Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Implementace automatického testování softwaru

Veronika Riegerová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Veronika Riegerová

Informatika

Název práce

Implementace automatického testování softwaru

Název anglicky

Implementation of automated software testing

Cíle práce

Hlavním cílem bakalářské práce je analýza možnosti automatického testování softwaru, výběr vhodného nástroje pro testování ve zvoleném scénáři a jeho následná implementace.

Dílčí cíle bakalářské práce:
- studium a analýza dostupných odborných informačních zdroju zaměřená na oblast automatického testování softwaru
- definice testovacího scénáře a specifikace požadavků na testování
- analýza vybraných nástrojů pro testování a výběr optimálního postupu
- zhodnocení zvoleného nástroje a postupu jeho zavedení

Metodika

Teoretická část práce je založena na studiu odborné literatury a relevantních elektronických zdrojů dostupných na internetu. Praktická část práce spočívá v definici testovacích scénářů, analýze a srovnání dostupných testovacích nástrojů a výběru vhodné varianty implementace vzhledem k požadavkům daného scénaře. Na základě syntézy poznatků teoretické části a výsledků praktické části budou formulovány závěry práce.

Doporučený rozsah práce

40-50

Klíčová slova

automatizace testování, testovací scénář, testovací případy, manuální testování, vývoj softwaru

---

Doporučené zdroje informací

Barnum, Carol M. a Dana Chisnell. Usability testing essentials: ready, set– test. 2nd ed. Burlington, MA: Morgan Kaufmann Publishers, c2011. ISBN 978-0123750921.

Bureš Miroslav, Renda Miroslav, Doležel Michal a kolektiv. Efektivní testování softwaru Praha: Grada Publishing. 2016, 229s., ISBN 978-80-247-5594-6.

Krishna Rungta. Learn Selenium in 1 Day: Definitive Guide to Learn Selenium for Beginners. 2017, 504s., ISBN B071YTMYXC.

Krishna Rungta. Learn Testing in 1 Day: Definitive Guide to Learn Software Testing for Beginners. 2017, 363s., ISBN 1522070443.

Patton Ron. Testování softwaru. Praha: Computer Press, 2002, 313 s., ISBN 80-7226-636-5.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Jan Pavlík

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 16. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2022

Čestné prohlášení

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

V Praze dne 15.3.2022 _____

Poděkování

Ráda bych touto cestou poděkovala svému vedoucímu práce Ing. Janu Pavlíkovi za jeho pomoc při tvorbě této práce, za ochotu a věnovaný čas.

Implementace automatického testování softwaru

Abstrakt

Bakalářská práce je zaměřená na oblast automatické testování softwaru. Testování je nedílnou fází životního cyklu vývoje softwaru. Hlavním cílem práce je analýza možností vybraných nástrojů určené pro tvorbu automatizovaného testování, výběr vhodného nástroje ve zvoleném scénáři a jeho následná implementace. Práce vychází ze studie poznatků teoretické části a z mojí tříleté praxe v oblasti testování.

Na základě poznatků teoretické části a výsledků praktické části jsou formulovány závěry práce.

Klíčová slova: automatizace testování, testovací scénář, testovací případy, manuální testování, vývoj softwaru

Implementation of automated software testing

Abstract

The bachelor thesis is focused on automatic software testing. Software testing is an integral phase in Software Development Life Cycle process. The main goal of this work is to analyze the possibilities of selected tools for the creation of automated testing, selection of a suitable tool in the selected scenario and its subsequent implementation. The work is based on a study of the knowledge of the theoretical part and my three years of experience in testing.

Based on the knowledge of the theoretical part and the results of the practical part, the conclusions of the work are formulated.

Keywords: testing automation, test scenario, test cases, manual testing, software development

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Kvalita softwaru určuje úspěch jakéhokoli softwarového produktu. Implementovat produkt s minimální nebo žádnou chybou je velmi obtížné, a proto vznikla myšlenka testování softwaru. Testování se stalo nezbytnou a velice rozsáhlou činností životního cyklu vývoje softwaru. Dnes pravděpodobně neexistuje žádná společnost, která by ve svém vývoji softwaru nezahrnula část testování. Pokud je testování v průběhu vývoje podceňováno a bráno za nadbytečnou činnost, může to mít fatální dopad na kvalitu softwaru a spokojenost koncových zákazníků.

Cílem testování je zajištění požadované kvality a odhalení co největšího množství vzniklých chyb za krátké časové období. Pokud je testování provedeno správně, zlepšuje výkon a přesnost softwarového systému. Identifikace závad a provedení oprav před vydáním softwaru do produkce pomáhá podnikům ušetřit dodatečné náklady na údržbu.

Všechny činnosti testování se provádí dvěma způsoby: manuálním a automatizovaným. Manuální testování se provádí ručně do softwarové aplikace podle testovacího plánu. V automatizovaném testování lze testovací činnost provádět pomocí testovacích nástrojů, aniž by bylo nutné zkoumat různé části aplikací ručně. Dříve bylo testování prováděno pouze manuálně, ale lidská chybovost je běžná a některé chyby se dají snadno přehlédnout. Některé programy a aplikace je téměř nemožné testovat manuálně. Naopak určité prvky grafického uživatelského prostředí není výhodné automatizovat a je potřeba dát při testování přednost lidskému uvažování a intuici. Tvorba automatizovaných testů je časově velice náročná a může být i velice nákladná. Pokud víme, v jakém případě se vyplatí dané testy automatizovat, ušetří nám to spoustu času a práce. Díky vývoji automatizovaného testování je proces testování mnohem rychlejší a přesnější. Jeho základem jsou nástroje používané k tvorbě a provádění samotného testování.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je analýza možností automatického testování softwaru, výběr vhodně zvoleného nástroje pro testování ve zvoleném scénáři a jeho následná implementace.

Dílčí cíle bakalářské práce:

- studium a analýza dostupných odborných informačních zdrojů zaměřená na oblast automatického testování softwaru
- definice testovacího scénáře a specifikace požadavků pro testování
- analýza vybraných nástrojů pro testování a výběr optimálního postupu
- zhodnocení zvoleného scénáře a postupu jeho zavedení

2.2 Metodika

Teoretická část práce je založena na studiu odborné literatury a relevantních elektronických zdrojů dostupných na internetu. Praktická část práce spočívá v definici testovacích scénářů, analýze a srovnání dostupných testovacích nástrojů a výběru vhodné varianty implementace vzhledem k požadavkům daného scénáře. Na základě syntézy poznatků teoretické části a výsledků praktické části budou formulovány závěry práce.

3 Teoretická východiska

3.1 Testování softwaru

Vývoj softwaru se neustále mění, dochází ke změnám požadavků na jeho funkcionalitu, samotného zdrojového kódu a dalších jeho součástí. Se změnami narůstá procento vzniklých chyb, které by mohly negativně ovlivnit kvalitu daného softwaru. Hlavním cílem testování je zajištění požadované kvality softwaru v průběhu jeho vývoje a odhalení co nejvíce softwarových chyb neboli nesrovnalostí oproti očekávaným výsledkům za co nejkratší časové období. Osoba, která testuje kvalitu softwaru, se nazývá *Tester*.

3.2 Chyba

Pojem *chyba* (někdy též bug) lze chápat a definovat několika způsoby. Každý uživatel má jiná očekávání, pohled a názor. Podle (Patton, 2002) lze definovat *softwarovou chybu*, pokud je splněna jedna z následujících podmínek:

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

Softwarové chyby jsou nevyhnutelnou součástí životního cyklu vývoje softwaru. Během procesu testování se testeři setkávají s různými druhy chyb. Pokud tyto chyby nejsou odstraněny v raných fázích, naruší tím další část procesu vývoje a jejich oprava bude mnohem nákladnější a časově náročnější. Nejběžnější typy softwarových chyb jsou (BrowserStack, 2021):

Funkční chyby – jsou spojeny s funkčností konkrétního softwaru. Představují širokou škálu chyb a mohou sahat od základních funkcí, jako jsou jednoduchá tlačítka, na která nelze kliknout, až po nemožnost používat hlavní funkce softwaru. Pokud funkce nefunguje, jak má, jedná se o funkční chybu.

Logické chyby – tyto chyby mohou způsobit neočekávané chování softwaru nebo dokonce i náhlé zhroucení. Logické chyby se primárně objevují z důvodu špatně naprogramovaného kódu, který způsobí nefunkčnost programu.

Chyby na úrovni jednotky – jsou velmi běžné a obvykle snadno opravitelné. Jakmile je kód naprogramován, vývojáři obvykle provádějí jednotkové testování. Otestují části kódu, aby se ujistili, zda fungují tak, jak mají. Během tohoto procesu se odhalují chyby na úrovni jednotek, jako jsou chyby ve výpočtech a základní logické chyby.

Integrační chyby – chyby integrace se primárně objevují, pokud dvě nebo více jednotek kódu napsaných různými vývojáři selhávají ve vzájemné interakci. Kompatibilita mezi dvěma nebo více komponentami není úplná. Takové chyby se obecně obtížněji opravují, protože je nutné prozkoumat více navzájem propojených softwarových systémů.

3.3 Kvalita softwaru

K určení kvality softwaru je zapotřebí stanovit, jaké vlastnosti budou brány v úvahu při hodnocení softwarového produktu. Pro stanovení, z jakých pohledů lze hodnotit kvalitu, existují normy. Norma ISO/IEC 25010, která je popsána podle (ISO 25000, 2021), obsahuje osm pozorovaných charakteristik.

1. **Funkčnost** – Stanovuje, do jaké míry software pracuje správně za předem stanovených podmínek definovaných ve specifikaci.
2. **Výkon** – Hodnotí dobu odezvy a zpracování při plnění velkého množství požadavků v poměru k množství využitých zdrojů.
3. **Kompatibilita** – Schopnost spolupracovat, komunikovat a sdílet společné prostředí s více systémy a komponenty.
4. **Použitelnost** – Stanovuje, jak je systém z pohledu uživatele efektivní, přehledný, náročný na ovládání a dosažení cíle. Především, zda je vhodný pro splnění jejich potřeb.
5. **Spolehlivost** – Sleduje, zda systém pracuje správně za normálního provozu, i za všech okolností, které mohou nastat. Určuje míru schopnosti reagovat na vzniklé hardwarové a softwarové chyby. Hodnotí obnovitelnost ovlivněných dat a stavů při výpadku, přerušení nebo selhání systému.
6. **Bezpečnost** – Míra rozsahu zajištění ochrany dat a informací, neoprávněnému přístupu a úpravám.

7. **Udržitelnost** – Efektivita systém upravovat, vylepšovat, opravovat nebo přizpůsobovat změnám prostředí a požadavkům. Sleduje dopad provedených změn na jednotlivé části nebo celého systému.
8. **Přenositelnost** – Možnost převést celý produkt nebo jeho části do jiného prostředí, případně nahrazení jiným systémovým produktem pro stejný účel.

3.4 Životní cyklus procesu vývoje

Vývoj softwaru je souhrn několika na sebe navazujících etap neboli kroků. Každá etapa je nedílnou součástí celého procesu a dohromady tvoří životní cyklus vývoje. Cílem procesu je zajistit vývoj kvalitního softwaru, který splňuje definované požadavky. Ve většině případů je tvořen následujícími kroky (Bureš, Renda, & Doležal, 2016):

1. Specifikace a definice požadavků
2. Analýza a návrh
3. Vývoj softwaru
4. Testování
5. Provoz a údržba

3.4.1 Specifikace požadavků

V rané fázi životního cyklu jsou formulovány požadavky. Podle (Bureš, Renda, & Doležal, 2016) je požadavek definován jako „*Podmínka nebo způsobilost potřebná k tomu, aby uživatel vyřešil problém nebo dosáhl cíle, který má systém nebo systémová komponenta splňovat nebo kterým má disponovat, a to tak, aby vyhovoval smlouvě, standardu, specifikaci nebo jinému formálně předepsanému dokumentu*.“

Celkový proces je tvořen několika úrovněmi požadavků, a to například Business cíle, které vycházejí z firemních plánů a nápadů. Klíčové je zde vytyčení přínosu a otázky, proč se mají plány realizovat. Další součást tvoří Uživatelské požadavky, které stanovují uživatelé, kteří daný software každodenně používají. Je vhodné brát v potaz jejich názor a individuální varianty řešení. Z business a uživatelských požadavků dále vycházejí Systémové požadavky, na kterých se podílejí IT odborníci. Variant pro realizaci návrhů bývá několik, avšak je nutné brát v potaz nadcházející vzniklé problémy a předcházet jim.

3.4.2 Analýza a návrh

Fáze návrhu slouží jako příprava podkladů pro realizaci při další fázi modelu implementace. Dochází zde například ke tvorbě schémat, návrhů vzájemných vztahů mezi komponentami, přehled funkčnosti každé komponenty a další.

3.4.3 Vývoj softwaru

Implementace neboli vývoj softwaru se provádí podle podkladů připravených z předchozích fází vývoje. Začíná zde samotný vývoj softwaru pomocí zvoleného programovacího jazyka. Vývojáři mají za úkol naprogramovat systém tak, aby odpovídal předem domluveným parametrům.

3.4.4 Testování

Před zpřístupněním daného softwaru do užívání je potřeba jej otestovat. Je důležité, aby software splňoval standardy a kvality, které byly definovány. Testování ověřuje, zda každá funkce funguje správně. Pokud je zjištěná závada, jsou o ni vývojáři informováni. Ověřené závady jsou opraveny a je vytvořená nová verze softwaru.

3.4.5 Provoz a údržba

Cílový stav představuje fáze Provoz a údržba. Po průchodu všemi fázemi testování by měla být zajištěna požadovaná kvalita softwaru. Produkt je nasazen do produkce a je využíván reálnými uživateli způsobem, pro který byl vytvořen. Údržba je nedílnou součástí každého provozu, při které dochází k opravě neodhalených bugů, zařazení nových funkcionalit a nahrazení softwaru novější verzí.

