Česká zemědělská univerzita v Praze

Technická fakulta

Diplomová práce

Automatizační nástroje pro testování online aplikací

Vedoucí diplomové práce:
Ing. Jan Lešetický, Ph. D.

Autor:
Bc. Lukáš Pazderka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Technická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Lukáš Pazderka

Informační a řídicí technika v agropotravinářském komplexu

Název práce

Automatizační nástroje pro testování online aplikací

Název anglicky

Automation tools for testing online applications

Cíle práce

Cílem práce je analyzovat a na základě provedeného rozboru realizovat automatizační nástroj pro testování vybraných metrik na webových stránkách. Pro vlastní realizaci je potřeba definovat pojmy a popsat dostupné metody, technologie a nástroje zabývající se zvolenou problematikou. V praktické části navrhnout testovací procesy a realizovat je ve zvoleném nástroji.

Metodika

1. Úvod
2. Cíl práce
3. Metodika práce
4. Metody a metriky testování
5. Přehled skriptovacích jazyků
6. Přehled testovacích nástrojů
7. Zhodnocení výběru pro praktickou realizaci
8. Návrh praktického řešení a realizace
9. Závěr

Doporučený rozsah práce

50 – 60

Klíčová slova

testování, testovací nástroje, skriptovací jazyky, automatizace

---

Doporučené zdroje informací

GUNDECHA, U., Learning Selenium Testing Tools with Python, Packt Publishing, 2014, ISBN 9781783983506
PAPAZOGLOU, M., Web Services: Principles and Technology, Prentice Hall, 2008, ISBN 9780321155559
PATTON, R., Testování softwaru, Computer Press, 2002, ISBN 8072266365
ROUDENSKÝ, P., HAVLÍČKOVÁ, A., Řízení kvality softwaru, COMPUTER PRESS, 2013, ISBN 9788025138168

Předběžný termín obhajoby

2021/2022 LS – TF

Vedoucí práce

Ing. Jan Lešetický, Ph.D.

Garantující pracoviště

Katedra technologických zařízení staveb

Konzultant

Ing. Marek Pačes

Elektronicky schváleno dne 3. 2. 2021

doc. Ing. Jan Malaťák, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 2. 2021

doc. Ing. Jiří Mašek, Ph.D.

Děkan

V Praze dne 24. 03. 2022

Čestné prohlášení

Já, Lukáš Pazderka prohlašuji, že jsem diplomovou práci zpracoval samostatně pod vedením Ing. Jana Lešetického, Ph.D. a vypsal veškeré zdroje informací, které byly použity.

Jsem si vědom, že odevzdáním diplomové práce souhlasím s jejím zveřejněním dle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů, ve znění pozdějších předpisů, a to i bez ohledu na výsledek její obhajoby.

Jsem si vědom, že moje diplomová práce bude uložena v elektronické podobě v univerzitní databázi a bude veřejně přístupná k nahlédnutí.

Jsem si vědom, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, ve znění pozdějších předpisů, především ustanovení § 35 odst. 3 tohoto zákona, tj. o užití tohoto díla.

V Praze dne . . . .

Lukáš Pazderka

Poděkování

Tímto bych chtěl poděkovat především Ing. Marku Pačesovi za směrování a důležité rady, které měly hlavní dopad na výsledek práce.

Abstrakt: V literární rešerši jsou popsána témata zabývající se testováním softwaru, testovacími nástroji a skriptovacími jazyky. Témata testování softwaru a testovací nástroje jsou popsány v širším kontextu, aby byla problematika dostatečně vysvětlena. V případě testování softwaru se jedná o metody, metriky, úrovně a typy testování. U testovacích nástrojů jsou to základní kategorie, jednotkové nástroje, integrační nástroje a API nástroje, GUI/E2E nástroje, výkonnostní nástroje, exploratory nástroje, nástroje pro test management a generátory testovacích dat. Následuje popis a charakteristiky skriptovacích jazyků JavaScript, Perl, PHP, Python a TypeScript. Praktická část začíná výběrem technologií pro vytvoření automatizovaného testovacího nástroje. Ve vybrané oblasti testování softwaru jsou porovnávány testovací nástroje, které spadají do této oblasti. Porovnání spočívá v možnostech, které nástroje nabízí, a také zkušenostech autora. Po výběru technologií se realizuje vytvoření automatizovaného testovacího nástroje. Pro realizaci je nutné mít nainstalovaný daný skriptovací jazyk, potřebné knihovny a vývojové prostředí. Po vytvoření testovacího nástroje je jeho celkové řešení tzn. návrh, potřebné instalace, návody a rozbor zdrojového kódu zapsané do práce.

Klíčová slova: testování; testovací nástroje; skriptovací jazyky; Selenium; Python

Automation tools for testing online applications

Abstract: In literal research are described topics dealing with testing software, testing tools and scripting languages. Topics testing software and testing tools are described in a border context so that issues are sufficiently explained. In case of software testing i.e., methods, metrics, levels and types of testing. For testing tools i.e., basic categories, unit tools, integration and API tools, GUI/E2E tools, performance tools, test management tools and generators of testing data. This is followed by a description and characteristics of the scripting languages JavaSript, Perl, PHP, Python, and TypeScript. The practical part begins with the selection of technologies to create automated test tool. Testing tools that fall within a selected area of software testing are compared. The comparison is based on the capabilities offered by the tools and the experience of the author. After the selection of the technologies, the creation of the automated test tool is implemented. For implementation, it is necessary to have the scripting language, the necessary libraries and the development environment installed. After the creation of the test tool, its overall solution i.e., design, necessary installations, instructions, source code analysis is written into the work.

Key words: testing; testing tools; scripting languages; Selenium; Python

SEZNAM POUŽITÝCH ZKRATEK

SW	Software
OOP	Objektově orientované programování
UAT	User acceptance testing
GUI	Graphic user interface
API	Application programming interface
E2E	End to end
JS	JavaScript
HTML	Hypertext Markup Language
APM	Application performace monitoring

SEZNAM ROVNIC

SEZNAM TABULEK

SEZNAM OBRÁZKŮ

Obsah

1 ÚVOD

Automatizace celosvětově zaznamenává rozkvět a firmy si začínají uvědomovat, že v této oblasti je budoucnost. Mezi primární benefit patří to, že program, resp. stroj dokáže provádět práci za člověka. To řeší dva velmi podstatné problémy. Prvním je, že společnost nemusí na tuto práci najímat člověka, neboť v dnešní době především v České republice je velice těžké nacházet nové zaměstnance v oblasti IT. Druhým je, že společnost nahradí zaměstnance programem a tento člověk může být přesunut a využit efektivněji. Sekundárními benefity jsou ušetření finančních prostředků, zrychlení pracovních procesů a s tím spojené množství vykonané práce. Možnými negativy jsou finanční náročnost investice a čas potřebný pro implementaci automatizačních procesů v této oblasti.

Automatizované testování webových aplikací je spojení automatizace a testování softwaru (dále jen SW) v online prostředí. Automatizace firemních procesů je v posledních letech velmi žádoucí, a jedná se o aktuální téma především v oblasti průmyslu a SW.

Společnosti mají větší požadavky na SW řešení a s tím roste i jeho komplexnost. Čím komplexnější, tím náročnější na údržbu. Do údržby se řadí i oblast testování SW, která hledá v SW nedostatky. V případě komplexního SW je potřeba disponovat většími kapacitami testerů, aby měl takový SW dostatečnou kvalitu. Tyto kapacity by bylo možno využít jinde, pokud by firma měla SW nástroj, který by dokázal automaticky otestovat určitou oblast SW.

Na toto téma jsem se zaměřil, jelikož se od roku 2020 pracovně věnuji manuálnímu testování a studuji obor, který se zabývá průmyslovou automatizací, dále vidím v tomto odvětví velký potenciál. Jelikož se spíše zajímám o oblast SW, resp. programovaní, vybral jsem si téma automatizované testování webových aplikací. Mojí motivací je seznámit veřejnost s touto oblastí a rozšířit si vlastní obzory.

2 CÍLE PRÁCE

Cílů má tato práce několik. Prvotním cílem je fundamentální rozbor oblasti testování, testovacích nástrojů a skriptovacích jazyků. Dále na základě těchto rozborů vybrat technologie, které budou použity k vlastní realizaci nástroje. V poslední řadě dle vybraných technologií vytvořit automatizační nástroj podle návrhu a následně popis nástroje, vytvoření variant testovacích scénářů v nástroji.

3 METODIKA PRÁCE

Teoretická část práce se zaměřuje na seznámení s oblastmi testování SW, testovacích nástrojů a skriptovacích jazyků. V případě testování SW se práce zaměřuje na metody testování, testovací metriky, úrovně testování, typy testování. U testovacích nástrojů se práce zabývá základními kategoriemi, nástroji pro jednotkové testování, integrační/application programming interface (dále jen API) neboli rozhraní pro programování aplikací testování, GUI/End to end (dále jen E2E) testování, výkonnostní testování, exploratory testování, test management testování a generátory dat na testování. U skriptovacích jazyků jsou rozebrány jazyky JavaScript (dále jen JS), Perl, PHP, Python, TypeScript.

Praktická část začíná výběrem technologií pro vytvoření automatizovaného testovacího nástroje. Ve vybrané oblasti testování SW se porovnávají testovací nástroje, které spadají do této oblasti. Porovnání spočívá v možnostech, které nástroje nabízí a také zkušenostech autora. Po výběru technologií se realizuje vytvoření automatizovaného testovacího nástroje. Pro realizaci je nutné mít nainstalovaný daný skriptovací jazyk, potřebné knihovny a vývojové prostředí. Po vytvoření testovacího nástroje je jeho celkové řešení (tzn. návrh, potřebné instalace, návody, rozbor zdrojového kódu) zapsáno do práce.

4 TESTOVÁNÍ WEBOVÝCH APLIKACÍ

Testování webových aplikací je testování online SW, resp. aplikací (dále jen SW), kdy je snahou testera nalézt chyby. Při testování SW se nabízí mnoho možností, resp. mnoho testovacích scénářů, přičemž tester nemůže pokrýt celou množinu těchto testovacích scénářů pro rozsáhlejší SW. V takovém případě tester nemůže říct, že SW funguje správně, ale pouze, že nenašel žádnou chybu. Dále je možné říct, že testovací scénář odpovídá jednomu testu a zároveň se jedná o ekvivalent testovacího případu.

Testování SW může probíhat různými metodami, dle kterých se testování řídí. Testovací metriky umožňují zaznamenávat nalezené chyby a dle nalezených chyb uzpůsobit samotný vývoj aplikace. Aby testování bylo snadnější, byly vytvořeny úrovně testování, kdy se systematicky otestuje celý SW tak, aby jeho chybovost byla co nejmenší a dosahoval požadované kvality. K testování SW patří také jednotlivé typy testování, které se věnují testování specifické oblasti SW a probíhají při různých metodách a úrovních.

4.1 Metody testování softwaru

Určení metody testování se odvíjí od skutečnosti kdo, jak a v jaké fázi bude SW testovat.

4.1.1 Statické a dynamické techniky

Statické techniky se využívají v raných fázích vývoje SW, resp. aplikací a není potřeba, aby byl daný SW v běhu nebo funkční. Ve statických technikách se praktikuje např. neformální review, analýza dokumentů, technické review a statická analýza. Specifikace požadavků nebo analýza zdrojového kódu jsou důvody, proč se tyto techniky praktikují.

Dynamické techniky jsou do jisté míry opakem statických, jelikož se využívají v pozdějších fázích vývoje a požadují, aby aplikace byla spuštěna. Zahrnuje je i provoz samotné aplikace. Tyto techniky lze rozdělit na metody jako např. White box, Black box a Grey box testování. **[2]**

4.1.2 White box, black box a grey box metody

Následující metody jsou rozděleny podle toho, jakými informacemi a zkušenostmi je třeba disponovat, aby mohlo být provedeno testování.

Pod pojmem white box neboli bílá skříňka si můžeme představit průhlednou skříňku, ve které je vložen zdrojový kód aplikace a tester vidí přesně každou část kódu. To znamená, že je umožněn veškerý přístup, jak k informacím, tak ke zdrojovému kódu. Přístupem k informacím a zdrojovému kódu se testerovi naskýtá větší škála možností testování např. revize kódu bez spuštění aplikace, optimalizace kódu nebo zamezení funkčním chybám. Je-li testerovi známa i vnitřní logika, můžou být testovány situace, které uživatel nemusí vnímat. Jedná se o metodu, která je z hlediska využití více komplexní, a proto je potřeba, aby testeři měli širší znalosti především v oblasti programování. **[1]**

Pod pojmem black box neboli černá skříňka si můžeme představit neprůhlednou černou skříňku, ve které je vložen zdrojový kód aplikace a tester vůbec neví, co se uvnitř skrývá. To znamená, že tester nemá přístup k veškerým informacím, zdrojovému kódu a ani vnitřní logice aplikace. Jedná se spíše o uživatelské testování, kdy jsou testerovi známy vstupy a výstupy. V případě, že vyhodnocené výstupy odpovídají výstupům dle návrhu, lze říct, že aplikace pracuje správně. Pokud neodpovídají, je někde chyba. Tester z hlediska požadavků znalostí nepotřebuje znát programovací jazyk ani algoritmizaci a tím je možno Black box metodu přenechat méně zkušeným zaměstnancům. **[1]**

V případě Gray box metody se jedná o kombinaci z White box a Black box metod. Tester zná do určité míry vnitřní strukturu. Tím jsou myšleny algoritmy a vnitřní datové struktury, které testerovi slouží k návrhu testovacích scénářů. Akt testování je již na úrovni uživatele. **[6]**

4.1.3 Manuální a automatické testování

Manuální testování je prováděno především testerem nebo programátorem, a to za použití testovacích scénářů, myši a klávesnice. V současné době je cílem, aby automatizované testování nahradilo právě manuální. Manuální testování však zcela nevymizí, neboť určité testy nejdou zautomatizovat. Je zde potřeba zmínit exploratory testování, které je velice podobné klasickému manuálnímu testování, avšak v tomto případě je rozdíl ve způsobu a záměru testování. Jedná se o testování, které ze svého procesu vynechává testovací scénáře, protože jsou tyto testy spíše průzkumného charakteru. Úkolem je nalézt chyby, které vzniknou při průzkumném testování SW.

Automatizované testy se momentálně dostávají do popředí, jelikož mají několik výhod jako např. ušetření finančních prostředků a možnost provádět testování v podstatě kdykoliv. I v případě automatizovaného testování jsou zapotřebí testovací scénáře, dle kterých se naprogramuje skript. Skripty jsou tvořeny skriptovacími, resp. programovacími jazyky, kterými jsou např. Python, Ruby, JS nebo VBScript. Další možností, jak tvořit automatizační testy jsou nástroje, které se pouze nakonfigurují a spustí. **[3]**

4.1.4 Agilní testování

Agilní testování je metoda řízena principy agilního vývoje SW. To znamená, že při vývoji probíhá souběžně i testování. Testovací scénáře se vytvářejí v menší míře a samotné testování je méně strukturované. V průběhu vývoje se často mění požadavky, které SW musí splňovat a těmto změnám se musí tester přizpůsobovat. Vývojář a tester spolu spolupracují na denní bázi a mají zodpovědnost za kvalitu SW. **[7]**

4.1.5 Ad hoc testování

Můžeme se setkat s označením náhodné nebo opičí testování. Jedná se o testování, které nevyužívá dokumentaci ani plánování testů. Charakteristické pro tuto metodu je, že testy jsou prováděny náhodně, neformálně a bez jakéhokoli postupu nebo očekávaných výsledků.

Postup testera je zcela nezávazný a jedná se spíše o improvizaci, kdy každý krok je náhodný. Tím vzniká i problém s reprodukcí postupu v případě nalezení chyby, jelikož nejsou vytvořeny testovací scénáře.

Metoda se využívá při akceptačním testování a provádí se obvykle pokud je omezený čas nebo zdroje dodání SW. Úspěch při tomto testování záleží na zkušenostech, zainteresovanosti, kreativitě a vytrvalosti testera. **[8]**

4.2 Testovací metriky

Při vývoji je potřeba testovat, aby byly objeveny skryté chyby, které nejsou žádané. Dle nalezených chyb se následně odvíjí zásahy jako např. kapacity testerů a programátorů na projektu. Pokud se nalezne značné množství chyb, je žádoucí, aby manažer na tuto skutečnost zareagoval doplněním kapacit a v opačném případě je snížil. Mezi základní metriky, které se využívají, jsou založené na chybách, testech a kódu. Tyto metriky slouží k průběžnému testování SW. **[4]**

4.2.1 Chybové metriky

Chybové metriky informují o množství nalezených chyb v projektu. Jedná se o velmi nepřesný údaj (viz tab. 1), který by měl být rozšířen o další informace.

Tabulka 1:Příklad počtu nalezených chyb během testů [56]

Období	Název testovaného SW	Celkový počet chyb
20.3.-1.4.2011	Účtenka beta	150

Rozšířené informace upřesňují např. kde se chyba nachází, škálu závažnosti chyb a v jakém stavu se chyba nachází (viz tab. 2).

Tabulka 2: Rozdělení chyb použitím metrik založených na chybách [56]

Funkční oblast	Vysoká závažnost	Střední závažnost	Nízká závažnost	Neopraveno	Celkem
Správa pohledávek	4	12	16	8	32
Tisk a export	0	4	6	4	10
Osobní zprávy	1	9	25	15	35

Jednou ze základních metrik založených na chybách je tzv. Bug fix rate (viz rovnice 1).

Rovnice 1: Metrika bug fix ratu

$$Bug\ fix\ rate = \frac{množství\ opravených\ chyb}{množství\ nalezených\ chyb} * 100\ [\%]$$

Rovnice udává, kolik procent z nalezených chyb je opraveno. Do množství opravených chyb se započítávají i chyby, které byly opraveny jakýmkoli způsobem. Například i v případě, že chyba byla zamítnuta, protože se jednalo o vlastnost aplikace.

Další metrikou založené na chybách je efektivnost testu (viz rovnice 2).

Rovnice 2: Metrika efektivnosti testu

$$Efektivnost\ testu = \frac{počet\ nalezených\ chyb\ pomocí\ testu}{množství\ chyb\ celkem} * 100\ [\%]$$

Touto rovnicí se získají informace o tom, jaký z použitých testů byl nejefektivnější, resp. udává procentuálně, kolik použitý test zachytil chyb. Výsledky bývají porovnány s jinými testy nebo s dobou trvání jiného testu. Následně jsou testy vyhodnoceny jako efektivní nebo neefektivní. Pokud jsou testy neefektivní, bývají nahrazeny jinými.

Rychlost objevení chyby je další metrika (viz rovnice 3).

Rovnice 3: Metrika rychlost objevení chyby

$$Rychlost\ objevení\ chyby = \frac{množství\ nalezených\ chy}{doba\ vykonávání\ testu} * 100\ [\%]$$

Tato metrika založená na chybách informuje o průměrné rychlosti nalezení chyby za hodinu. Pravidlem bývá, že na začátku testování se objeví nejvíce chyb a v průběhu se počet chyb snižuje, přičemž na konci testování je na nejmenší úrovni. **[4]**

Jednou z dalších metrik je doba chyby, která uvádí časový rozdíl mezi datem nalezení chyby a aktuálním datem nebo datem odstranění chyby. V případě, že je chyba nalezena se následně nahlásí a přiřadí kompetentní osobě. Časový rozdíl se počítá v hodinách nebo ve dnech. Status opravy chyby znamená, že chyba byla kódově opravena a následně otestována a uzavřena. Metrika poukazuje na rychlost reakce vývojového/testovacího týmu. **[9]**

4.2.2 Testové metriky

V případě testových metrik se můžeme setkat s několika situacemi. První z nich je, že testování proběhlo v pořádku a žádné chyby nebyly nalezeny. V tomto případě víme, že je aplikace připravena ke spuštění, nebo že kód může být přemístěn z testovacího prostředí na akceptační. Druhou možnou situací je, že jsou v aplikaci chyby, které musí být opraveny. Třetím případem je, že aplikace není dostatečně pokryta testovacími scénáři. Nejedná se o neobvyklou situaci, protože s robustnějším řešením je pokrytí menší a náročnější.

