Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Automatizace v testování webových aplikací

David Pilař

© 2021 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

David Pilař

Systémové inženýrství a informatika
Informatika

Název práce

Automatizace v testování webových aplikací

Název anglicky

Test automation of web applications

Cíle práce

Práce je zaměřena na problematiku automatizace v testování. Hlavním cílem práce je rozvoj prostředí pro automatizované testování zvolené webové aplikace. Dílčími cíli jsou analýza dostupných nástrojů a technologií a dále vytvoření samotných automatizovaných testů.

Metodika

Metodika teoretické části práce je založena na studiu a analýze odborných informačních zdrojů. Praktické řešení je realizováno formou rozvoje vybraného automatizačního frameworku, jeho nasazením do testovacího prostředí a následné tvorbě automatizovaných testů. Bude zpracováno porovnání časové efektivity vlastního automatizovaného testování s manuálním. Na základě syntézy poznatků z teoretické a praktické části budou vyvozeny závěry práce.

Doporučený rozsah práce

35-50

Klíčová slova

testování, automatizace, webové aplikace, zaručení kvality softwaru

---

Doporučené zdroje informací

BURNS David; Selenium 2 Testing Tools: Beginner's Guide; ISBN 9781849518307
PATTON Ron; Testování softwaru; ISBN 8072266365
Testování Webových Aplikací [online]; Dostupné na: https://goo.gl/YBKxnv
The Selenium Browser Automation Project [online]; Dostupné na:
https://www.selenium.dev/documentation/en/
Unmesh Gundecha; Selenium WebDriver 3 Practical Guide; ISBN 9781788999762

Předběžný termín obhajoby

2021/22 ZS – PEF

Vedoucí práce

Ing. Jan Pavlík

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 21. 11. 2021

Čestné prohlášení

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

V Praze dne 26. 11. 2021 ____________________________

Poděkování

Rád bych touto cestou poděkoval panu Ing. Janu Pavlíkovi za vedení mé bakalářské práce a cenné rady, které mi pro její zpracování poskytl. Dále bych rád poděkoval rodině a přítelkyni zejména za psychickou podporu a poskytnutí klidného prostředí pro vypracování této práce.

Automatizace v testování webových aplikací

Abstrakt

Bakalářská práce se zabývá problematikou automatizace v oboru testování softwaru, konkrétně webových aplikací.

V teoretické části popisuje a definuje základní principy testování, jeho životní cyklus, kategorie testů, role v testovacím týmu, definuje dokumentaci nezbytnou pro testovací proces a uvádí rozdíly mezi manuálním a automatizovaným testováním, přičemž u automatizovaného testování uvádí příklady využití, jeho výhody a představuje technologie využívané v automatizaci.

Praktická část se poté zabývá analýzou testovacího prostředí ve vybrané firmě. Jsou představeny použité softwarové nástroje, s jejichž pomocí je následně testovací prostředí dále rozvinuto. Jsou definovány nové elementy a metody ze dvou webových stránek, a tyto komponenty následně nalézají uplatnění při vývoji dvou nových automatizovaných testů, jenž vycházejí z testovacích případů vybraných v předešlé analýze. U zmíněných testů je následně zkoumána časová efektivita automatizovaného testování oproti manuálnímu. Ze zjištěných poznatků a dalších výsledků jsou vyvozeny závěry práce.

Klíčová slova: testování, automatizace, webové aplikace, software, zaručení kvality softwaru, Selenium, testovací případ, C#

Test automation of web applications

Abstrakt

The bachelor thesis deals with the issue of automation in the software testing field, specifically for web applications.

The theoretical part describes and defines the basic principles of testing, its life cycle, test categories, roles in the testing team, defines the documentation necessary for the testing process and lists the differences between manual and automated testing. For automated testing, it gives examples of use, its advantages and introduces technologies used in automation.

The practical part then deals with analysis of the test environment in the selected company. The used software tools are introduced, with the help of which the test environment is subsequently further developed. New elements and methods from two web pages are defined and these components are then used within tests which are based upon testcases from aforementioned analysis. For the mentioned tests, the time efficiency of automated testing compared to manual testing is then examined. The conclusions of the work are derived from the findings and other results.

Keywords: testing, automation, web applications, software, quality assurance, Selenium, test case, C#

Obsah

Úvod **11**

1 Cíl práce a metodika **12**

1.1 Cíle 12
1.2 Metodika 12

2 Teoretická východiska **13**

2.1 Testování softwaru 13
2.1.1 Definice testování 13
2.1.2 Cíle testování 13
2.2 Defekt, chyba a selhání softwaru 13
2.2.1 Defekt 14
2.2.2 Chyba 14
2.2.3 Selhání 14
2.3 Dokumentace v testování 15
2.3.1 Testovací plán 15
2.3.2 Testovací nápady 16
2.3.3 Testovací případ 16
2.3.4 Testovací scénář 19
2.3.5 Testovací skript 19
2.3.6 Hlášení defektu 19
2.4 Životní cyklus testování softwaru 21
2.4.1 Analýza požadavků 21
2.4.2 Plánování testování 22
2.4.3 Návrh testovacích scénářů 22
2.4.4 Příprava testovacího prostředí 22
2.4.5 Provedení testů 23
2.4.6 Vyhodnocení výsledků 23
2.5 Úrovně testování 23
2.5.1 Unit testy 24
2.5.2 Assembly testy 24
2.5.3 Smoke testy 24
2.5.4 Integrační testy 24
2.5.5 Systémové testy 25
2.5.6 Akceptační testy 25
2.5.7 Regresní testy 26
2.6 Způsoby testování 26
2.6.1 Manuální testování 26

2.6.2 Automatizované testování ... 27
2.7 Role v testování ... 30
2.7.1 Tester ... 30
2.7.2 Test analytik ... 31
2.7.3 Automatizační inženýr ... 31
2.7.4 Vedoucí testerů ... 31
2.7.5 Test manažer ... 32
3 Vlastní práce ... 33
3.1 Požadavky na automatizované testy ... 33
3.1.1 Finální výběr testovacích případů ... 34
3.1.2 Migrace testovacích případů do Azure Test Plan ... 36
3.2 Použité technologie a nástroje ... 37
3.2.1 Automatizační software ... 37
3.2.2 Testovací software ... 38
3.3 Definice stránek pro Selenium Webdiver ... 38
3.3.1 Seznam objednávek ... 38
3.3.2 Detail objednávky ... 41
3.4 Automatizace testovacích případů ... 46
3.4.1 Modify amount of goods in active uninvoiced order ... 46
3.4.2 Check orders page in BO ... 49
3.5 Časová analýza automatického a manuálního testování ... 51
4 Výsledky a diskuse ... 53
5 Závěr ... 56
6 Bibliografie ... 57

Seznam obrázků

Obrázek 1 - Znázornění defektu, chyby a selhání v systému [2] 14
Obrázek 2 - Životní cyklus testování softwaru [8] 21
Obrázek 3 - Selenium IDE 29
Obrázek 4 - Ukázka filtru v Confluence 34
Obrázek 5 - Kroky testovacího případu "Modify amount of goods in active uninvoiced order" 35
Obrázek 6 - Kroky testovacího případu "Check orders page in BO" 36
Obrázek 7 - Stránka s přehledem objednávek 39
Obrázek 8 - Definované elementy pro stránku s přehledem objednávek 40
Obrázek 9 - Definované metody pro stránku s přehledem objednávek 41
Obrázek 10 - Stránka s detailem objednávky (1) 42
Obrázek 11 - Stránka s detailem objednávky (2) 43
Obrázek 12 - Definované elementy pro stránku s detailem objednávky 44
Obrázek 13 - Definované metody pro stránku s detailem objednávky 45
Obrázek 14 - Testovací data pro test s modifikací počtu kusů položky v objednávce 47
Obrázek 15 - Třída s testem pro modifikaci počtu kusů položky v objednávce 48
Obrázek 16 - Testovací data pro test kontrolující stránku s přehledem objednávek 49
Obrázek 17 – Třída s testem pro kontrolu stránky s přehledem objednávek 50

Seznam tabulek

Tabulka 1 - ukázka testovacího případu 18
Tabulka 2 - Modify amount of goods in active uninvoiced order – čas automatického testu 51
Tabulka 3 - Check orders page in BO – čas automatického testu 51
Tabulka 4 - Modify amount of goods in active uninvoiced order – čas manuálního testu 52
Tabulka 5 - Check orders page in BO – čas manuálního testu 52
Tabulka 6 - graf ročního vývoje času stráveného nad manuálním a automatizovaným testováním 53

Úvod

V dnešní době je běžné, že téměř vše má svou webovou stránku, mobilní aplikaci či počítačový program. Lidé si svůj život zjednodušují použitím nejrůznějších softwarových pomocníků, ať už se jedná o obyčejnou kalkulačku, hudební přehrávač nebo dokonce celý e-shop. Software může být jednoduchý či velice komplexní a za každým z nich stojí minimálně jeden skutečný vývojář a hodiny jeho času stráveného vývojem. Čím je aplikace složitější a na počet funkcí objemnější, tím větší je riziko, že se do ní během vývoje zanese nějaký defekt, který se následně může na straně uživatele projevit chybou, ať už jde o drobný grafický nedostatek v uživatelském rozhraní či fatální selhání celého systému.

Výskyt takovýchto chyb není v žádném softwaru žádoucí, a proto by se v rámci vývoje mělo uplatňovat i testování softwaru. Mnohdy opomíjený a podceňovaný aspekt celého vývojového procesu je klíčový pro úspěch vyvíjeného softwaru. Testování softwaru spadá pod obecnější disciplínu zaručení kvality softwaru a jeho cílem je zkontrolovat vyvinutý software tak, aby se odhalilo co možná nejvíce potenciálních chyb a zamezilo se nepříjemnému dopadu na uživatele daného softwaru.

Testování je nedílnou součástí vývoje a mělo by se určitým způsobem uplatňovat po celou dobu vývojového cyklu, od návrhu až po vydání mezi zákazníky/uživatele. Ačkoliv je stále běžné mnoho věcí testovat manuálně a vždy bude určitá manuální práce v testování nutná, firmy stále více soustředí svou pozornost směrem k automatizaci, s vidinou zefektivnění testovacího procesu, ušetření lidského času, a především snížení finančních nákladů na testování.

Automatizace spočívá v tom, že namísto toho, aby tester procházel testovací případ manuálně, přepíše se testovací případ do speciálního automatizačního nástroje, který následně test provádí sám, s minimální nutností zásahu člověkem, který zpravidla test jen spustí a následně vyhodnotí výsledky.

Tato práce se zabývá právě problematikou automatizovaného testování, klade si za cíl pro konkrétní aplikaci rozvinout testovací prostředí, definovat nové automatické testy a následně vyhodnotit, zda se automatizace skutečně vyplatí oproti čistě manuální práci.

1 Cíl práce a metodika

1.1 Cíle

Práce je zaměřena na problematiku automatizace v testování. Hlavním cílem práce je rozvoj prostředí pro automatizované testování zvolené webové aplikace, konkrétně se jedná o e-shop společnosti Alza.cz a.s. Dílčími cíli jsou analýza dostupných nástrojů a technologií a dále vytvoření samotných automatizovaných testů.

1.2 Metodika

Metodika teoretické části práce je založena na studiu a analýze odborných informačních zdrojů. Praktické řešení je realizováno formou rozvoje vybraného automatizačního frameworku, jeho nasazením do testovacího prostředí a následné tvorbě automatizovaných testů. Bude zpracováno porovnání časové efektivity vlastního automatizovaného testování s manuálním. Na základě syntézy poznatků z teoretické a praktické části budou vyvozeny závěry práce.

2 Teoretická východiska

2.1 Testování softwaru

Testování je součástí vývoje softwaru a mělo by se vyskytovat určitým způsobem ve všech fázích vývoje, od samotného zadání, přes předání výsledného produktu zákazníkovi až po jeho následnou údržbu.

Dalo by se říct, že samotné testování softwaru je jen podmnožinou oblasti zajištění kvality softwaru. V angličtině se tento obor nazývá „Quality Assurance“ a odsud je také odvozena zkratka QA, která se pro toto odvětví softwarového vývoje obecně používá. [1]

2.1.1 Definice testování

Testování je proces hodnocení softwarového produktu za účelem zjištění, zda splňuje požadované podmínky či nikoliv. Proces testování zahrnuje vyhodnocení vlastností softwarového produktu z hlediska funkčnosti, použitelnosti, zabezpečení, spolehlivosti a výkonu.

Dělí se do několika kategorií podle způsobu testování, znalosti kódu či fáze vývoje, ve které se testování provádí. [1]

2.1.2 Cíle testování

Konkrétní cíl testování může být různý, podle toho, jakému druhu testování se zrovna tester věnuje. Obecně je však cílem testování především zjistit, zda daný software odpovídá specifikovanému zadání a identifikovat co nejvíce chyb a problémů, které se v softwaru vyskytují.

Je zde snaha o nalezení co největšího množství defektů, nelze však docílit stavu, kdy je produkt kompletně bezchybný. Často se odstraněním chyby jedné zanesou do softwaru dvě chyby jiné. Libovolný software tedy prakticky vždy bude obsahovat nějaké chyby. [1]

2.2 Defekt, chyba a selhání softwaru

Pojmy defekt, chyba a selhání jsou lidmi často užívány jako synonyma, jejich význam je však odlišný a zaměnitelnost těchto slov je tím pádem vyloučena. [2]

2.2.1 Defekt

V kontextu softwarového systému se defektem rozumí určitá vada v kódu nebo datech, zaviněna zpravidla programátorem, databázovým specialistou a podobně. Příčina vzniku defektu může být různá – chyba přímo v kódu, špatný návrh či nesprávné pochopení specifikace.

Defekt v kódu je považován za neaktivní až do doby, než dojde k vykonání tohoto kódu či zpracování vadných dat. Pokud se tedy z nějakého důvodu nachází v nedosažitelné části kódu, nemusí se defekt vůbec projevit. [2]

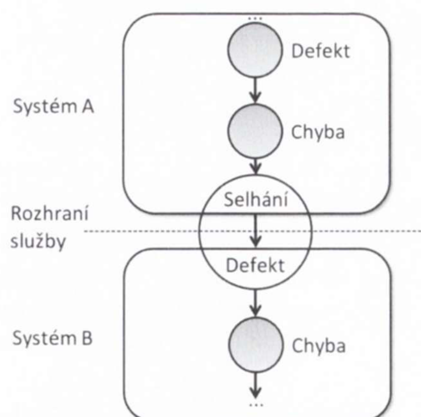
2.2.2 Chyba

Jako chybu se označuje odchýlení alespoň jednoho z externích stavů systému od správného stavu služby. Většinou defekt zapříčiní chybu nejdříve ve stavu určitého komponentu v rámci interního stavu systému. Jeho externí stav tak není ovlivněn okamžitě.