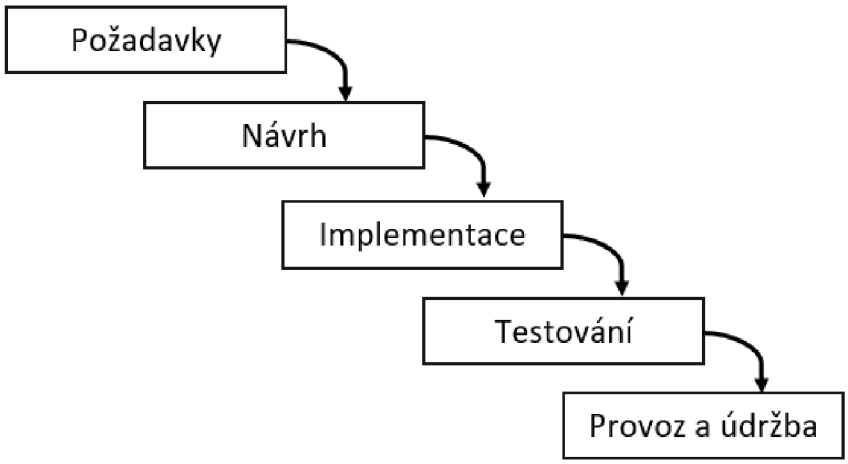
V případě, že je riziko při uvedení systému do reálného provozu příliš vysoké, přichází na řadu zkušební provoz, resp. simulace reálného využívání. Pokud v průběhu simulace nedojde k odhalení závažných chyb, přechází do produkce.

3.5 Modely procesu vývoje

Modely procesu vývoje znázorňují podrobný souhrn všech fází životního cyklu. Existuje několik modelů, přičemž každý z nich má své výhody a nevýhody. Je nutné zvážit, která metoda se nejvíce hodí pro konkrétní projekt.

3.5.1 Model vodopád (Waterfall)

Model Vodopád představuje definovaný plán, který se skládá z předem dané posloupnosti několika etap, které ovšem neumožňují cyklický návrat zpět. Spoléhá na dokončení předchozí fáze, aby mohla začít následující. Proces modelu, který odráží jeho název, plynule klesá fázemi implementace softwaru. Důležitým a zároveň prvním krokem je jasné stanovení požadavků a specifikace zadání, neboť po ukončení se k ní nelze vrátit a reagovat na dodatečné změny během vývoje. V případě nesprávné analýzy nebo špatně zadaných požadavků je celý návrh zrušen a celý proces opakován. (Insight, 2021)



Obrázek 1 Vodopádový model životního cyklu, zdroj: autor

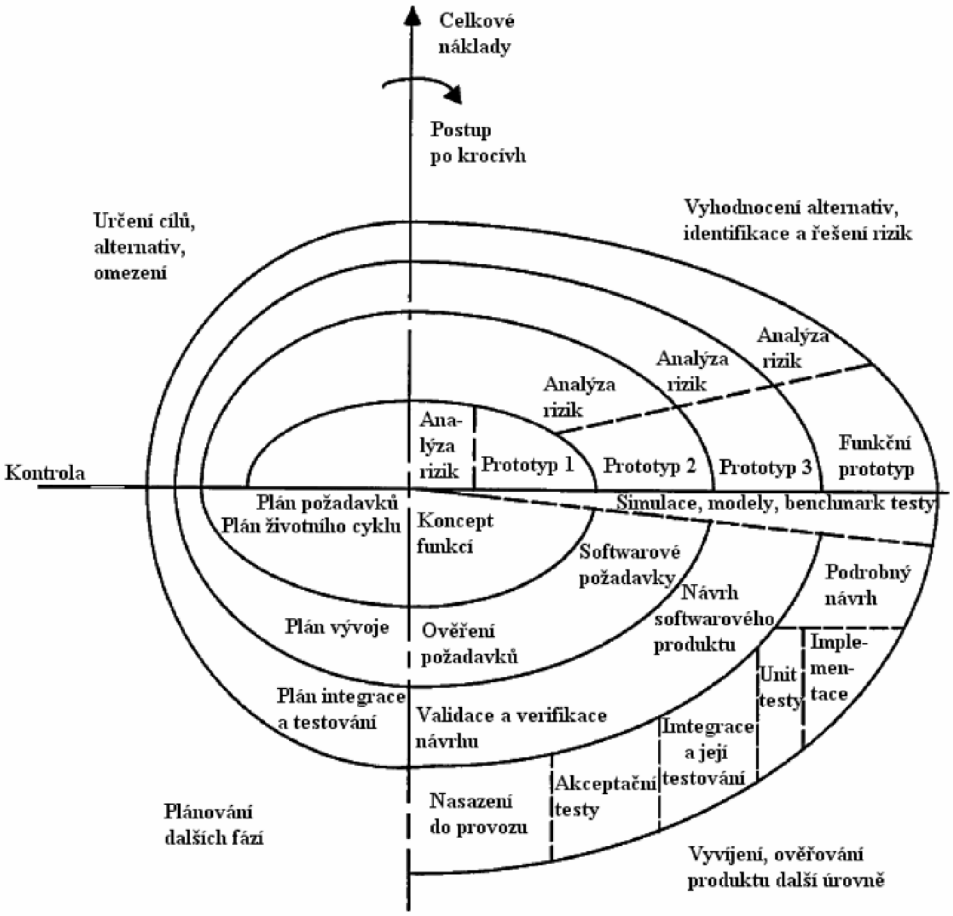
3.5.2 Spirálový model

Spirálový model poprvé představil Barry W. Boehm v roce 1988. Je reprezentován v podobě spirály a každá její smyčka představuje jednotlivou fázi procesu. V každé fázi je provedena analýza rizik, která nám udává, zda můžeme postoupit do další fáze či nikoli. Rizikem je chápána jakákoliv situace, která by mohla ohrozit daný SW. Tento model již neobsahuje vlastnost vodopádového modelu, a to nemožnost opakovat jednotlivé etapy.

Probíhá ve 4 krocích, které je možno několikrát iterovat, dokud není produkt přiveden do finálního stavu. Požadavky a analýzu lze tedy zpětně upravit neboli přizpůsobit dodatečným změnám a rozhodnutím. (Barry, 1988)

Průchod spirálou je rozdělen do 4 kroků:

1. Analýza – stanovení cílů
2. Hodnocení – identifikace rizik, vyhodnocení alternativ
3. Vývoj – vývoj produktu a jednotlivé fáze testování
4. Plánování – plánování následujících fází

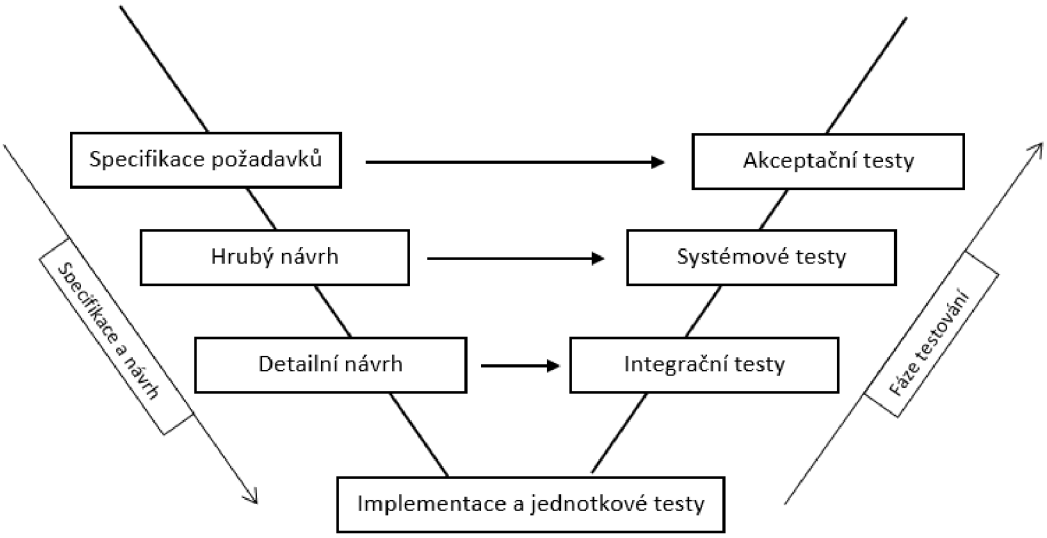


Obrázek 2 Spirálový model životního cyklu (Barry, 1988)

3.5.3 **V model**

V-model reprezentuje rozšířenou verzi vodopádového modelu do písmene V, podle kterého je také odvozen název. Je tvořen několika páry aktivit. Levou část tvoří činnosti spojené se specifikací požadavků a návrhem. Pravá část písmene je rozdělena do několika fází testování. V-model se tedy nezaměřuje pouze na analýzu a správně navrhnuté specifikace, ale zároveň jej doplňují fáze testování, které verifikují návrh a následnou realizaci. Zajišťuje tedy vysokou kvalitu softwaru. Jednotlivé úseky lze také definovat jako úrovně testování (angl. test levels). Podle (Bureš, Renda, & Doležal, 2016) Rex Black, klasik

v oblasti řízení testování, na základě své osobní zkušenosti tvrdí, že nejúspěšnější projekty jsou ty, jež mají jasně oddělené úrovně testování.



Obrázek 3 V model životního cyklu, zdroj: autor

3.6 Testovací dokumentace

Testovací dokumentace se skládá z několika dokumentů a je nedílnou součástí každého testování. Shromažďuje informace a určuje, jak bude probíhat náročnost testování. Sleduje požadavky, očekávané výsledky a další. Mezi běžně používané dokumenty patří například testovací plán, testovací případ a testovací scénář.

3.6.1 Testovací plán

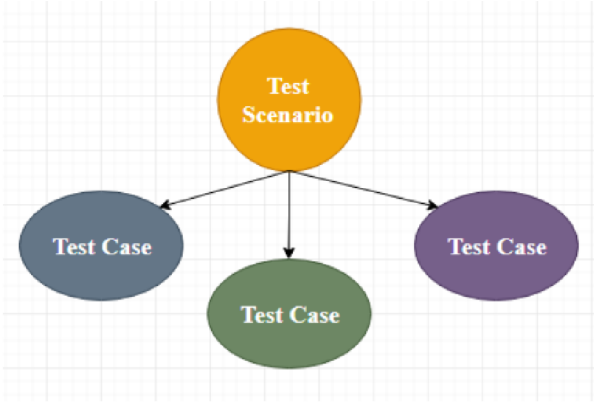
Testovací plán (angl. Test plan) je výchozí testovací dokument, který udává podmínky, za kterých bude testování probíhat a jsou podle něj řízeny veškeré části testování. Je vytvořen již v průběhu plánování projektu. Je postupně aktualizován a doplňován o informace až po vyhodnocení výstupů a konečné reportování výsledků. Obsahuje reference, tedy dokumenty zahrnující pokyny k instalaci nebo speciální vlastnosti, které s daným plánem souvisí. Dále seznam všech testovacích případů, seznam testovaných funkcí, vstupní a výstupní kritéria, seznam všech rizik spojených s testováním a plán cílů, kterých je třeba dosáhnout.

3.6.2 Testovací scénář

Testovací scénář (angl. Test scenario) je soubor testovacích případů, proto ve většině případech vyžaduje testování několika různými způsoby. Je méně podrobný a podává ucelené informace o tom, jaká oblast softwaru se má testovat.

3.6.3 Testovací případ

Testovací případ (angl. Test case) je jednou z primárních dokumentací životního cyklu. Poskytuje podrobné informace o tom, co je třeba testovat, jaká funkcionalita je testována, kroky prováděného testu a očekávaný výsledek. Dokument obsahuje všechny možné vstupy (pozitivní i negativní) a navigační kroky, které se používají k provedení testu. Testovací případ je akce první úrovně a je odvozen z testovacích scénářů. (JavaTpoint, 2021)



Obrázek 4 testovací případ (JavaTpoint, Test Case, 2021)

3.7 Funkční testování

Funkční testování se zaměřuje na funkční požadavky a specifikace softwaru, které jsou stanoveny v první fázi životního cyklu. Dochází k ověření správného chování systému ve všech možných situacích, které mohou nastat. Je obvykle rozděleno do čtyř úrovní testování – jednotkové, systémové, integrační a akceptační.

Úrovně testování neboli Typy testovacích prostředí, umožňují postupné, na sebe navazující testování softwaru, přičemž každá úroveň zahrnuje různé metody a přístupy. Dohromady tak zajišťují plné testování vyvíjeného softwaru a jeho požadovanou kvalitu. Pro zajištění nejvyšší kvality by mělo být předem určeno, jaké metody testů se v jednotlivých

fázích vývoje budou provádět. Nesprávná volba může mít za následek pozdní odhalení chyb až v následujících úrovní testování, kde bývá oprava nákladnější. (Corporation I. , 2022)

3.7.1 Jednotkové testování

Jednotkové testování (angl. Unit testing) jsou první fází testování a zpravidla je provádí sám vývojář. Testují se jednotlivé jednotky zdrojového kódu pomocí testovacích dat. Cílem testování je ověřit každou implementovanou část programu (metody, třídy) ještě před předáním k dalšímu testování. Odhalení chyb v prvním stádiu, tedy v průběhu jednotkového testování, šetří čas a opravy bývají nejméně nákladné.

3.7.2 Systémové testování

Další fází jsou systémové testy (angl. System test), zkráceně SYS. V některých případech se také můžeme setkat se zkratkou FAT z anglického výrazu *Factory Acceptace test* (Bureš, Renda, & Doležal, 2016). Systémové testování zahrnuje testování softwaru jako celku. Jsou zde propojeny hardwarové a softwarové komponenty systému, které již prošly jednotkovými testy. Testy jsou prováděny vývojovými testery a poté následuje předání testovacímu týmu do další úrovně.