Pokud je v jednom testovacím cyklu znám počet testovacích scénářů, je možné určit v jaké procentuální fázi se testování nachází (viz rovnice 4).

Rovnice 4: Metrika fáze testování

$$fáze\ testování = \frac{počet\ odtestovaných\ scénářů}{počet\ testovacích\ scénářů} * 100\ [\%]$$

To by platilo v případě, že by měl každý test stejnou váhu. Většinou testy mívají odlišnou váhu, kterou je potřeba brát v potaz. Testy se kategorizují a v případě, že má každý test jinou váhu, není ideální využívat tuto metriku pro všechny kategorie. Metrika je využívána zejména kategorií funkčních testů. **[4]**

4.2.3 Metriky programového kódu

Metrika programového kódu je také označována jako pokrytí kódu. Jak už název napovídá, jedná se o metriku z oblasti zdrojového kódu aplikace, která udává procentuálně kolik zdrojového kódu bylo otestováno a kolik je třeba otestovat.

Tato metrika může být konkretizována jednotlivými podtypy:

- Pokrytí příkazů
- Pokrytí hra
- Pokrytí podmínek
- Pokrytí cest

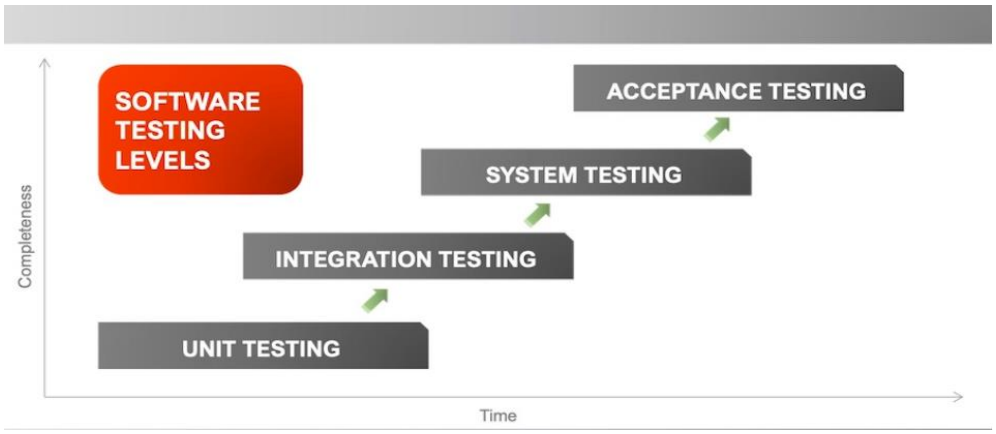
Podtypy specifikují, jak je na kód pohlíženo. Využívá se především při testování testovacích jednotek neboli unit testing. **[4]**

4.3 Úrovně testování

Vývoj SW má svůj vlastní životní cyklus, který lze dělit na různé úrovně (viz obr. 1), jež je potřeba otestovat. Mezi úrovně testování SW patří testování jednotek, integrační testování, systémové a akceptační.

Je třeba rozlišovat úrovně, metody a typy testování SW. Metody a typy se zaměřují na konkrétní vlastnosti SW. Například je možno provádět bezpečnostní testování pomocí metody White box a bude se jednat o systémovou úroveň testování. **[11]**

Obrázek 1: Úrovně testování [11]



4.3.1 Testování jednotek

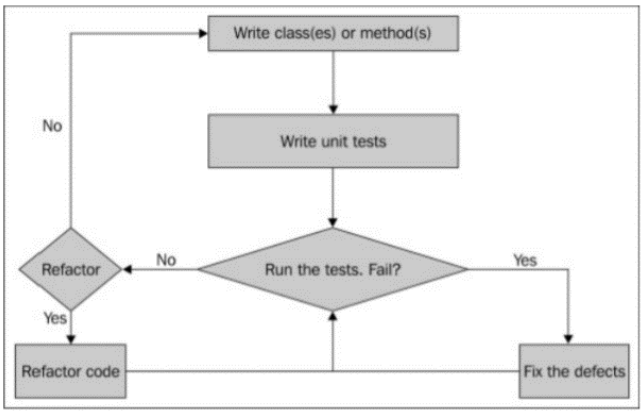
Testování jednotek neboli jednotkové testování je však více známé z anglického názvu unit testing nebo také component testing. Na této úrovni se testují jednotlivé jednotky nebo části SW, resp. kusy zdrojového kódu, které mají určitou funkcionalitu. Výsledkem je vyhodnotit, zda jednotky nebo komponenty fungují podle návrhu. **[5]**

Za jednotku je považována nejmenší testovatelná část SW, která obvykle obsahuje více vstupů s jedním výstupem. Jednotlivý program, funkce, procedura může být brána jako testovací jednotka v procedurálním programování. Metoda je opět nejmenší jednotkou u objektově orientovaného programovaní (dále jen OOP), která může být součástí abstraktní třídy, nadtřídy nebo podtřídy. K testování jednotek se využívají drivery, frameworky, falešné objekty atd. **[5]**

Jednotkové testování je většinou automatizované a využívá metodu White box. Chyby jsou opravovány zpravidla po zjištění a nebývají hlášeny ani sledovány. Jednotkové testování provádí vývojáři. Mohou je provádět i testeři, pokud mají znalosti návrhu a architektury projektu a mají přístup ke zdrojovému kódu. **[5]**

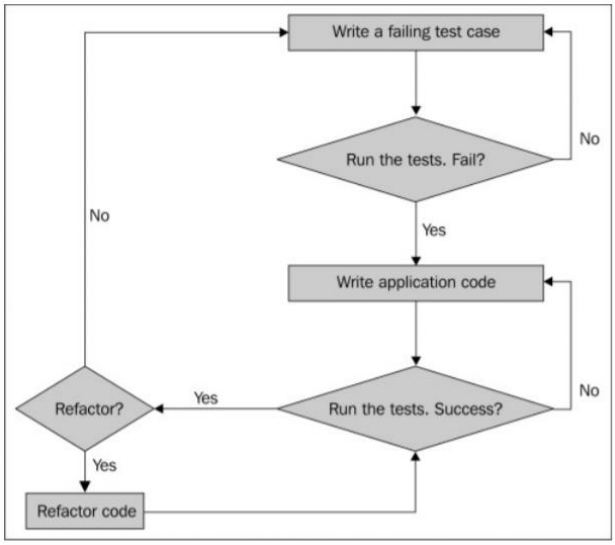
Testování jednotek může probíhat dvěma způsoby, a to Behavior Driven Development (dále jen BDD) nebo Test Driven Development (dále jen TDD). V prvním případě se testy vytvářejí standartně po vytvoření kódu aplikace (viz obr. 2). **[20]**

Obrázek 2: Algoritmus BDD [20]



V druhém případě jsou nejdříve vytvořeny a vyzkoušeny jednotkové testy. Pokud testy proběhnou v pořádku, je vytvořen potřebný kód aplikace (viz obr. 3). **[20]**

Obrázek 3: Algoritmus TDD [20]



4.3.2 Integrační testování

Integrační testování je druhou nejnižší úrovní testování SW, kdy jsou testovací jednotky spojeny a testovány jako celek. V případě integračního testování je snahou odhalit chyby, které vzniknout právě spojením testovacích jednotek, které spolu interagují. Především se jedná o chyby v integracích v systému nebo v rozhraních mezi integrovanými jednotkami.

U této úrovně testování se využívají metody White box, Black box a Gray box testování, kdy záleží na tom, co je potřeba integrovat. Testování může probíhat manuálně

i automatizovaně. Integrační testování jednotek je součástí procesu kontinuální integrace, pokud je použit iterativní nebo inkrementální vývoj.

Mezi integrační testování může být zařazeno také testování systémové integrace. Jedná se o typ testování, který se věnuje interakcím a rozhraním v rámci systémů. Systémy, které spolu budou integrovány, mohou být interní nebo externí. Takové systémy mají rozhraní pro mikroslužby a API. **[10]**

4.3.3 Systémové testování

Systémové testování je testování na systémové úrovni, kdy je testován kompletní a integrovaný SW. Na této úrovni se vyhodnocuje, zda systém odpovídá zadaným požadavkům. Testování obvykle probíhá metodou Black box, při níž není známa vnitřní struktura testovaného systému. Testování je prováděno manuálně, avšak do popředí se dostávají automatizační testy, které jsou využity zejména na regresní testování.

Jelikož se jedná o nejkomplexnější úroveň testování, je mnoho typů testů, které je možno využít jako např. smoke testování, funkční testování, regresní testování, výkonové testování, testování použitelnosti, bezpečnostní testování aj.

Systémové testování, jak už název napovídá je testování sytému tzn., že SW je testován jako celek, a proto by testovací prostředí mělo být totožné s prostředím produkčním. V opačném případě mohou nastat chyby, které nebyly brány v potaz nebo se nepodařilo je zachytit. **[11]**

4.3.4 Akceptační testování

Akceptační testování je konečná úroveň testování, ve které se posuzuje přijatelnost systému. Výsledkem testování je vyhodnocení, zda systém splňuje obchodní požadavky a je připraven na dodání na produkční prostředí.

V akceptačním testování se využívá metoda Black box a je prováděna manuálně. Na této úrovni testování nejsou striktní postupy, jak testovat. Testování probíhá spíše náhodným testováním.

Akceptační testování může mít několik typů:

- Interní
- Externí

- Testování zákazníkem
- Uživatelské testování

Interní akceptační testování je druh testování nazývané jako alfa testování. Probíhá v rámci firmy, která SW vyvinula. Testují ho však lidé, kteří se na projektu nepodíleli, a to např. z oblastí produktového managementu, prodeje nebo zákaznické podpory.

Externí akceptační testování je prováděno členy firmy, kteří nemají s vývojem daného SW nic společného.

Akceptační testování zákazníkem je prováděno zaměstnanci firmy, která si SW objednala. Vyjímaje SW, který je vyvinut firmou, která je objednavatelem a zároveň dodavatelem pro sebe sama.

Uživatelské akceptační testování, spíše známe z anglického názvu user acceptance testing (dále jen UAT) nebo také jako beta testování je prováděno koncovými uživateli SW, což mohou být potenciální nebo stávající zákazníci. Testování mohou provádět samotní zákazníci, zákazníci zákazníků nebo široká veřejnost. **[12]**

4.4 Typy testování

Typy testování probíhají na různých úrovních testování a metodách. Slouží k otestování specifické oblasti SW.

4.4.1 Smoke testování

Smoke testování je typ testování SW, který má za cíl otestovat nejdůležitější funkcionality SW. Jelikož se zabývá nejdůležitějšími funkcionalitami, neobsahuje kompletní sadu testů tzn. neobsahuje testy, které nejsou tolik potřebné, a vybrané funkce nejsou testovány do hloubky. Cílem je vyhodnotit, zda tato neúplná sada je dostatečně stabilní. Pokud je vyhodnoceno, že SW je dostatečně stabilní, následuje další testování. Vyhodnocení může rozhodovat i o produkční verzi, tzn. zda ji oznámit nebo vrátit.

Smoke testy odhalují stěžejní problémy v rané fázi vývojového cyklu a také integrační chyby. Mohou se provádět na novém SW, na zaběhlém SW, manuálně nebo automatizačními nástroji nebo skripty. V případě, že se smoke testování využívá často, měla by se upřednostnit automatizace řešení. Testování se koná na integrační, systémové a akceptační úrovni většinou pomocí metody Black box. **[13]**

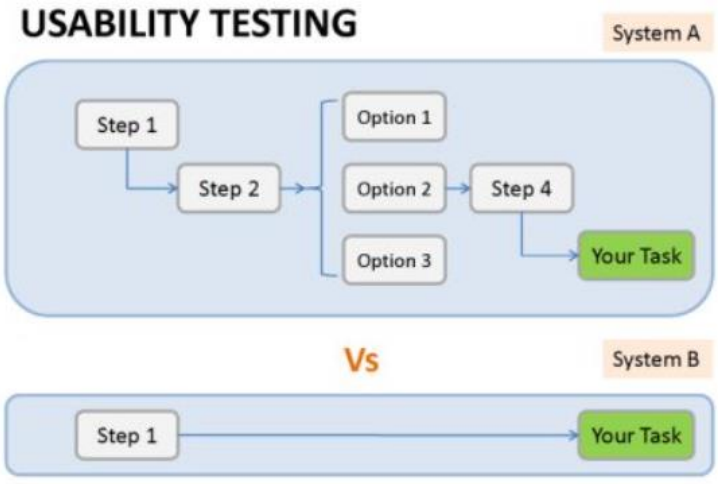
4.4.2 Funkční testování

V případě funkčního testování se jedná o typ testování SW, který zkoumá, jestli funkce SW odpovídají funkčním požadavkům. Jsou tedy testovány funkce SW, a to tak, že do testované funkce je vložen vstup a kontroluje se výstup, zda odpovídá požadavkům. Výsledkem testování není způsob zpracování, ale výsledky zpracování. Replikuje skutečné používání systému bez znalosti o vnitřní struktuře systému. Funkční testování je sestaveno dle požadavků klienta na výsledný SW z pohledu funkcionalit. Je konáno na všech úrovních testování a je prováděno metodou Black box. **[14]**

4.4.3 Testy použitelnosti

Testy použitelnosti zkoumají, jak je fakticky systém použitelný pro koncového uživatele. Bývá zahrnuto do tzv. nefunkčního testování. Koncové řešení by mělo být zpracováno tak, aby splňovalo funkční specifikace a zároveň bylo uživatelsky přívětivé. To však nebývá pravidlem (viz obr. 4).

Obrázek 4: Uživatelsky přívětivý systém vs nepřívětivý [15]



Testování probíhá na systémové a akceptační úrovni, kdy je využívána metoda Black box testování. **[15]**

4.4.4 Bezpečnostní testování

Bezpečnostní testování spadá do nefunkčního testování. Cílem je odhalit chyby, které by mohli vést k ohrožení systému a zároveň vyhodnotit, zda je systém, resp. data a zdroje chráněny v dostatečné míře vzhledem k riziku.

Při testování bezpečnosti je třeba se zaměřit na čtyři hlavní oblasti:

- Zabezpečení sítě
- Zabezpečení systémového SW
- Zabezpečení aplikace na straně klienta
- Zabezpečení aplikace na straně serveru

Zabezpečení sítě hledá zranitelná místa v síťové infrastruktuře neboli všechna zařízení, která přijímají a vysílají data a jsou připojené v počítačové síti.

Zabezpečení systémového SW hodnotí slabá místa v různých systémech jako např. operačních systémech, databázových systémech, informačních systémech aj., na kterých aplikace závisí.

Zabezpečení aplikace na straně klienta zajišťuje, aby klient nemohl být zmanipulován a nedošlo tak k možným rizikům.

Zabezpečení aplikace na straně serveru je zabezpečení kódu na serveru a jeho dalších technologií. Snahou je zabránit jakémukoliv vniknutí a následné ztrátě dat. **[16]**

4.4.5 Testování výkonu

Testování výkonu spadá do nefunkčního testování. Cílem je dostat informace o systému z hlediska odezvy a stability při daném zatížení. Testování může být prováděno např. testovacím nástrojem, který generuje zátěž pro testovaný prvek a zároveň sbírá informace o měření a výkonnosti. Výkonnostní testování se zaměřuje na následující atributy:

- Reakce
- Současnost
- Spolehlivost
- Stabilita
- Škálovatelnost

Reakce neboli odezva udává, jak rychle dokáže systém reagovat a dokončit zadané úkoly v čase.

Současnost umožňuje provádět paralelně více požadavků na SW vybavení systému.

Spolehlivost je schopnost SW dostát svým požadavkům. To znamená, že bude provádět funkce dle určitých podmínek po daný časový úsek bez žádných chyb.

Stabilita je schopnost SW být stabilní při různých úrovních zatížení v průběhu času.

Škálovatelnost je schopnost adekvátně měnit výkon SW v reakci na změnu požadavků na SW.

Zároveň existují čtyři druhy testování výkonu:

- Testování zátěže
- Zátěžové testování
- Testování odolnosti
- Spike testování

Testování zátěže je typ testování výkonu, který vyhodnocuje chování systému při rostoucím zatížení.

Zátěžové testování je typ výkonnostního testování, které vyhodnocuje chování systému při předpokládaném pracovním zatížením nebo za jeho hranicí.

Testování odolnosti je typ testování výkonnosti, kdy je cílem vyhodnocení chování systému při nepřetržitém zadávání požadavků, která vede k vysoké pracovní zátěži.

Spike testing je typ testování výkonnosti, kdy je náhle a podstatně zvýšena zátěž, při které se monitoruje chování systému. **[17]**

4.4.6 Regresní testování

Jedná se o typ testování SW, který spadá pod funkční i nefunkční testování. Zjišťuje, jestli provedené změny v SW neměli negativní vliv na jeho funkčnost a zda se nezměnila kvalita SW.

V případě jakékoliv změny zdrojového kódu je možné, že budou negativně ovlivněny funkce, a tedy i samotný chod SW. Aby se případné chyby včas zachytili, je třeba provést regresní testování. V případě regresního testování již existují předešlé testovací scénáře a není třeba vytvářet nové.

Testování je možné konat metodami White a Black box. Manuální testování je v tomto případě možné, avšak se z důvodů podstatné úspory času doporučuje regresní testování zautomatizovat. Dále testování probíhá na všech testovacích úrovních, kdy největší uplatnění má na systémové úrovni.

Automatizace regresního testování je využita při iterativních a inkrementálních vývojových cyklech. Za zmínění stojí agile metodika, která při nové funkci, změně stávajících funkcí, opravách chyb či refaktoringu kódu vede ke změnám v SW.

Potřeba regresního testování může vzniknout v důsledku některé z níže uvedených změn:

- Oprava závady
- Nová funkce
- Změna stávající funkce
- Refaktorizace kódu
- Změna technického návrhu
- Změna konfigurace

Není-li možno provést testování celého souboru regresních testů, jsou vybrány určité testovací scénáře, přičemž se ostatní vynechají. Výběr se zakládá na:

- Obecné stanovení priorit
- Prioritizace podle verze

V případě obecného stanovení priorit, jak už název napovídá, mají testovací scénáře svou prioritu, podle určitých faktorů jako např. dopad na podnikání, kritické funkce, často používané funkce, složité implementace a chybové oblasti SW.

U prioritizace podle verze jsou priority stanoveny, podle změn vykonaných ve verzi SW, a pravděpodobných oblastí SW, které mohly být ovlivněny těmito změnami. V tomto případě je zapotřebí provést důkladnou analýzu dopadu změn. **[18]**

4.4.7 Testování dodržování předpisů

Testování dodržování předpisů spadá do nefunkčního testování. V tomto případě se testují předpisy a normy s cílem zajistit shodu se systémem. Normy i předpisy mohou být externí i interní.