Chybu lze tedy definovat jako následek defektu, přičemž následně může vést k selhání systému. [2]

2.2.3 Selhání

Selhání je označení stavu, kdy se chyba stane součástí externího stavu systému, jinými slovy, když dosáhne rozhraní služby. Tímto se chyba stává viditelnou pro uživatele či jiného příjemce služby, projevuje se jako odchýlení od správného stavu služby a nastává selhání.



Obrázek 1 - Znázornění defektu, chyby a selhání v systému [2]

Pokud je příjemcem služby systému další systém, může selhání původního systému v tomto navazujícím systému způsobit další defekt, který vede k další chybě a ta se následně může opět projevit selháním. [2] Popsaný proces znázorňuje obrázek 1.

2.3 Dokumentace v testování

Testování softwaru se samozřejmě neobejde bez náležité dokumentace. V rámci procesu testování se lze setkat hned s několika druhy dokumentů, které slouží k různým účelům.

Dokumentace v testování softwaru je definována standardem IEEE 829, který byl v roce 2013 zapracován do novějšího ISO/IEC/IEEE 29119 – ten pojednává o softwarovém testování celkově, skládá se z pěti částí a třetí z nich se věnuje právě testovací dokumentaci.

Ačkoliv má dokumentace svou standardizovanou podobu, v rámci menších projektů může docházet k jejímu zjednodušování, nejsou například využívány úplně všechny druhy dokumentů, nebo je struktura jednotlivých dokumentů mírně upravena konkrétním požadavkům. [3]

V následujících podkapitolách jsou popsány nejdůležitější dokumenty na základě praxe, neshodují se tedy stoprocentně s tím, co definuje zmíněný standard.

2.3.1 Testovací plán

Testovacím plánem zpravidla začíná tvorba celé dokumentace a předchází testování jako takovému. Tímto dokumentem se řídí celý proces testování a jsou v něm definovány všechny klíčové náležitosti. [4]

- **Cíl testování** – definuje, co se od testování očekává nebo čeho by mělo být dosaženo. Může se jednat o ověření nové funkčnosti systému, případně všech funkčností v rámci systému či například ověření správného fungování aplikace na konkrétním druhu zařízení. Cílů bývá většinou více najednou.
- **Plánované testy** – jsou založeny na požadavcích testování. Vycházejí převážně z případů užití daného softwaru, případně z dalších požadavků zákazníka či businessu. Ideálně by mělo dojít k naplánování takových testů, aby byla pokryta celá funkčnost systému, a tím pádem se zvýšila šance na odhalení případných chyb.
- **Stanovení priorit** – souvisí úzce s plánovanými testy. Jednotlivým testovaným funkčnostem jsou určeny priority dle předem určené stupnice. Nejvyšší prioritu mají pro systém kriticky nezbytné a nejvýznamnější funkce, případně ty, které by mohly svou nefunkčností ohrozit cíle testování. Stanovení priorit probíhá mnohdy

ve spolupráci s businessem, který na základě požadavků zákazníka dokáže určit, co je pro otestování klíčové.

- **Testovací strategie** – určuje, jaké druhy testů budou pro daný projekt využity a co je jejich cílem. Může se jednat o testy zátěžové, funkční a podobně. U jednotlivých testů je nezbytné určit, v jaké fázi testování budou zařazeny, jakými technikami budou prováděny, jakým způsobem budou vyhodnoceny a podle jakých kritérií.
- **Požadavky na zdroje** – definují, co je vyžadováno k provedení testů. Těmito požadavky se rozumí především požadovaný hardware a software. Mezi požadavky může patřit například konkrétní testovací hardware, specifický software pro potřeby testování, konfigurace serverů nebo příprava testovacích dat. Zahrnuty jsou zde ale i požadavky na zdroje lidské, tedy konkrétní složení testovacího týmu a případně i pracovníků z jiných oddělení, pokud je vyžadována jejich součinnost při testování.
- **Definice rizik** – vymezuje situace, které mohou zapříčinit neúspěšnost testování. Ohrožením pro úspěšnost testování může být třeba nedostupnost potřebného testovacího nástroje či jiného zdroje, nedostatečné proškolení členů týmu nebo nedodání produktu ve stanoveném termínu. Každé riziko má stanovenou svoji míru závažnosti a ideálně by mělo být připraveno určité protiopatření, pokud by některá ze situací skutečně nastala.

2.3.2 Testovací nápady

Seznam testovacích nápadů je o něco méně formální dokument, kam si tester zapisuje myšlenky či návrhy toho, co by v aplikaci bylo vhodné otestovat. Bývají většinou rozděleny podle toho, jaké části aplikace se týkají. [3]

Pokud je například testovanou aplikací e-shop, může být nápadem na testování třeba vyhledání neexistujícího produktu, přidání vyprodané položky do košíku, nákup více kusů položky, než kolik je skladem, registrace s již zaregistrovaným emailem a podobně.

2.3.3 Testovací případ

Často je i v češtině označován anglickým výrazem „test case“. Formálně je dle slovníku ISTQB (International Software Testing Qualifications Board) definován jako „*sada vstupních podmínek, vstupů, očekávaných výsledků, výstupních podmínek a případně akcí, která je vypracována na základě testovacích podmínek.*“ [5]

Jinak řečeno, testovací případ dokumentuje sérii kroků, které tester provádí za specifikovaných podmínek za účelem zjištění, zda na dané vstupy systém reaguje definovanými očekávanými výstupy.

Testovací případy vznikají buď z předem sepsaných testovacích nápadů a myšlenek, nebo na základě případů užití ze specifikace daného produktu.

Většinou má test case podobu tabulky, která eviduje právě jednotlivé kroky, vstupy, výstupy a případně speciální podmínky. Kromě toho obsahuje ještě několik dalších atributů a může mít například následující strukturu. [2]

- Unikátní identifikátor
- Název testovacího případu
- Kategorie nebo zařazení
- Popis případu, jeho účel
- Specifikace vstupních dat, případně dalších prerekvizit
- Autor testovacího případu
- Specifikace kroků, vstupů a očekávaných výsledků

Pro lepší představu o možné podobě testovacího případu je přiložena následující tabulka, která popisuje testovací případ v rámci nespecifikované fiktivní aplikace.

<table>
<tr><td>ID:</td><td>TC-123</td><td colspan="2">Název:</td><td>Přihlášení neexistujícího uživatele</td></tr>
<tr><td rowspan="3">Popis:</td><td rowspan="3">Ověřuje správné chování systému při pokusu o přihlášení uživatele s emailem, který v systému zatím neexistuje.</td><td colspan="2">Prerekvizity:</td><td>V systému není přihlášen žádný uživatel.</td></tr>
<tr><td colspan="2">Kategorie:</td><td>Přihlášení</td></tr>
<tr><td colspan="2">Priorita:</td><td>Střední</td></tr>
<tr><td colspan="5">Testovací kroky:</td></tr>
<tr><td>Číslo</td><td>Popis</td><td colspan="2">Vstupní data</td><td>Očekávaný výsledek</td></tr>
<tr><td>1</td><td>Spusťte aplikaci</td><td colspan="2"></td><td>Aplikace je spuštěna</td></tr>
<tr><td>2</td><td>Přejděte na přihlašovací obrazovku</td><td colspan="2"></td><td>Je zobrazena přihlašovací obrazovka</td></tr>
<tr><td>3</td><td>Zadejte přihlašovací údaje</td><td colspan="2">Email: neexistujici@email.xyz
Heslo: heslo123</td><td>Přihlašovací údaje jsou zadány do patřičných vstupních polí</td></tr>
<tr><td>4</td><td>Klikněte na tlačítko „Přihlásit“</td><td colspan="2"></td><td>Aplikace zobrazí upozornění, že zadané přihlašovací údaje nejsou správné. Uživatel není přihlášen.</td></tr>
</table>

Tabulka 1 - ukázka testovacího případu

2.3.4 Testovací scénář

Testovací scénář je sada několika testovacích případů, které jsou seřazeny a vykonávány v přesně stanoveném pořadí, aby na sebe logicky navazovaly. Hodí se v případě komplexnějších funkcionalit systému, k jejichž otestování nestačí jednoduchý testovací případ. Stejně jako testovací případ by měl mít i testovací scénář určitou strukturu.

- Testovaná oblast
- Kategorie testů
- Testovací případy
- Metriky hodnocení

Zpravidla nacházejí testovací scénáře využití u větších projektů, jelikož pokrývají složitější funkcionality. Pokud je však v rámci projektu evidováno mnoho podrobných testovacích případů, které lze například pro pokrytí testu jedné funkce sloučit, může v takovém případě scénář značně ulehčit práci a zpřehlednit celou dokumentaci. [6]

2.3.5 Testovací skript

Testovací skript je prakticky testovací případ či testovací scénář přepsaný do programovacího jazyka. Testovací skripty se využívají k automatizaci testování – neprovádí jej manuálně člověk, ale specializovaný nástroj. [3]

Vzhledem ke své povaze nemají testovací skripty obecně stanovenou podobu či strukturu. Záleží vždy na pravidlech konkrétního automatizačního nástroje, programovacím jazyku, dalších podpůrných nástrojích a podobně.

2.3.6 Hlášení defektu

Hlášení defektu, častěji označovaný jako „bug report", je po testovacím případu takřka druhým nejdůležitějším dokumentem, se kterým testeři přicházejí do styku. Využívají se v případě, kdy tester narazí na selhání systému, tedy chybu vyvolanou nějakým defektem. Ačkoliv se dokument nazývá hlášení defektu, jeho autor zpravidla nedokáže přesně určit místo v kódu, které je chybné. V rámci hlášení se pouze popíše chybné chování systému či jiná odchylka od zadání a je až na vývojáři, aby dle popisu defekt našel a odstranil.

Podobně jako testovací případ má i hlášení defektu trochu komplexnější strukturu – nejedná se o pouhý odstavec textu. Hlášení lze psát i v obyčejném textovém či tabulkovém preprocesoru, zpravidla se však používá specializovaný software, který umožňuje jejich přehlednější správu, sledování aktuálního stavu hlášení či napojení na konkrétní testovací případ, kterého se týká.

Hlášení defektu by mělo mít přibližně následující strukturu. [2]

- **Unikátní identifikátor** – v rámci systému pro správu defektů. Každé hlášení je evidováno pod unikátním kódem, podobně jako testovací případy.
- **Shrnutí** – v rozsahu několika málo slov. Mělo by z něj být zřejmé, čeho se chyba týká, kde přesně se vyskytla a jaká akce k jejímu projevení vedla.
- **Priorita** – či závažnost, stejně jako testovací případy mají i hlášení defektů své priority, dle kterých jsou dále odbavovány.
- **Informace o prostředí** – jako například verze použité aplikace, operační systém zařízení, případně jazykové nastavení. Zkrátka relevantní informace, které zcela upřesní, na jaké platformě a konfiguraci aplikace se chyba vyskytla.
- **Autor** – je nálezce chyby. Může být odlišný od člověka, který vytváří hlášení v systému.
- **Řešitel** – určuje, na kterou osobu je hlášení aktuálně přiřazeno. V průběhu životního cyklu defektu se aktuální řešitel mění. Opakovaně si ho například mohou předávat vývojář a tester do doby, než je chyba zcela odstraněna.
- **Stav** – eviduje aktuální stav daného hlášení.
- **Popis** – je nejdůležitější částí hlášení a obsahuje podrobný popis projevů defektu. Ideálně by mělo zahrnovat i rozpis kroků k reprodukci dané chyby, díky čemuž si následně kdokoliv, kdo bude hlášení číst, dokáže sám vyvolat popisovanou chybu. V rámci popisu je vhodné uvézt, jak se systém aktuálně chová a jaké je naopak očekávané chování dle návrhu či specifikace.
- **Komentáře** – se vážou k hlášení defektu a slouží k průběžné komunikaci různých členů týmu, případně k doplnění dodatečných informací a podobně.
- **Přílohy** – jsou důležitým doplňkem k samotnému hlášení. Většinou se připojují snímky obrazovky zachycující popisovanou chybu, případně videonahrávky, logy z aplikace či systému a další soubory, které by mohly být při řešení nápomocné.
- **Odpracovaný čas** – slouží k evidenci času, který byl nad řešením chyby stráven.

Hlášení může být dále doplněno například o vazby na jiná související hlášení nebo může odkazovat na duplicitní hlášení, která pojednávají o totožné chybě. Mnohdy dokážou systémy pro evidenci defektů integrovat nástroje a systémy třetích stran, což dále v mnoha ohledech zjednodušuje práci. [2]

2.4 Životní cyklus testování softwaru

Životní cyklus testování softwaru (z angličtiny zkráceně STLC) označuje proces, který se skládá z jednotlivých kroků, které je třeba v konkrétním pořadí provést, aby bylo zajištěno, že produkt splňuje zadané požadavky a další nároky na kvalitu. STLC se může lišit napříč organizacemi, základ je ale vždy stejný.

Každá fáze má určitá vstupní a výstupní kritéria. Za ideálních podmínek by se nemělo přecházet k další fázi do té doby, než jsou splněna výstupní kritéria z fáze předchozí. Toho však v praxi občas nelze dosáhnout. [7]

Obrázek 2 - Životní cyklus testování softwaru [8]

Na obrázku 2 lze vidět grafickou interpretaci životního cyklu testování softwaru a jeho jednotlivé fáze, které jsou podrobněji vysvětleny v následujících podkapitolách.