3.7.3 Integrační testování

Integrační testování (angl. Integration testing) je určeno k testování jednotlivých komponent systému, které provádí testovací tým. Do tohoto prostředí se instalují již dokončené verze, které prošly testy vývojářů. Hlavním cílem je ověření bezchybné komunikace mezi jednotlivými komponentami, které dohromady tvoří kompletní aplikaci. Toto testování se obvykle provádí kombinací automatizovaných testů a manuálního testování v závislosti na tom, jak náročné je vytvořit automatické testy pro konkrétní součást systému.

3.7.4 Akceptační testování

Akceptační testování (angl. Acceptance testing) je závěrečnou fází funkčního testování softwaru. Je velice důležitým typem testování. Provádí jej testovací tým, který posoudí, zda software splňuje specifikace a uspokojuje požadavky klienta. Zjišťuje se a ověřuje splnění všech definovaných požadavků na daný projekt. Akceptační testy nemají poukázat pouze na jednoduché pravopisné a kosmetické chyby v aplikaci, ale také upozornit na případné chyby, které mohou způsobit pád celého systému. V případě odhalení bugů

v úrovni akceptačních testů, je nutné zajistit opravu v co nejkratším čase, neboť dlouhé časové zdržení může vést ke zpožděnému nasazení softwaru do produkce. (tutorialspoint, 2021)

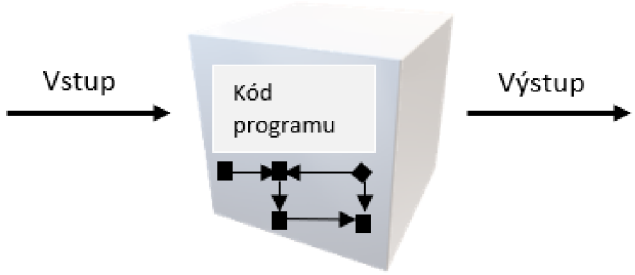
3.8 Metody testování

Existuje několik metod testování, každá z nich představuje různý způsob a přístup. Výběr způsobu testování závisí především na faktu, v jaké fázi vývoje se nacházíme, jaký produkt je testován a jakou funkcionalitu (chování) ověřujeme.

3.8.1 Znalost zdrojového kódu

Podle znalosti kódu dělíme metody testování do 3 kategorií a to: Černá, Bílá nebo Šedá skříňka. (Vskills, 2020)

Bílá skříňka (angl. White-Box) reprezentuje testování velice podrobné, zkoumá vnitřní logiku a struktury kódu. Aby mohl uživatel takové testy provádět, musí znát vnitřní fungování kódu. Hlavní cílem je na základě vstupů a pozorovaných výstupů zjistit, která část programu se chová nevhodně.



Obrázek 5 Bílá skříňka, zdroj: autor

Černá skříňka (angl. Black-Box) je metoda testování softwaru bez znalosti jeho zdrojového kódu. Tato metoda má omezenější pokrytí oproti Bílé skříňce, avšak testování může být provedeno i bez znalosti implementace daného softwaru nebo programovacího jazyka. Uživatel nemá přístup k vnitřní logice a zaměřuje se tedy především na uživatelské rozhraní systému. Na základě vstupů zkoumá výstupy, aniž by věděl, jak a kde se s nimi pracuje.

Obrázek 6 Černá skříňka, zdroj: autor

Šedá skříňka (angl.Gray-Box) je metodou, která představuje rozhraní mezi Bílou a Černou skříňkou. Uživatel má pouze omezený přístup ke znalostem o vnitřním fungování softwaru, jako jsou například návrhové dokumenty nebo databáze. Prostřednictvím těchto dat je testování oproti Černé skříňce rozšířenější a umožňuje tak vytvářet kvalitnější testovací scénáře. (Vskills, 2020)

3.8.2 **Statické a dynamické testování**

Podle nutnosti spuštění testovacího softwaru dělíme metody testování na Statické a dynamické. Tyto metody jsou velice významné v průběhu životního cyklu testování a je podstatné rozpoznat, jakou metodu pro stanovenou situaci použít.

Statické testování se provádí ručně, nedochází tedy ke spuštění kódu a software zůstává po celou dobu testování neaktivní. Probíhá v počátcích životního cyklu, nastává zde kontrola zdrojových kódů, testovacích skriptů, dokumentů, testovacích plánů a případů.

Dynamické testování je realizováno za běhu testovaného softwaru. Uživatel pracuje se vstupy a výstupy, které následně porovnává s očekávanými výsledky. (GeeksforGeeks, 2020)

3.8.3 **Manuální a automatické testování**

Manuální testování je založeno na ručním testování uživatelem podle předem vytvořených testovacích případů (angl. testcases), bez použití jakéhokoliv automatizovaného nástroje nebo skriptu. Je využíváno v různých fázích životního cyklu. Tento typ testování je vhodný v případech, kde je zapotřebí lidské uvažování.

Výhody:

- vhodné při testování neobvyklých situací
- pohotovost, reakce na změny softwaru
- snadné odhalení chyb v grafickém prostředí

Nevýhody:

- časová náročnost
- lidská chybovost
- ne vše lze otestovat ručně

Automatické testování je realizováno pomocí testovacích skriptů nebo pomocí jiného nástroje či softwaru. Lze jej jako manuální testování využívat na různých úrovních testovacího systému. Umožňuje opakované využití s velkým množstvím vstupních dat v různých kombinacích a ve srovnání s manuálním testováním je velice rychlé. V případě použití zcela automatizovaných testů, tedy bez přítomnosti lidského faktoru, se výrazně snižuje riziko chybného provedení postupu testování. Ne vždy se ale vyplatí přecházet na automatické testování, jelikož vývoj a údržba některých testů je velice nákladná a časově náročná. (Hamilton, 2022)

Výhody:

- opakovatelnost
- rychlost provedení
- přesnost, bezchybnost

Nevýhody:

- vysoké náklady na vývoj a údržbu
- složitá údržba
- výskyt chyb v testovacích skriptech

3.8.4 Manuální vs. automatizovaný test

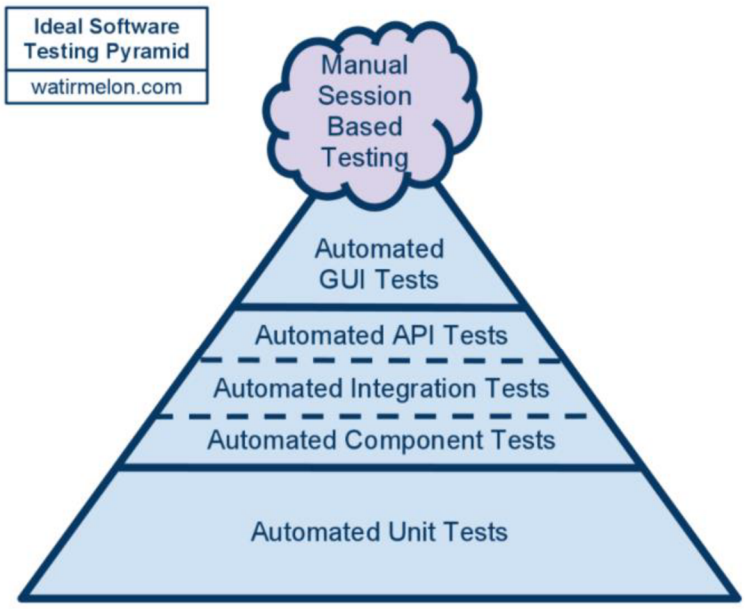
Podle (Bureš, Renda, & Doležal, 2016) je popsáno osm rozdílů, v čem se liší manuální a automatizovaný test:

1. Automat, na rozdíl od člověka, vykonává test „zdarma“ a relativně rychle.
2. Akce provedené automatizovaným testem jsou vždy stejné: automat v rámci toho, jak je naprogramován, zopakuje dané kroky vždy stejně a neudělá chybu.
3. Na rozdíl od člověka automat neumí samostatně řešit situace, pokud se chování testovaného systému odlišuje od očekávaného chování v testu.
4. Na rozdíl od člověka má automat omezenou možnost analýzy chybného chování testovaného systému.
5. Negativní výsledky běhu automatizovaného testu musí ještě nezávisle vyhodnotit tester.
6. Pokud vytváříme automatizovaný test, vlastně již testujeme.
7. Pro implementaci automatizovaných testů založených na simulování akcí v uživatelském rozhraní již potřebujeme zprovozněný a pokud možno stabilní testovaný systém.
8. Rozdíl je pochopitelně i v potřebné kvalifikaci. Manuální tester a vývojář automatizovaných testů jsou odlišné profese.

3.9 Principy automatizovaného testování

3.9.1 Testovací pyramida

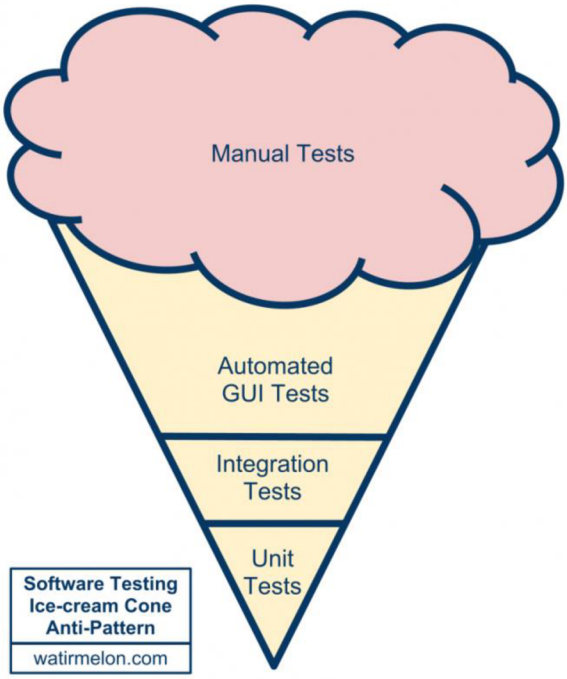
Testovací pyramida uvádí, jaké množství automatických testů by se mělo v daných fázích vývoje provádět. Největší část představuje jednotkové testování při samotném vývoji softwaru, postupuje přes integrační testy až po testování grafického prostředí. Automatizované GUI testy se vytvářejí velice obtížně a v některých případech je to téměř nemožné. Výhodou testovací pyramidy je včasné odhalení vývojářských chyb pomocí jednotkového testování, které se nepřenáší do dalších fází testování. Protiklad Testovací pyramidy představuje Testovací zmrzlina.



Obrázek 7 Testovací pyramida (Watirmelon, 2018)

3.9.2 **Testovací zmrzlina**

U softwarových testovacích kuželů zmrzliny většinu testů tvoří manuální testování, kde se testy provádějí ručně. Jednotkové testování je v tomto případě v pozadí a zaostává. Problém tohoto modelu představuje nedostatečné otestování samotného kódu programu.



Obrázek 8 Testovací zmrzlina (Watirmelon, 2018)

3.10 Nástroje pro automatizované testování

Nástroj pro automatizované testování je software, pomocí kterého se po samotné přípravě testů a testovacích dat dají opakovaně spouštět testy. Existuje celá řada nástrojů, vyvinutých pro použití na daném konkrétním projektu. Tato kapitola obsahuje popis vybraných nástrojů pro automatizované testování.

3.10.1 Randoop

Nástroj Randoop je generátor jednotkových testů pro programovací jazyk Java. Pomocí generátoru jsou automaticky vytvářeny testy, které kontrolují strukturu a chování vytvořeného programu. Randoop tak pomáhá vývojáři uspořit spoustu času při tvorbě základních testů. Používá se především k odhalení chyb v kódu, ale také k tvorbě regresních testů, které upozorňují na změny v chování programu. Výchozí nastavení tedy generuje dva druhy testů, zapsané do samostatných souborů. Nástroj lze bezplatně stáhnout z webových stránek v podobě samostatného externího nástroje. (Randoop Manual , 2020)

3.10.2 Test studio

Test studio je snadno použitelný, intuitivní nástroj, pro funkční testování webových a mobilních aplikací, testování výkonu softwaru a zátěžové testování, vyvinuté společností Telerik. Mezi podporované webové prohlížeče patří Internet Explorer, Chrome, Safari a Firefox. S pomocí vizuálního záznamníku testů (angl. Visual test recorder) je vhodný i pro méně zkušené vývojáře, kteří chtějí přejít z manuálního testování na automatizované. Mohou začít vytvářet automatizované testy bez nutnosti psaní kódu. V tomto případě Test Studio automaticky zaznamenává veškeré akce prováděné v testované aplikaci. Zkušeným testerům naopak umožňuje vytvářet pokročilé testy pomocí zabudovaných editorů C# nebo Visual Basic. Test Studio také umožňuje převádět funkční testy na zátěžové, popřípadě je snadno vytvářet již od začátku. Test Studio nabízí bezplatnou 30denní plnohodnotnou zkušební verzi. Podmínkou pro stažení instalačního souboru je vytvoření účtu Telerik. Po uplynutí zkušební verze je zapotřebí zakoupit jednu z nabízených licencí. Nejpoužívanější verze se všemi základními funkcemi je možné zakoupit v hodnotě 3500 $ za rok. Testy lze také vytvářet pomocí instalace plug-in do vývojového prostředí Visual Studio. (Corporation, 2022)

3.10.3 **Selenium**

Selenium je vhodné především pro automatizované testování webových aplikací. Jedná se o jednu z nejrozšířenějších testovacích sad pro automatizaci webového uživatelského rozhraní s otevřeným zdrojovým kódem (angl. open source). Podporuje automatizaci napříč různými prohlížeči, platformami a programovacími jazyky. Mezi podporované jazyky patří C#, Java, Perl, Python a Ruby. Selenium nepředstavuje jen jeden nástroj, ale sadu softwaru, z nichž každá má odlišný přístup k podpoře automatizovaného testování. (JavaTpoint, 2021) Skládá se ze 3 hlavních složek:

- Selenium WebDriver je nejdůležitější částí. Poskytuje programovací rozhraní pro vytváření a spouštění testovacích případů. Testovací skripty jsou psány za účelem identifikace webových prvků na webových stránkách a poté jsou na těchto prvcích prováděny požadované akce. Podporuje několik webových prohlížečů, jako jsou IE, Chrome, Firefox, Opera a Safari. Z důvodu bezpečného spojení s prohlížečem, používá Selenium WebDriver ovladače, které jsou specifické pro každý typ prohlížeče. V aplikaci lze testovací skripty vyvíjet pomocí libovolného podporovaného programovacího jazyka. Mezi podporované jazyky patří C#, Java, Python, PHP, Perl a Ruby. (JavaTpoint, 2021)
- Selenium IDE (Selenium Integrated Development Environment) je implementováno jako rozšíření webového prohlížeče, které poskytuje funkci nahrávaní a spouštění testovacích skriptů. Je k dispozici pro prohlížeče Google Chrome, Mozilla Firefox a Microsoft Edge. Nepotřebuje k provádění testování žádné znalosti programovacího jazyka. Uživatel má možnost jednoduše zaznamenávat své interakce s prohlížečem a vytvářet tak testovací případy. Umožňuje testerům exportovat vytvořené skripty v mnoha jazycích, jako je například HTML, Java, Ruby, C# a další. Exportované skripty se dají dále použít v Seleniu Webdriver. (JavaTpoint, 2021)
- Selenium Grid je také důležitou součástí. Umožňuje paralelně spouštět vytvořené testy na různých zařízeních s různými prohlížeči a operačními systémy. Grid využívá koncept hub-node, kdy test probíhá na centrálním zařízení a paralelní provádění testů se realizuje na různých vzdálených místech. (JavaTpoint, 2021)

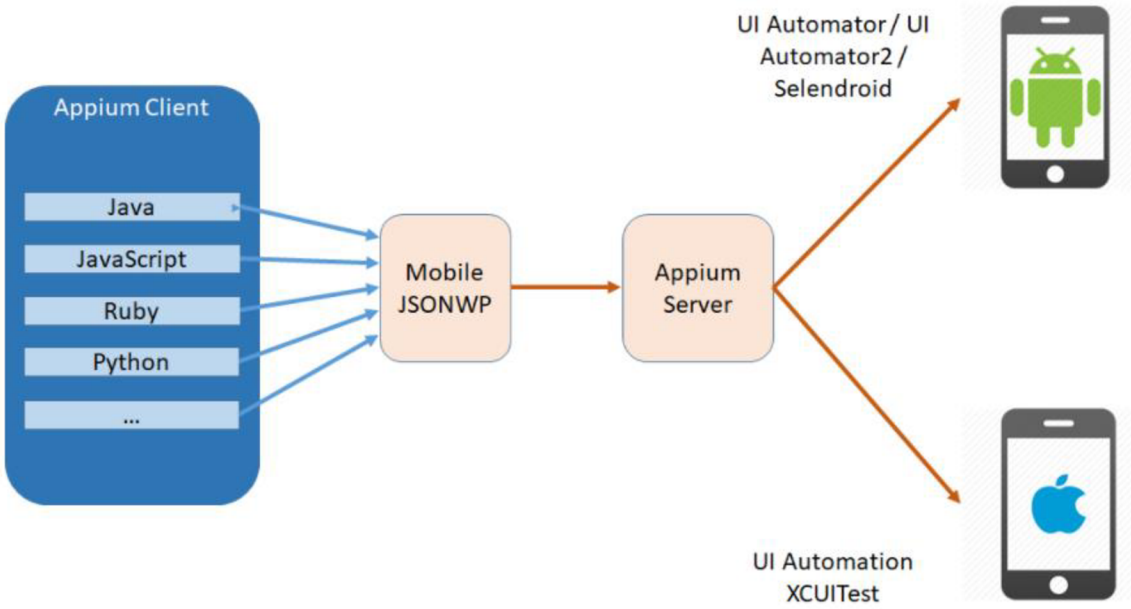
3.10.4 **Appium**

Appium je open-source nástroj, který se používá pro automatické testování mobilních aplikací. Je vyvinut a podporován společností Sauce Labs. Je velmi podobný testovacímu nástroji Selenium WebDriver. Jedná se o multiplatformní automatizační nástroj, což znamená, že umožňuje spouštění stejného testu na více platformách. Umožňuje paralelní provádění testovacích skriptů na více zařízeních, a tak urychluje proces testování. Nabízí opětovné použití stejného kódu na různých platformách zařízení, jako je iOS, Android a Windows. Podporuje tvorbu testů v několika jazycích jako je Java, PHP, C#, Python, JavaScript a Ruby.

Používá se pro automatizované testování nativních, hybridních a webových aplikací. Nativní aplikace jsou softwarové programy, které jsou vyvíjeny s ohledem na určitou platformu. Jsou vyvinuty pro použití na konkrétním zařízení a lze je nainstalovat z App Store, Google Play Store. Webové aplikace nejsou skutečné aplikace, ale webové stránky, které běží na prohlížečích. Jsou vyvíjeny pomocí HTML, CSS a JavaScriptu. Jelikož webové aplikace běží na webových prohlížečích, nevyžadují žádnou instalaci. Hybridní aplikace představuje kombinace nativních a webových aplikací. Stejně jako nativní aplikace lze tyto aplikace stáhnout z App Store/Google Play Store. Jedná se o mobilní aplikace, ke kterým lze také přistupovat prostřednictvím adresy URL přes webový prohlížeč. (Pai, 2021)

Architektura Appium se skládá ze 3 hlavních komponent (Section, 2021):

1. Klient Appium – obsahuje automatizovaný skript. Tento skript je napsán ve specifickém programovacím jazyce. Skript obsahuje podrobnosti o konfiguraci aplikace a mobilního zařízení, který se používá ke spuštění testovacích případů.
2. Server Appium – jedná se o http sever, který od klienta přijímá a shromažďuje příkazové požadavky (http požadavky) ve formátu JSON a odesílá je na příslušnou platformu. JSON Wire Protocol je komunikační prostředek mezi serverem a klientem. Spolupracuje s různými platformami jako je Android, iOS. Funguje tedy jako prostředník mezi klientem a koncovým zařízením. Appium server zpracovává požadavky pro iOS a Android odlišně. Na zařízeních se systémem iOS používá UIAutomation (knihovna poskytovaná společností Apple) pro připojení k prvkům uživatelského rozhraní aplikace. Na zařízeních se systémem Android používá Appium k interakci s prvky UIAutomator, který je vyvinutý společností Android.
3. Koncové zařízení – na tomto zařízení se provádějí testovací případy



Obrázek 9 Architerkura Appium, (Haidar, 2020)

4 Vlastní práce

V teoretické části práce byly popsány vybrané nástroje pro automatizované testování. Každý nástroj má své specifické vlastnosti a funkce. Nástroj lze využít k tvorbě skriptů za účelem otestování jednoduchého programu, aplikace či webové stránky, nebo také rozsáhlého projektu složeného z několika částí, komponent. Cílem této práce je srovnání testovacích nástrojů, výběr vhodné varianty implementace vzhledem k požadovaným vlastnostem nástroje a vlastností testovaného softwaru.

4.1 Analýza situace

Pro tuto práci jsem vybrala široce používanou webovou stránku, na které budu provádět testovaní. Jedná se o poštovní službu Gmail dostupnou na adrese https://mail.google.com/. Úspěch jakékoli automatizace testování závisí na výběru správného nástroje. Hlavní podmínkou pro výběr testovacího nástroje je tedy podpora tvorby testů pro webové aplikace. Cena nástroje je dalším důležitým kritériem. V případě potíží je možnost vznést dotaz, nebo vyhledat diskusi o obdobném problému výhodou. Dodavatelská společnost nástroje by tedy měla mít zavedenou zákaznickou podporu. Existuje několik klíčových vlastností, které je nutné zvážit před výběrem vhodného nástroje pro realizaci automatizovaného testování. Analýza vlastností vybraných nástrojů je provedena v další kapitole.

4.2 Analýza vybraných nástrojů

Aby bylo možné zahájit testování, je nutné vybrat správný testovací nástroj. Je třeba analyzovat řadu vlastností, abychom byli schopni danou aplikaci dostatečně otestovat a nástroj tak splnil naše očekávání.

Analýza a srovnání vybraných nástrojů spočívá v komparaci několika charakteristik a vlastností, které každý nástroj charakterizují a navzájem od sebe odlišují. Pozorované charakteristiky jsou:

- Dostupnost
- Cena
- Typy podporovaných testů
- Podporované jazyky pro tvorbu testů
- Náročnost práce s testovacím nástrojem
- Rozlehlost komunity, diskusní fóra
- Podpora a stáří

4.2.1 **Randoop**

Charakteristika	**Popis**
Dostupnost	Bezplatně dostupný
Cena	Zdarma
Typy podporovaných testů	Jednotkové a regresní testy
Podporované jazyky	Java
Náročnost práce s nástrojem	Snadná
Rozlehlost komunity, diskusní fóra	Uživatelské skupiny určené pro otevřené diskuse dostupné z: https://groups.google.com/g/randoop-discuss, https://groups.google.com/g/randoop-developers/c/kIFaRa7Xhjs
Podpora a stáří	Verze 4.3, vydaná 31.01. 2022

Tabulka 1 Charakteristiky nástroje Randoop, zdroj: autor

4.2.2 **Test studio**

Charakteristika	**Popis**
Dostupnost	Dostupný s bezplatnou trial verzí
Cena	2499 $ základní roční licence 3499 $ rozšířená roční licence zahrnující možnost spuštění zátěžových testů Veškeré doplňky a ceny jsou uvedeny na oficiálních webových stránkách: https://www.telerik.com/purchase/teststudio
Typy podporovaných testů	Funkční testování webových a mobilních aplikací, testování výkonu softwaru, zátěžové testování.
Podporované jazyky	C#, Visual Basic
Náročnost práce s nástrojem	Pro tvorba testů pomocí vizuálního záznamníku nejsou nutné žádné znalosti. Pro tvorbu automatizovaných skriptů je požadovaná znalost v programovacím jazyce.
Rozlehlost komunity, diskusní fóra	Diskusní fórum Test Studio podporované více než 3,5 miliony vývojářů dostupné i pro zkušební trial verzi. https://www.telerik.com/forums/teststudio
Podpora a stáří	Verze 2022.1.215, vydaná 15.02.2022

Tabulka 2 Charakteristiky nástroje Test Studio, zdroj: autor

4.2.3 **Selenium**

Charakteristika	**Popis**
Dostupnost	Bezplatně dostupný
Cena	Zdarma
Typy podporovaných testů	Funkční testování webových aplikací.
Podporované jazyky	C#, Java, Python, PHP, Perl, Ruby
Náročnost práce s nástrojem	Selenium IDE nevyžaduje žádné znalosti. Tvorba testovacích případů je velice snadná. Selenium WebDriver požaduje znalost v programovacím jazyce.
Rozlehlost komunity, diskusní fóra	Selenium ze svých oficiálních webových stránek https://www.selenium.dev/support/ nabízí několik možností, ze kterých lze vyhledat pomoc.
Podpora a stáří	Verze 4.1, vydaná 22.11.2021

Tabulka 3 Charakteristiky nástroje Selenium, zdroj: autor

4.2.4 **Appium**

Charakteristika	**Popis**
Dostupnost	Bezplatně dostupný
Cena	Zdarma
Typy podporovaných testů	Funkční testování mobilních aplikací
Podporované jazyky	Java, PHP, C#, Python, JavaScript, Ruby
Náročnost práce s nástrojem	Požadovaná znalost v programovacím jazyce
Rozlehlost komunity, diskusní fóra	Appium nabízí oficiální diskusní fórum dostupné na webové stránce: https://discuss.appium.io/
Podpora a stáří	Verze 1.22.2, vydaná 20.02.2022

Tabulka 4 Charakteristiky nástroje Appium, zdroj: autor

Testovací nástroj lze vybrat podle typu testovaného softwaru, finančního plánu a schopnosti provádět automatizované testování. Každý z pozorovaných nástrojů má své výhody i nevýhody.

Nástroj Randoop je skvělým pomocníkem při vytváření jednotkových a regresních testů. Je bezplatný a práce s ním je velmi snadná. Jednotkové testování je velmi důležitou součástí vývoje softwaru, ale jejich tvorba může být v některých případech obtížná a časově náročná. Pomocí náhodných, automaticky generovaných, ale smysluplných testů nám Randoop může ušetřit spoustu času. Pokud má uživatel při práci s Randoop nějaké obtíže, může se obrátit na jednu z oficiálních diskusních skupin, kde může vznést dotaz nebo si projít již zodpovězené od ostatních uživatelů. Jelikož testuji webovou aplikaci a nástroj Randoop nesplňuje požadovanou vlastnost, kterou je možnost testování webových aplikací, jeho výběr jsem zavrhla.

Appium je jedním z automatizačních nástrojů pro testování mobilních aplikací. Je zdarma a lze jej snadno stáhnout. Výhodou je podpora mnoho programovacích jazyků jako je Java, PHP, C#, Python, JavaScript, Ruby. Podpora Appium je napříč všemi platformami, napsaný test lze tedy spouštět v iOS i Androidu bez jakékoliv změny kódu. Je vyvinut a podporován společností Sauce Labs a má velkou podporu komunity. Appium je skvělým nástrojem pro tvorbu automatizovaných testů, ale stejně jako Randoop, nesplňuje možnost testování webových aplikací, proto ho pro tuto práci nemohu vybrat.