Testování se může pohybovat na úrovni velkých auditů při výběru testovacích vzorků nebo podrobné kontroly specifické normy. **[19]**

5 TESTOVACÍ NÁSTROJE SOFTWARU

Pod pojmem testovací nástroj je možné si představit SW, který umožňuje testerovi konat práci, ale především usnadňuje jeho práci, resp. testování. To znamená, že k testování aplikace je využit SW, resp. testovací nástroj, který dokáže zjednodušit, zrychlit, zefektivnit pracovní proces testera. **[20]**

5.1 Základní kategorie testovacích nástrojů

Testovacích nástrojů je celá řada a je možné je rozdělit do několika kategorií. Mezi základní kategorie patří bugtracking, test management, nástroje automatizace testování a podpůrné nástroje. **[1]**

5.1.1 Bugtracking

Bugtracking slouží k evidenci a správě nalezených chyb. Jedná se o základní nástroj, který by měl být používaný v každém projektu. Možností, jak bugtracking řešit je několik. Nejjednodušším řešením je např. zápis do sdíleného dokumentu, jako Google Docs a Excel, anebo využití specializovaných nástrojů určených přímo na bugtracking, jako např. Bugzilla, Mantis, Jira, Quality Center. Specializované bugtreckingové nástroje umožnují uživatelům širokou škálu funkcí, které využívají nejen testeři, ale celý tým. Mezi funkce, které takový nástroj nabízí patří např. konfigurovatelné workflow, hierarchie přístupových práv, snadné hledání, statistiky aj. **[1]**

5.1.2 Test management

V kategorii test management se jedná o nástroje, které se využívají pro přímé testování a umožňují vytvářet testovací scénáře s možností propojení s testovacími daty a spouštění testů. Spouštění testů je možné provést pomocí automatizačního nástroje nebo otevřením dokumentu s uloženými testovacími scénáři.

Test management se využívá pro ukládání všech testovacích scénářů, a to buď, že testovací scénáře jsou tvořeny přímo v nástroji nebo je možné je naimportovat jako jednotlivý testovací scénář nebo jejich množinu do nástroje. U většiny Test management nástrojů lze testovací scénáře sloučit do množin, a to na základě verze releasu, funkčností, modulů aj.

V případě spouštění testů jsou uživatelé využívající tento nástroj informování o spuštění, průběhu a výsledku testu. Dle těchto informací mohou uživatelé, resp. testeři monitorovat úspěšnost testů a plnění časového plánu testování.

Test management a Bugtracking jsou do jisté míry podobné nástroje, avšak Test management dává uživatelům hlubší informace o procesu testování. Na základě chyb vytvářejících se v průběhu testování, škály jejich závažnosti a opakování stejných chyb se může uživatel, resp. tester dále rozhodnout, zda nebude lepší testování pozastavit do doby, dokud nebudou zcela opraveny.

Do test managementu patří nástroje Quality Center, TestLink, VisualStudio aj. **[1]**

5.1.3 Automatizační nástroje

Automatizační nástroje tvoří rozsáhlou skupinu nástrojů a také oblast testování. Nástroj automatizovaného testování je SW, který zajišťuje testovací proces, a to buď v plném rozsahu nebo jen určitou část. Pokud má tento nástroj připravené testovací data, testy a konfigurace, umožňuje opakovaně a samostatně spouštět testy. Na projekty je možné využít tzv. krabicové řešení nebo vlastní řešení.

V případě krabicového řešení se jedná o nástroje, které jsou nabízeny jako hotová řešení, kdy uživatel pouze používané nástroje nakonfiguruje a připraví testovací data. Testovací případy jsou obvykle tvořeny skriptovacími jazyky, resp. skripty, které jsou v určité formě součástí těchto nástrojů. Také většinou obsahují tzv. záznamník akcí, jehož funkcí je zachytit akce nad grafickým uživatelským rozhraním (dále jen GUI) aplikace a transformovat je do skriptu. V případě potřeby se skript upravuje a připojuje na testovací data. Spojením skriptu a testovacích dat následně vzniká automatizovaný test. K používaným krabicovým nástrojům patří QuickTest, SilkTest, Selenium, TestComplete aj.

Krabicové řešení nemusí být vždy vhodné a v takových případech jsou vytvářeny vlastní testovací nástroje, které jsou vhodné přímo pro testování daného SW, resp. aplikace. Krabicové řešení zahrnuje obvykle celý proces testování, pokud je však potřeba otestovat určitou část, která např. nelze zautomatizovat v krabicovém řešení, nebo má smysl jen pro vlastní řešení, využívá se již zmíněné vlastní řešení. Mezi takové nástroje patří např. generování testovacích dat, kódování a dekódování dat, export a import dat, porovnávání výstupů aj. Takovéto nástroje bývají vytvářeny převážně vývojáři a mají podobu skriptu.

Existují i hybridní formy krabicového řešení, kdy se snadno vytváří skripty a není potřeba využívat služeb vývojářů. Mezi takové nástroje patří např. AutoIt. **[1]**

5.1.4 Podpůrné nástroje

Podpůrné nástroje zahrnují především nástroje nebo aplikace, které nesouvisí s testováním, ale jsou potřebné k samotnému testovacímu procesu.

Řadí se sem např. databázové nástroje, neboť při práci testera je nezbytné nahlížet do databáze, připravit nebo upravit v ní data. Patří sem např. SQL Developer, Toad, AquaData Studio, Db Visualizer aj.

Dalším nástrojem jsou editory, které využívá tester při práci s různými formáty. Testerovi umožňují zobrazovat a upravovat data v potřebném formátu. Příkladem může být analýza vytvořená business analytikem, která je vytvořena např. ve formátu UML nebo MS office a tester využije program Enterprise Architect. Odlišné formáty vstupních či výstupních souborů mohou být řešeny programem PSPad.

Dále do této kategorie patří nástroje, které provádí testy přímo, ale nejedná se o automatizované testování. Může se jednat o nástroj SoapUI, který zvládá integrační testování v rámci integrace služeb Webservis nebo REST servis. Dotazy se zasílají na testované rozhraní a následně se zobrazuje přijatá odpověď. Tento nástroj však umožňuje i samotnou automatizaci procesu.

Další nástroje, které se řadí do podpůrných nástrojů jsou WinSCP, PuTTY nebo nástroj pro virtualizaci Citrix. Tyto nástroje slouží k připojování na vzdálené testovací prostředí, které není možné dosáhnout na používaném zařízení. **[1]**

5.2 Testovací nástroje pro jednotkové testování

Jedná se o nástroje, které se využívají pro jednotkové testování.

5.2.1 Jasmine

Jasmine je open source framework, který umožňuje vykonávat jednotkové testy ve skriptovacím jazyce JS, a to synchronního i asynchronního kódu. Tento framework se řídí BDD procesem testování. Je vytvořen ve srozumitelném jazyce, tedy je pro něj charakteristický jasný popis, což umožňuje číst a rozumět jednotkovým testům

i neprogramátorům. Testy obsahují vyhodnocení, zda test uspěl či nikoli a podrobnější informace např. O neúspěšném testu. Může běžet samostatně a nevyžaduje žádné další frameworky a Document Object Model (dále jen DOM). Je vhodný pro testování webových stránek s různými prohlížeči, projekty Node.js nebo všude, kde může běžet JS. **[20] [21] [22]**

5.2.2 Criterion

Criterion je rozšiřitelný framework, který slouží k jednotkovému testování v jazycích C a C++ a má jednotné rozhraní mezi těmito jazyky. Zároveň je více kompatibilní např. s C99 a C++11. Umožňují automatickou registraci testů při deklaraci a struktura frameworku xUnit je implementována. Běh může být prováděn na systémech Linux, FreeBSD, Mac OS X, Windows.

V úvodu je dán vstupní bod, který umožňuje nedeklarovat funkci main v případě, že testování nebude náročné. Pro testovací frameworky, které se používají k testování v jazykách C a C++ je zapotřebí, aby byly vytvořeny šablony kódu při složitějších testech. Šablona obsahuje úvodní funkci main, ve které je potřeba zaregistrovat sady testů a samotné testy. Tyto šablony slouží k nastavení sad testů nebo samotných testů. Uživateli dodávají velkou kontrolu, avšak jsou složitější. **[24]**

5.2.3 jUnit 4

V případě jUnit4 se jedná o testovací framework, který byl vytvořen pro jednotkové testování v programovacím jazyce Java a je založen na anotacích. Patří mezi nejvíce využívané frameworky pro Javu. V případě jUnit můžeme říct, že disponuje větší podporou pro Mockito framework, který obsahuje i vlastní runner jUnit. Předešlé verze obsahovali mnoho nedostatků, které čtvrtá verze odstranila. **[23]**

5.2.4 xUnit

xUnit je nástroj určený pro jednotkové testování v prostředí .NET Framework. Je bezplatný s otevřeným zdrojovým kódem. Tento nástroj umožňuje testování následujících jazyků: C#, F#, VB.NET a dalších .NET jazyků. **[25]**

5.3 Testovací nástroje pro integrační a API testování

Testovací nástroje sloužící k integračnímu a API testování se používají především při práci se složitějším SW.

5.3.1 Citrus

Citrus je open source framework, který slouží k automatizovaným integračním testům. Zaměřuje se především na testování podnikových aplikací založených na zprávách. Rozhraní zpráv, které můžou být testovány jsou http REST, SOAP, JMS, Kafka, TCP/IP, FTP s dalšími aplikacemi jako klient-server. Citrus podporuje datové formáty jako XML, CSV, binární nebo JSON.

Citrus pracuje s knihovnami jako Spring Framework a Apache Camel a také s frameworky pro jednotkové testování jako jUnit, TestNG, Cucumber. Umožňuje integrace s buildovacími nástroji jako Maven nebo Grandle. **[26]**

5.3.2 Cypress

Cypress je framework, který slouží k testování moderních front endů neboli viditelných částí webových stránek. Mezi funkce, které umožňuje patří např. nastavení testů, psaní testů, spuštění a ladění. Jedná se o bezplatný open source, který bývá nainstalován lokálně na zařízení. Také zprostředkovává službu Dashboard, která zaznamenává testy.

Bývá využíván vývojáři nebo kvality (dále jen QA) inženýry, kteří budují webové aplikace pomocí JS frameworků. Testování probíhá formou TDD. Mezi testy, které Cypress umožňuje jsou end-to-end testy (dále jen E2E), integrační testy, unit testy, testování přes webový prohlížeč. **[27]**

5.3.3 Apache JMeter

Apache JMeter je open source aplikace, která vznikla z programovacího jazyka Java a slouží k testování funkčního chování a měření výkonu. Byla vytvořena za účelem testování webových aplikací, ale postupem času se její funkcionality rozšiřovaly.

V případě testování výkonu mohou být testovány statické i dynamické prvky aplikací. Mezi další využití patří simulování zátěže severu, skupiny serverů, sítě a zkoumání celkového výkonu při různých typech zátěže.

Mezi aplikace, servery a protokoly, u kterých lze testovat zátěž a výkon tímto nástrojem patří:

- Web – HTTP, HTTPS (Java, NodeJS,PHP, ASP.NET, ..)

- SOAP / REST webové služby
- FTP
- Databáze pomocí JDBC
- Middleware orientovaný na zprávy (MOM) prostřednictvím JMS
- Pošta – SMTP, POP3, IMAP
- Nativní příkazy nebo shellové skripty
- TCP
- Objekty Java

JMeter pracuje na úrovni protokolu a nejedná se o webový prohlížeč. **[28]**

5.3.4 Tavern

Tavern lze charakterizovat jako pytest plugin, nástroj příkazového řádku a knihovna Pythonu pro automatizované testování API s jasnou a stručnou syntaxí založenou na YAML. Využít ho lze i pro komplexní testování. Rozhraní API, které lze s tímto nástrojem testovat jsou REST a API, založené na protokolech MQTT.

Je možná integrace Tavernu do vlastního testovacího frameworku nebo nastavení kontinuální integrace za využití knihovny Python. Integrace je možná také pomocí nástroje příkazového řádku tavern-ci se skripty bash a úlohami cron. **[29]**

5.4 Testovací nástroje pro GUI/E2E

5.4.1 Appium

Jedná se o open source automatizační nástroj nativních, mobilních a hybridních webových aplikací. Umožňuje testování na mobilních platformách iOS, Android a Windows. Do nativních aplikací patří aplikace napsané sadami SDK pro všechny vypsané platformy. Testování mobilních webových aplikací probíhá pomocí prohlížeče, kdy pro iOS je podporovaný prohlížeč Safari a pro Android je podporován prohlížeč Chrome. V případě testování hybridních aplikací se provádí interakce s webovým obsahem. Zásadní je, že testování probíhá za použití jednoho rozhraní API pro více platforem. Používat tento nástroj lze s jazyky Java, Ruby, Python, PHP, JS a C#. **[30]**

5.4.2 Selenium

Selenium je projekt obsahující skupinu nástrojů a knihoven, které jsou využity za účelem automatizace webových prohlížečů. Jedná se o rozšíření, které umožňuje napodobit

interakce uživatele s prohlížeči, distribuční server pro škálování přidělení prohlížečů a infrastrukturu pro implementaci specifikace W3C WebDriver. Tato specifikace slouží k psaní zaměnitelného kódu, aby tomu porozuměli všechny hlavní prohlížeče. Využívané platformy jsou macOS, Windows a Linux.

Základem, ze kterého se Selenium skládá je WebDriver. Lze ho popsat jako rozhraní, které slouží k psaní sad instrukcí. Tyto instrukce je možné spouštět v různých prohlížečích jako Chrome, Internet Explorer, Safari, Firefox a Edge.

V případě práce se Seleniem je možné používat programovací jazyky jako Java, Python, Kotlin, Ruby, C#, nebo JS. **[31]**

5.4.3 Katalon

Katalon je komplexní nástroj, který umožňuje testování webu, API, mobilních a desktopových aplikací. Z operačních systémů podporuje Windows, macOS, Linux. Z webových platforem podporuje např. Firefox, Internet Explorer, Google Chrome, Microsoft Edge a Safari a z mobilních platforem Android, iOS a Cloud Services. Tvořit skripty lze v jazycích C#, Java, Groovy, Python a JS. **[32] [55]**

Katalon zajišťuje v oblasti funkčního testování především procesy jako testování odkazů, validace formulářů, testování souboru cookies, validaci Hypertext Markup Language (dále jen HTML) a CSS a kontrolu připojení k databázi. Dále v oblasti testování rozhraní zkoumá, jak samotný web reaguje na simulované přerušení nebo kompatibilitu a interakci mezi servery. V případě testování kompatibility se zaměřuje na kompatibilitu webu s různými prohlížeči a operačními systémy. Katalon dále umožňuje testování výkonu, kde se zabývá zatěžováním webových aplikací a testováním škálovatelnosti při vysokém využívání aplikace uživateli. **[32]**

5.4.4 SpecFlow

V případě SpecFlow se jedná o nástroj pro automatizaci testu pro .NET framework. Testy probíhají způsobem BDD. Byl vytvořen k správě a automatickému provádění akceptačních testů v projektech .NET.

Gherkin je jazyk, kterými jsou sepsány testy SpecFlow. Testovací scénáře se píšou přirozenými jazyky, které dále Gherkin transformuje. Tento jazyk využívá oficiální parser,

který umožňuje překlad ze 70 přirozených jazyků. Testy bývají svázány vazbami s kódem aplikace. Pomocí těchto vazeb je možné provádět testy testovacím frameworkem dle vlastního výběru. **[33]**

5.5 Testovací nástroje pro výkonnostní testování

Jsou to nástroje, které testují výkonnost jako odezva, chyby, topologie, obchodních transakcí, funkční kontroly aj.

5.5.1 AppDynamics

AppDynamics je nástroj pro testování výkonu aplikace. Application performance monitoring (dále jen APM) je nástroj, který je součástí platformy AppDynamics a umožňuje komplexní monitorování výkonu aplikace. Tento projekt využívá programovací jazyky, jako např. Java, .NET, Node.js, PHP, Python, C/C++ aj.

Mezi možnosti, které APM nabízí, patří řešení problému mále odezvy a chyby aplikace, zjišťování topologie aplikace, měření výkonu E2E obchodních transakcí se stavem jednotlivých uzlů aplikace a infrastruktury, přijímaní výstrah dle vlastních nebo vestavených pravidel a analýza aplikace na úrovni provádění kódu pomocí screenshotů. **[34]**

5.5.2 Gatling

Gatling je open source framework zaměřený na výkonnostní testování. Tento nástroj je vytvořen technologiemi jako Scala, Akku a Netty. Je naprogramován v již zmíněné Scale, taktéž testy jsou psány ve zdrojovém kódu tohoto jazyka. Spouštění testů je možné v kterémkoli operačním systému. Spuštění lze provádět pomocí příkazového řádku a zároveň je kompatibilní s jakoukoli platformou Continuous Integration.

Po sepsání a spuštění testů je možné zhlédnout přehled metrik bez přidání dalších pluginů. Metriky dané sady jsou uloženy jako soubor HTML, který dále slouží pro budoucí analýzy. Gatling pracuje s nástroji Akka, které pracují na modelu aktérů. Tento nástroj také nabízí integrované rozhraní API pro aserce. To znamená, že s tímto API je možné tvořit funkční kontroly s testováním výkonu zároveň. **[35]**

5.5.3 HammerDB

Tento bezplatný open source nástroj se využívá k zatěžování databází a k srovnávacím testům. V praxi se simuluje pracovní zátěž pomocí několika virtuálních uživatelů, kteří zatěžují databázi. Aby mohla být taková simulace spuštěna, je nutné v HammerDB vytvořit testovací schéma, do kterého se načtou data. Výsledkem bývají informace, které slouží k porovnání výkonu hardwaru a konfigurace SW.

Podporovány jsou databáze Oracle, SQL Server, IBM Db2, MySQL, MariaDB, PostrgreSQL. Je plně funkční na operačních systémech Linux a Windows. **[36]**

5.5.4 Iperf

Iperf je nástroj, resp. aplikace pro systémové administrátory na testování sítě vytvářením toků na protokolech TCP a UDP. Tato aplikace je součástí systému Raspbian. Iperf slouží jako klient-server. K práci je vyžadován další server, který slouží jako jiný počítat s nainstalovaným Iperfem. **[37]**

5.6 Testovací nástroje pro exploratory testování

Většinou tyto nástroje nejsou přímo určené na exploratory testování, ale obsahují funkce, které se při tomto testování využívají. Vzhledem k tomu, že exploratory testování je převážně neorganizované, je potřeba, aby takovéto testování bylo pečlivě zaznamenané.

5.6.1 Capture for Jira

Capture for Jira je nástroj, který je součástí Jiry a využívá se k různým druhům testování např. k exploratory testování nebo kolaborativnímu testování. Capture poskytuje zpětnou vazbu, která je odesílána přímo do Jiry. Zpětná vazba je brána jako komentovaný vizuální záznam, který může z hlediska testování více účelů.

Umožňuje nahrávání videí pro Chrome prohlížeč a následné sdílení v audiovizuální podobě. Dále je možné pracovat se samotným videem v podobě úprav. **[38]**

5.6.2 TestRail

TestRail je především webový nástroj, který je využíván pro správu testovacích scénářů. Správa zahrnuje sledování a organizaci testování SW. Mezi funkce, které TestRail umožňuje patří vytváření testovacích případů a jejich organizace v rámci sad, provádění testů

a kontrola výsledků. Všechny tyto funkce jsou v jednom rozhraní, a tedy není nutné žádný proces dělat separátně v jiném programu. **[39]**

5.7 Testovací nástroje pro test management

Tyto nástroje umožňují širokou škálu funkcionalit. Hlavní náplní je zajistit celkový proces správy testů.

5.7.1 Juno.one

Jedná se o český test management nástroj, který umožňuje spravovat a plánovat projekty, sledovat chyby, odhadovat čas projektu a správu testů aj. Dále nabízí funkci TODO, která říká, co by měl tester otestovat daný den. Využívá se zde také umělá inteligence k lepšímu uspořádání práce. **[40]**

5.7.2 QA Complete

QA Complete je aplikace, která umožňuje správu testovacích scénářů, prostředích, automatizovaných testů, defektů, a úkolů v projektu. Jedná se o komplexní řešení, které pomáhá testovaný SW udržovat ve vysoké kvalitě. **[41]**

5.8 Generátory testovacích dat

Generují testovací data, které mají pomoci při testování SW.

5.8.1 Test data generator

Test data generator je generátor, který slouží pro generování dat do databází. Využití nachází např., pokud nejsou žádné testovací data v nově vytvořené aplikaci. Data bývají čistě náhodná a nenarušují ničí soukromí. Takový generátor se hodí pro zátěžové testy, kdy je potřeba velké množství dat.

