

Automatizované testování webové aplikace Worklio

Bc. Jan Bartoník



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jan Bartoník**
Osobní číslo: **A13437**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Počítačové a komunikační systémy**
Forma studia: **prezenční**

Téma práce: **Automatizované testování webové aplikace Worklio**
Téma anglicky: **The Automated Testing of the Worklio Web Application**

Zásady pro vypracování:

1. Popište principy testování softwaru a jeho rozdělení.
2. Proveďte rešerši možností automatizovaného testování softwaru.
3. Popište nástroje rodiny Selenium, zaměřte se na Selenium WebDriver.
4. Představte aplikaci Worklio a popište způsob testování tohoto softwaru.
5. Navrhněte a proveďte automatizaci vybraných testovacích scénářů.
6. Zhodnoťte provedenou automatizaci a její další vývoj.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
2. ISTQB EXAM CERTIFICATION [online]. Dostupné také z: <http://istqbexamcertification.com/>
3. PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.
4. ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
5. ZHAN, Zhimin. Selenium WebDriver Recipes in C#. Second Edition. New York: Apress, 2015. ISBN 978-1-4842-1741-2.

Vedoucí diplomové práce:

Ing. Petr Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

3. února 2017

Termín odevzdání diplomové práce:

17. května 2017

Ve Zlíně dne 3. února 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan



Ing. Miroslav Matýsek, Ph.D.
ředitel ústavu


Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 14. 5. 2017


.....
podpis diplomanta

ABSTRAKT

Diplomová práce se zabývá automatickým testováním webové aplikace Worklio určené pro správu zaměstnanců, jejich mzdy a pojištění. Hlavním cílem práce je návrh a vytvoření sady automatických testovacích případů, která zvýší efektivitu testování výše zmíněné aplikace. Automatizace je provedena pomocí nástrojů Selenium WebDriver a jednotkového frameworku NUnit. Jako programovací jazyk je použit C#. Testovací případy je možné spouštět automaticky pomocí webové služby Visual Studio Team Services.

Klíčová slova: Automatické testování softwaru, NUnit, Selenium WebDriver, C#, Visual Studio Team Services

ABSTRACT

The master's thesis deals with the automated testing of the Worklio web application. The application is designed to manage employees, their payroll, and their benefits. The main goal of the thesis is to design and create a set of automated tests, which will increase the efficiency of the testing of the Worklio application. The automation is carried out with Selenium WebDriver and unit framework NUnit. The programming language used is C#. The test cases can be executed automatically via Visual Studio Team Services.

Keywords: Software Automation Testing, NUnit, Selenium WebDriver, C#, Visual Studio Team Services

Na tomto místě bych rád poděkoval panu Ing. Petru Šilhavému, Ph.D., vedoucímu mé diplomové práce, za odbornou pomoc a čas, který si na mě vyhradil. Dále pak společnosti Worklio, která umožnila vypracování této práce a v neposlední řadě své rodině a přítelkyni za podporu a trpělivost.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST.....	12
1 TESTOVÁNÍ SOFTWARE	13
1.1.1 Znamé historické chyby v testování.....	14
1.1.2 Disney, Lví král.....	14
1.1.3 Chyba dělení s pohyblivou čárkou.....	14
1.1.4 Nosná raketa Ariane 5 z roku 1996.....	14
1.2 ZÁKLADNÍ POJMY	14
1.3 ZÁKLADNÍ AXIOMY TESTOVÁNÍ SOFTWARE	15
1.3.1 Žádný netriviální software nemůže být kompletně otestován.....	15
1.3.2 Testování je věcí odhadu rizika.....	15
1.3.3 Testování může prokázat jen přítomnost defektů, nikoli jejich absenci	16
1.3.4 Čím více defektů v softwaru je nalezeno, tím více jich v produktu je.....	16
1.4 PŘÍČINY DEFECTŮ.....	16
1.5 NÁKLADY NA CHYBY	17
1.6 CYKLUS TESTOVÁNÍ	18
1.6.1 Analýza požadavků	19
1.6.2 Plánování testů	19
1.6.3 Vývoj testovacích případů.....	19
1.6.4 Příprava prostředí	19
1.6.5 Vykonání testů	19
1.6.6 Uzavření testovacího cyklu	20
1.7 TESTOVACÍ DOKUMENTACE.....	20
1.7.1 Testovací plán	20
1.7.2 Testovací scénáře	20
1.7.3 Testovací případy	20
2 ROZDĚLENÍ TESTOVÁNÍ SOFTWARE	22
2.1 STATICKE A DYNAMICKE TESTOVÁNÍ	22
2.2 TESTOVÁNÍ PODLE ÚROVNĚ.....	22
2.2.1 Jednotkové testování	22
2.2.2 Integrační testování	23
2.2.3 Systémové testování.....	23
2.2.4 Akceptační testování	23
2.2.5 Alfa a beta testování.....	23
2.2.6 Regresní testování	24
2.3 NEFUNKČNÍ TESTOVÁNÍ	24
2.3.1 Výkonnostní testování.....	24
2.3.2 Zátěžové testování.....	24
2.3.3 Stresové testování.....	25
2.3.4 Testování použitelnosti	25
2.3.5 Bezpečnostní testování.....	25
2.3.6 Testování přenositelnosti	26
2.4 ČERNÁ A BILÁ SKŘÍŇKA	26
2.4.1 Černá skříňka	26