Test Studio je navrženo především pro provádění funkčního testování webových a mobilních aplikací. Nabízí podporu napříč prohlížeči Internet Explorer, Chrome, Safari a Firefox. Pro méně zkušené testery nabízí pomocí vizuálního záznamníku vytváření automatizovaných testů bez nutnosti psaní kódu. Pro tvorbu automatizovaných skriptů podporuje pouze dva skriptovací jazyky, a to C# a Visual Basic, což může být jedna z nevýhod. Vývojáři automatizovaných testů nemají dostatečný výběr a zaškolení může být nákladné. Test studio nabízí bezplatnou trial verzi. Po uplynutí zkušební verze je nutné si zakoupit jednu z nabízených licencí. Základní licence nezahrnuje zátěžové testování a testování výkonu softwaru. Uživatelé mají ovšem možnost přejít na vyšší licenci, která toto testování zahrnuje. Test Studio je velice dobře podporováno a vyvíjeno, jednotlivé verze opravují chyby a přidávají nové funkcionality. Poslední verze 2022.1.215 byla uveřejněna 15.2.2022. Diskusní fórum Test Studia je velice rozsáhlé a podporované více než 3,5 miliony vývojářů. Je dostupné z oficiální webové stránky Telerik Test Studio: https://www.telerik.com/forums/teststudio. Pouze uživatelé se zakoupenou licencí nebo

zkušební verzí mohou do diskusního fóra položit jakoukoli otázku, popřípadě si přečíst již existující.

Pomocí Selenium lze provádět funkční testování webových aplikací. Podporuje několik webových prohlížečů. Poskytuje možnost spouštět testovací skripty v prohlížeči IE, Chrome, Firefox, Opera a Safari. Sada Selenium IDE poskytuje funkci nahrávaní a spouštění testovacích skriptů. Na rozdíl od sady Selenium WebDriver nepotřebuje k provádění testování žádné znalosti programovacího jazyka. Selenium WebDriver umožňuje psaní testovacích skriptů v mnoha jazycích, jako je C#, Java, Python, PHP, Perl a Ruby. To uživateli usnadňuje výběr programovacího jazyka podle jeho preference. Jednou z největších výhod Selenia je, že je pro všechny bezplatně dostupný. Jediné náklady s ním spojené mohou spočívat v zaškolení zaměstnanců, pokud společnost používá Selenium WebDriver jako primární testovací nástroj. Jednou z nevýhod Selenia jsou tedy požadované znalosti v programovacím jazyce. Jelikož podporuje různé programovací jazyky, je pro testery jednodušší vytvářet skripty v preferovaném programovacím jazyce. Nicméně každý tester musí mít pro psaní testovacích skriptů požadované znalosti alespoň jednoho z podporovaných skriptovacích jazyků. Pro všechny uživatele nabízí z oficiálních webových stránek https://www.selenium.dev/support/ několik možností, ze kterých lze v případě obtíží vyhledat pomoc. Jedná se například o uživatelskou skupinu, chatovací místnost a další. Selenium se stále vyvíjí a vychází nové verze, které opravují chyby a jsou obohaceny o nové vlastnosti. Poslední verze 4.1 byla vydána 22.11.2021.

Nástroj Test Studio i Selenium jsou skvělými kandidáty pro testování webových aplikací. Existují však mezi nimi rozdíly. Jak již bylo zmíněno, Test Studio podporuje pouze dva programovací jazyky pro psaní testovacích skriptů. Oproti tomu Selenium nabízí širokou škálu podporovaných jazyků a je tak snazší vybrat právě ten vyhovující. Test studio mimo testování webových aplikací nabízí možnost provádět zátěžové testování. V našem případě je to ale nadbytečná vlastnost, kterou u testování poštovní služby nevyužijeme. Největším, a především rozhodujícím rozdílem mezi těmito dvěma nástroji představuje cena. Zatímco Selenium je bezplatné, Test Studio nabízí pouze bezplatnou trial verzi a poté je nutné zakoupit licenci. Pro tuto práci jsem tedy vybrala nástroj Selenium.

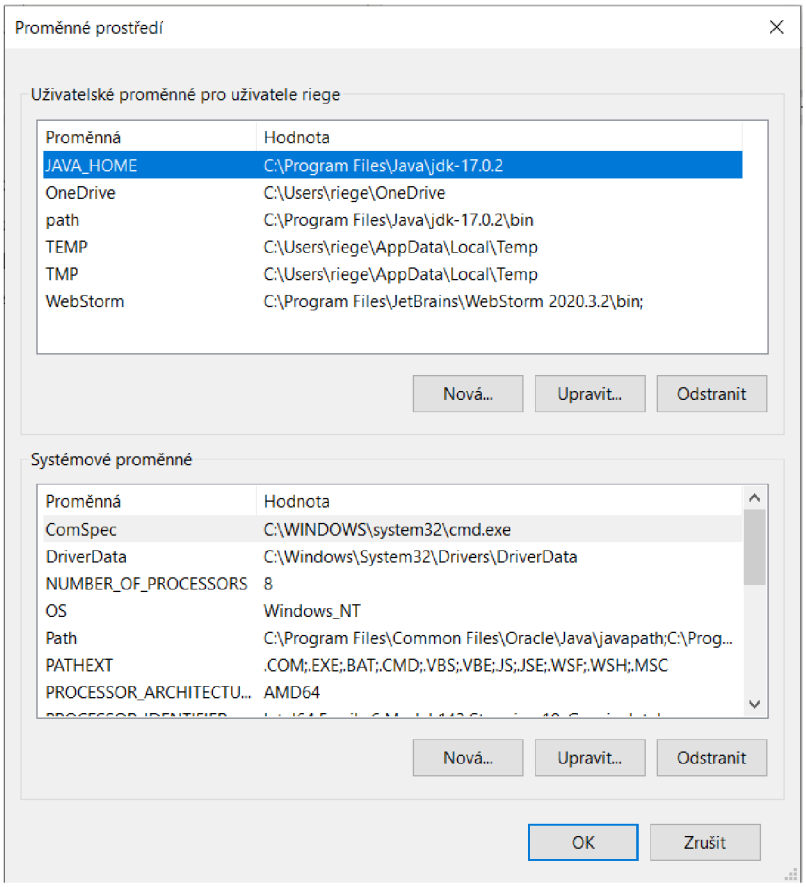
4.3 Instalace Selenium WebDriver

V první řadě je potřeba zvolit jeden z jazyků pro tvorbu automatizovaných testů, jelikož pro každý z nich se instalace liší. Selenium podporuje různé programovací jazyky, jako je Java, C#, Python, Perl, Ruby. Pro tvorbu automatizovaných testů jsem zvolila jazyk Java, jelikož je v dnešní době jedním z nejpoužívanějších jazyků pro Selenium.

Proces instalace zahrnuje následující kroky:

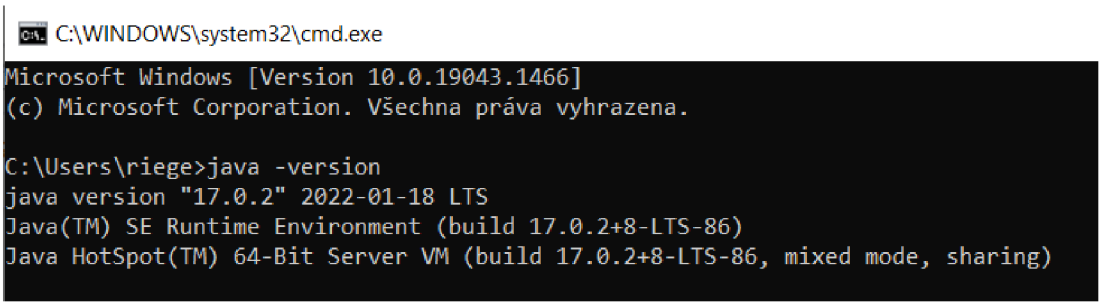
1. Instalace Javy
2. Instalace vývojového prostředí
3. Instalace Selenium WebDriver
4. Konfigurace Selenia

Nejnovější verze Java Development Kit (JDK) je bezplatně stažitelná na webové stránce https://www.oracle.com/java/technologies/downloads/. JDK obsahuje nástroj pro vývoj a testování programů běžících na platformě Java. Nejprve musíme potvrdit souhlas s licenčními podmínkami a poté je možné stáhnout instalační soubor v závislosti na verzi operačního systému. Po stažení instalačního souboru JDK je potřeba nastavit cestu k adresáři Java. Cestu je nutné nastavit pro ukládání zdrojových souborů. Pokud jej ukládáme právě do adresáře JDK/bin, není nutné nastavovat cestu, protože všechny nástroje budou dostupné v aktuálním adresáři. K nastavení trvalé adresy je potřeba provést několik kroků. V systému Windows 10 klikneme pravým tlačítkem na ikonu Tento počítač a z nabídky akcí vybereme možnost Vlastnosti. Otevře se nám okno nastavení a dále pokračujeme kliknutím na Upřesnit nastavení systému -> záložka upřesnit -> proměnné prostředí. Vytvoříme novou uživatelskou proměnnou s názvem Path a na místo hodnoty vložíme umístění složky JDK/bin. Podobným způsobem vytvoříme proměnnou s názvem JAVA_HOME s hodnotou cesty ke složce JDK.



Obrázek 10 Nastavení proměnných prostředí ve Windows 10, zdroj: autor

Pro kontrolu, zda instalace proběhla v pořádku, lze z příkazové řádky cmd spustit následující příkaz *java-version*, kde se zobrazí nainstalovaní verze Javy.



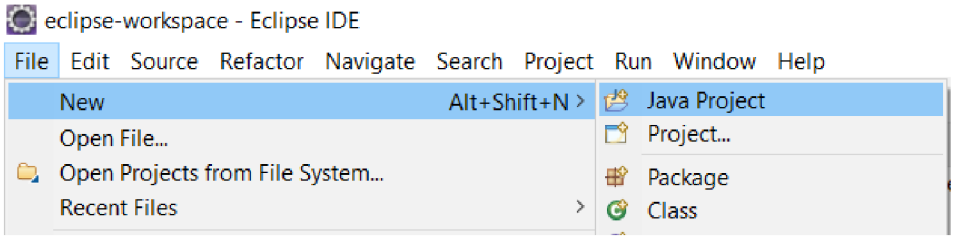
Obrázek 11 kontrola instalace přes cmd, zdroj: autor

Dalším krokem je instalace vývojového prostředí pro tvorbu automatizovaných testů. V tomto případě Eclipse, jelikož je určené pro programování v jazyce Java. Lze jej bezplatně stáhnout z webové stránky https://www.eclipse.org/downloads/ a samotná instalace je velmi snadná. Po úspěšné instalaci se spustí výchozí rozhraní Eclipse.

Instalace Selenium WebDriver se skládá ze dvou kroků. V první fázi je potřeba nainstalovat knihovny Selenium pro jazyk Java. Všechny dostupné verze můžeme najít na webové stránce https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java a jsou dostupně stažitelné z https://selenium-release.storage.googleapis.com/index.html. Soubor se stáhne se formátu ZIP, který je potřeba rozbalit na disk. Obsahuje veškeré soubory JAR, které je potřeba později naimportovat do prostředí Eclipse pro nastavení Selenia.

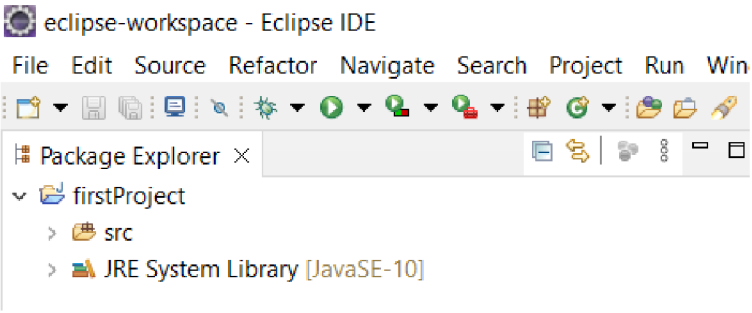
V dalším kroku je potřeba pomocí ovladačů nastavit systém tak, aby umožnil automatizaci vybraného prohlížeče. Prostřednictvím WebDriver podporuje Selenium všechny hlavní prohlížeče, jako je Chrome, Firefox, Internet Explorer, Edge, Opera a Safari. Pro každý testovaný prohlížeč je potřeba nainstalovat z webové stránky https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/ ovladač, v našem případě Chrome.

Nyní přichází na řadu konfigurace Eclipse pomocí Selenium WebDriver. Jinak řečeno, vytvoříme nový projekt Java v prostředí Eclipse a načteme všechny základní stažené soubory jar, abychom vytvořili testovací skript Selenium. Nový projekt vytvoříme z nabídky File -> New -> Java Project.