Generátor nabízí datové knihovny, ve kterých jsou shromažďovány různé typy dat pro různé využití. Tento typ generátoru také podporuje různé datové typy jako text, velký text, binární data, čísla, celá čísla, datum, čas, boolean a GUID. Každý z těchto typů je možno nastavit dle požadavků jako náhodná hodnota nebo náhodný soubor ze složky. **[42]**

5.8.2 Datprof

Generátor Datprof transformuje testovací data na tzv. syntetická, aby byly samotné testovací data maskovány. Cílem je ochránit data, s kterými se pracuje a zároveň je možno tyto data využít např. k prezentování.

Mezi funkce, které tento nástroj nabízí, patří zachovaní charakteru dat, konzistence v aplikacích a databázích, podpora formátů XML a CSV, generace syntetických dat a vysoký výkon v případě maskování velkých dat. **[43]**

6 SKRIPTOVACÍ JAZYKY NA TESTOVANÍ

Skriptovací jazyky bývají podmnožinou určitých programovacích jazyků, které skriptování umožňují. Skriptování je především určeno na jednodušší a menší bloky kódu, které dokáží interagovat s typy SW. Skriptovací jazyky se využívají na web scraping, testování, hacking, automatizace úloh, manipulace se stávajícími systémy atd.

6.1 Javascript

JS je jednoduchý skriptovací jazyk dříve nazývaný jako LiveScript, kterému se v posledních letech daří z hlediska počtu uživatelů. Základní myšlenou JS bylo dostat do statických HTML souborů dynamiku neboli interakci pro uživatele. Díky tomuto jazyku bylo možné nastavovat a ověřovat pravidla, nebo se vyhýbat vzdáleným serverům pro ověřování vstupů. Nyní je možné i přistupovat k DOM v prohlížeči, vytvářet asynchronní volání webového serveru a vývoj komplexních webových aplikací. Nepatří do seznamu univerzálních jazyků a má omezenou sadu knihoven. **[46] [48]**

Pokud je nutno vytvořit dynamické webové stránky, je zapotřebí nejen HTML a CSS, ale právě JS, který, jak již bylo zmíněno, zajišťuje právě samotnou dynamiku stránek. JS je jazyk citlivý na velká písmena, malá písmena a mezery. Program v tomto jazyce lze charakterizovat jako soubor příkazů, které jsou uvnitř značek (viz obr. 5). **[46]**

Obrázek 5: Příkazy nacházející se uvnitř značek [48]

V případě, že chce programátor zobrazit program napsaný v JS, musí být vyvolán z jiného prostředí např. prohlížeče tzn., že se spouští na straně klienta i pro webové aplikace. Samotné prohlížeče disponují vestaveným enginem JS. Jeho interpretace probíhá stejným způsobem jako probíhá čtení např. v Evropě, tedy shora dolů a zleva doprava. **[46] [48]**

Mezi funkce můžeme řadit podporu dynamického typování, OOP, podpora funkcionálního programování, nezávislost na platformě, pracování s prototypy, překlad pomocí interpretu, asynchronní zpracovávání, ověřování na straně klienta, větší kontrola v prohlížeči. **[48]**

JS se využívá k přidávání interaktivního chování na webové stránky, vytváření webových a mobilních aplikací, tvorbě webových serverů a vývoji serverových aplikací, vývoj her aj. **[49]**

6.2 Perl

Programovací jazyk, resp. skriptovací jazyk Perl, který se vyvíjí více než 30 let a nabízí širokou škálu funkcí. Využívá se na více než 100 platformách od přenosných počítačů po mainfraimy. Pomocí Perlu je možné vytvářet prototypy nebo i rozsáhlé vývojové projekty. Perl je součástí uskupení neboli rodin jazyků Raku dříve též známý jako Perl 6. Perl je však samostatný programovací jazyk.

Perl nabízí mnoho technologií, které lze použít v projektech jako např. databázové ORM, webový framework, profilovaní Perlu k zrychlení kódu, automatizované testování, reverzní proxy load balancer a webový server, Perl Plack/PSGI pro flexibilní vývoj webových stránek, Perl IPv6 pro práci na stejnojmenném protokolu.

Perl umožňuje mnoho užitečných funkcionalit. Může být využit k řešení kritických projektů ve veřejném i soukromém sektoru a podporuje OOP, procedurální i funkcionální programování. Lze ho rozšířit o 25000 modulů s otevřeným zdrojovým kódem, které jsou součástí sítě CPAN tedy Comprehensive Perl Archive Network. Dále umožňuje manipulaci s textem jako HTML, XMl a dalšími tagovacími a přirozenými jazyky. Podpora Unicodu verze 13 od verze Perlu 5.32. Rozhraní Perlu umožňuje integraci databází zkr. DBI s podporou databází jako Oracle, Sybase, Postgres, MySQL atd. Je možné využití externích knihoven C/C++.

V oblasti webu se jedná o programovací jazyk, který dokáže dobře manipulovat s textem a má schopnost rychlého vývojového cyklu. Tímto jazykem je sepsáno mnoho webových frameworků jako např. Catalyst. Perl dokáže pracovat s šifrovanými webovými daty jako jsou např. transakce elektronického obchodování, nebo také dokáže zrychlit zpracovávání o 2000 %, pokud je zabudován do webových serverů, a to např. při zabudování interpreteru Perl do webového serveru Apache. Umožňuje již zmiňované CPAN moduly a databázové integrace. **[47]**

6.3 PHP

Název PHP je rekurzivní zkratkou pro Hypertext Preprocessor a jedná se o velmi používaný open source interpretovaný OOP skriptovací jazyk, který slouží k všeobecnému použití především pro vývoj webových stránek. Kód PHP se vkládá do jednoho tagu (viz obr. 6) a tento tag může být vkládán do HTML struktury. Bývá rychlejší než jiné skriptovací jazyky jako např. ASP nebo JSP. **[52] [53]**

