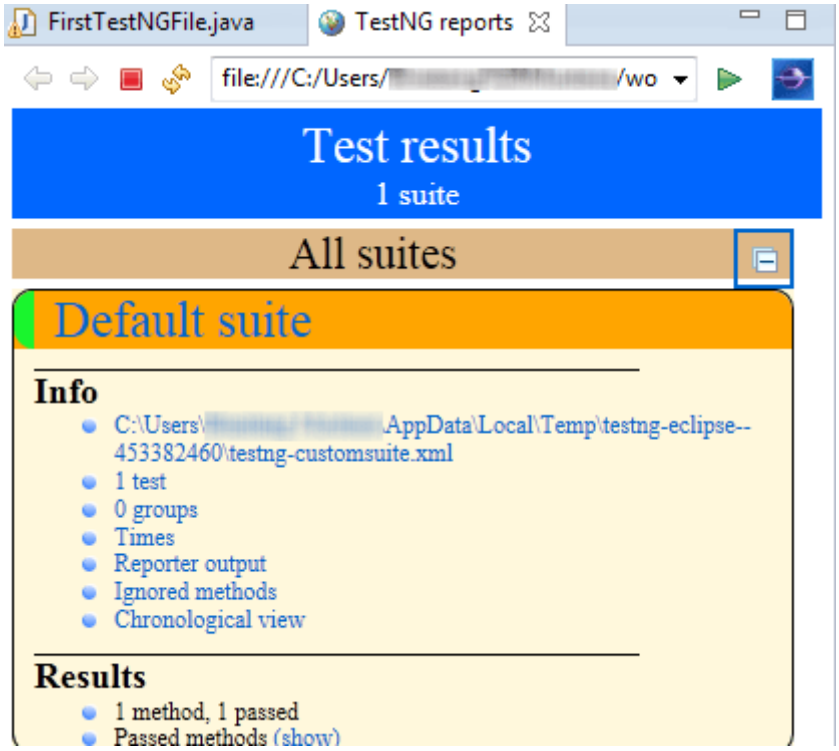
5.4. Závěry a výsledky testů

Při použití testovacího rozhraní TestNG se otevírají bohaté možnosti reportingu, který jinak není ve výchozím prostředí podporovaný mimo ukládání základních výsledků testů.

TestNG umožňuje generování reportů v XML i uživatelsky přívětivějším HTML formátu, které se generují automaticky v nově vytvořené složce *test-output* v projektu.

Tester se k takovému reportu dostane, pokud po spuštění testů podívá do této nově vytvořené složky a otevře soubor *index.html.* Tento HTML soubor obsahuje report s výsledky naposled provedeného testovacího běhu. Stejně tak se v této složce nachází report ve formátu XML *testing-results.xml* (Guru99, 2015)

V těchto souborech jsou vidět veškeré informace o běhu testu. Jedná se především o to, kde se nachází defekty, kolik testů a metod prošlo, či doba provádění těchto testů. Nacházejí se zde také informace o jednotlivých krocích, které byly prováděny.



Obrázek 3 - ukázkový report TestNG v HTML (TestNG, 2015)

5.5. Srovnání kapacity nákladů pro manuální a automatizované testování

Při rozhodování zda automatizovat či nikoliv vstupuje do hry mnoho rozličných faktorů. Manuální testování s sebou nenese počáteční náklady, ovšem doba provádění je delší než v případě automatizovaných skriptů. Automatizované testování s sebou ovšem nese větší náklady a je tedy třeba ověřit, zda se vynaložené náklady navrátí během času či nikoliv.

Konkrétně pro automatizaci webových aplikací pomocí nástroje Selenium s sebou automatizace nenese žádné náklady spojené s nákupem softwaru. Kvalifikovaná síla, tedy tester s potřebnými znalostmi, je ovšem zpravidla dražší než manuální tester. Taktéž tvorba automatizovaných skriptů či zavedení automatizovaného testovacího prostředí s sebou nese poměrně velkou časovou zátěž v porovnání s testováním manuálním.

Návratnost vynaložených investic záleží na tom, jak náročné je testovací scénář automatizovat a jak často je poté prováděno samotné testování. Veškeré ceny i časová zátěž jednotlivých skriptů jsou individuální a je zapotřebí důkladná analýza, která vyhodnotí, zda se automatizace vyplatí.