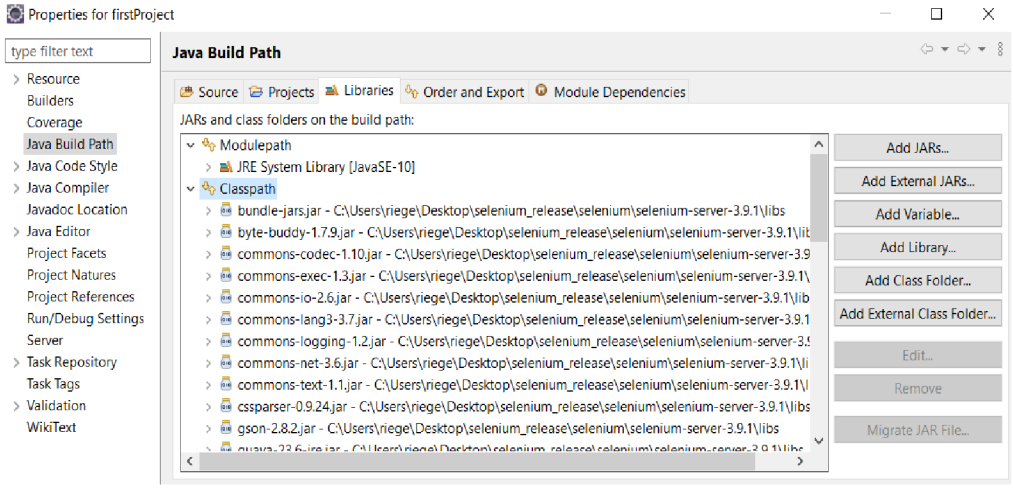
Obrázek 12 tvorba nového projektu Java, zdroj: autor

Zadáme název prvního projektu „firstProject“, ostatní pole necháme beze změny a klikneme na tlačítko dokončit. Vytvoří se nový projekt Java s následujícími adresáři:



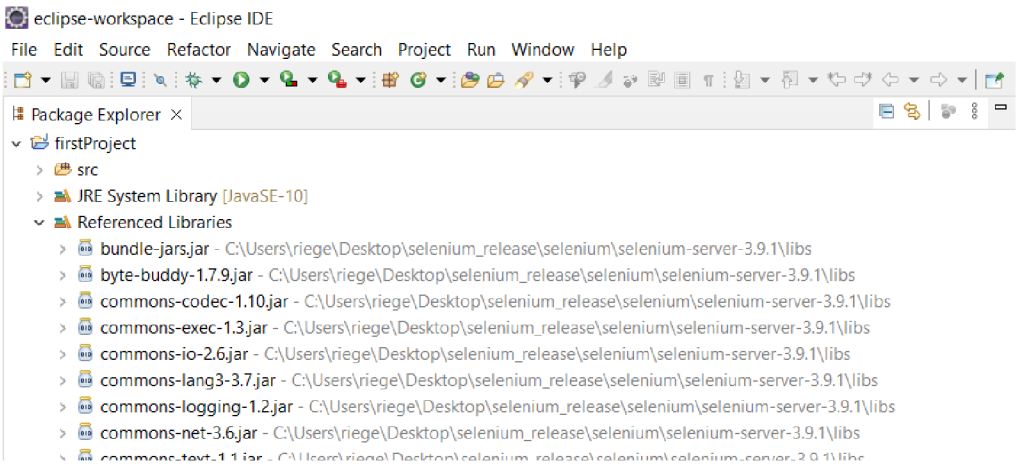
Obrázek 13 nový projekt s vytvořenými adresáři, zdroj: autor

Klikneme pravým tlačítkem na složku src poté na New -> Class a vytvoříme nový soubor neboli třídu. Zadáme název třídy „prvni“ a klikneme na tlačítko dokončit. Nyní přidáme soubory Selenium jar do naší testovací sady (firstProject). Přes pravé tlačítko na složku firstProject vybereme vlastnosti a otevře se nám okno. V levém panelu přejdeme do části Java Build Path, přes tlačítko Add External JARs nahrajeme všechny stažené soubory a uložíme.



Obrázek 14 konfigurace Selenia, zdroj: autor

V tuto chvíli jsme úspěšně nakonfigurovali Selenium WebDriver do prostředí Eclipse a můžeme začít vytvářet testovací skripty. Adresářová struktura naší testovací sady vypadá následovně:



Obrázek 15 adresářová struktura, zdroj: autor

4.4 Testovací scénář

V následující tabulce jsem vytvořila testovací scénář, který je rozdělen do několika testovacích případů, které jsou vedené pod svým unikátním ID. Každý testovací případ zahrnuje testovanou funkcionalitu aplikace, očekávaný výsledek a skutečný výsledek.

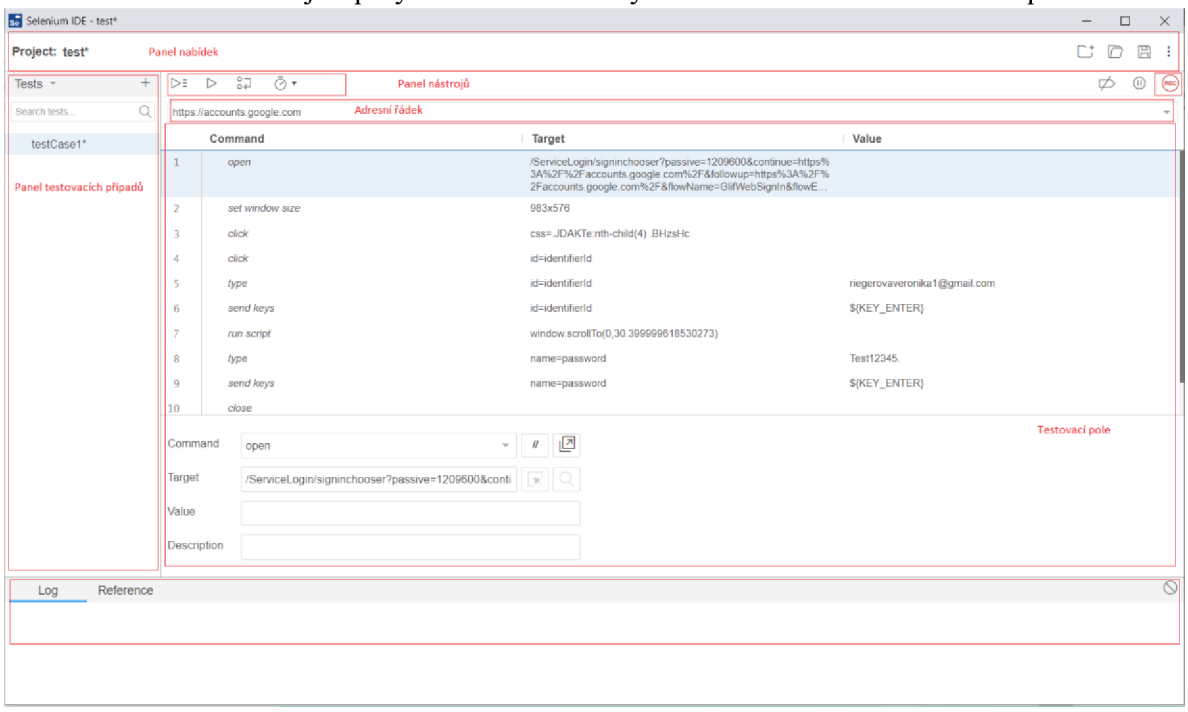
TestID	Funkcionalita	Očekávaný výsledek
001	Vytvoření nového účtu	Nový uživatelský účet byl úspěšně vytvořen
002	Přihlášení uživatele do poštovní služby	Uživatel je úspěšně přihlášen po zadání své e-mailové adresy a hesla.
003	Vyhledávání v poštovní službě pomocí filtru	Odpovídající e-mailové zprávy jsou úspěšně vyfiltrovány podle zadaného řetězce.
004	Odeslání e-mailu	E-mail byl úspěšně odeslán.
005	Ohlášení uživatele z poštovní služby	Uživatel je úspěšně odhlášen

Tabulka 5 Testovací scénář, zdroj: autor

4.5 Testování pomocí Selenium IDE

Nástroj Selenium IDE není potřeba instalovat. Lze stáhnout přímo a nakonfigurovat jako rozšíření prohlížeče Chrome. Poskytuje možnost automaticky zaznamenávat testovací případy na základě interakce s prohlížečem. Selenium IDE je rozděleno do několika částí, z nichž každá má své vlastnosti a funkce.

Panel nabídek je umístěn v horní části rozhraní Selenium IDE. Obsahuje název projektu a také umožňuje přejmenovat celý projekt, načíst jakýkoli existující projekt z disku nebo uložit již rozpracovaný projekt. Panel nástrojů obsahuje funkce pro řízení testovacích případů. Mezi nejčastěji používané funkce panelu nástrojů patří modifikace rychlosti provádění testovacích případů, spuštění aktuálně vybraného testu nebo celé testovací sady. Adresní řádek poskytuje rozbalovací nabídku, která obsahuje všechny předchozí hodnoty webových stránek neboli adresy URL, které byly dříve navštíveny a testovány. Panel testovacích případů zahrnuje všechny testovací případy, které IDE zaznamenává, takže uživatel mezi nimi může snadno procházet. Testovací pole zobrazuje veškeré uživatelské interakce, které IDE zaznamenalo. Každá interakce je zaznamenána ve stejném pořadí, v jakém byla provedena. Testovací pole je rozděleno do 3 sloupců: Příkaz, Cíl a Hodnota. Příkaz lze považovat za skutečnou akci, která se provádí na prvcích prohlížeče. Cíl určuje webový prvek, na kterém je akce provedena. Hodnota je vyplněna v případě, když odesíláme data neboli parametry. Pokud například zadáváme e-mailovou adresu, hodnota bude obsahovat vyplněné přihlašovací údaje. Tlačítko REC, které se nachází přímo vedle panelu nástrojů zaznamenává veškeré akce v prohlížeči, které jsou prováděné uživatelem. Poslední část Selenium IDE, která se nachází ve spodní části představuje panel, který je rozdělen do dvou částí: Log a Reference. Okno Log zobrazuje informační, chybové a varovné zprávy. Reference okno obsahuje úplný detail aktuálně vybrané interakce z testovacího pole.



Obrázek 16 Prostředí Selenium IDE zdroj: autor

4.6 Tvorba automatizovaných testů pomocí Selenium WebDriver

Abychom mohli testovat webovou aplikaci, potřebujeme znát identifikaci jednotlivých webových prvků, jako jsou například textová pole, tlačítka a další. Lokátory (angl. Locators) nám pomáhají najít a identifikovat konkrétní webový prvek. Lokátor každého prvku je zahrnut v HTML kódu stránky. Pokud ho chceme zjistit, klikneme na daný prvek pravým tlačítkem a vybereme možnost Prozkoumat. Otevře se okno obsahující veškerý specifický kód související s daným prvkem. WebDriver poskytuje k nalezení prvků metodu *findElement()*. Metoda findElement() přebírá jeden parametr, kterým je lokátor prvku.
driver.findElement(By.<Locator>);

Ve výše uvedeném příkazu je *By* třídou, kde můžou různé metody identifikovat prvek. Vzhledem k tomu, že na webové stránce HTML existují různé typy lokátorů, je třeba jej specifikovat k identifikaci webového prvku následovně:

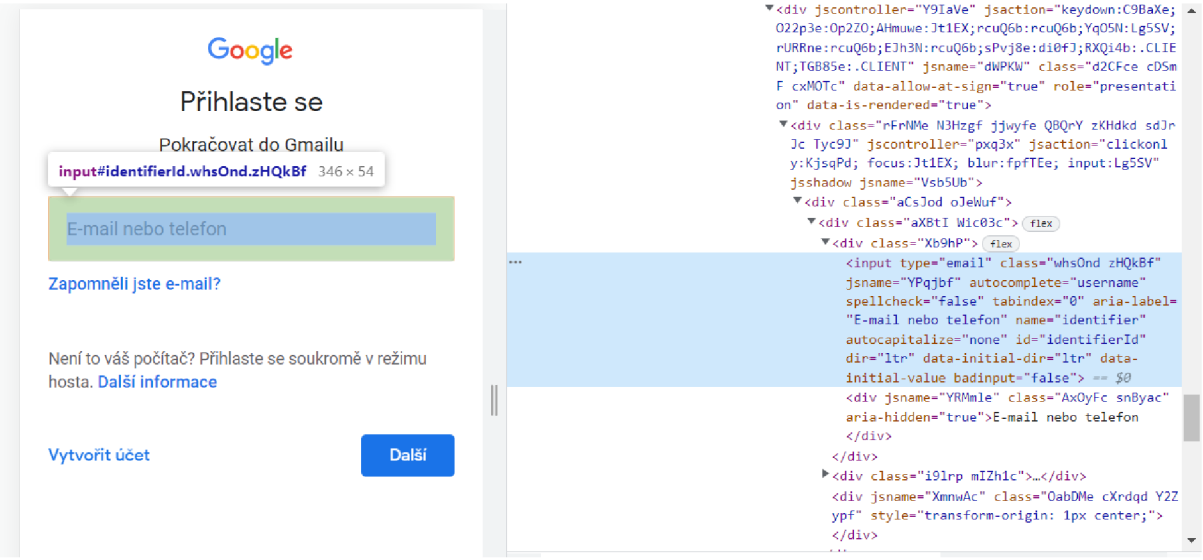
1. id – identifikace prvku pomocí id je nejpreferovanějším a nejefektivnějším způsobem. Id jsou obvykle jedinečná.
   findElement(By.id("nejakeID"))

2. name – lokátor name nalezne prvek s odpovídajícím jménem. Pokud na stránce existuje několik prvků se stejným jménem, je lepší vybrat jiný typ lokátoru.
   findElement(By.name("nejakeJmeno"))

3. class name – lokátor názvu třídy se používá stejně jako id nebo name. Pod stejným názvem třídy může existovat několik prvků, proto je lepší jej použít jen v případě, kdy název třídy identifikuje pouze jeden odpovídající prvek na stránce.
   findElement(By.className("nazevTridy"))

4. CCS selektor – identifikace webových prvků pomocí vlastností CSS se provádí pomocí ccsSelector lokátoru. CCS (Cascading Style Sheets) je jazyk, který popisuje styl a strukturu HTML stránky.
   findElement(By.ccsSelector("ccsSelectorxxx"))

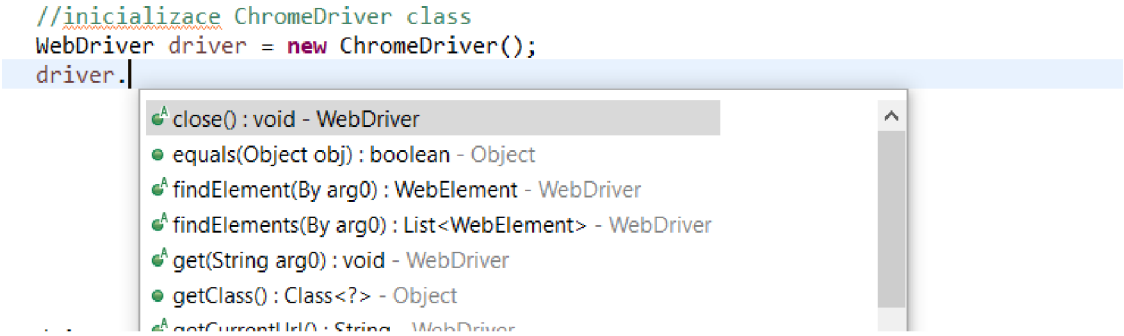
5. XPath – v případě, že neexistuje žádný jiný způsob, jak jednoznačně identifikovat vhodné ID nebo jméno pro prvek, který je třeba lokalizovat, pak se použije XPath lokátor. XPath je zkratka pro XML Path Language. Poskytuje způsob adresování různé části dokumentu XML, přičemž používá celý zápis cesty pro navigaci v hierarchické struktuře. Všechny lokátory XPath začínají na „/“. *findElement(By.xpath(“//xxxx”))*



Obrázek 17 Vyhledávání prvku HTML na webové stránce pomocí lokátoru zdroj: autor

4.6.1 Příkazy Selenium WebDriver

Příkazy jsou nedílnou součástí tvorby a spouštění testů. Jelikož používáme Selenium WebDriver s Javou, příkazy jsou jednoduché metody napsané v jazyce Java. Jedním z možných způsobů, jak zobrazit všechny dostupné metody, je vytvořit objekt ovladače pro WebDriver a stisknout klávesu s tečkou. Zobrazí se nám veškeré poskytované metody.