Obrázek 6: Vkládání PHP tagu do HTML struktury [52]

```
<!DOCTYPE html>
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

V případě PHP se vytvořený kód spouští na straně serveru a generuje HTML, které je zasláno klientovi. Po spuštění je klientovi prezentován výsledek skriptu. PHP umožňuje konfiguraci webového serveru tak, že bude zpracovávat soubory HTML. **[52]**

Pro PHP je charakteristická jednoduchost, kdy začátečník může začít psát jednoduché skripty v rámci hodin, a zároveň zkušenějším programátorům nabízí pokročilé funkce. **[52]**

Mezi funkcionality, kterými PHP disponuje, patří rychlé spouštění skriptů, multiplatformní využití, databázová podpora, reportování chyb, není potřeba deklarovat proměnné, podporu webových serverů, několik vrstev zabezpečení atd. **[53]**

Oblasti, ve kterých se PHP využívá, jsou e-commerce, motivy a pluginy pro WordPress, zdroj pro e-mailové šablony, sady uživatelských rozhraní, nástroje pro řízení projektů, GUI, vývoj Facebookových aplikací, generování PDF souborů, analýza XML souborů, zpracovávání a generování obrazu atd. **[54]**

6.4 Python

Python je programovací, resp. skriptovací jazyk, který se vyznačuje jednoduchostí a snadným naučením. Charakteristické pro tento jazyk je výkonnost, efektivní vysokoúrovňové datové struktury, jednoduchý přístup a manipulace s daty při využití OOP. **[44]**

S Pythonem je možné rychle vyvíjet aplikace v různých oblastech uplatnění na většině platforem. Oblasti, ve kterých se Python uplatňuje jsou například Data Science, strojové učení, AI, Web Scraping a různé úrovně testování. Rychlý vývoj je dán především tím, že tento jazyk má intuitivní syntaxi, dynamické typování a interpretovanou povahu. **[44]**

Oficiální stránky Pythonu jsou www.python.org. Na této stránce je zdarma k dispozici interpret a standartní knihovna zmíněného jazyka, a to v binární nebo zdrojové formě pro většinu hlavních platforem. Všechen obsah je možné volně šířit a stahovat. Zároveň se na stránkách objevují distribuce a odkazy na volně dostupný SW jako moduly, programy, nástroje a dokumentace jiných výrobců. **[44]**

Dále je možné rozšíření samotného interpretu o nové funkce a datové typy, které byly implementovány v jazycích C, C++ nebo jazycích volatelných z jazyka C. **[44]**

Python nabízí širokou škálu funkcionalit, do kterých patří již zmíněné snadné používání, interpretovaný jazyk, multiplatformní využívání či rozšiřitelnost. Dále sem můžeme zařadit funkcionality jako OOP, velké množství knihoven, podpora programování grafického uživatelského rozhraní neboli graphic user interface (dále jen GUI) a možnost integrace s jinými jazyky. **[50]**

6.5 TypeScript

V případě TypeScript se jedná o relativně nový programovací jazyk s otevřeným zdrojovým kódem, který slouží především k usnadnění vývoje webových aplikací a zvýšení jejich kvality. Byl vytvořen firmou Microsoft z již zmíněných důvodů. TypeScript má být nadstavbou jazyka JS, jelikož v jazyce JS se robustní aplikace špatně strukturují v udržitelné podobě.

Jedná se o staticky typovaný jazyk, který je kompilovaný a generuje kód jazyka JS. Tento vygenerovaný kód lze využít v multiplatformních řešeních. Jazyk nabízí širokou škálu sad

typů objektů a úrovní přístupnosti, které jsou podobné OOP. Jazyk nabízí zjednodušené možnosti dědičnosti, zapouzdření a abstrakce. Mezi vlastnosti převzaté z OOP patří statické typování, třídy, rozhraní, generika a moduly.

Nadstavbou TypeScriptu nad JS je myšleno hlavně přidání vrstvy statického typování nad ním. Tato statická vrstva je spuštěna pomocí kompilátoru, který v první řadě přezkoumá sepsaný kód v TypeScriptu a následně ho převádí na čistý JS. Kompilátor zajišťuje nejen překlad, ale i včasné nalezení chyb, které se v projektu objeví.

TypeScript také zavedl tzv. členy, kteří jsou soukromí nebo veřejní. V případě, že objekt nebo funkce přistupuje k jinému objektu, který se vyznačuje jako soukromý člen, pak kompilátor zjistí, že sepsaný kód je neplatný a vyhlásí chybu.

Lepší testovatelnost unit testů vytvořeného kódu je díky rozhodnutí o zahrnutí rozhraní do specifikace jazyka. **[45]**

Mezi jednu velkou funkcionalitu patří to, že TypeScript je nadmnožinou JS, tedy všechny funkcionality nebo knihovny, které patří k JS, patří také k TypeScriptu, avšak TypeScript je kompilovaný do jazyka JS. **[51]**

7 VÝBĚR TECHNOLOGIÍ PRO VLASTNÍ REALIZACI

Předtím než budou vybrány technologie, které budou použity, je důležité vybrat oblast testování, které se testovací nástroje věnují. Autorem byla vybrána oblast GUI/E2E testování, jelikož z možných oblastí vyšla tato možnost nejlépe (viz tab. 3). Tabulka se vztahuje pouze k vypsaným testovacím nástrojům v této práci.

Tabulka 3: Porovnání oblastí testovacích nástrojů s možnostmi testování [Autor]

Oblast testování	Nevyžaduje připravený kód, projekt či technologie	Umožňuje testování web aplikací	Umožňují automatizaci	Znalost autora
Jednotkové testování	Ne	Ano	Ano	Střední
Integrační a API	Ano	Ano	Ano	Malá
GUI/E2E	Ano	Ano	Ano	Střední
Výkonnostní	Ano	Ano	Ano	Malá
Exploratory	Ano	Ano	Ano	Malá
Test Management	Ne	Ne	Ne	Malá

V GUI/E2E oblasti jsou vypsány čtyři testovací nástroje. Z těchto nástrojů připadají v úvahu Selenium nebo Katalon (viz tab. 4), jelikož možností využití programovacími jazyky, platformami a prohlížeči je nejvyšší. Autor vybírá testovací nástroj Selenium z důvodu osobního zájmu a disponuje širší podporou pro prohlížeče tzn., že je komplexnějším nástrojem.

Tabulka 4: Možnosti GUI/E2E nástrojů [Autor]

GUI/E2E testovací nástroj	Jazyky	Platformy	Prohlížeče
Appium	Java, Ruby, Python, PHP, JS, C#	iOS, Windows, Android	Chrome, Safari

Selenium	Java, Python, Kotlin, Ruby, C#, JS	macOS, Windows, Linux	Chrome, Internet Explorer, Safari, Firefox, Edge
Katalon	C#, Java, Ruby, Python, Groovy	Android, iOS, Cloud Services, macOS, Linux, Windows	Firefox, Internet Explorer, Chrome, Edge, Safari
SpecFlow	Gherkin	-	-

Použitým skriptovacím jazykem je Python, jelikož s tímto jazykem má autor největší zkušenosti a umožní mu pracovat efektivněji, zároveň je velice používaný s kombinací se Seleniem.

8 VYTVOŘENÍ AUTOMATIZAČNÍHO NÁSTROJE

Dle vybraných technologií se začne vytvářet automatizační testovací nástroj.

8.1 Návrh

Tento nástroj již byl vytvořen jako Testovací nástroj pro funkční testování. Název je odvozen od základního účelu samotného nástroje a zároveň může sloužit k regresnímu testování a funkčnímu testování. Nástroj spadá do oblasti GUI/E2E testování.

Cílem je vytvořit nástroj pro testování webových aplikací, který budou moct využívat uživatelé s minimálními požadavky na znalosti z oblasti ICT. Testovací scénáře jsou ukládány ve formátu xlsx neboli Excel. Důraz je kladen na možnost testovat velké množství testovacích scénářů a zároveň ponechává uživateli volnost v samotném nastavení testovacího scénáře. Testovací nástroj po skončení testování vytváří výsledný soubor s výsledkem testování. Samotné testování bude probíhat tak, že nástroj pomocí driveru příslušného internetového prohlížeče otevře samotný prohlížeč a bude postupovat webovými stránkami dle nastavení v Excelu.

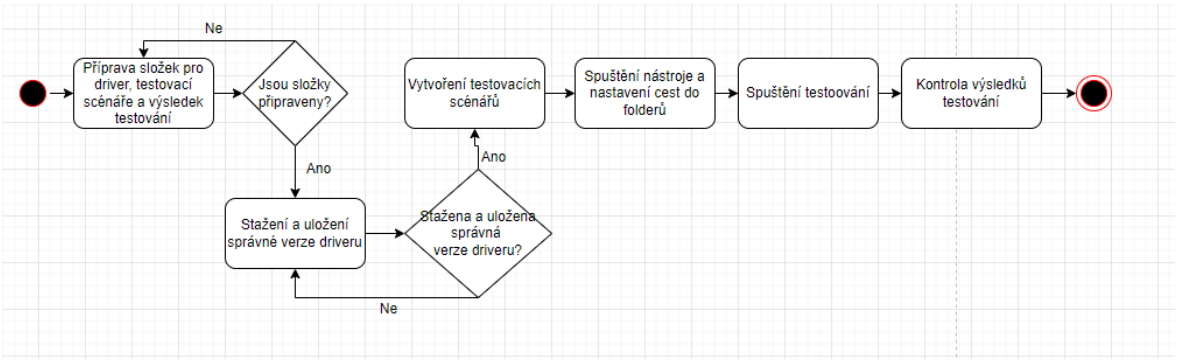
Nástroj je vhodný při regresním testování, funkčním testování, agilním nebo waterfall stylu vývoje. V některých situacích se stává, že při nasazení nových funkcionalit v kódu se rozbijí jiné funkcionality. U webových aplikací to mohou být tlačítka, odkazy, hledací panely aj. A právě na to bude sloužit tento nástroj.

K tomu, aby mohl nástroj správně pracovat, bude zapotřebí internetové připojení a počítač s dostatečným uložištěm na disku, to je však v nynější době zanedbatelný požadavek, jelikož bude celkově zapotřebí paměť v řádech stovek MB. Dále dvě rozdílné složky pro testovací scénáře a výsledek testu. Posledním požadavkem je driver, s kterým pracuje testovací nástroj.

8.1.1 Procesní diagram testování

Diagram (viz obr. 7) slouží k pochopení procesů, které uživatele provázejí při využívání vytvořeného nástroje.

Obrázek 7: Procesní diagram testování nástrojem [Autor]



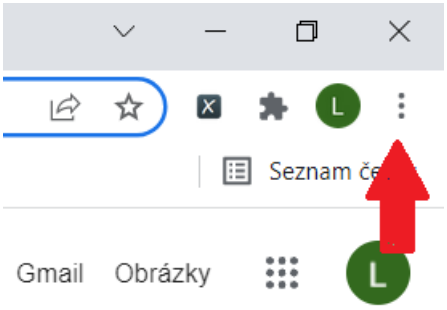
8.2 WebDriver a jeho nastavení

Jedná se o program, který automaticky spouští a pracuje s webovým prohlížečem. Práce driveru už dále záleží na konfiguraci testovacího scénáře, dle kterého se driver řídí. Driver je možné použít pro všechny známé prohlížeče jako Google Chrome, Microsoft Edge či Safari.

Driver je potřeba stáhnout a uložit do složky. Její cestu je třeba si zapamatovat, jelikož tato cesta bude používána testovacím nástrojem. Verze driveru, která se bude stahovat musí být totožná s verzí používaného prohlížeče, jinak hrozí, že driver nebude správně pracovat.

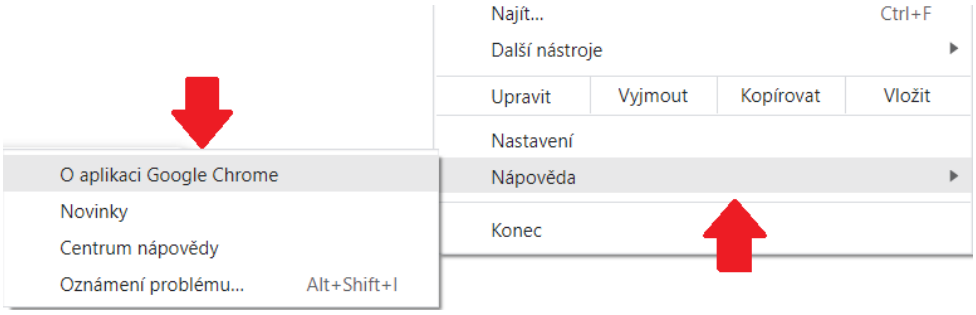
V případě Google Chrome je potřeba zjistit jeho verzi, a to následně. Otevře se prohlížeč dvojklikem. Klikne se na tři horizontální tečky v pravém horním rohu (viz obr. 8).

Obrázek 8: Horizontální tečky v Chromu [Autor]



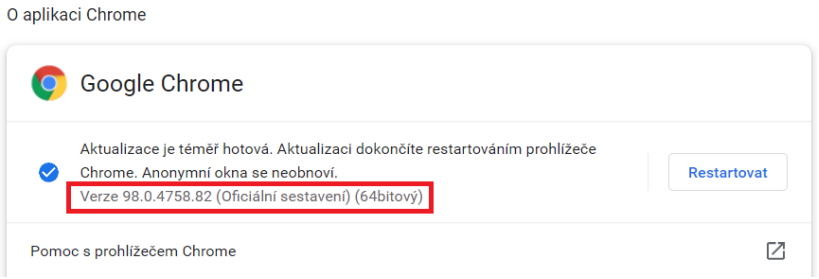
Po rozbalení možností je potřeba najet kurzorem na Nápověda a kliknout na záložku O aplikaci Google Chrome (viz obr. 9)

Obrázek 9: O aplikaci Google Chrome [Autor]



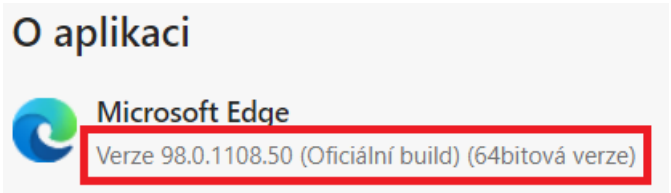
Následně Google Chrome zobrazí potřebnou informaci o verzi prohlížeče (viz obr. 10).

Obrázek 10: Verze Google Chrome [Autor]



V případě Microsoft Edge je postup stejný. Dvojklikem se otevře Edge a klikne se na tři vertikální tečky v pravém horním rohu. Po rozbalení možností se na konci seznamu najede kurzorem na Nápověda a zpětná vazba a po dalším rozbalení možností se klikne na O aplikaci Microsoft Edge. Výsledkem by mělo být přesměrování v prohlížeči na nastavení, kde se zobrazují informace o prohlížeči (viz obr. 11).

Obrázek 11: Verze Microsoft Edge [Autor]



Následovat bude výčet stránek, kde lze Webdriver stáhnout:

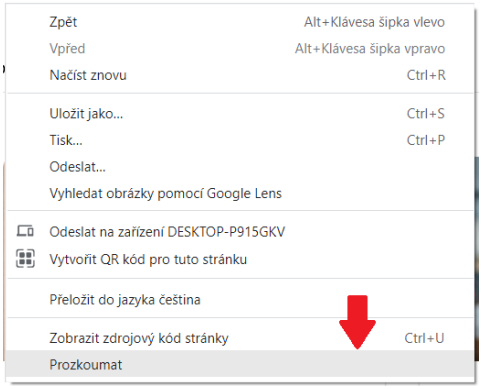
- Chrome: https://chromedriver.chromium.org/getting-started
- Edge: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

V případě, že Webdriver je stažený, je určen pro použití s totožným prohlížečem a má totožnou verzi s prohlížečem, pak je připraven k použití nástrojem.

8.3 Lokátory

Typy lokátorů, resp. atributy elementů s jejich parametry se vkládají do testovacích scénářů jako parametry pro testovací nástroj. Aby testovací nástroj mohl pracovat s webovými aplikacemi, je potřeba nějakým způsobem vyhledávat elementy neboli tlačítka, vyhledávací pole atd. K tomu slouží několik typů lokátorů, s kterými tento nástroj pracuje, jako XPath, ID, Class, Tag, Link Text, Name, CSS Selector, Partial Link. Aby mohl být element nalezen, je potřeba si zobrazit HTML kód webové aplikace, kterou chceme testovat. Zobrazení se provádí pravým kliknutím na stránce a po rozbalení nabídky se klikne levým tlačítkem na poslední možnost dole Prozkoumat v případě Google Chrome (viz obr. 12) nebo Zkontrolovat v případě Microsoft Edge. Je nutno říci, že lokátor na stránce musí být unikátní, jinak testovací nástroj nebude přesně vědět, jaký lokátor má zvolit.

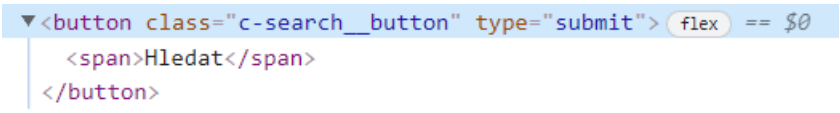
Obrázek 12: Možnost prozkoumat Google Chrome [Autor]



8.3.1 Lokalizace elementu

Po zobrazení zdrojového kódu v pravé části obrazovky se mohou elementy vyhledávat. Hledaní elementů bude probíhat pouze v záložce Elements. Pokud je potřeba na stránce nalézt element, např. pro tlačítko, najede se na tlačítko kurzorem, klikne se pravým tlačítkem a z možností se vybere Prozkoumat pro Chrome nebo Zkontrolovat pro Edge. Následně bude ve zdrojovém kódu stránky modře zbarvený obdélník s částí kódu uvnitř (viz obr. 13). Tento kód reprezentuje vybrané tlačítko na stránce.

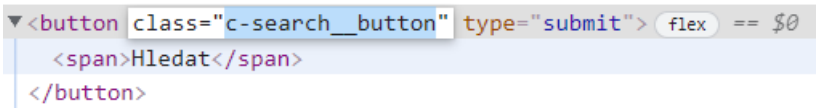
Obrázek 13: Lokalizovaný element [Autor]



8.3.2 Zvolení lokátoru

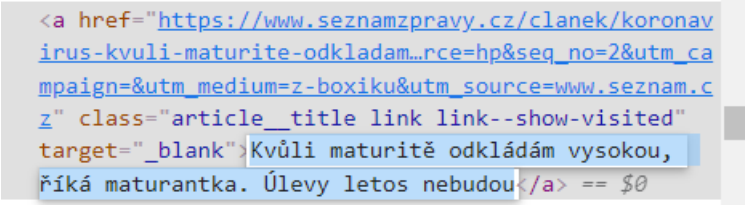
Po vyhledání elementu záleží, jaký typ lokátoru element obsahuje. Z příkladu vyhledávání tlačítka (viz obr. 13) lze vidět, že tento element nabízí lokátor typu Class. Nyní stačí zkopírovat parametr uvnitř uvozovek (viz obr. 14). Tento způsob hledání platí pro lokátory Class, ID, Name.

Obrázek 14: Kopírovaní parametru lokátoru pro ID, Class, Name [Autor]



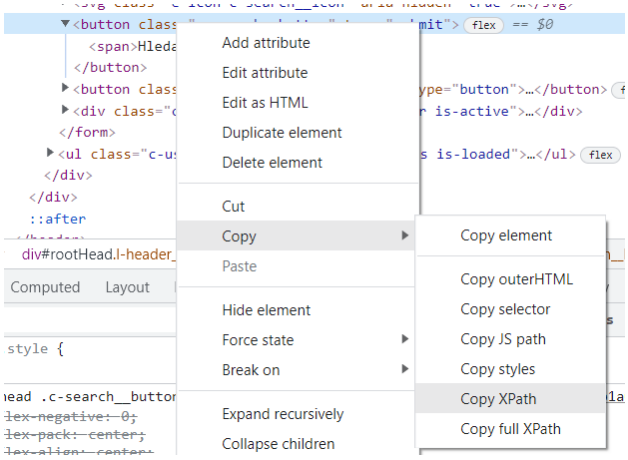
V případě Link textu se jedná o zvolení odkazu, avšak jako parametr se vkládá text, který je uvnitř elementu (viz obr. 15). Partial link je zcela totožný s Link textem jen zde stačí jako parametr část textu a ne celý text.

Obrázek 15: Kopírování parametru lokátoru pro Link, Partial Link [Autor]



Při zvolení XPath jako typ lokátoru stačí po nalezení elementu na zdrojové stránce kliknout na element pravým tlačítkem. Po zobrazení možností najet myší na copy a po zobrazení dalších možností kliknout na Copy XPath (viz obr. 16).

Obrázek 16: Kopírování parametru lokátoru pro XPath [Autor]



Nyní je parametr XPath zkopírovaný a je možné ho vložit do testovacího scénáře do Excelu. Stejným způsobem lze postupovat v případě CSS Selectoru.

Poslední možností je tag. Jedná se o HTML tagy, kterých je celá řada (viz https://www.w3schools.com/TAgs/default.asp). Pokud si uživatel vybere tag, který chce použít pro testování, musí si uvědomit, že tag může obsahovat více než jeden element a testovací nástroj vybere první, který bude za tímto tagem následovat.

8.3.3 Kontrola unikátnosti parametru lokátoru

Může nastat situace, že bude vybrán lokátor s parametrem, který se na HTML stránce objevuje více než jednou. Taková situace bude mít za následek, že testovací nástroj nebude vědět, jaký element má být použit pro testování a test by nebyl dokončen.

Dříve než bude lokátor a jeho parametr vložen do testovacího scénáře je potřeba si ověřit, že je tento parametr unikátní. V případě vybrání lokátoru a parametru stačí zkopírovaný parametr vložit do vyhledávacího pole, které má uvnitř popis „Find by string, selector or XPath“. Pokud má uživatel kliknuto v boxu, kde se nachází HTML struktura, stačí stisknout klávesy ctrl + F a pod tímto boxem se zobrazí již zmíněné vyhledávací pole (viz obr. 17).

Obrázek 17: Vyhledávací pole v HTML kódu [Autor]

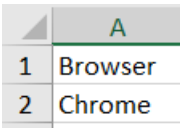
8.4 Funkce testovacího nástroje a jejich zápis

Funkcemi nástroje jsou myšleny funkce, které se vkládají do excelu při vytváření testovacího scénáře. Podle těchto funkcí se řídí samotný testovací nástroj. Do prvního řádku se vždy zadávají názvy funkcí, které se mají provést a v dalších řádcích se zadávají parametry, dle kterých se program řídí. Každý sloupec může být reprezentován jako krok.

8.4.1 Browser

Browser je funkce, která slouží k tomu, aby byl vybrán Webdriver, s kterým má nástroj pracovat. Tato funkce musí být pokaždé zadána a v excelu má absolutní místo ve sloupci A. Pokud nebude zadána, nebo nebude vybrán Webdriver nástroj, nemůže provést test. Do prvního řádku je potřeba zadat o jakou funkci se jedná, tedy Browser. Do druhého řádku se zadává parametr, který uvádí, s jakým Webdriverem se bude pracovat. Momentální možnosti jsou Chrome nebo Edge. Správné zapsání by tedy mělo být A1: Browser, A2: Chrome nebo Edge (viz obr. 18).

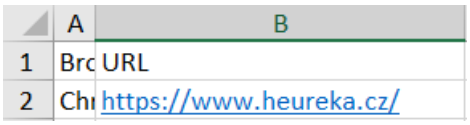
Obrázek 18: Zápis funkce Browser v excelu [Autor]



8.4.2 URL

Stejně jako browser, tak i URL má absolutní místo v sloupci B a musí být zadána. Po výběru browseru je nutné zadat URL, aby nějakým způsobem začal test. Prestože má URL absolutní místo, pokud bude potřeba, je možné tuto funkci používat opakovaně i v následujících krocích. V prvním řádku bude název funkce, tedy URL, a do druhého řádku se zadává parametr, což je adresa webové stránky. Webová stránka musí obsahovat začáteční část webové adresy https:// jinak testovací nástroj, resp. webdriver takovou adresu nerozpozná. Správné zapsání tohoto kroku je následující B1: URL, B2: https://www.heureka.cz/ (viz obr. 19).

Obrázek 19: Zápis funkce URL v excelu [Autor]

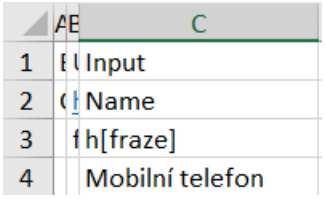


8.4.3 Input

Jedná se o funkci, která nemá absolutní pozici a může být vložena po sloupci B kdekoliv a opakovaně dle potřeby. Input funkce zajišťuje vkládaní jakéhokoliv textu do vyhledávacího pole. V prvním řádku musí být název funkce Input. Do druhého řádku se zadává název vybraného lokátoru. Do třetího řádku se vkládá parametr lokátoru a do

čtvrtého řádku se zadává text, který chceme vložit do vyhledávacího pole. Správné zapsání kroku je např. C1: Input, C2: Name, C3: h[fraze], C4: Mobilní telefon (viz obr. 20).

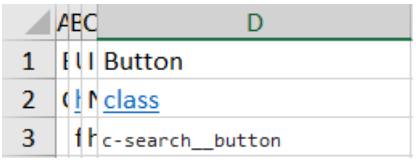
Obrázek 20: Zápis funkce Input v excelu [Autor]



8.4.4 Button

Button je funkce, která má na starost klikání na elementy. Element musí být samozřejmě takový, aby se na něj dalo kliknout. Jsou to elementy jako tlačítka, checkboxy, radiobuttony atd. Button funkce nemá absolutní pozici a může se v testovacím scénáři opakovat. Správný zápis tohoto kroku je např. D1: Button, D2: Class, D3: c-search__button (viz obr. 21).

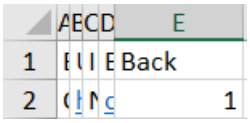
Obrázek 21: Zápis funkce Button v excelu [Autor]



8.4.5 Back

Back je jednoduchá funkce, která slouží k přesměrovaní na předešlou stránku. Tato funkce jako parametr bere číslo, které reprezentuje počet kroků zpět. Tato funkce nemá absolutní pozici, ale měla by být používaná v případě, že je možné jít o stránku zpět, a to maximálně tolikrát, kolikrát je možno jít zpět. Správný zápis tohoto kroku je např. E1: Back, E2: 1 (viz obr. 22)

Obrázek 22: Zápis funkce Back v excelu [Autor]

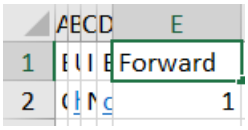


8.4.6 Forward

Forward funkce je podobná funkci back. Pouze s tím rozdílem, že místo na zpáteční stránky jde stránkami dopředu. Parametr funkce reprezentuje číslo, které udává počet kroků vpřed.

Funkce nemá absolutní pozici, ale měla by být používána v případě, že je možno jít o stránku vpřed, a to maximálně tolikrát, kolikrát je možno jít vpřed. Správný zápis této funkce v excelu je např. E1: Forward, E2: 1 (viz obr. 23).

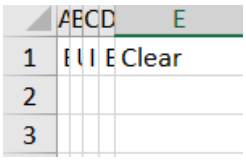
Obrázek 23: Zápis funkce Forward v excelu [Autor]



8.4.7 Clear

Tato funkce dokáže odstranit text z textového pole, který do něj byl vložen. Funkce nevyžaduje žádný parametr, ale měla by být použita pouze pokud v textovém poli obsahuje text a zároveň nemůže být použita, pokud by po použití funkce input následovaly funkce jako button nebo další input. Správný zápis této funkce je E1: Clear (viz obr. 24).

Obrázek 24: Zápis funkce Clear v excelu [Autor]



8.4.8 Wait

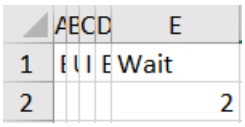
Wait patří mezi užitečnou funkci, která pozastavuje procesovaní testovacího nástroje. Důvodem je to, že program někdy pracuje rychleji, než se dokáže načíst HTML kód testované stránky a tím je způsobeno, že program nedokáže vyhledat požadované elementy. Z těchto důvodů byla tato funkce přidána jako užitečný nástroj pro odstranění tohoto problému.

Wait funkce má číselný parametr, který reprezentuje čas v sekundách, po který má testovací nástroj čekat, než bude pokračovat.

V případě využívání nástroje se doporučuje implementovat tuto funkci pokaždé, když se stránka bude muset načítat znovu, a to s parametrem alespoň 2.

Správný zápis této funkce je např. E1: Wait, E2: 2 (viz obr. 25)

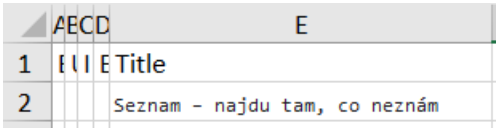
Obrázek 25: Zápis funkce Wait v excelu [Autor]



8.4.9 Title

Title funkce porovnává titulek stránky s parametrem vloženým do testovacího scénáře. Je možné si ověřit, zda titulek stránky odpovídá předpokladům. Správný zápis funkce je např. E1: Title, E2: Seznam – najdu tam, co neznám (viz obr. 26).

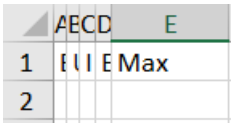
Obrázek 26: Zápis funkce Title v excelu [Autor]



8.4.10 Max

Max funkce pracuje se samotným prohlížečem. Nemá absolutní pozici a je možné ji použít nanejvýš jednou. Tato funkce maximalizuje okno prohlížeče přes celou obrazovku. Správným zápisem v excelu je např. E1: Max (viz obr. 27)

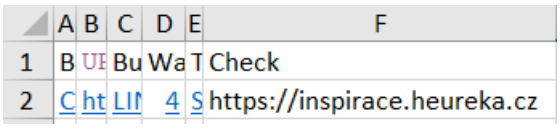
Obrázek 27: Zápis funkce Max v excelu [Autor]



8.4.11 Check

Tato funkce dokáže porovnat adresu zadanou v testovacím scénáři s aktuální adresou, která se nachází v prohlížeči. Funkce nemá absolutní pozici a může být použita opakovaně. U této funkce je třeba dávat pozor na správný zápis v testovacím scénáři. Zápis musí vždy obsahovat počátek adresy takto: https:// a následuje adresa, která musí být zakončena /. Pokud nebude mít porovnávaná adresa tuto strukturu, testovací nástroj vyhodnotí, že adresa stejná není. Funkce má jako parametr již zmíněnou webovou adresu. Správný zápis této funkce v testovacím scénáři je např. F1: Check, F2: https://inspirace.heureka.cz (viz obr. 28).

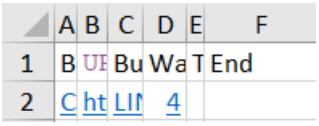
Obrázek 28: Zápis funkce Check v excelu [Autor]



8.4.12 End

Jedná se o funkci, která testovací nástroj informuje, že testovaní je u konce a může být ukončeno. Pokud je tato funkce spuštěna, zastavují se všechny procesy a prohlížeč je uzavřen. Tato funkce by měla být použita nejdříve ve čtvrtém kroku testovacího scénáře a může být použita pouze jednou. Funkce nemá absolutní pozici a nepřijímá žádný parametr. Správný zápis funkce je F1: End (viz obr. 29).

Obrázek 29: Zápis funkce End v excelu [Autor]



8.5 Vytvoření testovacího scénáře

Nejdříve je nutné vytvořit nebo vybrat složku, kde se samotné testovací scénáře budou vytvářet. Následně se vytvoří Excel soubor ve formátu xlsx a vhodně se pojmenuje podle toho, jaká oblast se testuje. Po vytvoření souboru se tento soubor otevře a uživatel začne vytvářet testovací scénář tak, že se do sloupců vkládají funkce, kterým rozumí testovací nástroj. Když je testovací scénář vytvořen, stačí pouze uložit.

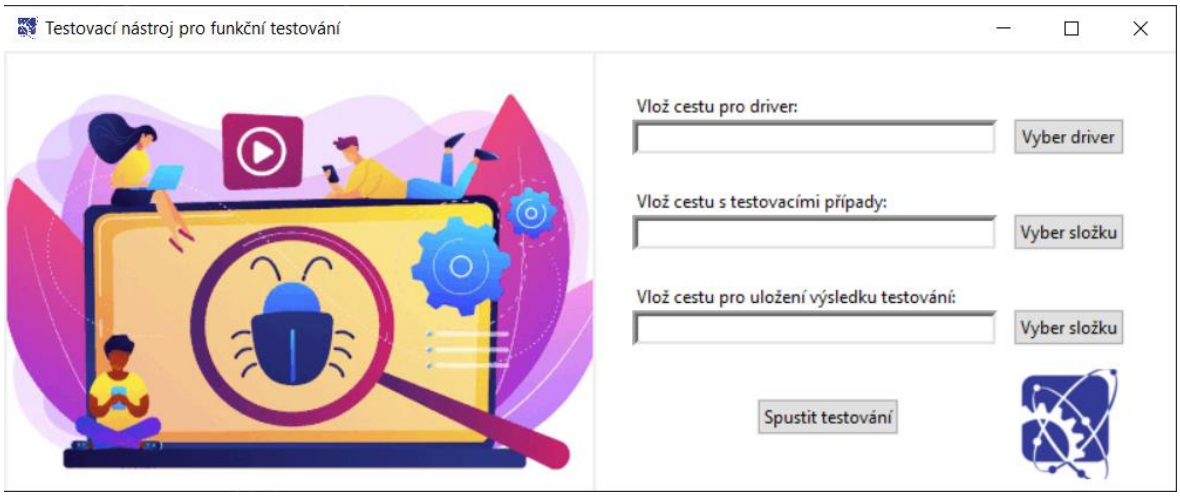
8.5.1 Vkládání funkcí s parametry

Vytváření probíhá tak, že každý sloupec reprezentuje krok, při kterém se musí provést nějaká funkce. Po provedení kroku následuje přechod do druhého sloupce a provedení funkce v tomto sloupci.

8.6 GUI

Aby bylo možné testovací nástroj používat, bylo vytvořeno GUI (viz obr. 30), které se zaměřilo především na jednoduchost používání. GUI obsahuje název, obrázek, logo fakulty, tři texty, tři textové pole a čtyři tlačítka.

Obrázek 30: Vzhled GUI [Autor]

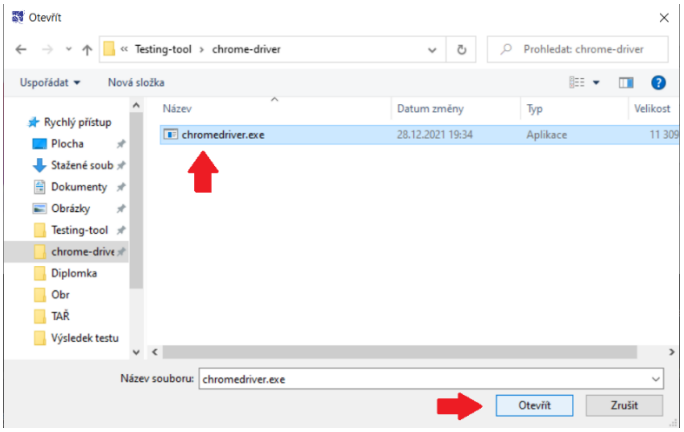


GUI je rozděleno na dvě poloviny, kdy na levé polovině je obrázek, který poukazuje na hledání defektů. V pravé časti jsou již strukturovaně vloženy widgety, resp. texty, tlačítka atd. Dále je v pravé části v pravém dolním rohu vloženo logo fakulty.

8.6.1 Práce s GUI

Začíná se výběrem driveru, kdy se klikne na tlačítko Vyber driver. Po kliknutí se zobrazí okno pro vybrání souboru (viz obr. 31). Následně se uživatel prokliká do složky, kde je uložený driver, který se označí a klikne se na otevřít. Po vybrání driveru se do příslušného textového pole vyplní cesta k driveru.

Obrázek 31: Zvolení driveru [Autor]



Analogicky se postupuje při vybírání složky testovacích scénářů a výsledku testování. Rozdíl je však v tom, že se nevybírá soubor ale pouze složka, kde jsou uloženy testovací scénáře a kde bude uložen výsledek testování.

Pokud jsou všechny textové pole vyplněny a soubory se složkami jsou správně vybrány, může být testování spuštěno kliknutím na tlačítko.

8.7 Proces testování

Jelikož je testování precizní práce, u které je třeba mít data strukturované a k dispozici, vytvoří se složka testování libovolně na počítači, s kterou bude později pracovat.

Dále je nutné zjistit verzi prohlížeče, kterou používáme a následně k němu stáhnout daný Webdriver. Po stažení se ve složce testování vytvoří složka driver, do které se vloží Webdriver formátu exe.

Ve složce testovaní se nyní vytvoří složka testovací scénáře. Jakmile je složka vytvořena, vytvoří se excel soubor s názvem testovacího scénáře a v něm se začne vytvářet testovací scénář. Testovacích scénářů může být ve složce více.

Ve složce testovaní se naposledy vytvoří složka výsledek testování, která bude sloužit k ukládání výsledku testování.

Nyní je vše připravené a testovací nástroj může být spuštěn. Po spuštění testovacího nástroje a zobrazení GUI se vyberou potřebné cesty k driveru a složkám a spustí se testování. Program začne pracovat a začne spouštět prohlížeč. Po dokončení testu je ve složce výsledek testování excel s výsledkem testování.

8.8 Rozbor zdrojového kódu

Kód se skládá ze tří souborů formátu py (viz obr. 32), a to excelimport, funcionaltetsing a gui. Každý soubor je rozdělen podle funkce, kterou v programu vykonává.

Obrázek 32: Soubory kódu [Autor]

8.8.1 Použité knihovny

Pandas je knihovna, která se stará o nahrávání a úpravu dat v excelu.

Selenium je knihovna, která zajišťuje práci s driverem, resp. s prohlížečem. Většina funkcí testovacího nástroje používá právě funkcionality Selenia.

Time je knihovna, která umožňuje pracovat s časem.

Os je knihovna, která obsahuje funkce pracující s operačním systémem.

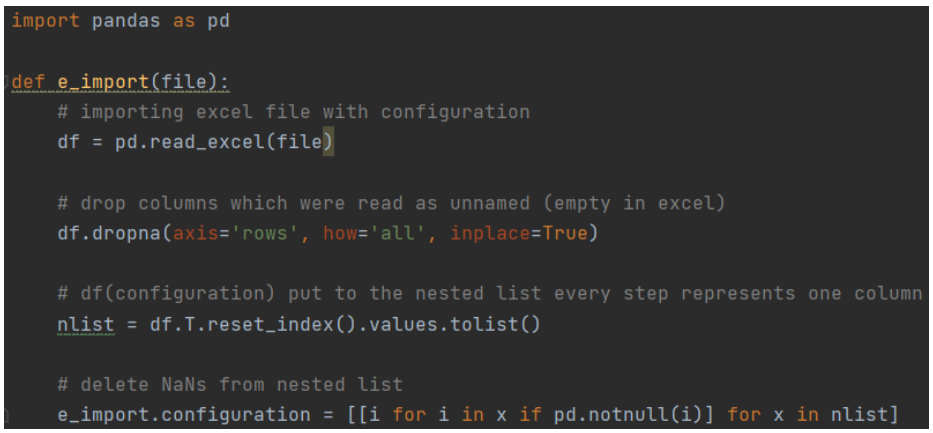
Tkinter je knihovna, která obsahuje funkce umožňující vytvářet desktopové aplikace.

8.8.2 excelimport.py

Tento soubor (viz obr. 33) zajišťuje nahrávání excelu testovacího scénáře a jeho úpravu, aby mohl být použit v dalších souborech. Obsahuje knihovnu Pandas, která dále bude používána pod zkratkou pd.

Byla vytvořena funkce *e_import*, která má jako parametr file reprezentující cestu, kde se nachází testovací scénář. Funkce obsahuje proměnnou df, do které byl uložen testovací scénář. Uložení umožňuje metoda *pd.read_excel*. Následně ve funkci probíhají úpravy proměnné df, resp. dat v excelu. Metoda *dropna*, vymaže všechny sloupce, které program přečetl jako unnamed. Následně byla vytvořena proměnná nlist, která transformuje strukturu excelu do struktury vnořených listu. Poslední proměnnou je configuration, která vymaže NaNs z listů a jedná se o konečnou úpravu.

Obrázek 33: Soubor excelimport.py [Autor]

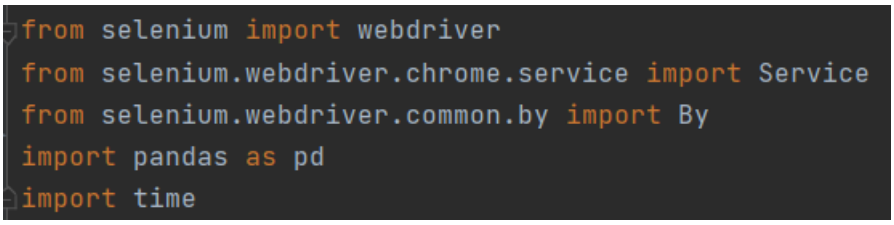