2.4.1 Analýza požadavků

Nejprve je třeba analyzovat zadání, zpravidla má tester k dispozici business analýzu, kde jsou specifikovány požadavky na výsledný produkt. Tester má za úkol především zjistit, zda jsou požadavky testovatelné či nikoliv. Je zde prostor pro diskusi s ostatními zainteresovanými stranami a je nezbytné odstranit veškeré nejasnosti ohledně případných

netestovatelných požadavků. Požadavky mohou být buď funkčního charakteru (jak má produkt fungovat) nebo nefunkčního (design, výkon, zabezpečení a podobně). [7]

2.4.2 Plánování testování

Další fází je sestavení plánu testování. Jde o nejdůležitější ze všech kroků, protože je zde definována testovací strategie a testovacím plánem se dále řídí všechny zbylé kroky. V rámci plánování je nutné definovat celkový rozsah testování, aby byly pokryty všechny požadavky ze zadání. V plánu je dále třeba stanovit odhad náročnosti na zdroje, a to jak lidské (kolik testerů je na projekt potřeba) tak i časové (na jak dlouho je odhadováno celé testování). Zejména časový odhad je důležitý například v rámci projektového řízení.

V plánu je dále zahrnut výčet testovacích nástrojů, definice testovacího prostředí, případně definice testovacích dat, která jsou k tomu nezbytná. [7]

2.4.3 Návrh testovacích scénářů

Když je připravený plán testování, může se přejít k tvorbě testovacích scénářů. Každý testovací scénář popisuje detailně krok za krokem, jak postupovat za účelem dosažení určitého výsledku v rámci funkcionality daného systému. V podstatě každý ze zadaných požadavků by měl být pokrytý nějakým testovacím scénářem.

Pokud je například v rámci vznikajícího e-shopu jedním ze zadaných požadavků možnost vkládání zboží do košíku, bude se k němu vázat testovací scénář popisující, jak v katalogu najít konkrétní zboží a „proklikat" se jednotlivými kroky až k cílovému stavu, tedy stavu, kdy je zboží vloženo v košíku. [7]

2.4.4 Příprava testovacího prostředí

Testovací prostředí je jednou z klíčových částí testovacího procesu. Určuje totiž, v jakých podmínkách bude software testovaný. Přesněji řečeno, definuje, oproti jaké hardwarové a softwarové konfiguraci se bude produkt testovat.

S přípravou testovacího prostředí mnohdy asistuje i vývojový tým, případně databázoví specialisté a podobně, dle požadavků.

Obzvlášť v případě, kdy je testována například nová funkcionalita již existujícího softwaru, je dobrá definice a příprava testovacího prostředí velmi důležitá. Teoreticky totiž může, třeba chybou nastavení prostředí, dojít k situaci, kdy se testuje na produkčním prostředí. To není žádoucí, jelikož tímto se přímo zasahuje do systémů, ke kterým

přistupují uživatelé a je zde větší riziko, že se něco pokazí, přičemž následky chyb jsou pak horší, než když je testování realizováno na vyhrazeném prostředí, do kterého nikdo jiný než testovací a případně vývojový tým nepřistupuje. [7]

2.4.5 Provedení testů

Fáze provedení testů je přímo závislá na dokončení dvou předchozích fází, tedy definici scénářů a přípravě prostředí. V této fázi tým testerů provádí jednotlivé testy a je zaznamenávána jejich úspěšnost. Pokud se podaří všemi kroky testovacího scénáře projít až do jeho cílového stavu, je test považován za úspěšný. Pokud se ale v rámci některého scénáře setká tester s odlišným než očekávaným výsledkem, test není úspěšný a je nutné zaznamenat hlášení o nalezené chybě. To je poté spárováno se scénářem, během kterého byla chyba nalezena, a vývojář má následně za úkol nalézt příčinu chyby a zajistit její nápravu. Následně tester znovu opakuje daný scénář a ověřuje, zda byla nahlášená chyba opravena. [7]

2.4.6 Vyhodnocení výsledků

Když jsou všechny testy dokončeny, dojde k jejich vyhodnocení a jsou vyvozeny závěry celého testování. Zpětně se vyhodnocuje úspěšnost testů či celková časová náročnost testování oproti původnímu odhadu. U testů, které neskončily s úspěšným výsledkem, se v této fázi čeká na opravu jejich příčiny, případně je upraven testovací scénář, pokud chyba není v softwaru, ale byl jen špatně sestaven konkrétní scénář.

Může proběhnout i porada, kde se prodiskutuje, co se během testování podařilo, případně co je třeba v procesu do příště vylepšit. [7]

2.5 Úrovně testování

Testování není jednorázový proces, je svázáno s celým vývojem a promítá se do všech fází životního cyklu vývoje softwaru. V různých fázích vývoje se uplatňují různé druhy testů, jinými slovy každý druh testu slouží k ověření odlišné fáze vývoje. Principem takového přístupu je otestovat software od malých částí, přes větší celky až po jeho kompletní podobu. [9]

2.5.1 Unit testy

Úplně první fázi testování mají v rukách samotní vývojáři. Unit testy se zaměřují na testování jednotlivých komponent, objektů a tříd. Unit testy nespadají do plánu testování a zpravidla si pomocí nich jen vývojáři ověřují, že kód, který napsali, funguje sám o sobě korektně a dělá to, co vývojář očekává. [9]

2.5.2 Assembly testy

Assembly testy jsou trochu pokročilejší obdobou unit testů a mají za úkol ověřit, že jednotlivé komponenty aplikace lze zkombinovat do funkčního celku. Stejně jako unit testy má i assembly testy zpravidla na starost sám vývojář, v případě složitějších celků se na tomto úkolu může podílet více zkušenějších členů týmu. [9]

2.5.3 Smoke testy

Smoke testy předcházejí samotnému testování vyvíjené aplikace a mají za úkol zjistit, zda je připravené korektně testovací prostředí a také software, který se bude testovat. Typicky jde o kontrolu, zda je software správně nainstalovaný, spuštěný a nakonfigurovaný vůči danému testovacímu prostředí. Zmíněné náležitosti lze otestovat například jednoduchým průchodem testovaným programem, čímž se snadno ověří všechny vyjmenované atributy.

Na rozdíl od prvních dvou úrovní jsou již smoke testy v režii týmu testerů, případně jsou prováděny za asistence člověka, který má na starosti správu testovacích prostředí, což však mnohdy bývají samotní testeři. [9]

2.5.4 Integrační testy

U integračních testů je rozlišováno mezi dvěma druhy integrace – vnitřní a vnější. Vnitřní spočívá ve vzájemné komunikaci součástí samotného testovaného softwaru. Vnější integrace je o komunikaci našeho softwaru s nějakými externími aplikacemi, které dohromady tvoří větší celek. Integrační testy mají velký význam v obou případech.

Obvykle se testují nejprve jednotlivé moduly a postupuje se směrem k větším celkům. Velmi často se při testování používají takzvané falešné moduly, které jen napodobují chování těch skutečných. Uplatnění naleznou například v situaci, kdy je nutné otestovat komunikaci jednoho konkrétního modulu s ostatními. Falešné moduly simulují chování těch reálných a lze díky tomu ověřit, že námi testovaný modul korektně odesílá

a přijímá data právě od ostatních modulů. V dalších fázích se postupně testuje více modulů spojených dohromady, až se nakonec otestuje daný software jako celek.

U vnější integrace je často testování náročnější, jelikož jednotlivé aplikace mohou pocházet od různých výrobců. Je zde nutná minimálně distribuce důležitých informací, nějaké dokumentace a podobně. V komplikovanějších situacích může být nezbytná spolupráce vývojových týmů všech zainteresovaných firem, z čehož vyplývá větší náročnost jak na časové, tak i lidské zdroje. Oproti tomu u vnitřní integrace si lze vystačit s lidmi a znalostmi v rámci vlastní firmy. [9]

2.5.5 Systémové testy

Systémové testy jsou zpravidla přesně to, co si většina lidí představí, když se řekne testování. Jde o kontrolu, zda software jako celek funguje korektně a dle návrhu. Testuje se, zda software dělá to, pro co byl vytvořen, že přijímá očekávané vstupy a vrací správné výstupy, případně že umí zpracovat a zareagovat i na množství neočekávaných vstupů. Především otestování mnoha nestandardních situací a okrajových případů bývá klíčové při odhalování chyb, na které nemusel nikdo při návrhu či vývoji pomyslet. V neposlední řadě je důležité otestovat, že byly splněny všechny požadavky zákazníka.

Systémové testy v drtivé většině případů probíhají v několika iteracích, jelikož během testování jsou nacházeny a průběžně opravovány chyby. Jen výjimečně se stane, že již při prvním kole testů nejsou nalezeny žádné chyby. Je tedy nutné po každé úpravě projít celý software znovu, ověřit především že byly opraveny chyby nahlášené, a že se neobjevily žádné nové. [9]

2.5.6 Akceptační testy

Akceptační testy navazují na testy systémové a jde o nejpodstatnější část testování z pohledu zákazníka. V této fázi se ověřuje, zda software splňuje veškeré zákazníkovy požadavky. Akceptační testy může provádět rovněž tým testerů, bývá ale zvykem, že si akceptační testy realizuje sám zákazník, případně jsou provedeny jím stanoveným týmem.

V rámci akceptačních testů se software porovnává s akceptačními kritérii, která jsou obvykle dohodnuta a stanovena buď před testováním, nebo ještě před samotným vývojem. Kritéria mohou definovat například maximální možný počet nalezených chyb, požadované výkonové parametry a podobně. Pokud jsou tyto požadavky splněny, je software téměř připraven k předání zákazníkovi. [9]

2.5.7 Regresní testy

Prakticky poslední fází testování jsou regresní testy. Jejich úkolem je ověřit, že provedené zásahy do softwaru nenarušili jeho již existující funkcionalitu. Provádějí se jak při samotném vývoji (po přidání každé nové funkce se regresně ověří, že se nepokazily předchozí přidané funkce), tak i později před každým uvedením nových změn do již vyvinutého softwaru.

Zpravidla bývá regresními testy pokryta kritická funkcionalita daného systému, mnohdy vzhledem k jeho rozsahu není možné testovat kompletně všechny jeho části. Jsou tedy definovány klíčové oblasti, které bezpodmínečně musejí fungovat, a na ty se následně regresní testy zaměřují. Tento soubor testů se může případně modifikovat, pokud jsou ze systému kritické funkce odebrány, nebo jsou naopak přidávány nové, u kterých se rozhodne o nutnosti jejich zařazení do regrese.

Jelikož jde o dlouhodobě ustálenou sadu testů se známým výsledkem, jsou mnohdy regresní testy automatizovány, což následně šetří čas a výrazně se tím testerům usnadňuje provádění těchto testů. [9]

2.6 Způsoby testování

Rozlišuje se v zásadě mezi dvěma hlavními způsoby, jak lze realizovat testování softwaru. Jejich charakteristiku, výhody a případné nevýhody popisují následující podkapitoly. [10]

2.6.1 Manuální testování

Manuální testování bylo v podstatě popsané v kapitole „Životní cyklus testování softwaru". Jde o testy, které ručně provádí sám tester dle scénářů. Může být výhodné například při testování uživatelského rozhraní (UI), kde je lidský instinkt stále nenahraditelný. Zároveň se při manuálním testování dá lépe zjistit už z pohledu testera, kde přesně vzniká příčina určité chyby – člověk dokáže sledovat a vnímat souvislosti.

Lidský faktor zde ale může být i negativem, jelikož může nějaké věci přehlédnout, špatně pochopit či na jejich otestování úplně zapomenout. Manuální testování tak nikdy nelze považovat za 100% spolehlivé.

Speciálním druhem manuálního testování je pak takzvané explorativní testování. Od klasického manuálního se liší v tom, že se tester neřídí předepsanými scénáři, ale prozkoumává přirozeným způsobem daný software, čímž se snaží zjistit, jak se daný

software chová, jaké má vlastnosti a funkce. I při explorativním testování přitom může dojít k nalezení mnoha chyb. [10]

2.6.2 Automatizované testování

Automatizace je považována za nejpokrokovější disciplínu testování softwaru, ačkoliv má již poměrně bohatou historii. Automatizované testování je prakticky kód, který kontroluje kód. Jde o krátké programy, psané v různých jazycích, které mají za úkol ověřit určitou funkcionalitu.

Jeho výhoda je především v rychlosti. To, co by tester procházel několik minut, má automatizovaný test mnohdy hotové jen za několik sekund. Kromě toho jsou testy jednoduše škálovatelné a zpravidla i cenově výhodnější.

Nevýhoda automatizace může být například ve vysokých počátečních nákladech na její zavedení. Samotná údržba testů a jejich konfigurace zabírá mnohdy spoustu času, zejména pokud se testovaný software často mění. S každou změnou podoby softwaru je třeba zrevidovat i související automatizované testy, aby skutečně kontrolovaly to, co mají.

Na rozdíl od manuálního testera nedokáže automatizovaný test posoudit faktory UX, takže v tomto ohledu stále vítězí lidský přístup. Automatizace může být také poměrně neefektivní pro menší software, u kterého by vývoj automatizovaných testů zabral neúměrně dlouhou dobu a nevyplatil by se oproti manuálnímu otestování. [3]

Automatizace nalezne dobré uplatnění zejména u regresních testů, které vyžadují opakované vykonávání těch samých testovacích scénářů na pravidelné bázi. Manuální testeři mohou takto opakovanou činností časem začít přehlížet i dobře patrné chyby, test již nemusí provádět s takovou pečlivostí a rychlostí, jako z počátku. Oproti tomu automatizované testy dodávají konzistentní výsledky, jejich exekuce trvá pokaždé přibližně stejnou dobu a nemůže se stát, že by takový test přehlédl nějakou chybu.