Obecně platí, že pokud je automatizace možná a funkcionalita se musí pravidelně testovat, investice do automatizace se časem navrátí. U funkcionalit, které se testují sporadicky, se automatizace obvykle nedoporučuje. Právě z důvodu vhodnosti automatizace u často testovaných scénářů se automatizace doporučuje především na regresních testech.

Konkrétně na příkladu testovacího příkladu v kapitole 6, jsem níže provedl ukázkovou kalkulaci. Tato kalkulace slouží především k ilustraci, jelikož konkrétní ceny a časové náklady jsou individuální.

U testovacího scénáře se tedy testuje přihlašování do elektronického bankovnictví. K možné kalkulaci je nutné si definovat některé základní věci. Banka testuje toto přihlašování přibližně 30 krát ročně. V testu se kontroluje přihlášení a text zobrazený v elektronickém bankovnictví.

Manuální tester s hodinovou sazbou 100 korun na hodinu by tento scénář prováděl přibližně 2 minuty, tj. 1 hodinu ročně. Náklad by tedy byl 100 korun ročně.

Nyní se neberou v potaz počáteční náklady k zavedení automatizace, které mohou být přibližně jeden pracovní den. U tohoto velmi jednoduchého scénáře by automatizovanému testerovi, který má hodinovou sazbu 150 korun na hodinu, stačilo k sepsání testu v Javě za pomocí Selenia půl hodiny. Jednorázový náklad by tedy byl 75 korun.

Na ilustrativním příkladu je tedy vidět, že by se automatizace testovacího případu vyplatila. Samozřejmě by se podobné předběžné kalkulace musely provést i u ostatních scénářů, aby se zjistilo,

zda se navrátí i počáteční investice do zavedení testovacího prostředí, která by v tomto případě činila 8 hodin, tedy 1200 korun.

Také záleží na frekvenci testování, jak bylo zmíněno výše. Kdyby se v tomto případě provádělo testování 10 krát ročně, již by byla počáteční investice do tohoto scénáře vyšší než roční náklady na manuální otestování (přibližně 33 korun). Návratnost u tohoto konkrétního příkladu by tedy byla již přes 2 roky. Neberou se také v potaz náklady spojené s údržbou testů, které u některých scénářů mohou vstupovat také do kalkulace.

6. Závěr

Hlavním cílem bakalářské práce byl popis automatizace testování softwaru pomocí nástroje Selenium WebDriver, vytvoření praktického příkladu a popisu postupu vytváření testů. Tohoto bylo docíleno především v kapitole 5, kde byl postupný popis veškerých kroků u konkrétního skriptu. Dá se zde nalézt jak popis tvorby skriptů v elektronickém bankovnictví, tak spouštění skriptů i stručné srovnání nákladů mezi manuálním a automatizovaným testováním. V kapitole 4 se dají nalézt informace o krocích automatizace obecně. Kompletní praktické příklady jsou také přiloženy v přílohách.

Dílčím cílem bylo představení nástroje Selenium, čehož bylo docíleno v kapitole 4, kde mimo představení nástroje Selenium byl představen i způsob automatizace pomocí tohoto nástroje, identifikace jednotlivých prvků a operace v rámci webové aplikace obecně.

Dalším dílčím cílem bylo shrnutí možností testování softwaru a aplikace testování v elektronickém bankovnictví. Obou těchto částí bylo docíleno v kapitole 3. Zde byly nejdříve ukázány možnosti testování a poté zaměření specificky na elektronické bankovnictví.

Mezi přínosy práce tedy patří představení možností automatizovaného testování pomocí nástroje Selenium, jmenovitě v oblasti elektronického bankovnictví. Mezi přínosy dále patří možnost použití popsaných postupů automatizace i k automatizaci jiných oblastí testování softwaru.

V této bakalářské práci jsem se zabýval automatizací testování softwaru pomocí nástroje Selenium. Tato práce popisuje celý proces od představení nástroje, pomocí kterého se automatizace provádí, jeho možností až po samotný proces oné automatizace.