```
import pandas as pd

def e_import(file):
    # importing excel file with configuration
    df = pd.read_excel(file)

    # drop columns which were read as unnamed (empty in excel)
    df.dropna(axis='rows', how='all', inplace=True)

    # df(configuration) put to the nested list every step represents one column
    nlist = df.T.reset_index().values.tolist()

    # delete NaNs from nested list
    e_import.configuration = [[i for i in x if pd.notnull(i)] for x in nlist]
```

8.8.3 funcionaltesting.py

Tento soubor obsahuje knihovny (viz obr. 34) Selenium, Pandas a time. Funcionaltesting tvoří hlavní logiku programu, která zajišťuje průchod vnořenými listy v proměnné configuration, spouštění metod Selenia a zápis do výsledku testu do souboru.

Obrázek 34: Použité knihovny ve funcionaltesting.py [Autor]

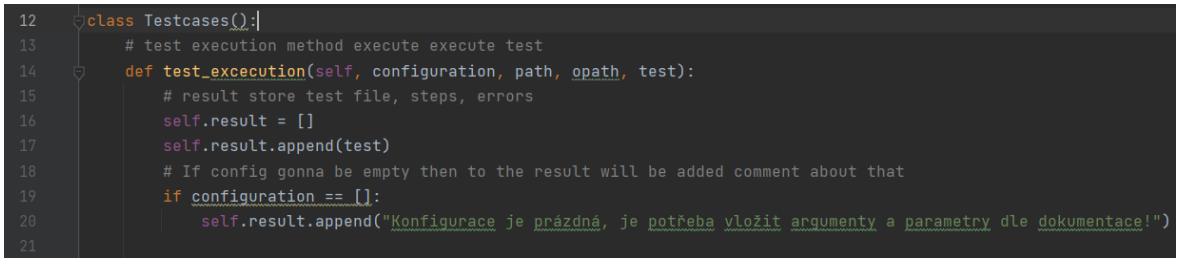


```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import pandas as pd
import time
```

Je vytvořena třída Testcases, která obsahuje metodu *test_execution*, *choose_browser*, *append_to_result*, *gourl*, *input_text*, *clicks*, *locators*, *back*, *forward*, *clear*, *wait*, *check_title*, *max*, *check_url*.

Metoda *test_execution* má na starost procesovat jednotlivé testovací scénáře. Zároveň zapisovat prošlé kroky do result filu s jeho závěrečným vyhodnocením. Obsahuje parametry configuration, path, opath, test (viz obr. 35).

Obrázek 35: Třída Testcases + počátek metody test_execution [Autor]



```
12  class Testcases():
13      # test execution method execute execute test
14      def test_excecution(self, configuration, path, opath, test):
15          # result store test file, steps, errors
16          self.result = []
17          self.result.append(test)
18          # If config gonna be empty then to the result will be added comment about that
19          if configuration == []:
20              self.result.append("Konfigurace je prázdná, je potřeba vložit argumenty a parametry dle dokumentace!")
21
```

V 16. řádku *test_execution* obsahuje list s názvem result, do kterého se ukládají jednotlivé kroky testovaní, název testu, výsledek testování nebo případné chyby. V 17. řádku se vkládá do listu result název testu, který je uložen v parametru test. V 19. řádku je první podmínka, která se provede, pokud list configuration bude prázdný. V takovém případě bude do listu result vložen text s chybou a test bude ukončen.

V 23. řádku je druhá podmínka (viz obr. 36), která říká, že v prvním a druhém kroku, resp. v prvním vnořeném listu a druhém vnořeném listu v listu configuration musí být na první pozici název funkce, resp. parametr text Browser a následně URL. V případě, že je podmínka splněna pokračuje kód na 26. řádku. Try otestuje blok kódu, který bude následovat. V tomto případě se jedná o iteraci přes list configuration (viz obr. 37). Při iteraci se do proměnné step ukládá list, který reprezentuje daný krok (viz obr. 37). V 30. řádku se inicializuje list step, aby mohl být využit v rámci celé třídy.

Obrázek 36: test_execution podmínka + začátek try bloku [Autor]

```
23          elif configuration[0][0].upper() == "BROWSER" and configuration[1][0].upper() == "URL":
24
25              # Try to loop over config
26              try:
27
28                  # Looping over configuration
29                  for step in configuration:
30                      self.step = step
```

Obrázek 37: Reprezentace configuration a krok [Autor]

```
Configuration=  [['Browser', 'Chrome'], ['URL', 'https://www.heureka.cz/'], ['Button', 'LINK', 'Co teď frčí'], ['Wait', 4.0], ['End']]
Krok = step =  ['Browser', 'Chrome']
```

Následně jsou vytvořeny if podmínky (viz obr. 38), které jsou splněny v případě, že první parametr v kroku je roven názvu některé z funkcí testovacího nástroje. Pokud je podmínka splněna spustí se blok kódu, pro který je charakteristické přidání kroku do result listu a následné volání metody, která představuje provedení dané funkce.

Obrázek 38: If podmínky v try bloku [Autor]

```
if "BROWSER" == step[0].upper():
    self.result.append(self.step)
    self.choose_browser(path)
elif "URL" == step[0].upper():
    self.result.append(self.step)
    self.gourl()
elif "INPUT" in step[0].upper():
    self.result.append(self.step)
    self.input_text()
elif "BUTTON" in step[0].upper():
    self.result.append(self.step)
    self.clicks()
elif "BACK" in step[0].upper():
    self.result.append(self.step)
    self.back()
elif "FORWARD" in step[0].upper():
    self.result.append(self.step)
    self.forward()
elif "CLEAR" in step[0].upper():
    self.result.append(self.step)
    self.clear()
elif "WAIT" in step[0].upper():
    self.result.append(self.step)
    self.wait()
elif "TITLE" in step[0].upper():
    self.result.append(self.step)
    self.check_title()
elif "MAX" in step[0].upper():
    self.result.append(self.step)
    self.max()
elif "CHECK" in step[0].upper():
    self.result.append(self.step)
    self.check_url()
```

Pokud bude krok obsahovat parametr end (viz obr. 39), znamená to, že test je na konci. Do listu result se vloží text s úspěšným vyhodnocením testu. Následně se vypne prohlížeč a ukončí se procesování testu.

Obrázek 39: Konečný krok End [Autor]

```
elif "END" == step[0].upper():
    self.result.append("Test uspěšně proveden!")
    self.driver.quit()
    break
```

V případě, že název funkce neodpovídá funkcím testovacího nástroje v daném kroku, je spuštěn blok kódu else (viz obr. 40). Tento blok má na starost zapsaní kroku do result listu, zapsání chybové hlášky do result listu, vypnutí prohlížeče a zastavení kódu. Tímto blokem kódu končí blok kódu od 29. řádku.

Obrázek 40: Else blok v try bloku kódu [Autor]

```
else:
    self.result.append(step[0])
    self.result.append("Neznamý krok! Projdi dokumentaci pro seznam kroků a jejich parametry.")
    self.driver.quit()
    break
```

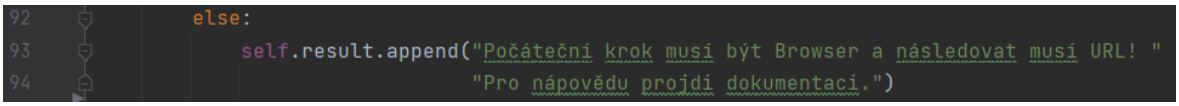
Jak již bylo zmíněno, od 26. řádku (viz obr. 36) se testoval celý blok kódu až po konec bloku kódu v else podmínce (viz obr. 40). V případě, že nastane chyba ve zmíněném bloku kódu, spustí se blok kódu expect ze 77. řádku (viz obr. 41), který slouží k oznamování chyb uživateli v result filu. Podmínka v 79. řádku je zaměřená na funkce, které nemohou být provedeny ve 3. kroku jako end, back, forward, clear, browser. Pokud se taková funkce objeví na 3. místě v configuration, pak je podmínka splněna a do result listu bude vložen text s chybovou hláškou. Následně bude vypnut prohlížeč a test skončí. Druhá podmínka z 85. řádku je zaměřena na kroky od čtvrtého a dále, s výjimkou funkcí button a input jejichž indikace chyb jsou prováděny v jiné oblasti kódu. Pokud bude podmínka z 85. řádku splněna, do listu result bude zapsána chybová hláška a prohlížeč bude uzavřen. Tímto blokem kódu končí blok kódu od 23. řádku.

Obrázek 41: Except blok [Autor]

```
77          except:
78              # If third step in config will be steps that cannot be processed then it will be save to result var.
79              if configuration[2][0].upper() not in ("WAIT", "BUTTON", "INPUT", "TITLE", "MAX", "CHECK"):
80                  self.result.append("Krok v konfiguraci musí obsahovat krok, který se dá provést! "
81                                     "Pro nápovědu použij dokumentaci.")
82                  self.driver.quit()
83
84              # If will be error in any step of config then it will be saved in result var.
85              elif self.step[0].upper() not in ("BUTTON", "INPUT"):
86                  self.result.append("V kroku " + self.step[0] + " nastala chyba! "
87                                                                 "Pro správné spuštění oprav chybu."
88                                                                 " Zkontroluj parametry."
89                                                                 " Pro nápovědu použij dokumentaci.")
90                  self.driver.quit()
```

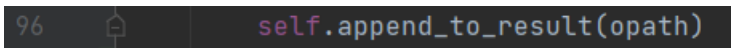
Poslední možností, která může nastat je, že configuration nebude prázdná, avšak nebude podle návrhu testovacího nástroje a nebude splněna podmínka z 23. řádku. V takovém případě bude proveden blok kódu podmínky else z 92. řádku (viz obr. 42). Jedná se o přidání chybové hlášky do result listu.

Obrázek 42: Else v metodě test_execution [Autor]



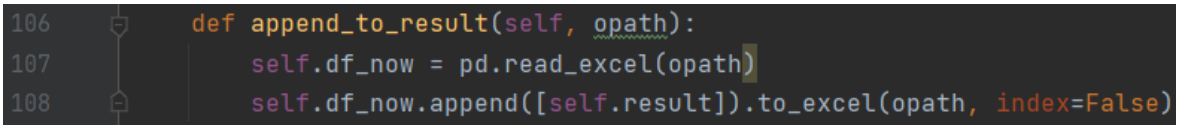
Poslední, co se v metodě *test_execution* nachází, je volání metody *append_to_result* (viz obr. 43), která přijímá parametr opath.

Obrázek 43: Volání metody v metode test_execution [Autor]



Opath je parametr, který má v proměnné uloženou cestu, kde se nachází result_file, resp. výsledek testu. V podstatě se nyní provedl test jednoho testovacího scénáře a jeho kroky s výsledkem jsou uloženy v listu result. Po zavolání této metody se začne provádět blok kódu této metody (viz obr. 44). Metoda *pd.read_excel* přecte excel v cestě opath a jeho hodnoty uloží do proměnné df_now. Následně do této struktury přidá vše, co bylo uloženo do listu result, což se uloží do stejného excel souboru v cestě opath. Parametr index, který je nastaven na hodnotu false zajišťuje, že se celý excel nepřepisuje, ale pouze se přidávají řádky.

Obrázek 44: Metoda append_to_result [Autor]



Metoda *choose_browser* (viz obr. 45) přijímá parametr path, za kterým se skrývá cesta k místu, kde se nachází uložený WebDriver. Metoda slouží k rozhodnutí, který WebDriver bude použit. Rozhodnutí zaleží na tom, jaký parametr je uložen ve funkci browser.

Obrázek 45: Metoda choose_browser [Autor]

V případě přechodu na určitou URL, je potřeba do excelu vložit funkci input, pak je podle programu zavolána metoda *gourl*. *Driver.get* (viz obr. 46) reprezentuje přechod na určitou stránku. Do url se vkládá parametr, který byl vložen u funkce input v excelu.

Obrázek 46: Metoda gourl [Autor]