Obrázek 18 metody poskytované WebDriver, zdroj: autor

Metody mohou vracet hodnotu nebo nevracet nic (void). Pokud je za metodou void, znamená to, že nevrací žádnou hodnotu. Pokud vrací hodnotu, pak musí zobrazit typ hodnoty např. *get.Title():String*.

Níže jsou uvedené některé z nejčastěji používaných příkazů:

- Vytvoření nové instance prohlížeče Chrome
  ```
  WebDriver driver = new ChromeDriver();
  ```
- Načtení webové stránky pomocí URL adresy
  ```
  driver.get("https://www.gmail.com/");
  ```
- Vyhledání prvku a odeslání uživatelských vstupů pomocí metody sendKeys().
  ```
  driver.findElement(By.id("identifierId")).sendKeys("xxx@gmail.com");
  ```
- Vymazání uživatelských vstupů
  ```
  driver.findElement(By.name( "q" )).clear();
  ```
- Načtení dat přes libovolný prvek
  ```
  driver.findElement(By.id( "identifierId" )).getText();
  ```
- Provedení akce stisknutí tlačítka
  ```
  driver.findElement(By.id( "button" )).click();
  ```
- Navigace zpět v historii prohlížeče
  ```
  driver.navigate().back();
  ```
- Pohyb vpřed v historii prohlížeče
  ```
  driver.navigate().forward();
  ```
- Obnovení webové stránky
  ```
  driver.navigate().refresh();
  ```
- Zavření prohlížeče
  ```
  driver.close();
  ```

4.6.2 Vstupní data

Jedním z chování uživatelů, které se dá automatizovat, je vstup dat pomocí metody sendKeys. Tato metoda se používá k odeslání vstupů z klávesnice do textových polí, jako jsou znaky, čísla a symboly. Po úspěšném otevření webové stránky a nalezení webového prvku pomocí lokátoru, v tomto případě textové pole, vyhledávací okno nebo formulář, lze poslat vstupní data pomocí metody sendKeys. Stejně jako přihlašování do e-mailové poštovní služby má většina aplikací přihlašovací formulář. Každá e-mailová adresa má společný vzor, který začíná uživatelským jménem, za ním následuje znak @ a nakonec název

domény. Zde je uveden příklad pro ověření úspěšného přihlášení do poštovní služby se vstupními daty emailové adresy *riegerovaveronika1@gmail.com*:

```
driver.findElement(By.id("identifierId")).sendKeys("riegerovaveronika1@gmail.com");
```

Obrázek 19 ověření vstupů pomocí metody sendKeys, zdroj: autor

Další možnost představuje nahrání souboru. Pokud se tedy v uživatelském prostředí nachází možnost nahrát soubor, vstupní data jsou opět odeslána pomocí metody sendKeys. Nejprve musíme zkontrolovat prvek pro nahrání a poté pomocí metody procházet cestu, ve které je soubor uložen.

```
driver.findElement(By.id("identifierId")).sendKeys("C:\\Users\\riege\\Desktop\\data\\data.txt");
```

Obrázek 20 ověření vstupů pomocí metody sendKeys, zdroj: autor

4.7 Automatizované testy

Testovací skripty byly vytvořeny v programovacím jazyce Java a vývojovém prostředí Eclipse. Prvním krokem testování je tvorba testovacího scénáře (viz kapitola 4.4). Ke každému testovacímu skriptu je uvedený testovací případ, pomocí kterého byl testovací skript vytvořen. Pro příklad jsou zde uvedeny dva automatizované testy – ověření úspěšného přihlášení uživatele do poštovní služby a úspěšné odhlášení uživatele z poštovní služby. Veškeré testy vytvořené podle testovací scénáře jsou přiloženy na závěr práce.

TestID	**Funkcionalita**	**Očekávaný výsledek**
002	Přihlášení uživatele do poštovní služby	Uživatel je úspěšně přihlášen po zadání své e-mailové adresy a hesla.

Tabulka 6 Testovací případ, zdroj: autor

```
package co.edureka.selenium.demo;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo {

	public static void main(String[] args) {
```

```
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            //inicializace ChromeDriver
            WebDriver driver = new ChromeDriver();
            // Test name: prihlaseniUzivatele
             // 1 | open web site
            driver.get("https://mail.google.com/mail");
            //the system might need more time to find the element
           driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
           // 2 | setWindowSize
            driver.manage().window().maximize();
            // 3 | id=identifierId | riegerovaveronika1@gmail.com
            driver.findElement(By.id("identifierId")).sendKeys("riegerovaveroni
            ka1@gmail.com");
            // 4 | sendKeys | id=identifierId
            driver.findElement(By.id("identifierId")).sendKeys(Keys.ENTER);
            // 5 | name=password | Test12345.
            driver.findElement(By.name("password")).sendKeys("Test12345.");
            // 6 | sendKeys | name=password
            driver.findElement(By.name("password")).sendKeys(Keys.ENTER);

        }
    }
```

005	Ohlášení uživatele z poštovní služby	Uživatel je úspěšně odhlášen

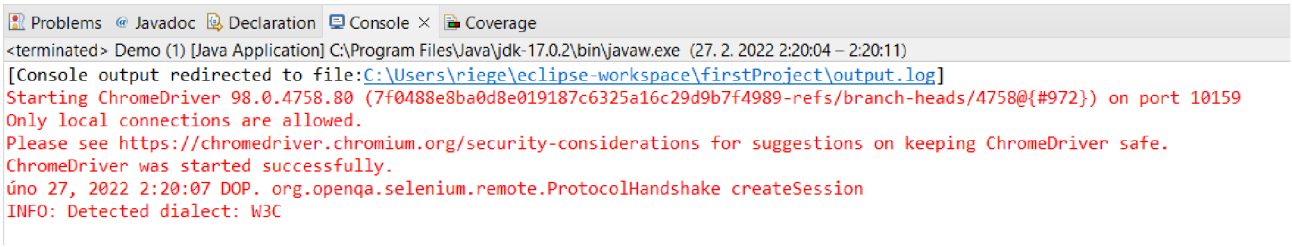
Tabulka 7 Testovací případ, zdroj: autor

```
package co.edureka.selenium.demo;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo {
      public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            //inicializace ChromeDriver
            WebDriver driver = new ChromeDriver();
            // Test name: odhlaseniiUzivatele
           // 1 | open web site
            driver.get("https://mail.google.com/mail/u/0/?pli=1");
            // 2 | setWindowSize
            driver.manage().window().maximize();
            // 3 | click | css=.gb_Aa
            driver.findElement(By.cssSelector(".gb_Aa")).click();
            // 4 | selectFrame | index=5
            driver.switchTo().frame(5);
            // 5 | click | css=.jAcX2 > .wBFtm
            driver.findElement(By.cssSelector(".jAcX2 > .wBFtm")).click();
      }
}
```

4.7.1 Zprávy a výsledky testů

Po provedení výše uvedených testů můžeme vidět jejich odpovídající výsledky. Nalezneme je v konzoli Eclipse. Výsledky prostřednictvím konzole můžeme zobrazit ve spodní části prostředí Eclipse. Mimo výstupů se zde zobrazují všechny chybové a varovné zprávy. V některých případech mohou být zprávy tak dlouhé, že jsou v konzoli oříznuty jen na poslední řádky, a tím pádem je nejsme schopni shlédnout celé. Existuje způsob, jak přesměrovat výstup konzole přímo do souboru a pohodlně si jej pak projít. Přejdeme do nabídky Run -> Configuration a přejdeme na kartu Common. V části standardní výstup zaškrtneme políčko Output File, zadáme název souboru a cestu, kam chceme, aby se nám soubor uložil. Spustíme test a pokud jsme vše nastavili správně, na prvním řádku konzole se zobrazí zpráva o přesměrování všech výstupů do souboru.



Obrázek 21 výstup konzole v prostředí Eclipse, zdroj: autor

5 Výsledky a diskuse

5.1 Práce se Selenium IDE

Příprava pro testování pomocí nástroje Selenium IDE byla velice snadná. Nebylo zapotřebí žádná instalace, pouze jej nakonfigurovat jako rozšíření přímo do prohlížeče Chrome. Pro práci se sadou Selenium IDE nebyly zapotřebí žádné předchozí znalosti. Celý proces vytváření testovacího skriptu pomocí nahrávání uživatelských interakcí s prohlížečem proběhl bez jakýchkoli problémů nebo potíží.

5.2 Práce se Selenium WebDriver

Instalace a následná konfigurace Selenium WebDriver do prostředí Eclipse vyžaduje trochu více času, avšak proběhla v pořádku bez jakýchkoli komplikací. Samotná práce se Selenium WebDriver a tvorba automatizovaných testů proběhla bez větších problémů. Testy se spustily a proběhly podle očekávání. Pro vývoj skriptů je zapotřebí ovládat znalost v programovacím jazyce Java, avšak pokud je tato podmínka splněna, je tvorba testů poměrně snadná a intuitivní. Práci se Selenium WebDriver bych zhodnotila kladně, splnila má očekávání a pro moji práci nástroj vyhověl veškerým požadavkům.

5.2.1 Problémy při implementaci

V některých případech jsem po spuštění automatizovaného testu narazila na chybu „Nelze najít prvek“, ke které došlo v důsledku pomalého načtení webové stránky. Chování webové aplikace závisí na mnoha vnějších faktorech, jako je mimo jiné rychlost sítě. Díky těmto faktorům nelze přesně určit potřebný čas k načtení konkrétního prvku. Abych předešla těmto chybám, vložila jsem do testovacího kódu příkaz:

driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);

Tento příkaz vyvolá čekání, které informuje webový ovladač, aby pro stanovenou dobu čekal na všechny webové prvky přítomné na stránce. V uvedeném příkazu jsem zadala čekání 60 sekund, což znamená, že maximální doba čekání je 60 sekund, než se konkrétní prvek načte. Pokud se však prvek načte dříve než za 60 sekund, skript po zbývající čas nečeká a pokračuje dál.

Další problém, se kterým jsem se v průběhu testování setkala, byla nesrovnalost mezi verzí ovladače a aktuální verzí prohlížeče. V průběhu instalace jsem pro testovaný prohlížeč, v našem případě Chrome, naistalovala ovladač ve verzi 98.0.4758.80. Tato verze byla poslední vydanou a shodovala se s aktuální verzí prohlížeče Chrome. Pokud se v prohlížeči Chrome vyskytne nová verze, automaticky se aktualizuje. Tak se stalo i v mém případě a některé testy v důsledku neshody verzí neproběhly v pořádku. V konzoli vývojové prostředí Eclipse se objevila odpovídající varovná zpráva: `[1647044608.962][WARNING]: This version of ChromeDriver has not been tested with Chrome version 99.` V tomto případě je však nutné nainstalovat nejnovější verzi ovladače.

5.3 Využití v praxi

Ve vybraném scénáři, který zahrnoval testování poštovní služby se výběr Selenium osvědčil. Pro testování všech obdobných i rozšířenějších webových aplikací bych tento nástroj rozhodně doporučila. Je nutné brát v potaz, že je pro tvorbu testů pomocí Selenium WebDriver nutné mít předem požadované znalosti. Pro všechny uživatele nabízí Selenium z oficiálních webových stránek několik možností, ze kterých lze v případě obtíží vyhledat pomoc. Pokud si při vývoji automatických testů nevíme rady, můžeme na daný problém vznést dotaz nebo o něm s ostatními účastníky prostřednictvím chatu diskutovat. I když je nástroj bezplatný, zaškolení uživatelů od úplných začátků může být ve výsledku také velice nákladné. Zaškolení však představuje opakované výdaje po určitou dobu a poté už je pro nás tvorba testů bezplatná. Pokud zvolíme placený nástroj, výdaje spojené s vývojem automatizovaných testů nás provázejí od úplného začátku až po jeho ukončení. Selenium je určené pouze pro testování webových aplikací, pokud bychom chtěli provádět i jiné druhy testů, bylo by nutné zvolit jiný nástroj.

6 Závěr

V teoretické části práce jsem se zaměřila na oblast testování softwaru. Veškeré studie odborných informačních zdrojů byly podkladem pro zpracování vlastní práce.