Nejdříve je ukázána samotná instalace nezbytná k umožnění tvorby těchto testovacích případů. Poté je od tvorby projektu až po popis možností automatizace ukázáno, co nástroj dokáže. Celý postup tvorby je na stránkách této práce zdokumentovaný a to i včetně možné realizace samotných testů i reportingu jejich výsledků.

Téma automatizace testování softwaru je poměrně široké a v práci jsem vyzdvihl možnosti tohoto řešení, jeho omezení i širší představení problematiky testování softwaru obecně. Z důvodu požadavků na udržení kvality softwaru, je toto téma ve světě IT aktuální a relevantní.

7. Terminologický slovník

Tabulka 3 - terminologický slovník (autor)

Termín	Zkratka	Význam (Zdroj)
Elektronické bankovnictví	IB (internet banking)	Webová aplikace sloužící k vyřízení bankovních služeb (autor)
Testování softwaru	Testování SW	Proces zkoušení funkčnosti softwaru na základě testovacích případů (autor)
Selenium WebDriver		Rozhraní sloužící k tvorbě a provádění automatizovaných testů webových aplikací (autor)
Java		Populární programovací jazyk (autor)
Software	SW	Komplex programů (autor)
Automatizace		Převod manuálních kroků na automatické (autor)
Metodika MMSP		Metodika pro menší softwarové projekty vytvořena lokalizací a customizací metodiky OpenUP (MMSP, 2015)
Tester		Člověk, který se zabývá testováním softwaru (autor)
Testovací případ	TC (test case)	Konkrétní akce prováděné se softwarem a jejich očekávané výsledky (Testování softwaru, 2015)

8. Bibliografie

Avasarala, S., 2014. *Selenium WebDriver practical guide interactively automate web applications using Selenium WebDriver.* Birmingham: Packt Pub.

Bezpečný internet.cz, 2015. *Bezpečný internet | Bezpečnost internetového bankovnictví obecně.* [Online]
Available at: http://www.bezpecnyinternet.cz/pokrocily/internetove-bankovnictvi/bezpecnost.aspx
[Přístup získán 19 Duben 2015].

Google Code, 2015. *Overview.* [Online]
Available at: http://selenium.googlecode.com/git/docs/api/java/index.html
[Přístup získán 19 Duben 2015].

Gundecha, U., 2012. *Selenium 2 Cookbook.* 2. editor Birmingham: Packt Publishing.

Guru99, 2015. *All About Manual Testing.* [Online]
Available at: http://www.guru99.com/manual-testing.html
[Přístup získán 19 Duben 2015].

Guru99, 2015. *How TestNG makes Selenium tests easier.* [Online]
Available at: http://www.guru99.com/all-about-testng-and-selenium.html
[Přístup získán 28 Duben 2015].

Húska, J., 2012. *Automated Testing of the Component-based Web Application User Interfaces,* Brno: Masarykova Univerzita.

Chmurčiak, D., 2013. *Automatické testování webových aplikací,* Brno: Masarykova Univerzita.

Mazoch, B., 2012. *Funkční testování webových aplikací,* Brno: Masarykova Univerzita.

Microsoft, 2015. *Regression Testing.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/aa292167(VS.71).aspx
[Přístup získán 28 Duben 2015].

MMSP, 2015. *Metodika MMSP.* [Online]
Available at: mmsp.czweb.org
[Přístup získán 30 Duben 2015].

Pietrik, M., 2012. *Automatizované testování webových aplikací,* Brno: Masarykova Univerzita.

Rejnková, P., 2011. *Lokalizace a přizpůsobení metodiky OpenUP,* Praha: Vysoká škola ekonomická v Praze.

Rejnková, P., nedatováno *Metodika MMSP.* [Online]
Available at:
http://mmsp.czweb.org/MMSP/guidances/templates/testovaci_pripad_sablona_C67B8B99.html
[Přístup získán 30 Duben 2015].

Roudenský, P. & Havlíčková, A., 2013. *Řízení kvality softwaru: průvodce testováním.* 1. editor Brno: Computer Press.

SeleniumHQ, 2015. *Importing Sel2.0 Project into Eclipse using Maven - Selenium Documentation.* [Online]
Available at: http://docs.seleniumhq.org/docs/appendix_installing_java_driver_Sel20_via_maven.jsp
[Přístup získán 19 Duben 2015].