Při volbě testů ke zautomatizování je také dobré brát v potaz jejich prioritu z hlediska businessu a z technického hlediska. Pokud je testovací případ příliš složitý pro manuální testování a lze jej zautomatizovat, je určitě vhodné to zvážit. To samé v případě, že je testovaná oblast z pohledu businessu klíčová a její funkčnost je stěžejní pro daný software. [11]

Pro automatizaci testů slouží specializované nástroje, lišící se v několika oblastech. Rozdílný je například způsob zápisu testů. Některé nástroje dokážou zaznamenávat akce

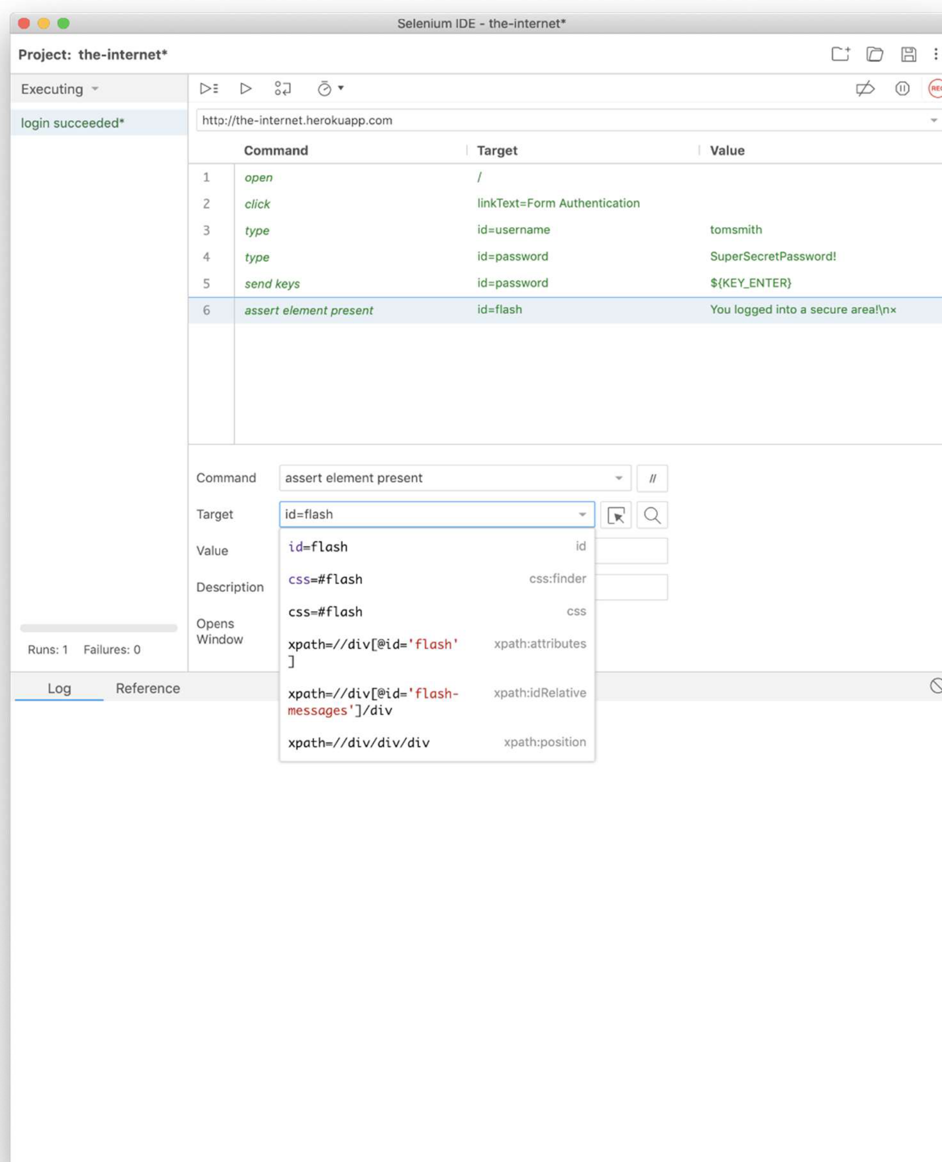
prováděné ručně uživatelem a ty následně automaticky reprodukovat, běžnější jsou však nástroje, kde se testy zapisují skriptovacím či plnohodnotným programovacím jazykem.

Jednotlivé automatizační softwary většinou dokážou zajistit automatizaci jen pro jednu platformu. Pokud tedy například firma vyvíjí program pro desktop, web i mobilní zařízení, je zpravidla nutné pro automatizaci na každé platformě volit specifický automatizační nástroj. [12]

Liší se pak i cena samotných nástrojů. Některé jsou jako open-source dostupné zcela zdarma, jiná řešení mohou být zpoplatněna v řádu až desítek tisíc korun ročně za jednu licenci. [13]

Mezi nejpopulárnější nástroje pro webové testování patří Selenium. Umožňuje tvorbu testů v několika programovacích jazycích a jednoduše se integruje s dalšími frameworky, což značně rozšiřuje celkové možnosti jeho užití. Jedná se o sadu celkem čtyř jednotlivých programů, z nichž každý nějakým způsobem umožňuje automatizaci v procesu testování.

Selenium IDE (Integrated Development Environment) je rozšíření do internetového prohlížeče, jenž umožňuje nahrávat akce prováděné uživatelem, generovat na základě nich testovací skripty a ty následně automaticky opakovat. Výhodou je, že jako jediný ze Selenium sady nevyžaduje znalost programování či skriptování a je tedy vhodný i pro úplné začátečníky v automatizaci. Nevýhodou je velmi omezená podpora – lze jej využít jen v prohlížeči Mozilla Firefox. Na obrázku 3 lze vidět uživatelské rozhraní této aplikace.



Obrázek 3 - Selenium IDE

Dalším nástrojem je Selenium Grid. Ten umožňuje simultánní běh několika testů na různých prohlížečích, operačních systémech a rozlišných zařízeních. Test jako takový běží na centrálním hubu, přičemž je zajištěna paralelní exekuce na dalších vzdálených zařízeních. Na jednom stroji lze současně testovat až v pěti prohlížečích zároveň.

Dále Selenium nabízí Selenium RC (Remote Control), to umožňuje testování uživatelského rozhraní pomocí příkazů v programovacím jazyce (Java, C#, PHP, Python, Ruby a PERL). Selenium RC provádí testy v internetovém prohlížeči za použití JavaScriptu. Celý nástroj se skládá ze dvou komponent. RC Server zahrnuje sadu JavaScriptové příkazy pro samotné ovládání prohlížeče a zároveň slouží jako HTTP

(Hypertext Transfer Protocol) proxy ověřující zprávy mezi prohlížečem a aplikací. Druhou komponentou je RC Client poskytující rozhraní mezi programovacím jazykem a serverem. Nevýhodou Selenium RC je jeho omezená schopnost paralelního běhu více testů, má problémy s rychlostí a komplexita některých jeho funkcí nakonec vedla k nahrazení nástrojem Selenium Webdriver.

Posledním z nástrojů je Selenium Webdriver, přičemž ten je nejrozšířenější z celé sady. Nabízí jednodušší a stručnější programovací rozhraní a řeší spoustu nedostatků, kterými trpělo Selenium RC. Umožňuje na základě příkazů v kódu ovládat internetový prohlížeč. Jednotlivé prvky na stránce se v kódu zapisují jako proměnné a k jejich identifikaci na reálné stránce se využívá HTML (HyperText Markup Language) lokátorů, jako jsou třídy, id a podobně.

Mezi jeho největší výhody patří podpora více internetových prohlížečů, široká podpora programovacích jazyků pro tvorbu testů, jednoduše se integruje s dalšími testovacími frameworky, disponuje integrovanou správou defektů a umožňuje jejich přímé hlášení, a v neposlední řadě je dostupný zcela zdarma.

Nevýhodou Webdriveru je, že podporuje pouze testování webových aplikací, vyžaduje velmi dobrou znalost programovacího jazyka pro tvorbu testů, kvůli open-source distribuci není k dispozici oficiální technická podpora, a pro jeho maximální využití je nutné jej doplnit a integrovat s dalšími nástroji a frameworky, které rozšíří jeho funkcionalitu pro konkrétní potřeby dané firmy. [14]

2.7 Role v testování

V rámci týmu testerů se uplatňují lidé s různými rolemi, jinými slovy, ne každý tester má na starosti ty samé věci.

2.7.1 Tester

Role „obyčejného“ testera se může lišit podle toho, pro jak velkou společnost pracuje. Většinou ale práce testera spočívá především v provádění manuálních testů dle scénářů. Zpravidla je nutné, aby se tester před testováním s aplikací důkladně seznámil a věděl tak s předstihem, na co se má připravit. Úkolem testera je během testování zaznamenávat chyby, hlásit je vývojářům a následně se postarat o opětovné testování po jejich opravě.

Tester nemusí mít nutně znalosti z oboru IT (informační technologie), je však důležité, aby měl správné testerské myšlení. Musí se na daný software dívat z různých úhlů, přemýšlet tak trochu za uživatele a být vynalézavý, aby dokázal mnoha způsoby a různými cestami dosáhnout cílových stavů v jednotlivých procesech. Jen tak je možné docílit toho, že se během testování odladí většina případných chyb.

Důležitá je také schopnost komunikace. Tester musí pochopit principy testovaného softwaru a jakákoliv neobjasněná záležitost, třeba už v zadání, může způsobit problém. Proto je nezbytné, aby tester kladl dobře formulované otázky, pokud mu cokoliv není jasné. Obdobně to platí i z druhé strany, když tester zadává hlášení o nalezené chybě. Její popis musí být dostatečně jasný a srozumitelný, aby byl pochopitelný každým, kdo si ho přečte. Výrazně se tím urychluje celý proces testování. [15]

2.7.2 Test analytik

Úkolem test analytika je především zpracovávání testovacích analýz. Podle již zpracované business analýzy vznikajícího softwaru by měl být test analytik schopen pochopit jeho principy a na základě toho zpracovat dokumentaci k testovacímu procesu, případně rovnou testovací scénáře. V rámci analýzy by měly být definovány požadavky na testovací data, prostředí a podobně. [2]

Mnohdy se stává, že roli test analytika zastává zároveň sám tester. V takovém případě tedy ta stejná osoba zároveň navrhuje testovací analýzu a následně se stará o exekuci testů.

2.7.3 Automatizační inženýr

Tento člověk má na starosti tvorbu a správu automatizovaných testů. Většinou úzce spolupracuje s test analytiky a dle technologie, strategie a požadavků pomocí vhodného nástroje automatizuje jednotlivé testovací scénáře.

Kromě tvorby a údržby testů také vytváří potřebnou dokumentaci, zaznamenává výsledky jednotlivých běhů testů a spravuje také celkové prostředí, kde automatizované testy běží. [2]

2.7.4 Vedoucí testerů

Jeho úkolem je tvorba plánu testování v souladu se strategií a kontrola jeho dodržování. Je zodpovědný za nastavení a správu systému pro hlášení defektů, zadává

úkoly jednotlivým členům týmu a kontroluje jejich plnění. Z testování generuje pravidelná hlášení pro své nadřízené.

Měl by mít bohaté znalosti z teorie i praxe testování softwaru, manažerské a organizační dovednosti. U velkých projektů bývá často přítomno více vedoucích testerů, tým každého z nich pak pracuje na odlišné části systému (například frontend a backend). [2]

2.7.5 Test manažer

Test manažer je zodpovědný za vedení veškerých aktivit souvisejících s testováním, spolupracuje s vedoucím testování a případně schvaluje jeho rozhodnutí.

Jeho úkolem je příprava celkového přístupu k testování a návrh testovací strategie, nastavení jeho cílů, akceptačních kritérií či vyjednávání s managementem, zákazníkem a vývojovým oddělením. Dále odpovídá za zajištění a správu hardwarových a softwarových zdrojů pro potřeby testování.

V případě menších projektů bývá často role test manažera a vedoucího testerů zastávána jedním člověkem. [2]

3 Vlastní práce

Praktická část se zabývá rozvojem prostředí pro automatizované testování ve vybrané firmě, konkrétně se jedná o společnost Alza.cz a.s. Samotné práce se pak týkají problematiky testování webové aplikace, tedy e-shopu.

Nejprve je proveden sběr požadavků a analýza testovacích případů, což je nutná prerekvizita pro následný definitivní výběr testovacích případů, které je možné zautomatizovat.

3.1 Požadavky na automatizované testy

Proces automatizace není jen o psaní kódu. V první řadě je nutné zanalyzovat dostupné testovací případy a posoudit jejich vhodnost ke zautomatizování. Konkrétně v případě webového testování je například nutné odfiltrovat si testy, v jejichž rámci se přistupuje do jiných aplikací. Mezi ty může patřit například interní informační systém, logovací nástroje, databázový server a podobně. Nástroje pro webovou automatizaci zkrátka umožňují manipulovat pouze s internetovým prohlížečem.

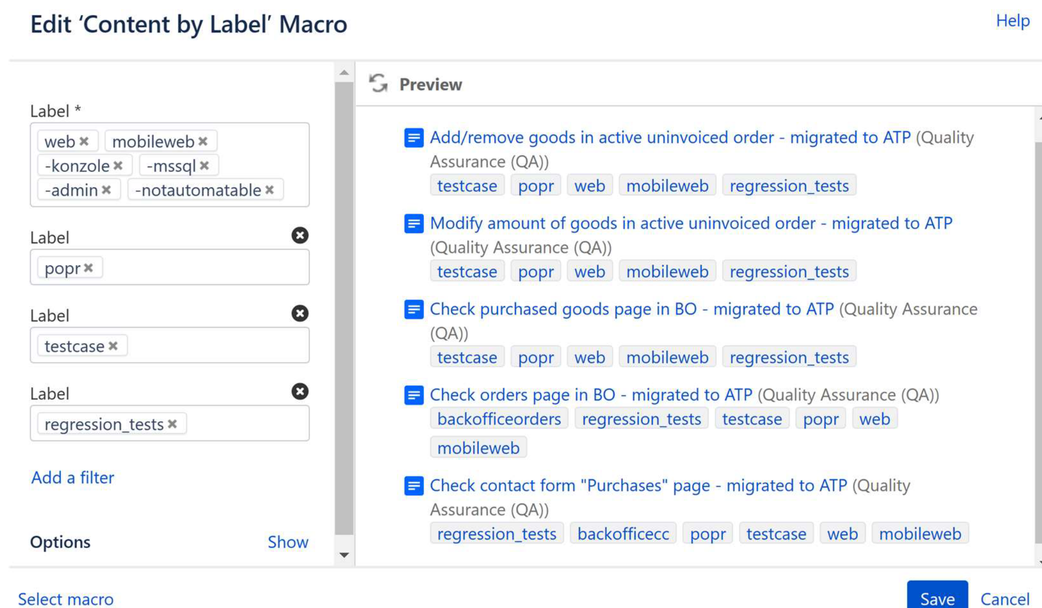
Jako další požadavek a výběrový parametr je stanovena priorita. Každý testovací případ má na škále „low, medium, high“ nastavenou prioritu, která určuje, jak moc důležité je mít danou oblast otestovanou. Autor si vybírá z testů, které mají prioritu medium nebo high, jelikož tyto testy pokrývají většinou kriticky důležité komponenty webové aplikace.