```
def gourl(self):
    self.driver.get(url=self.step[1])
```

Ve zdrojovém kódu se vyskytuje také metoda *locators* (viz obr. 47), která slouží k nalezení hledaného elementu na stránce. Voláním této metody se nejdříve zkusí blok try, který je ohraničený na obrázku. Tento blok kódu obsahuje if a elif podmínky, které se spouští dle parametru, který se nachází v kroku. Zároveň je tento parametr metodou *upper* převeden na velké znaky, čímž se program vyhýbá striktnímu zápisu v excelu, tzn., že může začínat malým i velkým písmenem, či být celá psána malými a velkými písmeny nebo kombinací. V případě splnění podmínky tzn., pokud text parametru bude shodný s některým textem v podmínkách kódu, se spouští blok kódu uvnitř podmínky. Pro všechny tyto bloky v podmínkách je charakteristické, že do proměnné locator se uloží výsledek metody *driver.find_element*, která vyhledává element na stránce. Do této funkce jsou vloženy dva parametry. První z nich je druh lokátoru, který bude použit a druhý z nich cesta k elementu na stránce. Může nastat situace, kdy bude lokátor špatně zapsaný, a v takovém případě se bude provádět blok kódu v else. Tento blok vloží do result listu chybovou hlášku a zavře prohlížeč. Pokud v bloku try nastane chyba, začne se provádět blok kódu v except. Tento kód vloží do result listu chybovou hlášku a zavře prohlížeč.

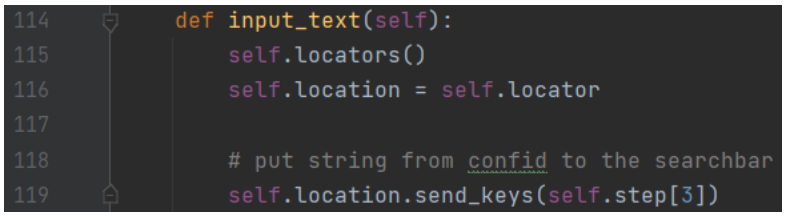
Obrázek 47: Metoda locators [Autor]

```
def locators(self):
    try:
        if self.step[1].upper() == "XPATH":
            self.locator = self.driver.find_element(By.XPATH, self.step[2])
        elif self.step[1].upper() == "ID":
            self.locator = self.driver.find_element(By.ID, self.step[2])
        elif self.step[1].upper() == "CLASS":
            self.locator = self.driver.find_element(By.CLASS_NAME, self.step[2])
        elif self.step[1].upper() == "LINK":
            self.locator = self.driver.find_element(By.LINK_TEXT, self.step[2])
        elif self.step[1].upper() == "NAME":
            self.locator = self.driver.find_element(By.NAME, self.step[2])
        elif self.step[1].upper() == "TAG":
            self.locator = self.driver.find_element(By.TAG_NAME, self.step[2])
        elif self.step[1].upper() == "CSSSELECTOR":
            self.locator = self.driver.find_element(By.CSS_SELECTOR, self.step[2])
        elif self.step[1].upper() == "PARTLINK":
            self.locator = self.driver.find_element(By.PARTIAL_LINK_TEXT, self.step[2])
        else:
            self.result.append("Chyba v názvu lokátoru!")
            self.driver.quit()
    except:
        self.result.append("Chyba v cestě lokátoru!")
        self.driver.quit()
```

Voláním metody *input_text* (viz obr. 48) se spustí metoda *locators*, vyhledávající element na stránce, který se uloží do proměnné locator. Následně se vytvoří proměnná location, do

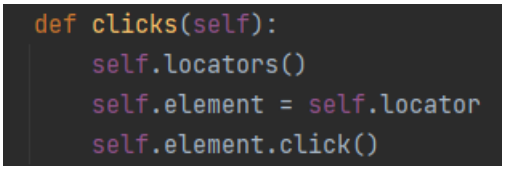
které se uloží nalezený element, který byl uložený v proměnné locator. V 119. řádku se již do vyhledaného elementu vkládá text.

Obrázek 48: Metoda input_text [Autor]



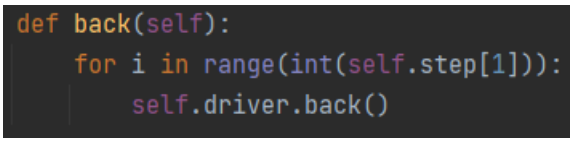
Mezi další metody se řadí *clicks*. Ta slouží ke klikání na daný element. Nejdříve se spustí metoda locators k nalezení elementu. Následně se vytvoří proměnná element, do které se uloží proměnná locator. Na takovýto element se nyní dá kliknout pomocí metody *click*.

Obrázek 49: Metoda clicks [Autor]



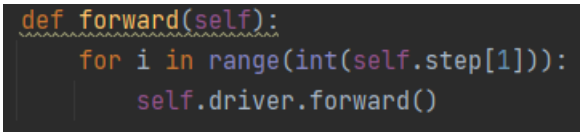
Metoda *back* (viz obr. 50) obsahuje for cyklus, který se bude opakovat podle parametru zadaného v kroku. Ve for cyklu je metoda *driver.back*, která se vrací jednou zpět na stránce.

Obrázek 50: Metoda back [Autor]



U metody *forward* (viz obr. 51) platí to samé jako u metody *back* s rozdílem, že u *forward* metody jde prohlížeč vpřed.

Obrázek 51: Metoda forward [Autor]



Metoda *clear* (viz obr. 52) z lokace, kde se nachází text, tento text vymaže.

Obrázek 52: Metoda clear [Autor]

```
def clear(self):
    self.location.clear()
```

Metoda *wait* (viz obr. 53) slouží k pozastavení programu. Metoda *time.sleep* pozastavuje program na čas, který je dán jako parametr. Tento parametr se vezme z daného kroku na dané pozici.

Obrázek 53: Metoda wait [Autor]

```
def wait(self):
    time.sleep(int(self.step[1]))
```

Metoda *check_title* (viz obr. 54) slouží k porovnávání titulku stránek. If podmínka říká, jestliže parametr v daném kroku je shodný s textem, který vyprodukuje metoda *driver.title*, pak se do result listu vloží text s kladným výsledkem. Pokud to tak není, je spuštěn blok kódu else. V takovém případě se do result listu vloží text se záporným výsledkem.

Obrázek 54: check_title [Autor]

```
def check_title(self):
    if self.step[1] == self.driver.title:
        self.result.append("Titulek souhlasí!")
    else:
        self.result.append("Titulek nesouhlasí!")
```

Metoda *max* (viz obr. 55) slouží k maximalizaci prohlížeče přes celou obrazovku. To umožňuje metoda *driver.maximize_window*.

Obrázek 55: Metoda max [Autor]

```
def max(self):
    self.driver.maximize_window()
```

Poslední metodou je *check_url* (viz obr. 56), která porovnává url adresy. If podmínka znamená, pokud parametr v daném kroku tzn. step[1] je roven adrese, na které se prohlížeč nachází tzn. *driver.current_url*, pak se do result listu vloží text s úspěšným výsledkem. V opačném případě se provede blok kódu v else, který vloží text do result listu se záporným výsledkem.

Obrázek 56: Metoda check_url [Autor]

```
def check_url(self):
    if self.step[1] == self.driver.current_url:
        self.result.append("URL je totožná!")
    else:
        self.result.append("URL není totožná!")
```

8.8.4 gui.py

Tento soubor slouží k vizualizaci programu, spouštění funkcí a metod jiných souborů. Obsahuje knihovny Tkinter, os a pandas (viz obr. 57). Je vytvořena třída screen, která obsahuje metody *__init__*, *ifolder_path*, *dfolder_path*, *ofolder_path*, *dir_files*, *start_test*.

Obrázek 57: Knihovny gui.py [Autor]

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import os
import excelimport
import funcionaltesting
import pandas as pd
```

Metoda *__init__* (viz obr. 58) se zavolá pokaždé, pokud je třída screen inicializována. V této metodě se nachází celé GUI. Prvotně se do gui proměnné uloží metoda *Tk*, která vytváří rámeček aplikace a s tímto rámečkem se dále bude pracovat. V 12. řádku se tomuto rámečku nastaví geometrie. V 13. řádku se nastaví titulek rámečku a v dalším řádku se nastaví bitmapová ikona, která se nachází v levém horním rohu (viz obr. 30).

Dále je vidět, že od 16. do 18. řádku jsou vytvořeny proměnné dfolerPath, ifolderPath a ofolderPath. Do těchto proměnných se vloží metoda *StringVar*, která slouží k následné jednodušší práci s hodnotami widgetů.

Do proměnné left_frame se ukládá widget frame, který obsahuje parametry gui, width a height. Díky parametru gui se ví, do kterého rámečku, resp. gui tento frame patří. Width a height nastavují výšku a šířku frame. V 21. se pomocí metody grid nastaví, v jaké části se tento frame bude nacházet. *Grid* metoda má parametry row a column, které reprezentují řádek a sloupec. Stejným způsobem je nastavený pravý frame v 27. a 28. řádku, který je označený pod proměnnou right_frame s rozdílem, že parametr column, resp. sloupec má

hodnotu 1. Tyto dva framy rozdělují gui na dvě stejné části přesně v polovině, a to sloupcově. Následně je možné s těmito rozdělenými částmi pracovat samostatně.

V 23. řádku se do proměnné bg ukládá widget photoimage, který představuje vkládání fotografie. Photimage obsahuje parametry file, width a height. File slouží k označení obrázku ve složce. Width a height nastavují šířku a výšku widgetu. Do proměnné bg_label se ukládá widget label, který obsahuje parametry left_frame a image. Tento label slouží k zobrazení obrázku. Parametr left_frame je zde, aby program věděl, kam label patří. Parametr image očekává widget, který představuje obrázek. Posledním krokem je umístění pozice labelu pomocí metody *grid*, která má parametry řádku a sloupce na 0 v levém framu.

V 30. řádku se vytvoří proměnná logo, která slouží taktéž k vložení obrázku, a nastavení je skoro stejné jako v přechozím případě. Změny jsou v různých hodnotách u parametrů šířky a výšky, jelikož má obrázek jiné rozlišení. V případě labelu je vložen do right_frame framu a přibyl parametr bd, který slouží k nastavení tloušťky ohraničení. Poslední změna je ve způsobu umístění obrázku, což znamená, že místo metody *grid* byla použita *place*, která slouží k absolutnímu umístění ve framu podle pixelů. *Place* metoda obsahuje parametry *x* a *y*, které slouží k umístění obrázku.

Obrázek 58: Třída Screen a počátek metody __init__ [Autor]

```
9   class Screen():
10      def __init__(self):
11          self.gui = Tk()
12          self.gui.geometry("800x300")
13          self.gui.title("Testovací nástroj pro funkční testování")
14          self.gui.iconbitmap("logo.ico")
15
16          self.dfolderPath = StringVar()
17          self.ifolderPath = StringVar()
18          self.ofolderPath = StringVar()
19
20          self.left_frame = Frame(self.gui, width=400, height=300)
21          self.left_frame.grid(row=0, column=0)
22
23          self.bg = PhotoImage(file="photo.gif", width=400, height=300)
24          self.bg_label = Label(self.left_frame, image=self.bg)
25          self.bg_label.grid(row=0, column=0)
26
27          self.right_frame = Frame(self.gui, width=400, height=300, bg="white")
28          self.right_frame.grid(row=0, column=1)
29
30          self.logo = PhotoImage(file="logo.gif", width=75, height=75)
31          self.logo_label = Label(self.right_frame, image=self.logo, bd=0)
32          self.logo_label.place(x=285, y=215)
```

Metoda *__init__* pokračuje (viz obr. 59) tvorbou widgetů jako label, button neboli tlačítko, entry neboli textové pole a zobrazením celkového gui.

Do proměnné driver_text je uložen ve widgetu label, který má parametry right_frame, text a bg. Do parametru text se vkládá text, který chceme zobrazit. Do parametru bg se vkládá text s názvem barvy, kterou má mít pozadí labelu. V 35. řádku se label umístí pomocí metody *place*. Do proměnné dpath_field je uložen widget entry neboli textové pole, který má parametry right_frame, textvariable, bd a width. Tento widget slouží jako textové pole, do kterého je možno zapisovat. Do parametru textvariable se ukládá proměnná dfolderPath. Width parametr nastavuje šířku widgetu. Následně je tento widget umístěn na určitou pozici. Do proměnné dfind_path je uložen widget button neboli tlačítko, které má parametry right_frame, text a command. Do parametru command je uložena metoda dfolder_path, která se spustí v případě kliknutí na tlačítko. Dva bloky kódu od 42. řádku do 46. řádku a od 48. řádku do 53. řádku vytvářejí stejné widgety. V těchto widgetech jsou však změněny parametry text, textvariable, command a pozice v place metodách. U parametrů text, textvariable a command záleží k jaké oblasti náleží jako driver, input neboli vstup, output neboli výstup. Oblast input je reprezentována počátečním názvem proměnné i nebo input. Oblast output je reprezentována počátečním názvem proměnné o nebo output. Oblast driver je reprezentována počátečním názvem proměnné d nebo driver. Posledním widgetem je button v proměnné start_test, které má parametry right_frame, text, command a je umístěn na určité místo. V 58. řádku je metoda *mainloop*, která zobrazuje celé gui.

Obrázek 59: Pokračování metody __init__ [Autor]

Při zavolání metody *ifolder_path* (viz obr. 60) se do proměnné ifolder_selected uloží cesta do složky, která se vybere. Metoda *filedialog.askdirectory* otevírá okno pro výběr složky

a zároveň ukládá cestu do proměnné. Následně metodou set se do proměnné ifolderPath nastaví hodnota, resp. cesta do složky. To má za následek automatické propsání do wigdetu entry. V této metodě se také volá metoda *dir_files*.

Obrázek 60: Metoda ifolder_path [Autor]

```
def ifolder_path(self):
    self.ifolder_selected = filedialog.askdirectory()
    self.ifolderPath.set(self.ifolder_selected)
    self.dir_files()
```

Metoda *ofolder_path* (viz obr. 61) má stejnou funkci jako metoda *ifolder_path*. Rozdílem je jiná proměnná, která je použita jako parametr pro metodu *set*. Také neobsahuje volání metody *dir_files*.

Obrázek 61: Metoda ofolder_path [Autor]

```
def ofolder_path(self):
    self.ofolder_selected = filedialog.askdirectory()
    self.ofolderPath.set(self.ofolder_selected)
```

Metoda *dfolder_path* (viz obr. 62) je podobná jako metody *ofolder_path* a *ifolder_path*. Rozdíl je v metodě *filedialog.askopenfilename*, která očekává vybrání souboru místo složky. Jinak princip je zachován.

Obrázek 62: Metoda dfolder_path [Autor]

```
def dfolder_path(self):
    self.file_selected = filedialog.askopenfilename()
    self.dfolderPath.set(self.file_selected)
```

Metoda *dir_files* (viz obr. 63) vytváří listy files a filename. Metoda obsahuje for cyklus, který iteruje přes názvy souborů v cestě složky, která je uložená v proměnné ifolder_selected. Metoda *os.listdir* vrací list se soubory, které se nachází v určité složce. Při každé iteraci se do listu filenames vloží filename tzn. název souboru. Následně je do proměnné f uložena metoda *os.path.join* s parametry ifolder_selected, filename. Tato metoda spojuje uložené textové řetězce v proměnných ifolder_selected a filename tzn., že při každé iteraci se vytvoří cesta ke konkrétnímu souboru. Dále je vytvořena if podmínka, při které se pomocí metody *os.path.isfile* s parametrem proměnné f určuje, zda tento

parametr je soubor. Pokud se jedná o soubor, zapíše se tato cesta k souboru do listu files. Pokud se nejedná o soubor, pak program pokračuje dále.

Obrázek 63: Metoda dir_files [Autor]

```
def dir_files(self):
    self.files = []
    self.filenames = []
    for filename in os.listdir(self.ifolder_selected):
        self.filenames.append(filename)
        f = os.path.join(self.ifolder_selected, filename)
        # checking if it is a file
        if os.path.isfile(f):
            self.files.append(f)
        else:
            pass
```

Zavoláním metody *start_test* (viz obr. 64) se začne provádět testování. Nejdříve se vytvoří proměnná create_result, do které se uloží metoda *pd.DataFrame().to_excel* s parametrem excel_writer. Tato metoda vytvoří prázdný excel v místě, které je dáno parametrem excel_writer, resp. hodnotou uloženou v proměnné ofolder_selected a názvem souboru, tj. result_file.xlsx, resp. výsledek testu. Po vytvoření excelu se provádí for cyklus, který iteruje přes list files. Při iteraci se vytvoří proměnná number, která pomocí metody *index* a parametru file vrací index z listu files. Poté se spustí v souboru excelimport funkce *e_import* s parametrem file. Dále se vytvoří proměnná self.file, která je odlišná od předešlého parametru file. Do této proměnné se uloží výsledek funkce *e_import* ze souboru excelimport, a to zápisem excelimport.e_import.configuration. Posledním krokem v tomto cyklu je spuštění metody *test_execution* ve třídě Testcases a souboru funcionaltesting. V případě volání této metody se do ní musí vložit parametry, které používá tedy configuration, path, opath, test. Po skončení for cyklu se použije funkce *quit*, která ukončí metodu. Voláním třídy Screen se provede kód v __init__ metodě.

Obrázek 64: Metoda start_test a spuštění třídy Screen [Autor]

```
    def start_test(self):
        self.create_result = pd.DataFrame().to_excel(excel_writer=self.ofolder_selected + "/result_file.xlsx")
        for file in self.files:
            self.number = self.files.index(file)
            excelimport.e_import(file)
            self.file = excelimport.e_import.configuration
            funcionaltesting.Testcases().test_excecution(configuration=self.file, path=self.file_selected,
                                                         opath=self.ofolder_selected + "/result_file.xlsx", test=self.filenames[self.number])
        quit()
Screen()
```

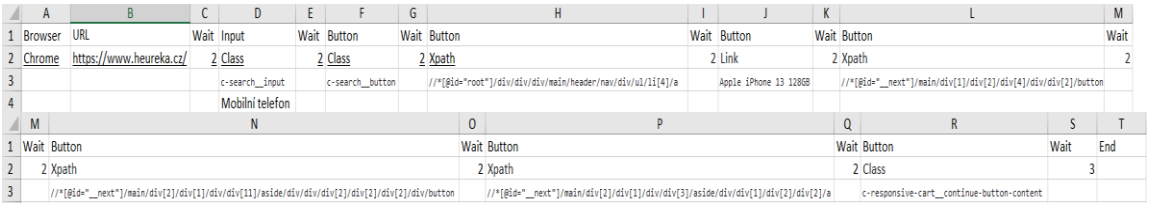
8.9 Varianty testovacích scénářů

Variant testovacích scénářů může být více. Záleží na potřebách a dovednostech testera, jak dokáže využít a kombinovat funkce, které nástroj nabízí. Testy mohou být spouštěny jednotlivě nebo dohromady.

8.9.1 Výběr a přidání výrobku do košíku až k následné platbě

Jednou z variant testovacích scénářů (viz obr. 65) může být testování funkcionalit e-shopu, resp. snaha napodobovat chování uživatele na stránce a pokus o zachycení nějaké chyby. Jedním z typů chování uživatele na stránce je nakoupení produktu. Nástrojem je možné simulovat procházení uživatele na stránce a následné přidávání produktů do košíku, dále projít košíkem až k platebním metodám. V tomto případě bude testovací scénář úspěšný, pokud testovací nástroj dojde na stránce k platebním metodám.

Obrázek 65: Testovací scénář výběru a přidání výrobku do košíku v excelu [Autor]



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Browser	URL	Wait	Input	Wait	Button	Wait	Button	Wait	Button	Wait	Button	Wait
2	Chrome	https://www.heureka.cz/	2	Class	2	Class	2	Xpath	2	Link	2	Xpath	2
3				c-search__input		c-search__button		//*[@id="root"]/div/div/div/main/header/nav/div/ul/li[4]/a		Apple iPhone 13 128GB		//*[@id="__next"]/main/div[1]/div[2]/div[4]/div/div[2]/button	
4				Mobilní telefon									

	M	N	O	P	Q	R	S	T
1	Wait	Button	Wait	Button	Wait	Button	Wait	End
2	2	Xpath	2	Xpath	2	Class	3	
3		//*[@id="__next"]/main/div[2]/div[1]/div/div[11]/aside/div/div/div[2]/div[2]/div[2]/div/button		//*[@id="__next"]/main/div[2]/div[1]/div/div[3]/aside/div/div[1]/div[2]/div[2]/a		c-responsive-cart__continue-button-content		

8.9.2 Porovnání titulku

Testování titulku je jednoduchý testovací scénář (viz obr. 66), který otestuje, zda má stránka titulek, jaký předpokládáme. V jednom testovacím scénáři lze testovat jednu nebo více stránek najednou.