SeleniumHQ, 2015. *Maven Information.* [Online]
Available at: http://docs.seleniumhq.org/download/maven.jsp
[Přístup získán 19 Duben 2015].

SeleniumHQ, 2015. *Selenium - Web Browser Automation.* [Online]
Available at: http://docs.seleniumhq.org/docs/
[Přístup získán 19 Duben 2015].

SeleniumHQ, 2015. *Selenium - Web Browser Automation.* [Online]
Available at: http://www.seleniumhq.org/
[Přístup získán 19 Duben 2015].

SeleniumHQ, 2015. *Selenium WebDriver - Selenium Documentation.* [Online]
Available at: http://docs.seleniumhq.org/docs/03_webdriver.jsp
[Přístup získán 19 Duben 2015].

Shah, M., 2015. *Selenium WebDriver Read PDF Content - Testing Diaries.* [Online]
Available at: http://www.testingdiaries.com/selenium-webdriver-read-pdf-content/
[Přístup získán 19 Duben 2015].

SlideShare, 2013. *Testing banking apps.* [Online]
Available at: http://www.slideshare.net/chrix2/testing-banking-apps
[Přístup získán 19 Duben 2015].

Software Testing Class, 2015. *What is Manual Testing.* [Online]
Available at: http://www.softwaretestingclass.com/what-is-manual-testing/
[Přístup získán 19 Duben 2015].

Software Testing Help, 2015. *How To Test Banking Applications - Software Testing Help.* [Online]
Available at: http://www.softwaretestinghelp.com/testing-banking-applications/
[Přístup získán 19 Duben 2015].

Stack OverFlow, 2015. *Stack OverFlow.* [Online]
Available at: http://stackoverflow.com/
[Přístup získán 19 Duben 2015].

SW Testování, 2015. *SW Testování.* [Online]
Available at: www.swtestovani.cz/index.php?option=com_content&view=article&id=22:funkni-vs-nefunkni-testovani
[Přístup získán 28 Duben 2015].

Testing Tools, 2015. *Test Automation Tools - Popular Automated Testing Tools and Software.* [Online]
Available at: http://www.testingtools.com/test-automation/
[Přístup získán 19 Duben 2015].

TestNG, 2014. *TestNG.* [Online]
Available at: http://testng.org/doc/documentation-main.html#testng-xml
[Přístup získán 28 Duben 2015].

TestNG, 2014. *TestNG.* [Online]
Available at: http://testng.org/doc/documentation-main.html
[Přístup získán 28 Duben 2015].

TestNG, 2015. *TestNG.* [Online]
Available at: http://testng.org/doc/documentation-main.html#test-results
[Přístup získán 30 Duben 2015].

Testování softwaru, 2015. *Druhy, Typy A Kategorie Testů.* [Online]
Available at: http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu/
[Přístup získán 19 Duben 2015].

Testování softwaru, 2015. *Testování softwaru.* [Online]
Available at: http://testovanisoftwaru.cz/dokumentace-v-testovani/test-case/
[Přístup získán 30 Duben 2015].

The Apache Software Foundation, 2015. *Maven - Welcome to Apache Maven.* [Online]
Available at: http://maven.apache.org
[Přístup získán 30 Duben 2015].

ToolsQA, 2014. *Selenium WebDriver Tutorials | Junit Test with Selenium WebDriver - Selenium WebDriver Tutorials.* [Online]
Available at: http://www.toolsqa.com/java/junit-framework/junit-test-selenium-webdriver/
[Přístup získán 28 Duben 2015].

ToolsQA, 2015. *Selenium WebDriver Tutorials.* [Online]
Available at: http://www.toolsqa.com/selenium-webdriver/
[Přístup získán 19 Duben 2015].