Pro firmu je důležité především automatizovat pravidelně prováděné testy z regresní sady. Jedná se o sadu testovacích případů, které je nutné vykonat pravidelně každý týden v rámci vydání nové verze aplikace. Výběr tedy bude dále omezen jen na testovací případy z této sady.

V neposlední řadě je výběr omezen jen na testovací případy spadající pod produktovou skupinu „Poprodejní procesy“ (dále jen POPR), ve které autor působí. Tím je zároveň zajištěno, že se na práci nepodílejí ostatní kolegové z jiných produktových skupin, jelikož jsou v tomto směru kompetence striktně rozdělené.

3.1.1 Finální výběr testovacích případů

Na základě výše uvedených parametrů byly tedy vyfiltrovány odpovídající testovací případy. Za tímto účelem bylo pomocí integrovaného nástroje v rámci Confluence (firemní wiki) vytvořeno makro, které dle požadovaných výše uvedených kritérií zobrazilo jen testovací případy, které lze zvažovat k automatizaci.



Obrázek 4 - Ukázka filtru v Confluence

Obrázek 4 znázorňuje postup při výběru vhodných testovacích případů. Filtr v levém sloupci zajišťuje, že ve výsledném výběru vpravo budou jen testovací případy, které se týkají webu, mobilního webu, spadají pod produktovou skupinu POPR a jsou součástí sady regresních testů. Naopak není žádoucí, aby tyto případy vyžadovaly práci s konzolí (interním IS), SQL databází, adminem (portál webové administrace) a také ty, které jsou přímo označeny labelem jako nevhodné pro automatizaci (notautomatable) z jakéhokoliv jiného důvodu.

Z tohoto výčtu si autor pro další práci vybral testovací případy „Modify amount of goods in active uninvoiced order“ a „Check orders page in BO“.

Modify amount of goods in active uninvoiced order

Hlavním cílem testovacího případu je ověřit, že na detailu aktivní nevyfakturované objednávky korektně funguje modifikace počtu kusů zakoupené položky. Úkolem testera je tedy zjistit, zda když na stránce s detailem objednávky klikne na tlačítko „+“ či „-“ u počtu kusů položky, dojde správně k přidání dalšího či odebrání jednoho kusu totožné položky v rámci dané objednávky.

Test je vyhodnocen jako úspěšný v případě, že se správně aktualizuje počet kusů položky a vypočítá se správná výsledná cena celé objednávky.

Test Steps

Test Step	Step Detail	Test Data	Expected results
1	Open https://alfa.alza.#domain#/muj-ucet/objednavky.htm and log in	Domains: • .cz • .sk • .hu • .at • .de • .co.uk email: password:	User logged in and orders page is opened
2	Open detail of the only active order that is displayed on the page		Order detail page is opened
3	Increase the amount of pieces of the item in the order		The amount is increased Correct price per piece is shown Correct total price is shown
4	Decrease the amount of pieces of the item in the order.		The amount is decreased Correct product price is shown Correct total price is shown
5	Log out		User logged out

Obrázek 5 - Kroky testovacího případu "Modify amount of goods in active uninvoiced order"

Check orders page in BO

Testovací případ má za úkol komplexní kontrolu stránky s přehledem objednávek. V rámci manuálního testu se stránka kontroluje především z vizuálního aspektu – zda jsou na stránce přítomny všechny podstatné prvky, vše odpovídá grafickému návrhu a jsou funkční základní ovládací prvky.

Test Steps

Test Step	Step Detail	Test Data	Expected results
1	Open https://alfa.alza.#domain#/muj-ucet/objednavky.htm and log in	Domains: .cz .sk .hu .at .de .co.uk email: password:	User logged in and orders page is opened
2	Check list of orders		Orders are displayed correctly Important elements are present: • Search field input • Orders page button • Purchased goods page button • Order list • Archive orders list • Archive orders - next page button • Archive orders - previous page button • Archive orders - page size selector Archive orders pagination works (next, previous)

Obrázek 6 - Kroky testovacího případu "Check orders page in BO"

3.1.2 Migrace testovacích případů do Azure Test Plan

Jelikož v průběhu zkoumání testovacích případů bylo zjištěno, že jejich dokumentace je zajištěna jen formou jednoduchých tabulek na Confluence, proběhla nad rámec původního plánu i jejich migrace do nástroje Azure Test Plan (dále jen ATP) od společnosti Microsoft. Firma ATP již nějakou chvíli využívá, ovšem nejsou jeho součástí zdaleka všechny testovací případy, včetně těch, které jsou považovány za kandidáty na automatizaci.

ATP umožňuje efektivnější správu testovacích případů, jejich přímou exekuci a zaznamenávání výsledků z jednotlivých iterací. Testovací případy lze shromažďovat do složek, generovat sady testů podle specifických kritérií či značek jednotlivých testů nebo umožňuje kroky, které se opakují napříč více případy, seskupit do takzvaných „sdílených kroků“.

Generování sady testů dle jejich parametrů umožňuje jednoduše vytvořit menší specifické testovací sady, které naleznou uplatnění například při testování určité konkrétní oblasti aplikace, případně pokud je třeba otestovat jen jednu doménu a podobně.

Díky sdíleným krokům pak často se opakující úkony, jako například přihlášení uživatele, není nutné popisovat pokaždé znovu. Testy sdílejí jejich jednotnou definici a v případě, že by bylo nutné tyto kroky upravit, změna se ihned promítne do všech testovacích případů a není nutné úpravu provádět všude zvlášť.

ATP nabízí rovněž dashboard s přehledem úspěšnosti běhu testů v čase. Lze tak jednoduše získat přehled například o tom, jaké testy jsou nejvíce problémové. Testovací případy lze také snáze identifikovat, jelikož v rámci ATP má každý svůj vlastní číselný identifikátor, který zůstává neměnný i v případě kdy se změní třeba celý název testovacího případu.

3.2 Použité technologie a nástroje

V této kapitole jsou představeny nástroje využité pro automatizaci a testování. Jsou uvedeny důvody jejich výběru a výhody oproti dostupným alternativám.

3.2.1 Automatizační software

Stěžejním prvkem celého projektu je samotný software, který umožňuje automaticky na základě příkazů v kódu ovládat webový prohlížeč. Pro tento účel byl zvolen nástroj Selenium WebDriver. Jde o program s dlouholetou historií a v oboru je považován za jakýsi dlouhodobý standard.

Umožňuje ovládání mnoha běžných druhů prohlížečů a jeho obrovskou výhodou je, že podporuje hned několik programovacích jazyků, které lze pro psaní testů s WebDriverem využít (JavaScript, C#, Java, Perl, PHP, Python, ...).

Mezi populární a modernější alternativy patří například Cypress, který na rozdíl od WebDriveru nevyžaduje téměř žádnou konfiguraci, příkazy vykonává přímo v prohlížeči a změny v testech dokáže aplikovat v reálném čase. Jediným podporovaným jazykem je JavaScript.

Selenium WebDriver byl zvolen zejména proto, že v rámci firmy již má zažitou historii, nabízí více možností přizpůsobení individuálním potřebám a je dobře škálovatelný. Jde o open-source nástroj, který je navíc k dispozici zdarma, s jeho implementací tedy kromě samotné práce nejsou spojeny žádné dodatečné náklady. Navíc

díky podpoře více programovacích jazyků je možné psát testy ve stejném jazyce, ve kterém je psaná samotná aplikace e-shopu, tedy C#.

3.2.2 Testovací software

Selenium WebDriver se stará pouze o ovládání prohlížeče, samotné testování zajišťuje framework NUnit. Jde o open-source verzi frameworku JUnit, který se používá pro vývoj v Javě, zatímco NUnit umožňuje testování na platformě C# .NET.

NUnit umožňuje spouštění testů příkazy přímo z konzole, případně za využití některého z nástrojů třetích stran. Spouštění přímo z konzole je užitečné zejména v procesu samotného vývoje, lze si tak jednoduše test spustit lokálně a zjistit, zda správně funguje. Mezi jeho další výhody patří paralelní běh více testů, dobrá podpora pro využití testovacích dat v rámci testů a kategorizace testů pro jejich selektivní spouštění.

3.3 Definice stránek pro Selenium Webdiver

Jednotlivé stránky, které je nutné v průběhu testu navštívit, musejí mít svou specifickou definici v rámci Selenia. Každou stránku reprezentuje v projektu jedna třída a samotná stránka se definuje pomocí elementů (prvky na stránce) a metod (činnosti, které lze na stránce vykonávat). Jako příklad elementu je možné si představit obyčejné tlačítko „Přihlásit se". Metoda pak může reprezentovat celý proces, respektive sérii kroků, které reálně vedou k přihlášení konkrétního uživatele.

Při vývoji bylo zjištěno, že základní definice kostry e-shopu a rovněž stěžejní prvky v definici několika stránek již v projektu existují. Autor se tedy zaměří na obohacení těchto stránek o definice chybějících elementů a metod, které následně bude moct využít při vývoji vlastních testů.

3.3.1 Seznam objednávek

Nejprve je nutné dobře se seznámit s reálnou podobou stránky tak, jak ji vidí standardní zákazník. Na obrázku 5 lze vidět stránku s přehledem aktivních a archivních objednávek. Aktivní objednávky jsou řazeny na vrchu a vypisují se pod sebe do nekonečna.

Ve spodní části pod aktivními objednávkami jsou objednávky archivní. To jsou ty, které již zákazník převzal, případně stornoval. Na rozdíl od těch aktivních jsou rozděleny do omezeného výpisu (se stránkováním), přičemž zákazník si jeho rozsah může nastavit

a ke starším či novějším objednávkám se jednoduše dostane kliknutím na tlačítko „Další objednávky“, případně pomocí šipek.

Mezi další klíčové prvky této stránky patří vyhledávání nebo přepínač mezi sekcemi „objednávky“ a „zakoupené zboží“

Stránku je nutné nyní definovat v Seleniu. K tomuto účelu se využije v projektu v rámci složky Pages již existující třída MyOrdersPage.cs, která navíc dědí ještě definici základní kostry webu z BasePage.cs, jelikož některé prvky jsou pro všechny stránky shodné (záhlaví, zápatí, postranní bannery, …).

Před zásahem autora již třída obsahovala základní definici tří elementů (vyhledávací pole, tlačítko pro přístup na detail objednávky a prvek obsahující seznam objednávek) a zároveň dvě jednoduché metody – pro otevření detailu objednávky podle jejího čísla a pro získání seznamu čísel aktivních objednávek.

Obrázek 7 - Stránka s přehledem objednávek

Pro účely vybraných testů toto nestačí, a tak je nutné definovat zejména všechny chybějící elementy a následně pak metody, které následně naleznou uplatnění v rámci testů. Nejprve je třeba podívat se do HTML kódu stránky a najít u jednotlivých prvků nějaké jedinečné identifikátory, které jednoznačně definují daný prvek na stránce. Vhodné je využít zejména unikátní id nebo třídy těchto elementů.

Na stránce s přehledem objednávek autor dodatečně definuje tyto elementy:

- Tlačítko směrující na stránku s objednávkami
- Tlačítko směřující na stránku se zakoupeným zbožím
- Samostatný seznam archivních objednávek
- Tlačítko pro přechod na další stránku v seznamu archivních objednávek
- Tlačítko pro přechod na předchozí stránku v seznamu archivních objednávek
- Tlačítko s rozbalovacím seznamem pro výběr velikosti stránky archivních objednávek

Technické řešení implementace těchto prvků lze vidět na obrázku 8. Pro každý z prvků je v rámci třídy založena nová proměnná, do které se pomocí integrované funkce Selenia dosazuje daný prvek dle zvoleného selektoru, zde ve všech případech podle CSS.

```
28
29  /// <summary>
30  /// Button leading to Orders section.
31  /// </summary>
    1 reference
32  public readonly Button _ordersPageButton = new(By.CssSelector(".top-navigation app-button:nth-child(1) button"));
33
34  /// <summary>
35  /// Button leading to Purchased goods section.
36  /// </summary>
    1 reference
37  public readonly Button _purchasedGoodsPageButton = new(By.CssSelector(".top-navigation app-button:nth-child(2) button"));
38
39  /// <summary>
40  /// List of archive orders.
41  /// </summary>
    1 reference
42  public readonly Simple _archiveOrderList = new(By.CssSelector(".orders"));
43
44  /// <summary>
45  /// Archive orders - previous page button.
46  /// </summary>
    3 references
47  public readonly Button _archiveOrdersPreviousPageButton = new(By.CssSelector(".mat-paginator-navigation-previous"));
48
49  /// <summary>
50  /// Archive orders - next page button.
51  /// </summary>
    3 references
52  public readonly Button _archiveOrdersNextPageButton = new(By.CssSelector(".mat-paginator-navigation-next"));
53
54  /// <summary>
55  /// Archive orders - page size selector.
56  /// </summary>
    1 reference
57  public readonly Button _archiveOrdersPageSizeSelector = new(By.CssSelector(".mat-paginator-page-size-select"));
58
```

Obrázek 8 - Definované elementy pro stránku s přehledem objednávek

Poté jsou definovány dvě nové metody. Obě slouží pro navigaci v seznamu archivních objednávek – jedna obstarává přechod na další stránku, druhá přechod na stránku předchozí. Zmíněné metody jsou ukázány na obrázku 9.