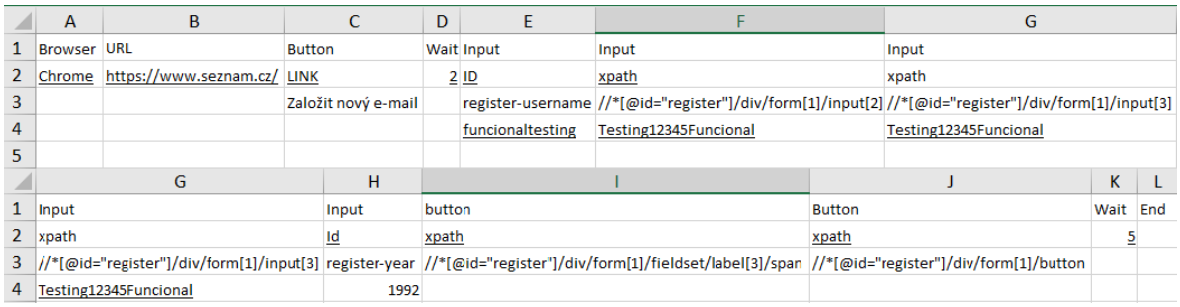
Obrázek 66: Testovací scénář porovnání titulku v excelu [Autor]

	A	B	C	D
1	Browser	URL	Title	End
2	Chrome	https://www.seznam.cz/	Seznam – najdu tam, co neznám	

8.9.3 Vytvoření účtu na stránce

Další variantou může být registrace na webové stránce nebo vytvoření emailu (viz obr. 67). U této varianty, pokud vše proběhne v pořádku a testovací nástroj vyhodnotí test jako úspěšný, by nebylo od věci zkontrolovat registraci v databázi, aby bylo ověřeno, že funkce registrace funguje opravdu správně.

Obrázek 67: Testovací scénář vytvoření účtu na stránce v excelu [Autor]



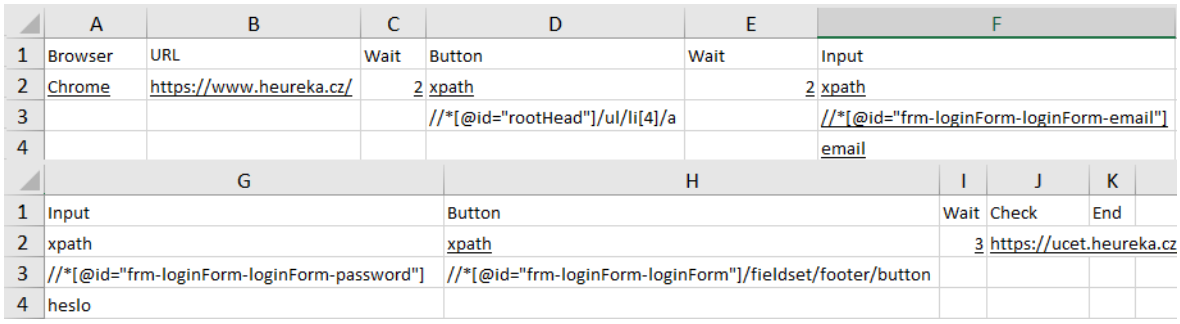
	A	B	C	D	E	F	G
1	Browser	URL	Button	Wait	Input	Input	Input
2	Chrome	https://www.seznam.cz/	LINK	2	ID	xpath	xpath
3			Založit nový e-mail		register-username	//*[@id="register"]/div/form[1]/input[2]	//*[@id="register"]/div/form[1]/input[3]
4					funcionaltesting	Testing12345Funcional	Testing12345Funcional
5							

	G	H	I	J	K	L
1	Input	Input	button	Button	Wait	End
2	xpath	Id	xpath	xpath	5	
3	//*[@id="register"]/div/form[1]/input[3]	register-year	//*[@id="register"]/div/form[1]/fieldset/label[3]/span	//*[@id="register"]/div/form[1]/button		
4	Testing12345Funcional	1992				

8.9.4 Přihlášení

Testování přihlášení je jednou z dalších variant (viz obr. 68). Nelze ho však provést u všech stránek, jelikož při přihlášení uživatel nemusí být přesměrován na jinou stránku, což je pro tento nástroj klíčové. Při vložení emailové adresy, hesla a následného kliknutí na tlačítko přihlášení se porovnává stránka, na kterou byl uživatel přesměrován se stránkou, kde se uživatel přesměroval. Toto porovnání lze provést funkcí check.

Obrázek 68: Testovací scénář přihlášení v excelu [Autor]



	A	B	C	D	E	F
1	Browser	URL	Wait	Button	Wait	Input
2	Chrome	https://www.heureka.cz/	2	xpath	2	xpath
3				//*[@id="rootHead"]/ul/li[4]/a		//*[@id="frm-loginForm-loginForm-email"]
4						email

	G	H	I	J	K
1	Input	Button	Wait	Check	End
2	xpath	xpath	3	https://ucet.heureka.cz	
3	//*[@id="frm-loginForm-loginForm-password"]	//*[@id="frm-loginForm-loginForm"]/fieldset/footer/button			
4	heslo				

9 ZÁVĚR A DOPORUČENÍ

Cíl analýzy oblastí testování, testovacích nástrojů a skriptovacích jazyků byl splněn. Všechny tyto oblasti v práci jsou analyzovány a obsahují strukturovanou logickou posloupnost, která obsahuje většinu fundamentálních informací potřebných k pochopení problematiky. Informace vycházely z knižních a online zdrojů, především z cizojazyčných, neboť tato celková problematika je v tuzemsku nedostatečně popsána. Tato analýza je následně využívána k rozhodovacímu procesu výběru technologií v praktické části.

Cíl výběru technologií byl splněn. Zde se vycházelo z vypracované analýzy literární rešerše. Byly vytvořeny tabulky pro výběr oblasti testování a dle této oblasti i tabulka pro výběr technologií. Vybrané technologie jsou následně použity k vytvoření testovacího nástroje. Z tabulek vyplývá, že technologie nabízí srovnatelné možnosti, kdy spíše záleží na preferencích programátora.

Cíl vytvoření testovacího nástroje byl splněn. Při vytváření nástroje byl nejdříve vytvořen návrh, dle kterého se postupovalo. Nástroj funguje podle navrženého cíle. Potřebné návody, důležité popisy, testovací varianty i rozbor zdrojového kódu byly navrženy a popsány. Testovací nástroj funguje dle očekávání.

Jednou ze silných stránek tohoto nástroje je jeho variabilita tzn., že je tento nástroj velice univerzální a je možno si ho přizpůsobit dle potřeby. Druhou silnou stránkou je jeho rozšiřitelnost dvěma směry. První je rozšiřitelnost funkcionální tzn., že nástroj může pracovat s více funkcemi, které je možné kdykoli přidat. Druhá rozšiřitelnost je platformní, kdy je možné, aby nástroj pracoval s více prohlížeči jako Safari či Firefox. Obě vypsané rozšiřitelnosti zvyšují univerzálnost nástroje a zvyšují množinu využití možných uživatelů.

Celkově práce splňuje všechny požadavky, které byly zadány. Práce utváří aktuální pohled na oblast testování a její nástroje v rámci SW. Ukazuje propojení jednotlivých odvětví automatizace a testování.

V případě potřeby zlepšení kvality nástroje bych doporučil rozšíření o platformy, resp. prohlížeče a dále přidaní funkcionalit Selenia do nástroje. V případě samotného nástroje bych doporučoval zaměřit se na result_file neboli výsledek testování, který se dá upravit dle požadavků testera.

Vytvořený program, tak může sloužit jako šablona pro vytvoření individuálního nástroje s vlastními požadavky na testování.

10 SEZNAM POUŽITÝCH ZDROJŮ

1. **Testování software.** *Černá vs. bílá skříňka*[online]. [cit. 2021-09-30]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=23:erna-vs-bila-skika&catid=3:zaklady&Itemid=11
2. **KITNER, Radek**. Přehled testovacích technik. *Radek Kitner - konzultant, lektor testování softwaru* [online]. [cit. 2021-09-30]. Dostupné z: https://kitner.cz/testovani_softwaru/prehled-testovacich-technik/
3. **KITNER, Radek**. Typy testování software. *Radek Kitner - konzultant, lektor testování softwaru* [online]. [cit. 2021-09-30]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/
4. **HUTCHESON, Marnie L**. *Software testing fundamentals: methods and metrics*. Indianapolis, Ind.: Wiley Pub., 2003. ISBN 978-0471430209.
5. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Unit Testing - SOFTWARE TESTING Fundamentals* [online]. 13.09.2020 [cit. 2021-10-10]. Dostupné z: https://softwaretestingfundamentals.com/unit-testing/
6. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Gray Box Testing - SOFTWARE TESTING Fundamentals* [online]. 17.09.2020 [cit. 2021-10-10]. Dostupné z: https://softwaretestingfundamentals.com/gray-box-testing/
7. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals**. *Agile Testing - SOFTWARE TESTING Fundamentals* [online]. 31.08.2020 [cit. 2021-10-10]. Dostupné z: https://softwaretestingfundamentals.com/agile-testing/
8. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Ad hoc Testing - SOFTWARE TESTING Fundamentals* [online]. 31.08.2020 [cit. 2021-10-10]. Dostupné z: https://softwaretestingfundamentals.com/ad-hoc-testing/
9. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Defect Age - SOFTWARE TESTING Fundamentals* [online]. 06.09.2020 [cit. 2021-10-10]. Dostupné z: https://softwaretestingfundamentals.com/defect-age/
10. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Integration Testing - SOFTWARE TESTING Fundamentals* [online]. 13.09.2020 [cit. 2021-10-25]. Dostupné z: https://softwaretestingfundamentals.com/integration-testing/
11. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *System Testing - SOFTWARE TESTING Fundamentals* [online]. 21.09.2020 [cit. 2021-10-25]. Dostupné z: https://softwaretestingfundamentals.com/system-testing/
12. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Acceptance Testing - SOFTWARE TESTING Fundamentals* [online]. 21.09.2020 [cit. 2021-10-25]. Dostupné z: https://softwaretestingfundamentals.com/acceptance-testing/
13. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Smoke Testing - SOFTWARE TESTING Fundamentals* [online]. 02.09.2020 [cit. 2021-10-25]. Dostupné z: https://softwaretestingfundamentals.com/smoke-testing/

14. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Functional Testing - SOFTWARE TESTING Fundamentals* [online]. 07.09.2020 [cit. 2021-10-25]. Dostupné z: https://softwaretestingfundamentals.com/functional-testing/
15. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Usability Testing - SOFTWARE TESTING Fundamentals* [online]. 06.09.2020 [cit. 2021-11-07]. Dostupné z: https://softwaretestingfundamentals.com/usability-testing/
16. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Security Testing - SOFTWARE TESTING Fundamentals* [online]. 08.09.2020 [cit. 2021-1-07]. Dostupné z: https://softwaretestingfundamentals.com/security-testing/
17. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** Performance Testing - SOFTWARE TESTING Fundamentals [online]. 02.09.2020 [cit. 2021-11-07]. Dostupné z: https://softwaretestingfundamentals.com/performance-testing/
18. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Regression Testing - SOFTWARE TESTING Fundamentals* [online]. 07.09.2020 [cit. 2021-11-07]. Dostupné z: https://softwaretestingfundamentals.com/regression-testing/
19. **Software Testing Fundamentals (STF) ! - SOFTWARE TESTING Fundamentals.** *Compliance Testing - SOFTWARE TESTING Fundamentals* [online]. 07.09.2020 [cit. 2021-11-15]. Dostupné z: https://softwaretestingfundamentals.com/compliance-testing/
20. **SALEH, Hazem.** *JavaScript Unit Testing*. Birmingham, UK.: Packt Publishing, 2013. ISBN 978-1-78216-062-5.
21. **GitHub: Where the world builds software · GitHub.** *GitHub - jasmine/jasmine: Simple JavaScript testing framework for browsers and node.js* [online]. [cit. 2021-11-21]. Dostupné z: https://github.com/jasmine/jasmine#installation
22. **GitHub: Where the world builds software · GitHub.** *Your_first_suite* [online]. [cit. 2021-11-21]. Dostupné z: https://jasmine.github.io/tutorials/your_first_suite
23. **ACHARYA, Sujoy.** *Mastering Unit Testing Using Mockito and JUnit*. Birmingham, UK.: Packt Publishing, 2014. ISBN 978-1-78398-250-9.
24. **GitHub: Where the world builds software · GitHub.** *GitHub - Snaipe/Criterion: A cross-platform C and C++ unit testing framework for the 21st century* [online]. [cit. 2021-11-21]. Dostupné z: https://github.com/Snaipe/Criterion
25. **Microsoft – cloud, počítače, aplikace a hry.** *Testování v .NET - .NET | Microsoft Docs* [online]. 2022 [cit. 2021-11-21]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/testing
26. **Citrus Framework.** *Citrus* [online]. 2021 [cit. 2021-12-03]. Dostupné z: https://citrusframework.org/citrus/reference/3.1.0/html/index.html
27. **JavaScript End to End Testing Framework | cypress.io testing tools.** *Why Cypress? | Cypress Documentation* [online]. [cit. 2021-12-03]. Dostupné z: https://docs.cypress.io/guides/overview/why-cypress#Our-mission

28. **Apache JMeter – Apache JMeter™** [online]. [cit. 2021-12-08]. Dostupné z: https://jmeter.apache.org/
29. **GitHub: Where the world builds software · GitHub.** *Note: 2.0 release | Tavern* [online]. [cit. 2021-12-08]. Dostupné z: https://taverntesting.github.io/
30. **Appium: Mobile App Automation Made Awesome.** *Introduction - Appium* [online]. [cit. 2021-12-15]. Dostupné z: https://appium.io/docs/en/about-appium/intro/
31. **Selenium.** *The Selenium Browser Automation Project | Selenium* [online]. [cit. 2021-12-15]. Dostupné z: https://www.selenium.dev/documentation/
32. **Katalon | Simplify Web, API, Mobile, Desktop Automated Tests.** *Web Application Testing | Complete Guide for Beginners* [online]. [cit. 2021-12-15]. Dostupné z: https://katalon.com/web-testing
33. **BDD Framework for .NET - SpecFlow - Enhance Your Automated Tests.** *Welcome to SpecFlow's documentation! —* documentation [online]. [cit. 2021-12-25]. Dostupné z: https://docs.specflow.org/projects/specflow/en/latest/
34. **The world's # 1 APM solution | AppDynamics.** *Application Monitoring* [online]. [cit. 2021-12-25]. Dostupné z: https://docs.appdynamics.com/21.11/en/application-monitoring
35. **The Complete Continuous Testing Platform | BlazeMeter.** *8 Reasons You Should Use Gatling for Your Load Testing | BlazeMeter* [online]. 2021 [cit. 2021-12-25]. Dostupné z: https://www.blazemeter.com/blog/eight-reasons-you-should-use-gatling-for-your-load-testing
36. **The Complete Continuous Testing Platform | BlazeMeter.** *HammerDB* [online]. [cit. 2021-12-25]. Dostupné z: https://hammerdb.com/about.html
37. **J. KULA, Piotr.** *Raspberry Pi Server Essentials*. Birmingham, UK.: Packt Publishing, 2014. ISBN 9781783985692.
38. **Explore apps for Atlassian products | Atlassian Marketplace.** *Capture for Jira | Atlassian Marketplace* [online]. [cit. 2021-12-25]. Dostupné z: https://marketplace.atlassian.com/apps/306183/capture-for-jira?tab=overview&hosting=cloud
39. **TestRail: Test Management & QA Software for Agile Teams.** *Introduction to TestRail - TestRail [*online]. [cit. 2021-12-25]. Dostupné z: https://www.gurock.com/testrail/docs/user-guide/getting-started/walkthrough/
40. **Business Software and Services Reviews | G2.** *Juno.one Reviews 2022: Details, Pricing, & Features | G2* [online]. [cit. 2021-12-28]. Dostupné z: https://www.g2.com/products/juno-one/reviews
41. **Test Management.** *Test Management with QA Complete* [online]. [cit. 2021-12-30]. Dostupné z: https://www.testmanagement.com/qacomplete/
42. **Upscene: Database tools for Developers.** Database development tools and database developer tools for InterBase, Firebird, Oracle, MS SQL Server, MySQL, NexusDB, SQL Anywhere and Advantage Database. *Test Data Generator / Test Data Generation / Advanced Data Generator @ Upscene Productions* [online]. [cit. 2021-12-30]. Dostupné z: https://www.upscene.com/advanced_data_generator/
43. **Test Data Simplified - DATPROF.** *Data masking with DATPROF Privacy - DATPROF* [online]. [cit. 2021-12-30]. Dostupné z: https://www.datprof.com/products/datprof-privacy/

44. **VAN ROSSUM, Guido a Fred L. DRAKE, JR.** *An Introduction to Python*. Bristol: Network Theory, 2011. ISBN 978-1906966133.
45. **NANCE, Christopher.** *TypeScript Essentials*. Birmingham, UK.: Packt Publishing, 2014. ISBN 978-1-78398-576-0.
46. **SRINIVAS SRIPARASA, Sai.** *JavaScript and JSON Essentials*. Birmingham, UK.: Packt Publishing, 2013. ISBN 978-1-78328-603-4.
47. **The Perl Programming Language - www.perl.org.** About Perl - www.perl.org [online]. [cit. 2021-12-30]. Dostupné z: https://www.perl.org/about.html
48. **Studytonight - Best place to Learn Coding Online.** *Introduction to JavaScript - Studytonight* [online]. [cit. 2021-12-30]. Dostupné z: https://www.studytonight.com/javascript/introduction-to-javascript
49. **Coding Bootcamps & Software Engineering Courses | Hack Reactor | Hack Reactor.** *What is JavaScript used for? | Hack Reactor* [online]. [cit. 2021-12-30]. Dostupné z: https://www.hackreactor.com/blog/what-is-javascript-used-for
50. **Tutorials List - Javatpoint.** *Python Features - javatpoint* [online]. [cit. 2021-12-30]. Dostupné z: https://www.javatpoint.com/python-features
51. **Tutorials List - Javatpoint.** *TypeScript Features - javatpoint* [online]. [cit. 2021-12-30]. Dostupné z: https://www.javatpoint.com/typescript-features
52. **PHP: Hypertext Preprocessor.** *PHP: What is PHP? - Manual* [online]. [cit. 2021-12-30]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php
53. **Tutorials List - Javatpoint.** *Learn PHP Tutorial - javatpoint* [online]. [cit. 2021-12-30]. Dostupné z: https://www.javatpoint.com/php-tutorial
54. **Free How-To Tutorials & Online Courses by Envato Tuts+.** *15 Wonderfully Creative Uses for PHP* [online]. [cit. 2021-12-30]. Dostupné z: https://code.tutsplus.com/tutorials/15-wonderfully-creative-uses-for-php--net-4714
55. **AltexSoft - Technology & Solution Consulting Company.** *Pros and Cons of Katalon Studio Automation Testing Tool | AltexSoft* [online]. [cit. 2021-12-30]. Dostupné z: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/
56. **Testovanisoftwaru.** Hodnocení testů – metriky [online]. [cit. 2021-09-20]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/hodnoceni-testu-metriky