Hlavním cílem práce byla analýza možností automatického testování a výběr vhodně zvoleného nástroje pro testování ve zvoleném scénáři. Tohoto bylo docíleno především v kapitole 4.2, která zahrnovala analýzu a srovnání vybraných testovacích nástrojů a výběr vhodné varianty v rámci stanovených požadavků. Mezi vybrané nástroje patřil Randoop, Test Studio, Selenium a Appium. Ze všech nástrojů nejlépe splňoval požadované vlastnosti nástroj Selenium. Následně byla popsána a provedena instalace vývojového prostředí Eclipse, které je určené pro vývoj v programovacím jazyce Java, instalace nástroje Selenium a jeho konfigurace do vývojového prostředí Eclipse.

Dílčím cílem práce byla definice testovacího scénáře, kterého bylo docíleno v kapitole 4.4. Ta obsahovala tvorbu testovacího scénáře, který zahrnuje veškeré testovací případy, pomocí kterých byly vytvářeny testovací skripty.

Poslední část práce zahrnovala samotnou implementaci automatizovaného testování. Pro tuto práci jsem vybrala široce používanou webovou stránku, na které bylo realizováno testovaní podle vytvořeného scénáře. Jedná se o poštovní službu Gmail. Testování bylo zprvu provedeno prostřednictvím nástroje Selenium IDE pomocí funkce nahrávání a spouštění testů. Jednotlivé kroky a interakce s prohlížečem byly zaznamenány a byly tak vytvořeny testovací případy. Poté bylo testování provedeno pomocí Selenium WebDriver. Testovací skripty byly vytvořeny v programovacím jazyce Java a vývojovém prostředí Eclipse. Tvorba automatizovaných testů proběhla bez větších problému. Výběr zvoleného nástroje Selenium se osvědčil.

Jako hlavní přínos práce bych uvedla obecné shrnutí základů testování a představení vybraných testovacích nástrojů. Výběru vhodně zvoleného nástroje, který se především osvědčil. Dále tento výzkum může sloužit jako podklad pro obdobné situace při vývoji automatického testování.

Práce lze v budoucnu rozšířit o několik dalších nástrojů pro tvorbu automatizovaného testování, představení práce a srozumitelných vzorových případů pro práci s nimi. Vybrané nástroje však mohou mít v důsledku vývoje nové funkce, proto je lze případně zmínit a o nové funkcionality doplnit.

7 Seznam použitých zdrojů

1. Barnum, C. M. a. D. C., 2011. *Usability testing essentials: ready, set– test. 2nd ed.*. Burlington: MA: Morgan Kaufmann Publishers, 480s. ISBN 978-0123750921.
2. Barry, B. W., 1988. *A Spiral Model of Software Development and Enhancement.* Redondo Beach: TRW Defense Systems Group.
3. BrowserStack, 2021. *BrowserStack.* [Online] Dostupné z: https://www.browserstack.com/guide/types-of-software-bugs
4. Bureš, M., Renda, M. & Doležal, M., 2016. *Efektivní testování softwaru.* Praha: Grada Publishing. 229s. ISBN 978-80-247-5594-6.
5. Corporation, I., 2022. *Software Testing Methodologies - Learn The Methods & Tools.* [Online] Dostupné z: https://www.inflectra.com/ideas/topic/testing-methodologies.aspx
6. Corporation, P. S., 2022. *Welcome to Test Studio.* [Online] Dostupné z: https://docs.telerik.com/teststudio/welcome
7. DataArt, 2019. *QA for beginners: Automated Testing vs. Manual Testing.* [Online] Dostupné z: https://www.dataart.com.ar/news/qa-for-beginners-automated-testing-vs-manual-testing/
8. Gatson, S., 2015. *The Software Testing Ice Cream Cone.* [Online] Dostupné z: https://saeedgatson.com/the-software-testing-ice-cream-cone/
9. GeeksforGeeks, 2020. *Difference between Static and Dynamic Testing.* [Online] Dostupné z: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing/
10. Haidar, B., 2020. *Visual Testing with Applitools, Appium, and Amazon AWS Device Farm.* [Online] Dostupné z: https://applitools.com/blog/visual-testing-appium-amazon/
11. Hlava, T., 2011. *Fáze a úrovně provádění testů.* [Online] Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/
12. Hamilton Thomas, 2022. *Automated Testing Vs. Manual Testing. What's the Difference?* [Online] Dostupné z: https://www.guru99.com/difference-automated-vs-manual-testing.html

13. Insight, 2021. *The Types of Software Development Models.* [Online] Dostupné z: https://ca.insight.com/en_CA/content-and-resources/2016/07152016-types-of-software-development-models.html#waterfall-model
14. ISO25000, 2021. *ISO/IEC 25010.* [Online] Dostupné z: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=0
15. JavaTpoint, 2021. *Selenium Integrated Development Environment (IDE).* [Online] Dostupné z: https://www.javatpoint.com/selenium-ide
16. JavaTpoint, 2021. *What is Selenium Grid?.* [Online] Dostupné z: https://www.javatpoint.com/selenium-grid
17. JavaTpoint, 2021. *Selenium Tutorial.* [Online] Dostupné z: https://www.javatpoint.com/selenium-tutorial
18. JavaTpoint, 2021. *Selenium WebDriver.* [Online] Dostupné z: https://www.javatpoint.com/selenium-webdriver
19. JavaTpoint, 2021. *Test Case.* [Online] Dostupné z: https://www.javatpoint.com/test-case
20. Otta, J., 2016. *Testování v procesu implementace informačního systému.* [Online] Dostupné z: https://www.systemonline.cz/erp/testovani-v-procesu-implementace-informacniho-systemu.htm
21. Pai, A., 2021. *Appium Tutorial for Mobile Application Testing.* [Online] Dostupné z: https://www.browserstack.com/guide/appium-tutorial-for-testing
22. Patton, R., 2002. *Testování softwaru.* Praha: Computer Press. 313 s. ISBN 80-7226-636-5.
23. Randoop, 2022. *Randoop Manual.* [Online] Dostupné z: https://randoop.github.io/randoop/manual/
24. Rungta, K., 2017. *Learn Selenium in 1 Day: Definitive Guide to Learn Selenium for Beginners.* 504 s. ISBN B071YTMYXC.
25. Sakshi Dewan, S. t., 2015. *Difference Between Static Testing And Dynamic Testing.* [Online] Dostupné z: https://www.softwaretestingclass.com/difference-between-static-testing-and-dynamic-testing/
26. Section, 2021. *Mobile Automation Testing with Appium - An Introduction.* [Online] Dostupné z: https://www.section.io/engineering-education/mobile-automation-testing-with-appium/

27. Tutorialspoint, 2021. *Software Testing - Documentation.* [Online] Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_documentation.htm
28. Tutorialspoint, 2021. *Software Testing - Levels.* [Online] Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm
29. Tutorialspoint, 2021. *Software Testing - Methods.* [Online] Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm
30. Vskills, 2020. *Software Testing Tutorial | Testing Methods – The box approach.* [Online] Dostupné z: https://www.vskills.in/certification/tutorial/testing-methods-the-box-approach/
31. Watirmelon, 2018. *Testing pyramits.* [Online] Dostupné z: https://watirmelon.blog/testing-pyramids/
32. Watirmelon, 2012. *Ice Cream cone.* [Online] Dostupné z: http://watirmelon.com/tag/ice-cream-cone/
33. Anon., nedatováno *Spirálový model.* [Online] Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

8 Přílohy

```
public class Demo {

	public static void main(String[] args) {
		System.setProperty("webdriver.chrome.driver",
		"C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
		_win32\\chromedriver.exe");
		WebDriver driver = new ChromeDriver();

		driver.get("https://accounts.google.com/signup/v2/webcreateaccount?
		service=mail&continue=https%3A%2F%2Fmail.google.com%2Fmail%2Fu%2F0%
		2F&biz=false&flowName=GlifWebSignIn&flowEntry=SignUp");
		driver.manage().window().maximize();
		driver.findElement(By.id("firstName")).sendKeys("Veronika");
		driver.findElement(By.id("lastName")).click();
		driver.findElement(By.id("lastName")).sendKeys("Riegerová");
		driver.findElement(By.id("username")).click();

		driver.findElement(By.id("username")).sendKeys("riegerovaveronika1"
		);
		driver.findElement(By.name("Passwd")).click();
		driver.findElement(By.name("Passwd")).sendKeys("Test12345.");
		driver.findElement(By.name("ConfirmPasswd")).click();

		driver.findElement(By.name("ConfirmPasswd")).sendKeys("Test12345.")
		;
		driver.findElement(By.cssSelector(".VfPpkd-LgbsSe-OWXEXe-k8QpJ >
		.VfPpkd-vQzf8d")).click();
		driver.findElement(By.id("phoneNumberId")).click();
		driver.findElement(By.id("phoneNumberId")).sendKeys("724828935");
		driver.findElement(By.id("day")).click();
		driver.findElement(By.id("day")).sendKeys("13");
		driver.findElement(By.id("month")).click();
		{
			WebElement dropdown = driver.findElement(By.id("month"));
			dropdown.findElement(By.xpath("//option[. =
			'květen']")).click();
		 }
		driver.findElement(By.id("year")).click();
		driver.findElement(By.id("year")).sendKeys("1998");
		driver.findElement(By.id("gender")).click();
		{
			WebElement dropdown = driver.findElement(By.id("gender"));
			dropdown.findElement(By.xpath("//option[. =
			'Žena']")).click();
		}
		driver.findElement(By.cssSelector(".VfPpkd-LgbsSe-OWXEXe-k8QpJ >
		.VfPpkd-RLmnJb")).click();
		driver.findElement(By.cssSelector(".qs41qe .t5nRo")).click();
		driver.findElement(By.cssSelector(".VfPpkd-vQzf8d")).click();
		driver.findElement(By.cssSelector(".qs41qe .t5nRo")).click();
		driver.findElement(By.cssSelector(".qs41qe .t5nRo")).click();
		driver.findElement(By.cssSelector(".VfPpkd-ksKsZd-mWPk3d-OWXEXe-
		Tv8l5d-lJfZMc > .VfPpkd-RLmnJb")).click();
```

```
            driver.findElement(By.cssSelector(".qs41qe .t5nRo")).click();
            driver.findElement(By.cssSelector(".VfPpkd-ksKsZd-mWPk3d-OWXEXe-
            Tv8l5d-lJfZMc > .VfPpkd-vQzf8d")).click();
            driver.findElement(By.cssSelector(".qs41qe .t5nRo")).click();
            driver.findElement(By.cssSelector(".VfPpkd-ksKsZd-mWPk3d-OWXEXe-
            Tv8l5d-lJfZMc > .VfPpkd-RLmnJb")).click();
            driver.findElement(By.id("c27")).click();
            driver.findElement(By.cssSelector(".VfPpkd-ksKsZd-mWPk3d-OWXEXe-
            Tv8l5d-lJfZMc > .VfPpkd-vQzf8d")).click();
            driver.findElement(By.cssSelector(".VfPpkd-ksKsZd-mWPk3d-OWXEXe-
            Tv8l5d-lJfZMc > .VfPpkd-vQzf8d")).click();
            driver.findElement(By.cssSelector(".VfPpkd-LgbsSe-OWXEXe-k8QpJ >
            .VfPpkd-vQzf8d")).click();

      }

}
```

Příloha č. 1 Testovací script: Vytvoření nového účtu

```
public class Demo {

      public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            WebDriver driver = new ChromeDriver();
            driver.get("https://mail.google.com/mail");
            driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
            driver.manage().window().maximize();
            driver.findElement(By.id("identifierId")).sendKeys("riegerovaveroni
            ka1@gmail.com");
            driver.findElement(By.id("identifierId")).sendKeys(Keys.ENTER);
            driver.findElement(By.name("password")).sendKeys("Test12345.");
            driver.findElement(By.name("password")).sendKeys(Keys.ENTER);

      }
}
```

Příloha č. 2 Testovací script: Přihlášení uživatele do poštovní služby

```
public class Demo {

      public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            WebDriver driver = new ChromeDriver();
            driver.get("https://mail.google.com/mail/u/0/?zx=ehu2fxox4r5b");
            driver.manage().window().maximize();
            driver.findElement(By.name("q")).click();
            driver.findElement(By.cssSelector(".gsas-d")).click();
```

```
            driver.findElement(By.name("q")).sendKeys("důležité");
      }
}
```

Příloha č. 3 Testovací script: Vyhledávání v poštovní službě pomocí filtru

```
public class Demo {

      public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            WebDriver driver = new ChromeDriver();
            driver.get("https://mail.google.com/mail/u/0/");
            driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
            driver.manage().window().maximize();
            driver.findElement(By.cssSelector(".T-I-KE")).click();
            driver.findElement(By.id(":a5")).sendKeys("riegerovaveronika1@gmail
            .com");
            driver.findElement(By.id(":9n")).click();
            driver.findElement(By.id(":9n")).sendKeys("test");
            driver.findElement(By.id(":at")).click();
            {
                  WebElement element = driver.findElement(By.id(":at"));
                   js.executeScript("if(arguments[0].contentEditable ===
                   'true') {arguments[0].innerText = 'Toto je testovací
                   email'}", element);
                }
                driver.findElement(By.id(":9d")).click();
      }
}
```

Příloha č. 4 Testovací script: Odeslání e-mailu

```
public class Demo {
      public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
            "C:\\Users\\riege\\Desktop\\selenium_release\\drivers\\chromedriver
            _win32\\chromedriver.exe");
            WebDriver driver = new ChromeDriver();
            driver.get("https://mail.google.com/mail/u/0/?pli=1");
            driver.manage().window().maximize();
            driver.findElement(By.cssSelector(".gb_Aa")).click();
            driver.switchTo().frame(5);
            driver.findElement(By.cssSelector(".jAcX2 > .wBFtm")).click();
       }
}
```

Příloha č. 4 Testovací script: Ohlášení uživatele z poštovní služby