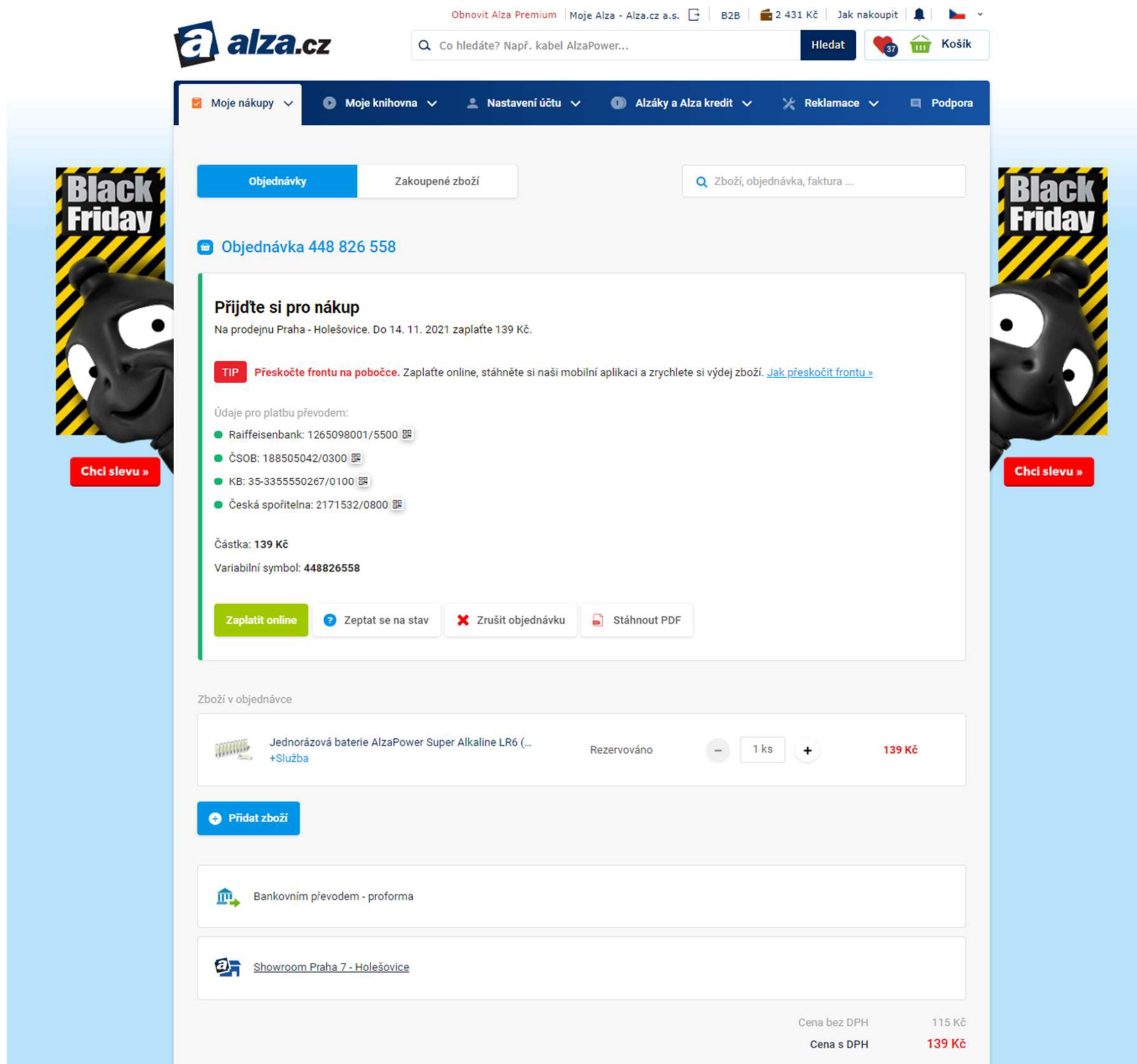
```
/// <summary>
/// Goes to next page in the list of archive orders.
/// </summary>
1 reference
public void GoToNextInArchiveOrders() {
    _archiveOrdersNextPageButton.ScrollTo();
    _archiveOrdersNextPageButton.Click();
}

/// <summary>
/// Goes to previous page in the list of archive orders.
/// </summary>
1 reference
public void GoToPreviousInArchiveOrders() {
    _archiveOrdersPreviousPageButton.ScrollTo();
    _archiveOrdersPreviousPageButton.Click();
}
```

Obrázek 9 - Definované metody pro stránku s přehledem objednávek

3.3.2 Detail objednávky

Stránka reprezentuje detail jedné konkrétní objednávky. Jde o poměrně komplexní stránku – v úvodní části se nachází číslo objednávky, její stav a níže jeho podrobnější vysvětlení, poté další informace k objednávce, jako platební údaje pro bankovní převod či jiné dodatečné informace, a dále akční tlačítka (Zaplatit online, Zeptat se na stav, Zrušit objednávku, …).

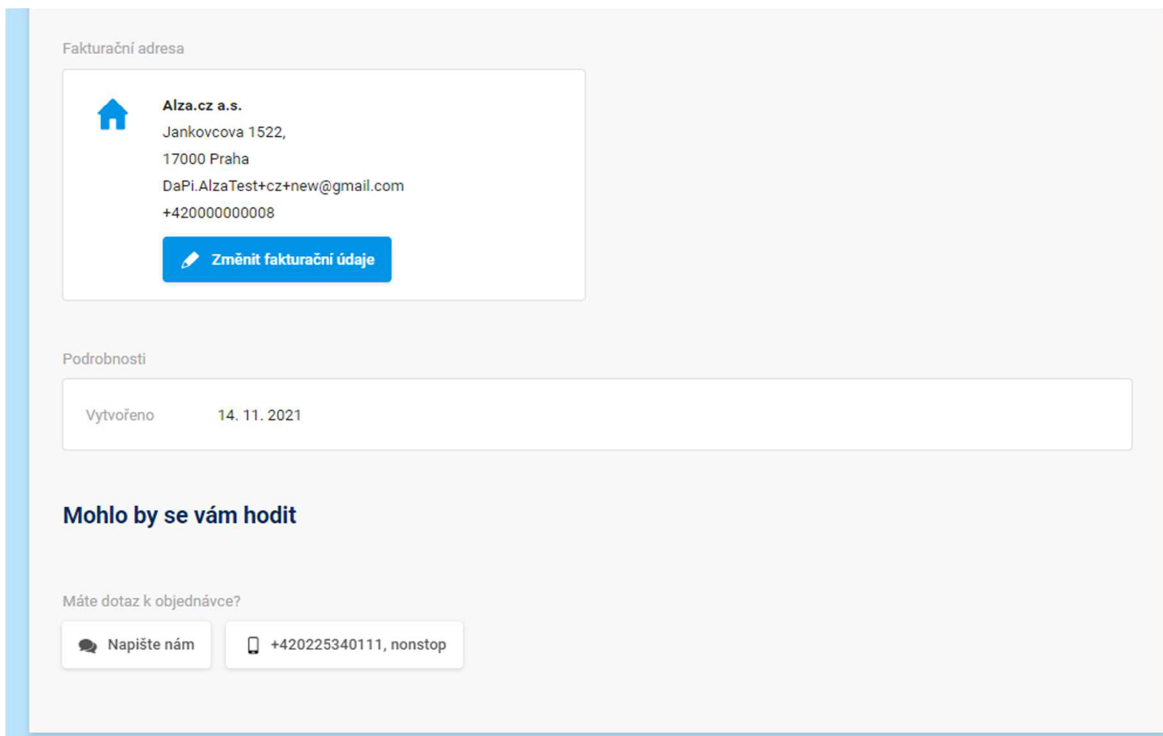


Obrázek 10 - Stránka s detailem objednávky (1)

Podoba většiny těchto prvků je úzce spjata s konkrétním stavem objednávky v čase a dle toho se prvky dynamicky obměňují (stavový text se mění, tlačítka přibývají či ubývají…).

V další sekci je výpis produktů v objednávce, přičemž u každého produktu se zobrazuje jeho obrázek, název, stav expedice, počet kusů (u některých stavů i tlačítka pro modifikaci počtu kusů) a nakonec cena produktu, v případě více kusů je zde cena celková i cena za kus.

Následuje pak přehled platební metody a zvoleného typu dopravy, celková suma částky za objednávku. Na úplném konci stránky lze najít fakturační, případně dodací adresu, informaci o datu vytvoření objednávky a tlačítka sloužící pro kontaktování zákaznického servisu.



Obrázek 11 - Stránka s detailem objednávky (2)

Stránka s detailem objednávky je definována ve třídě MyOrderDetailPage.cs, opět dědí z BasePage.cs a stejně jako výpis objednávek, i tato stránka má již v rámci Selenia určitou vlastní základní strukturu definovanou, chybí však takřka všechny elementy, se kterými je nutno interagovat v rámci vybraného testu „Modify amount of goods in active uninvoiced order".

Na stránce s detailem objednávky je nutné dodatečně definovat tyto elementy:

- Tlačítko pro navýšení počtu kusů zakoupené položky
- Tlačítko pro snížení počtu kusů zakoupené položky
- Text indikující aktuální počet kusů zakoupené položky

Opět lze na obrázku 12 vidět konkrétní technické řešení definice těchto prvků, i v tomto případě na základě CSS selektorů.

```
69   /// <summary>
70   /// Button for increasing item pieces count.
71   /// </summary>
     2 references
72   private readonly Button _increaseItemCountButton = new(By.CssSelector("span.count-plus"));
73
74   /// <summary>
75   /// Button for decreasing item pieces count.
76   /// </summary>
     2 references
77   private readonly Button _decreaseItemCountButton = new(By.CssSelector("span.count-minus"));
78
79   /// <summary>
80   /// Text label with item pieces count.
81   /// </summary>
     2 references
82   private readonly Simple _itemCount = new(By.CssSelector(".item .count .count-input .count-edit .count-text"));
```

Obrázek 12 - Definované elementy pro stránku s detailem objednávky

Zároveň v rámci třídy MyOrderDetailPage je možné rovnou definovat metody, které bude test využívat při svém běhu. Jelikož test potřebuje ověřovat počet kusů položky v objednávce, musí vzniknout metoda pro získání tohoto údaje. Sice již došlo k definici elementu s textem počtu kusů, nicméně text je ve formátu „1 ks“. Je tedy nutné z tohoto řetězce dostat ještě osamocenou číslovku.

Je tedy definována veřejná metoda návratového typu int s názvem GetItemPiecesCount. Ta nejprve vyčká na samotné zobrazení elementu, se kterým pracuje – k tomu se využívá funkce WaitForDisplayed. Jakmile je detekována přítomnost očekávaného prvku na stránce, získá se aktuální textová hodnotu tohoto elementu a za použití funkce ParseNumber dojde k navrácení osamocené číselné hodnoty.

V dalších dvou metodách autor definuje samotný mechanismus modifikace počtu kusů položky. Jedna metoda slouží pro přidávání kusů, druhá pro ubírání kusů. Jejich povaha je až na tyto odlišnosti prakticky stejná.

Veřejné metody IncreaseItemPiecesCount i DecreaseItemPiecesCount jsou typu void, to znamená že jen vykonají definovanou činnost a nevrací žádnou hodnotu. Ve svém úvodu si do proměnné initialPiecesCount (datový typ int) uloží původní počet kusů. K tomu využívají již výše definovanou metodu GetItemPiecesCount.

Následně jsou do prohlížeče vyslány instrukce, aby se doscrollovalo k tlačítku pro zvýšení/snížení počtu kusů a následně proběhne kliknutí na toto tlačítko.

```
193         /// <summary>
194         /// Returns count of item pieces in order.
195         /// </summary>
196         /// <returns>Item pieces count as int</returns>
            4 references
197         public int GetItemPiecesCount()
198         {
199             _itemCount.WaitForDisplayed();
200             return Utils.ParseNumber(_itemCount.GetText());
201         }
202
203         /// <summary>
204         /// Increases the quantity of goods in order.
205         /// </summary>
            0 references
206         public void IncreaseItemPiecesCount()
207         {
208             int initialPiecesCount = GetItemPiecesCount();
209
210             _increaseItemCountButton.ScrollTo();
211             //_decreaseItemCountDisabledButton.WaitForDisplayed();
212             _increaseItemCountButton.Click();
213
214             WaitWithRefresh(() => (GetItemPiecesCount() == (initialPiecesCount+1)));
215         }
216
217         /// <summary>
218         /// Decreases the quantity of goods in order.
219         /// </summary>
            0 references
220         public void DecreaseItemPiecesCount()
221         {
222             int initialPiecesCount = GetItemPiecesCount();
223
224             _decreaseItemCountButton.ScrollTo();
225             //_decreaseItemCountDisabledButton.WaitForNotDisplayed();
226             _decreaseItemCountButton.Click();
227
228             WaitWithRefresh(() => (GetItemPiecesCount() == (initialPiecesCount-1)));
229         }
```

Obrázek 13 - Definované metody pro stránku s detailem objednávky

V závěru obou metod je ještě definované vyčkání na podmínku, že aktuální počet kusů, tedy po již provedené modifikaci, odpovídá původnímu počtu kusů poníženého či povýšeného o jedna. Tím je již předem zaručeno, že metoda úspěšně neskončí do doby, než reálný stav odpovídá očekávanému výsledku v rámci testu.

3.4 Automatizace testovacích případů

Po definici zejména všech potřebných prvků je možné přejít k samotnému procesu automatizace manuálních testovacích případů. Jelikož v rámci firmy jsou pro automatizované testy nastavena určitá specifická pravidla vývoje, je nutné se oproti původnímu manuálnímu testovacímu případu v několika krocích lehce odchýlit, respektive vyřešit určité záležitosti lehce odlišným způsobem, nicméně test jako takový stále kontroluje totožnou funkcionalitu, což je podstatné.

3.4.1 Modify amount of goods in active uninvoiced order

Jak již bylo popsáno dříve, cílem testu je kontrolovat funkčnost modifikace počtu kusů zakoupené položky v objednávce.

Test se v rámci Selenia definuje opět do vlastní třídy. Pro účely testů tvořených v rámci této práce vznikla nová složka „Orders“ v rámci složky „Tests“, krerá zastřešuje veškeré testy. Pro přehlednost a lepší orientaci v celém projektu odpovídá název třídy názvu testovacího případu.

Struktura třídy je poměrně jednoduchá. Obsahuje jen jednu metodu, která reprezentuje samotný test a její název zahrnuje i číslo testovacího případu, které se získá právě z ATP. Aby framework NUnit poznal, že daná metoda zahrnuje test, je nutné nad název metody uvést označení [Test]. Obdobným způsobem za pomoci hranatých závorek je rovněž metoda, respektive celý test, označena názvy kategorií, do kterých patří. Umožní to následně spouštět selektivně testy jen v rámci určité kategorie. Rovněž je takto u testu uveden odkaz na samotný testovací případ v ATP.

Důležité je také předání testovacích dat. V případě tohoto testu se předává informace o zemi, respektive doméně, na které test probíhá, a následně data o nákupu. V rámci manuálního testu se využívá dedikovaný zákaznický účet s předpřipravenou objednávkou, především pro urychlení práce, nicméně v automatizovaných testech je definován pro každou doménu jen jeden testovací uživatelský účet, kde takto manuálně definovat objednávku by nebylo dlouhodobě udržitelné, jelikož jiný test by teoreticky mohl tuto objednávku ovlivňovat. Z tohoto důvodu jsou kroky testovacího případu v rámci automatizace lehce upraveny, a test si nejprve sám vytvoří pro další potřeby vždy novou objednávku.