9. Seznam tabulek, obrázků a kódů

10. Přílohy

Testovací skript 1 – Přihlášení do elektronického bankovnictví (autor)

Třída se stránkou

```
public class LoginPage {
    @FindBy(css = ".buttonLogin")
    private WebElement loginButton;
    @FindBy(id = "username")
    private WebElement usernameField;
    @FindBy(id = "password-view")
    private WebElement passField;
    public void setUsername(String username) {
        usernameField.sendKeys(username);
    }
    public void setPassword(String password) {
        passField.sendKeys(password);
    }
    public void login() {
        this.setUsername("testovaciuzivatel1");
        this.setPassword("heslo1");
        this.loginButton.click();
    }
    public void login(String user, String pass) {
        this.setUsername(user);
        this.setPassword(pass);
        this.loginButton.click();
      }
}
```

Třída s testovacím skriptem

```
public class LoginTest {
    public LoginTest() {
        super();
    }
    @Test
    public void loginTest() {
      WebDriver driver = new FirefoxDriver();
        driver.get("www.internetovebankovnictvi.cz");
        LoginPage loginPage = new LoginPage();
        loginPage.login();
```

```
        assertTrue(isTextPresent("Vitejte v elektronickem bankovnictvi"));
        driver.quit();
    }
}
```

Testovací skript 2 – založení kontokorentu (autor)

Třída se stránkou 1

```
public class HomePage {
    @FindBy(xpath = "//span[contains(text(),'Produkty a sluzby')]")
    private WebElement productsAndServicesButton;
    @FindBy(xpath = "//a[contains(text(),'Pujcky')]")
    private WebElement loanButton;
    @FindBy(xpath = "//img[@alt='Kontokorent']")
    private WebElement kontokorentButton;
    @FindBy(xpath = "//a[contains(text(),'Kde sjednat')]")
    private WebElement whereToMakeButton;
    @FindBy(xpath = "//a[contains(@href, '/klient/forms/FRM-
kontokorent?produkt=PPU&s=private')]")
    private WebElement makeOnlineButton;
    public void clickLoanButton() {
        loanButton.click();
    }
    public void clickOverdraftButton() {
        kontokorentButton.click();
    }
    public void clickWhereToMakeButton () {
        whereToMakeButton.click();
    }
    public void clickMakeOnlineButton() {
        makeOnlineButton.click();
    }
    public void clickProductsAndServicesButton() {
        productsAndServicesButton.click();
}
```

Třída se stránkou 2

```
public class OverdraftPage {
    @FindBy(xpath = "//div[@location='amount']//input")
```

```
    private WebElement overdraftAmountField;
    @FindBy(xpath = "//div[@location='number']//input")
    private WebElement certificateField;
    @FindBy(xpath = "//div[@location='doesNotExpire']//input")
    private WebElement unlimitedValidityButton;
    @FindBy(xpath = "//span[contains(.,'Pokracovat')]")
    private WebElement continueButton;
    @FindBy(xpath = "//div[@location='monthlyNetIncome']//input")
    private WebElement monthSalaryField;
    @FindBy(xpath = "//div[@location='monthlyInstallments']//input")
    private WebElement actualPaymentsField;
    @FindBy(xpath = "//div[contains(@location, 'employerName')]//input")
    private WebElement employerNameField;
    @FindBy(xpath = "//div[contains(@location, 'employerIC')]//input")
    private WebElement employerICField;
    @FindBy(xpath = "//div[contains(@location, 'employerPhone')]//input")
    private WebElement employerPhoneField;
    @FindBy(xpath = "//div[contains(@location,
'employeePosition')]//div[contains(@class, 'button')]")
    private WebElement employeePositionDroplist;
    @FindBy(xpath = "//td[contains(@class, 'MenuItem')]//span[contains(text(),
'vedouci pracovnik na dobu neurcitou')]")
    private WebElement employeePositionDroplistChef;
    @FindBy(xpath = "//div[contains(@location, 'employedFrom')]//input")
    private WebElement employedFromField;
    @FindBy(xpath = "//div[contains(@location,
'maritalStatus')]//div[contains(@class, 'button')]")
    private WebElement maritalStatusDroplist;
    @FindBy(xpath = "//td")
    private WebElement maritalStatusDroplistFirstItem;
    @FindBy(xpath = "//div[contains(@location, 'householdMembersNumber')]//input")
    private WebElement householdMembersNumberField;
    @FindBy(xpath = "//div[contains(@location,
'householdMembersWithoutIncome')]//input")
    private WebElement householdMembersWithoutIncomeField;
    @FindBy(xpath = "//div[contains(@location, 'education')]//div[contains(@class,
'button')]")
    private WebElement educationDroplist;
    @FindBy(xpath = "//span[contains(text(), 'VS')]")
    private WebElement educationDroplistUniversity;
```

```
    @FindBy(xpath = "//div[contains(@location,
'housingType')]//div[contains(@class, 'button')]")
    private WebElement housingTypeDroplist;
    @FindBy(xpath = "//span[contains(text(), 'dum / byt v os. nebo druzst.
vlastnictvi')]")
    private WebElement housingTypeOwnApartment;
    @FindBy(xpath = "//div[contains(@location,
'stayOnPreferredAddressFrom')]//input")
    private WebElement stayOnPreferredAddressFromField;
    @FindBy(xpath = "//div[contains(@location, 'agreementCheck')]//input")
    private WebElement agreementCheckBox;
    @FindBy(xpath = "//div[contains(@location, 'form')]//input")
    private WebElement agreementRequestCheckBox;
    @FindBy(xpath = "//span[contains(.,'Souhlasim')]")
    private WebElement agreementButton;
    @FindBy(xpath = "//span[contains(.,'Dokoncit')]")
    private WebElement finishButton;
    @FindBy(xpath = "//span[contains(.,'Souhlasim s uvedenymi parametry')]")
    private WebElement agreementParametersConfirmationButton;
    public void fillOverdraftAmountField() {
        overdraftAmountField.sendKeys("35000");
    }
    public void fillCertificateNumber() {
        certificateField.sendKeys("123456789");
    }
    public void clickUnlimitedValidityButton() {
        unlimitedValidityButton.click();
    }
    public void clickContinueButton() {
        continueButton.click();
    }
    public void fillIncomeOutcome() {
        monthSalaryButton.sendKeys("30000");
        actualPaymentsButton.sendKeys("0");
        employerNameField.sendKeys("Josef Novak");
        employerICField.sendKeys("29146453");
        employerPhoneField.sendKeys("+420777777777");
        employeePositionDroplist.click();
        employeePositionDroplistChef.click();
        employedFromField.sendKeys("2000/01");
```

```
        continueButton.click();
    }
    public void fillHouseholdInfo() {
        maritalStatusDroplist.click();
        maritalStatusDroplistFirstItem.click();
        householdMembersNumberField.sendKeys("2");
        householdMembersWithoutIncomeField.sendKeys("0");
        educationDroplist.click();
        educationDroplistUniversity.click();
        housingTypeDroplist.click();
        housingTypeOwnApartment.click();
        stayOnPreferredAddressFromField.sendKeys("2008");
        continueButton.click();
    }
    public void agreementCheck() {
        agreementCheckBox.click();
        continueButton.click();
    }
    public void clickAgreementButton() {
        agreementButton.click();
    }
    public void clickFinishButton() {
        finishButton.click();
    }
}
```

Třída s testovacím skriptem

```
public class OverdraftIsFilledAndAuthorisationIsMadeTest {
    public OverdraftIsFilledAndAuthorisationIsMadeTest() {
        super();
    }
    @Test
    public void OverdraftIsFilledAndAuthorisationIsMadeTest() {
        WebDriver driver = new FirefoxDriver();
        driver.get("www.internetovebankovnictvi.cz");
        LoginPage loginPage = new LoginPage();
        String USER = "ukazkovyuzivatel";
        String PASS = "ukazkoveheslo";
        loginPage.login(USER, PASS);
```

```
        HomePage homePage = new HomePage();
        homePage.clickProductsAndServicesButton();
        homePage.clickLoanButton();
        homePage.clickOverdraftButton();
        homePage.clickWhereToMakeButton();
        homePage.clickMakeOnlineButton();
        OverdraftPage overdraftPage = new OverdraftPage();
        overdraftPage.fillOverdraftAmountField();
        overdraftPage.fillCertificateNumber();
        overdraftPage.clickUnlimitedValidityButton();
        overdraftPage.clickContinueButton();
        overdraftPage.fillIncomeOutcome();
        overdraftPage.fillHouseholdInfo();
        overdraftPage.agreementCheck();
        overdraftPage.clickContinueButton();
        overdraftPage.clickAgreementButton();
        overdraftPage.clickFinishButton();
        assertTrue(overdraftPage.contains("Dekujeme za trpelivost"));
    }
}
```