Na obrázku 14 lze vidět definici testovacích dat ve třídě TestData. V rámci definovaného enumerátoru jsou testu předávány informace o doméně, a následně data související s nákupem. Zde se uvádí kupovaná položka, typ dopravy, platby a zákazník.

```
721         /// <summary>
722         /// Returns test data for modifying amount of goods in active uninvoiced order.
723         /// </summary>
724         /// <returns>TestCaseData</returns>
            0 references
725         public static IEnumerable BackOffice_ActiveUninvoicedOrder_RegisteredUser()
726         {
727             yield return new TestCaseData(Country.CZ, new Purchase
728             {
729                 Product = ProductList.CoaxialCable[Country.CZ],
730                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.CZ],
731                 PaymentMethod = PaymentMethodList.BankTransfer[Country.CZ],
732                 User = UserList.GetRegisteredUser(Country.CZ)
733             });
734             yield return new TestCaseData(Country.SK, new Purchase
735             {
736                 Product = ProductList.CoaxialCable[Country.SK],
737                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.SK],
738                 PaymentMethod = PaymentMethodList.BankTransfer[Country.SK],
739                 User = UserList.GetRegisteredUser(Country.SK)
740             });
741             yield return new TestCaseData(Country.HU, new Purchase
742             {
743                 Product = ProductList.CoaxialCable[Country.HU],
744                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.HU],
745                 PaymentMethod = PaymentMethodList.BankTransfer[Country.HU],
746                 User = UserList.GetRegisteredUser(Country.HU)
747             });
748             yield return new TestCaseData(Country.AT, new Purchase
749             {
750                 Product = ProductList.CoaxialCable[Country.AT],
751                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.AT],
752                 PaymentMethod = PaymentMethodList.BankTransfer[Country.AT],
753                 User = UserList.GetRegisteredUser(Country.AT)
754             });
755             yield return new TestCaseData(Country.DE, new Purchase
756             {
757                 Product = ProductList.CoaxialCable[Country.DE],
758                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.DE],
759                 PaymentMethod = PaymentMethodList.BankTransfer[Country.DE],
760                 User = UserList.GetRegisteredUser(Country.DE)
761             });
762             yield return new TestCaseData(Country.FR, new Purchase
763             {
764                 Product = ProductList.CoaxialCable[Country.FR],
765                 DeliveryMethod = DeliveryMethodList.PersonalPickup[Country.FR],
766                 PaymentMethod = PaymentMethodList.BankTransfer[Country.FR],
767                 User = UserList.GetRegisteredUser(Country.FR)
768             });
769         }
```

Obrázek 14 - Testovací data pro test s modifikací počtu kusů položky v objednávce

Nyní lze již přistoupit k samotnému testu. Na řádcích 24–34 na obrázku číslo 15 lze vidět již zmíněné vytvoření objednávky. Pracuje se zde s proměnnou purchase, která se předává z testovacích dat, a pro průchod košíkem test využívá již definovaných

stránek a metod. Od skončení tvorby objednávky jsou již kroky téměř shodné s manuálním testem.

```
10  namespace Alza.UI.Web.Full.Tests
11  {
12      [TestFixture]
        0 references
13      public class ModifyAmountOfGoodsInActiveUninvoicedOrder : PurchaseBaseTest
14      {
15          [Test]
16          [Category("FULLWEB")]
17          [Category("POPR")]
18          [Category("ORDERS")]
19          [Category("CZ"), Category("SK"), Category("HU"), Category("AT"), Category("DE"), Category("FR")]
20          [Property("ManualTestCaseLink", "https://dev.azure.com/betaalza/Alza.QA/_workitems/edit/2521")]
21          [TestCaseSource(typeof(TestData), nameof(TestData.BackOffice_ActiveUninvoicedOrder_RegisteredUser))]
            0 references
22          public void AT_2521_ModifyAmountOfGoodsInActiveUninvoicedOrder(Country country, Purchase purchase)
23          {
24              // Create purchase
25              purchase = Utils.CloneObject<Purchase>(purchase);
26
27              DetailPage detailPage = DetailPage.Login(country, purchase);
28
29              Assert.That(detailPage.IsUserLoggedIn, Is.True);
30              Assert.That(WebDriver.Url, Does.Contain(purchase.Product.Url));
31
32              CheckProductDetails(detailPage, purchase);
33
34              CrossSellPage crossSellPage = detailPage.Buy(purchase);
35              CheckCrossSellPage(crossSellPage, purchase);
36
37              MyOrderDetailPage orderDetailPage = Checkout(crossSellPage, purchase, false);
38              CheckOrderDetails(orderDetailPage, purchase);
39
40              // Increase item count
41              orderDetailPage.IncreaseItemPiecesCount();
42
43              // Update purchase values
44              purchase.Product.Quantity = (purchase.Product.Quantity+1);
45              purchase.TotalPrice = (purchase.TotalPrice+purchase.Product.Price);
46              purchase.Product.Price = (purchase.Product.Price*2);
47
48              CheckOrderDetails(orderDetailPage, purchase);
49              Assert.That(orderDetailPage.GetItemPiecesCount() == purchase.Product.Quantity);
50
51              // Decrease item count
52              orderDetailPage.DecreaseItemPiecesCount();
53
54              // Update purchase values
55              purchase.Product.Quantity = (purchase.Product.Quantity-1);
56              purchase.Product.Price = (purchase.Product.Price/2);
57              purchase.TotalPrice = (purchase.TotalPrice-purchase.Product.Price);
58
59              CheckOrderDetails(orderDetailPage, purchase);
60              Assert.That(orderDetailPage.GetItemPiecesCount() == purchase.Product.Quantity);
61
62              // Cancel order and logout
63              orderDetailPage.CancelOrder();
64              orderDetailPage.Logout();
65          }
66      }
67  }
```

Obrázek 15 - Třída s testem pro modifikaci počtu kusů položky v objednávce

Kupuje se jeden kus vybrané položky, je tedy nejprve nutné provést navýšení počtu kusů. Zde již autor využívá vlastní definovanou metodu, která se o přidání dalšího kusu postará. Poté, co je dokončeno přidání dalšího kusu, manuálně se pomocí jednoduchých výpočtů aktualizují údaje v proměnné purchase, aby hodnoty odpovídaly očekávanému reálnému stavu v objednávce.

Ačkoliv již metoda pro navýšení a vlastně i na snížení počtu kusů obsahuje mechanismus, který by měl zaručit, že proces neskončí úspěšně pokud modifikace počtu kusů neproběhne dle očekávání, i tak je dobré ještě v rámci testu použít takzvaný assert, což je to samotné testované kritérium. V assertu je definováno určité očekávání, a pokud toto očekávání není splněno, test je vyhodnocen jako neúspěšný. V tomto případě se očekává, že aktuální počet kusů položky v objednávce odpovídá navýšenému počtu kusů, který byl dříve vypočítán.

V případě, že by z nějakého důvodu toto očekávání nebylo naplněno, test v této fázi skončí jako neúspěšný. V opačném případě pokračuje dál, nyní již pokusem o snížení počtu kusů. Postup je obdobný jako výše, opět se využívá autorem definovaná metoda, aktualizují se opět pomocí výpočtů údaje v purchase, a následně je zde další assert, který kontroluje definované očekávání, že počet kusů nyní odpovídá vypočtenému sníženému počtu.

V případě, že toto všechno proběhne úspěšně, test po sobě na konci stornuje objednávku, kterou si z počátku vytvořil, a tím celý test končí. Pokud v průběhu nedojde k nějakému timeoutu nebo negativnímu vyhodnocení některého z assertů, test je vyhodnocen jako úspěšný.

3.4.2 Check orders page in BO

Stejně jako předchozí test, i tento je definován do vlastní třídy v rámci složky Orders. Většina štítků a rozřazení do kategorií je stejné, jako u testu pro modifikaci počtu kusů položky, liší se až zdrojová testovací data. Pro potřeby tohoto testu je nutné vědět opět doménu, poté uživatelský účet, a nakonec url stránky s objednávkami, aby nebylo nutné vypisovat ji pokaždé ručně.

```
770
771         /// <summary>
772         /// Returns test data for orders page.
773         /// </summary>
774         /// <returns>TestCaseData</returns>
            1 reference
775         public static IEnumerable OrdersPage()
776         {
777             yield return new TestCaseData(Country.CZ, UserList.GetRegisteredUser(Country.CZ), Sitemap.BackOffice.Orders[Country.CZ]);
778             yield return new TestCaseData(Country.SK, UserList.GetRegisteredUser(Country.SK), Sitemap.BackOffice.Orders[Country.SK]);
779             yield return new TestCaseData(Country.HU, UserList.GetRegisteredUser(Country.HU), Sitemap.BackOffice.Orders[Country.HU]);
780             yield return new TestCaseData(Country.AT, UserList.GetRegisteredUser(Country.AT), Sitemap.BackOffice.Orders[Country.AT]);
781             yield return new TestCaseData(Country.DE, UserList.GetRegisteredUser(Country.DE), Sitemap.BackOffice.Orders[Country.DE]);
782             yield return new TestCaseData(Country.UK, UserList.GetRegisteredUser(Country.FR), Sitemap.BackOffice.Orders[Country.UK]);
783         }
784
```

Obrázek 16 - Testovací data pro test kontrolující stránku s přehledem objednávek

Metoda s testem začíná definicí proměnné, která reprezentuje stránku s objednávkami, a následně je provedeno otevření této stránky v prohlížeči. Hned poté se

ověřuje, zda url v prohlížeči skutečně odpovídá očekávané url stránky s objednávkami. Následně dojde k přihlášení zákazníka a potom je zde opět assert ověřující, zda skutečně došlo k přihlášení zákazníka.

Poté se na stránce kontroluje přítomnost téměř všech prvků, které jsou pro její funkčnost zásadní. Mezi tyto prvky patří jak ty, které již byly na stránce definovány (vyhledávací pole, seznam objednávek), tak i všechny ty, které nově definoval autor práce.

```
10  namespace Alza.UI.Web.Full.Tests
11  {
12      [TestFixture]
        0 references
13      public class CheckOrdersPageInBO : BaseTest
14      {
15          [Test]
16          [Category("FULLWEB")]
17          [Category("POPR")]
18          [Category("ORDERS")]
19          [Category("CZ"), Category("SK"), Category("HU"), Category("AT"), Category("DE"), Category("FR")]
20          [Property("ManualTestCaseLink", "https://dev.azure.com/betaalza/Alza.QA/_workitems/edit/2517")]
21          [TestCaseSource(typeof(TestData), nameof(TestData.OrdersPage))]
            0 references
22          public void AT_2517_CheckOrdersPageInBO(Country country, User user, string ordersURL)
23          {
24              // Open orders page
25              MyOrdersPage myOrdersPage = new(country, ordersURL);
26              myOrdersPage.Open();
27
28              // Assert that browser url is the same as expected orders url
29              Assert.That(WebDriver.UrlPathAndQuery(), Does.Contain(ordersURL));
30
31              // Login user
32              myOrdersPage.Login(user);
33
34              // Assert that user is actually logged in
35              Assert.That(myOrdersPage.IsUserLoggedIn, Is.True);
36
37              // Assert that all important elements are present on the page
38              Assert.That(myOrdersPage._searchInput.IsDisplayed(), Is.True);
39              Assert.That(myOrdersPage._orderList.IsDisplayed(), Is.True);
40              Assert.That(myOrdersPage._ordersPageButton.IsDisplayed(), Is.True);
41              Assert.That(myOrdersPage._purchasedGoodsPageButton.IsDisplayed(), Is.True);
42              Assert.That(myOrdersPage._archiveOrderList.IsDisplayed(), Is.True);
43              Assert.That(myOrdersPage._archiveOrdersPreviousPageButton.IsDisplayed(), Is.True);
44              Assert.That(myOrdersPage._archiveOrdersNextPageButton.IsDisplayed(), Is.True);
45              Assert.That(myOrdersPage._archiveOrdersPageSizeSelector.IsDisplayed(), Is.True);
46
47              // Go to next page in archive list
48              myOrdersPage.GoToNextInArchiveOrders();
49
50              // Assert that browser url is the same as expected archive page url
51              Assert.That(WebDriver.UrlPathAndQuery(), Does.Contain(ordersURL+"?p=2"));
52
53              // Go to previous page in archive list
54              myOrdersPage.GoToPreviousInArchiveOrders();
55
56              // Assert that browser url is the same as expected archive page url
57              Assert.That(WebDriver.UrlPathAndQuery(), Does.Contain(ordersURL+"?p=1"));
58
59              // Logout user
60              myOrdersPage.Logout();
61
62          }
63      }
64  }
```

Obrázek 17 – Třída s testem pro kontrolu stránky s přehledem objednávek

Jelikož manuální tester toto všechno ověřuje opticky oproti obrázku, má šanci si všimnout i případných grafických nesrovnalostí. V rámci automatického testu je třeba se

spokojit s tím, že definované elementy jsou na stránce zobrazené, nelze již vyřešit ověření jejich správného vzhledu.

Po kontrole elementů se ověřuje funkčnost stránkování v archivních objednávkách. Nejprve se testuje přechod na další stránku, poté přechod zpět na stránku předchozí. V obou případech se využívá nově definovaných metod pro tuto stránku. Po každém přechodu v archivu se assertem ověřuje, zda url v prohlížeči odpovídá očekávané url dané archivní stránky. Využívá se zde toho, že jednotlivé stránky jsou pomocí specifických url parametrů přístupné i z odkazu, a i při manuálním procházení stránek se tyto parametry v url dynamicky mění.

Pokud v průběhu není některý z assertů vyhodnocen negativně, končí tímto úspěšně celý test.

3.5 Časová analýza automatického a manuálního testování

Po dokončení všech nezbytných prací na automatizovaných testech bylo provedeno několik kontrolních běhů obou testů, aby se ověřila jejich funkčnost. Zároveň byl v rámci třech běhů zaznamenán čas trvání.

V tabulkách níže lze vidět časové údaje o běhu obou testů vyvinutých v rámci této práce, přičemž čas se týká vždy běhu pouze na jedné doméně, konkrétně na Alza.cz. Časové údaje jsou zaokrouhleny na celé sekundy.

Modify amount of goods in active uninvoiced order – automatický test			
Číslo běhu	1	2	3
Čas běhu	53 sekund	49 sekund	51 sekund
Průměrný čas	51 sekund		

Tabulka 2 - Modify amount of goods in active uninvoiced order – čas automatického testu

Check orders page in BO – automatický test			
Číslo běhu	1	2	3
Čas běhu	17 sekund	18 sekund	17 sekund
Průměrný čas	17 sekund		

Tabulka 3 - Check orders page in BO – čas automatického testu

Následně autor práce společně s dalšími dvěma testery prošli ty samé testy manuálně a měřili při tom čas, který testováním stráví. Je nutné poznamenat, že do času manuálního

testu se počítá celá doba strávená prací, která s vykonáváním testu souvisí, tedy od otevření kroků testovacího případu až do zaznamenání konečného výsledku do k tomu určené tabulky.

Výsledky z měření manuálního testování lze opět vidět v následujících tabulkách. Na první pohled je přitom patrné, že časy jsou zde znatelně vyšší než u testů automatizovaných.

Modify amount of goods in active uninvoiced order – manuálně			
Tester	Tester #1	Tester #2	Tester #3
Čas testu	1 minuta 55 sekund	1 minuta 37 sekund	1 minuta 32 sekund
Průměrný čas	1 minuta 41 sekund		

Tabulka 4 - Modify amount of goods in active uninvoiced order – čas manuálního testu

Check orders page in BO – manuálně			
Tester	Tester #1	Tester #2	Tester #3
Čas testu	1 minuta 8 sekund	59 sekund	1 minuta 12 sekund
Průměrný čas	1 minuta 6 sekund		

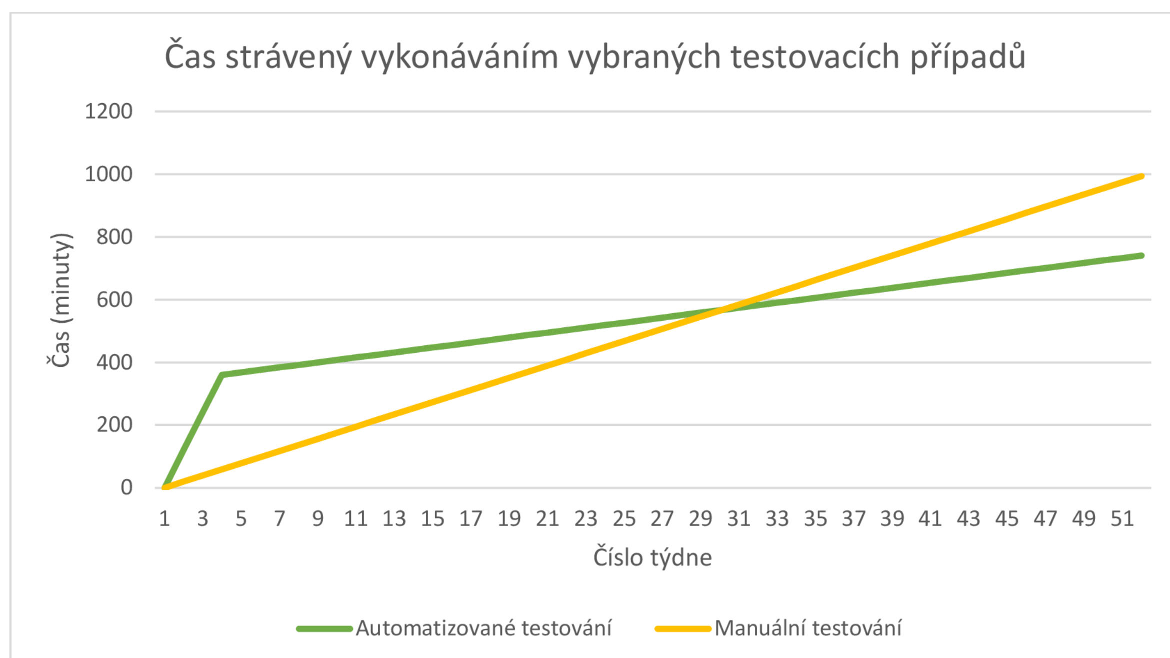
Tabulka 5 - Check orders page in BO – čas manuálního testu

Pokud by se bral v potaz pouze čas strávený samotným výkonem daného testovacího případu, lze tvrdit, že automatizací se v průměru ušetří u prvního z testů 49,5 % času, u druhého testu dokonce 74,2 % času. Dohromady u těchto dvou testů by tedy celková průměrná úspora byla 59,3 % času. Na dvou testech vyvinutých v této práci tedy tester konkrétně ušetří v průměru 1 minutu a 39 sekund.

Nicméně zemí, ve kterých Alza oficiálně působí a má pro ně dedikovanou doménu, je aktuálně 7. Oba nové testy jsou kompatibilní se všemi doménami, jak je patrné z definice jejich testovacích dat. Dohromady na všech doménách tedy automatické testování zabere týdně 7,93 minut, zatímco manuální 19,48 minut. Celkem je tedy při jedné regresi ušetřeno v rámci nových testů 11,55 minut oproti manuálnímu testování. Ročně automatizované testování zabere 412,36 minut, oproti tomu manuální 1012,96 minut, bez započítání vývojového času se tedy automatizací ročně ušetří 600,6 minut, tedy 10 hodin.

4 Výsledky a diskuse

Nejprve bude zhodnoceno srovnání manuálního a automatizovaného testování na základě dat z časové analýzy. Na následujícím grafu lze vidět dlouhodobý vývoj průměrného času stráveného jak nad manuálním, tak nad automatickým testováním obou v práci řešených testovacích případů.



Tabulka 6 - graf ročního vývoje času stráveného nad manuálním a automatizovaným testováním

Z grafu je dobře patrné, že automatizace si vyžádala velkou počáteční časovou investici. V prvních třech týdnech bylo nad vývojem testů a dalších záležitostí stráveno dohromady 6 hodin (360 minut). Poté již hodnota každý další týden roste jen o čas, který zabere samotný běh těchto testů. U manuálního testování čas lineárně roste od počátku.

Je možné si všimnout, že zhruba ve třicátém týdnu se křivky manuálního a automatizovaného testování protínají. Znamená to, že počínaje třicátým týdnem se již reálně začíná více vyplácet automatizované testování, vrací se počáteční časová investice do vývoje a na konci sledovaného období, tedy po roce od zahájení vývoje, je zde úspora (po zaokrouhlení) 4 hodiny a 26 minut.

Na základě zjištěných poznatků a naměřených dat je možné dopočítat, jak moc se automatizace vyplatí ve větším měřítku. Oba vyvinuté testy jsou, jak již bylo zmíněno dříve, součástí regresní sady, kterou je třeba otestovat každý týden před vydáním nové verze webové aplikace. Regresní sada obsahuje přibližně 180 testovacích případů a průměrně u testování na jedné doméně stráví manuální tester čtyři hodiny.

Pokud by se podařilo zautomatizovat i pouhou polovinu všech testovacích případů z regresní sady, znamená to v rámci jedné domény časovou úsporu 1 hodinu 11 minut a 9 sekund, za předpokladu že počítáme se zde vypočtenou průměrnou úsporou 59,3 % z celkového času.

Pokud by polovina regresní sady byla zautomatizovaná a spouštěna na všech doménách, je zde dohromady úspora 8 hodin 18 minut a 8 sekund. Osm hodin je přitom standardní pracovní doba pro jednoho člověka. Znamená to tedy, že zautomatizováním poloviny regresní sady se dá ušetřit jeden celý člověkoden strávený nad manuálním testováním.

Stále je přitom řeč pouze o jedné iteraci tohoto testování. Pokud by mělo být přihlédnuto k dlouhodobému horizontu, například jednomu roku stejně jako v předchozím grafu, byla by zde časová úspora 431,73 hodin. To je téměř 18 dnů, respektive necelých 54 člověkodnů. Všechen tento uspořený čas by tedy bylo možné využít pro kreativnější práci či úkoly s vyšší prioritou.

Na základě těchto úvah a výpočtů lze tedy konstatovat, že navzdory vysoké časové náročnosti vývoje automatizovaných testů se dlouhodobě jejich využití místo manuálních testů vyplatí, přičemž z těchto poznatků mohou těžit i jiné firmy při zvažování zavedení automatizace do svého testovacího procesu.

Kromě časové úspory dosažené implementací nových testů rovněž práce přispěla k dalšímu rozšíření prostředí pro automatizaci a zefektivnění budoucích prací. I dílčí prvky v kódu jako metody a zejména elementy definované pro potřeby nových testů najdou široké uplatnění v mnoha dalších testovacích případech, jakmile bude přistoupeno k jejich zautomatizování. Nad rámec původního zadání navíc došlo u dotčených testovacích případů k zajištění jejich efektivnější evidence pomocí nástroje Azure Test Plan, kde v případě selhání automatizovaného testu může manuální tester snáze najít postup k reprodukci daných kroků.

Celkově se pak automatizací dlouhodobě a pravidelně vykonávaných činností navíc eliminuje riziko lidského faktoru. Jakmile si manuální tester zapamatuje kroky z testovacího případu, již nemusí nutně nahlížet do dokumentace, což může být problémové v případě, že se postup testovacího případu nějakým způsobem změní. Zároveň se může stát, že tester přehlédne nějakou chybu, nevšimne si chybějícího prvku, zkrátka vinou lidské chyby dojde ke špatnému vyhodnocení testu. To je něco, co se v rámci automatizovaného testu stát nemůže, jelikož zde je zaručeno, že tak, jak byl test

z počátku napsán, je vykonáván pokaždé, a to ve stejném rozsahu i kvalitě. Automatizované testování je tedy více konzistentní.

5 Závěr

Cílem práce bylo zajistit rozvoj prostředí pro automatizované testování ve vybrané firmě. V teoretické části práce bylo vysvětleno obecně testování, včetně jeho dokumentace, typů defektů, druhů testů a způsobů testování. Teoretická část tedy představila do detailu většinu podstatných aspektů testování softwaru včetně záležitostí souvisejících s automatizací a byla tak pro praktickou část práce užitečným východiskem.

Praktická část se poté zabývala prací, která postupně vedla k naplnění hlavního cíle, tedy celkovému rozvoji automatizace ve vybrané firmě. Podařilo se téměř hned v úvodu udělat něco nad rámec zadání a byla vylepšena správa testovacích případů. Následně v rámci Selenium Webdriveru došlo k obohacení dvou stránek o definice nových elementů a metod, byla definována testovací data a následně vyvinuty dva nové automatizované testy, které firmě pomohou dlouhodobě ušetřit čas, který by byl jinak stráven u manuálního testování.

Nakonec byla zhodnocena časová efektivita automatizovaného testování vůči manuálnímu, včetně odhadu dlouhodobého vývoje. Na základě analýzy naměřených a vypočtených dat bylo možné vyvodit závěr, že automatizované testování se v dlouhodobém horizontu vyplatí oproti manuálnímu testování. Konkrétně u testů vyvinutých v rámci této práce se jen v prvním roce ušetří 4 hodiny a 26 minut. Prokazatelně pak i další provedené činnosti přispěly nad rámec potřeb této práce k rozvoji automatizačního prostředí ve vybrané firmě.

6 Bibliografie

[1] KITNER, Radek. O čem je testování software? (pro znalé). *Radek Kitner - konzultant, lektor testování softwaru* [online]. Modřice, 2017 [cit. 2021-02-25]. Dostupné z: https://kitner.cz/testovani_softwaru/co-je-testovani-software/

[2] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. 1. vydání. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

[3] BOROVCOVÁ, Anna. *Testování webových aplikací*. Praha, 2008. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra teoretické informatiky a matematické logiky.

[4] Testovací dokumentace - plán, scénář, případ. *Testování software* [online]. [cit. 2021-02-25]. Dostupné z: http://test.swtestovani.cz/index.php?view=article&id=15

[5] Slovník pojmů ISTQB CTFL. In: *Czech and Slovak Testing Board* [online]. Praha, 2020 [cit. 2021-02-25]. Dostupné z: https://castb.org/wp-content/uploads/2020/05/ISTQB-Slovnik-CTFL-v20200521.pdf

[6] HLAVA, Tomáš. Test Script – testovací scénář. *Testování softwaru* [online]. Praha, 2011 [cit. 2021-02-25]. Dostupné z: http://testovanisoftwaru.cz/dokumentace-v-testovani/test-script-testovaci-scenar/

[7] Software Testing Life Cycle (STLC). *GeeksforGeeks* [online]. Noida, 2019 [cit. 2021-02-25]. Dostupné z: https://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/

[8] Sequential Software Testing Life Cycle. In: *Wikimedia Commons* [online]. San Francisco [cit. 2021-11-20]. Dostupné z: https://commons.wikimedia.org/wiki/File:Software_Testing_Life_Cycle.jpg

[9] HLAVA, Tomáš. Fáze a úrovně provádění testů. *Testování softwaru* [online]. Praha, 2011 [cit. 2021-02-25]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/

[10] KITNER, Radek. Typy testování software (třídění testů). *Radek Kitner - konzultant, lektor testování softwaru* [online]. Modřice, 2017 [cit. 2021-02-25]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

[11] Automation Testing Tutorial: Getting Started. *BrowserStack* [online]. Mumbai [cit. 2021-11-20]. Dostupné z: https://www.browserstack.com/guide/automation-testing-tutorial

[12] What is a Test Automation Tool? Definition and How to Choose One for You. *Automated Functional Testing - Software Testing Tool - Testim.io* [online]. Tel Aviv [cit. 2021-11-21]. Dostupné z: https://www.testim.io/blog/what-is-a-test-automation-tool/

[13] Top 21 Best Automation Testing Tools In 2021. *Software Test Tips* [online]. [cit. 2021-11-21]. Dostupné z: https://www.softwaretesttips.com/best-automation-testing-tools/

[14] *A Comparative Study of Automated Software Testing Tools*. St. Cloud, Minnesota, 2016. Dostupné také z: https://repository.stcloudstate.edu/csit_etds/12/. Starred Paper. St. Cloud State University.

[15] Software Testing Roles and Responsibilities. *TEST INSTITUTE* [online]. Wollerau [cit. 2021-02-25]. Dostupné z: https://www.test-institute.org/Software_Testing_Roles_And_Responsibilities.php