2.4.2	Bílá skříňka	26
2.4.3	Šedá skříňka	26
2.5	POZITIVNÍ A NEGATIVNÍ TESTOVÁNÍ	27
2.6	MANUÁLNÍ A AUTOMATICKÉ TESTOVÁNÍ	27
2.6.1	Rozdíl manuálního a automatického testování	27
2.6.2	Výhody automatického testování	28
2.6.3	Mýty automatického testování	28
3	MOŽNOSTI POUŽITÍ AUTOMATICKÉHO TESTOVÁNÍ	30
3.1	JEDNOTKOVÉ TESTOVÁNÍ	30
3.1.1	MSTest	30
3.1.2	JUnit	31
3.1.3	NUnit	31
3.2	INTEGRAČNÍ TESTOVÁNÍ	33
3.2.1	SoapUI	33
3.3	TESTOVÁNÍ UŽIVATELSKÉHO PROSTŘEDÍ	33
3.3.1	HP Unified Functional Testing	33
3.3.2	Rational Functional Tester	34
3.3.3	SilkTest	34
3.3.4	Watir	34
3.3.5	Ranorex	34
3.3.6	TestComplete	34
3.3.7	TestStudio	35
3.4	NÁSTROJE PRO PRŮBĚŽNOU INTEGRACI	35
3.4.1	Jenkins	35
3.4.2	TeamCity	36
3.4.3	Bamboo	36
3.4.4	Visual Studio Team Services	36
3.5	VÝKONNOSTNÍ TESTOVÁNÍ	36
3.5.1	HP Load Runner	37
3.5.2	Apache JMeter	37
3.5.3	Visual Studio	37
4	NÁSTROJE SELENIUM	38
4.1	SELENIUM REMOTE CONTROL	39
4.2	SELENIUM IDE	39
4.3	SELENIUM GRID	40
4.4	SELENIUM WEBDRIVER	40
5	SELENIUM WEBDRIVER V C#	42
5.1	ROZHRANÍ IWEBDRIVER	42
5.2	ROZHRANÍ IWEBELEMENT	43
5.3	VYHLEDÁVÁNÍ ELEMENTŮ VE WEBDRIVER	44
5.3.1	Id	44
5.3.2	Name	44
5.3.3	LinkText	44
5.3.4	PartialLinkText	44
5.3.5	XPath	44

5.3.6	ClassName.....	45
5.3.7	CSS.....	45
5.3.8	TagName	45
II	PRAKTICKÁ ČÁST	46
6	POPIS APLIKACE WORKLIO A JEJÍ TESTOVÁNÍ.....	47
6.1	TESTOVÁNÍ WEBOVÉ APLIKACE WORKLIO	48
6.1.1	Testování nových funkcionalit.....	49
6.1.2	Regresní testování	50
6.1.3	Správa defektů.....	51
7	NÁVRH AUTOMATIZACE.....	55
7.1	VÝBĚR NÁSTROJŮ.....	55
7.2	VÝBĚR TESTOVACÍCH PŘÍPADŮ	56
7.3	NÁVRH PROJEKTU	57
7.3.1	Návrhový vzor Page Object Model	57
7.3.2	Page Factory	58
7.3.3	Pořadí testů.....	59
7.3.4	Parametrické testování	59
7.3.5	Kontrola dat.....	60
7.3.6	Nahrávání souborů do systému	60
7.3.7	Testování s různými prohlížeči	60
7.3.8	Vytvoření chybových hlášení.....	61
7.3.9	Využití třídy a metody	62
7.4	VYTVOŘENÍ TESTŮ	62
7.4.1	TestCreateCompany	62
7.4.2	TestLocations	63
7.4.3	TestDepartments	63
7.4.4	TestAddExistingEmployee	63
7.4.5	TestTerminateEmployee	64
7.4.6	TestAddingFullAccessAdmin	64
7.4.7	TestCompanyIdea	64
7.4.8	TestDisciplineRecords	65
7.4.9	Test515Report	65
7.4.10	TestAddContractor	65
7.4.11	TestSetupPTO	65
7.4.12	TestCompanyInfo.....	66
7.5	SPOUŠTĚNÍ TESTŮ	66
7.5.1	Visual Studio	67
7.5.2	Visual Studio Team Services	68
7.6	ÚDRŽBA AUTOMATICKÝCH TESTŮ	71
8	VYHODNOCENÍ PROVEDENÉ AUTOMATIZACE	73
8.1	OMEZENÍ PROVÁDĚNÍ TESTŮ	74
8.1.1	Mozilla Firefox.....	74
8.1.2	Microsoft Edge.....	74
8.1.3	Internet Explorer	75

8.2	EKONOMICKÉ ZHODNOCENÍ.....	75
8.3	DALŠÍ VÝVOJ.....	75
ZÁVĚR		77
ZÁVĚR V ANGLIČTINĚ.....		78
SEZNAM POUŽITÉ LITERATURY.....		79
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....		82
SEZNAM OBRÁZKŮ		84
SEZNAM TABULEK.....		86
SEZNAM PŘÍLOH.....		87

ÚVOD

Doménou dnešní doby je rychlý vývoj informačních technologií. To stejné platí také pro vývoj softwaru, který se stal do značné míry součástí našeho života a obklopuje nás na každém kroku. Nedílnou částí životního cyklu vývoje softwaru je jeho testování, které je stejně důležitou fází jako například analýza požadavků, návrh designu nebo implementace. Cílem testování je odhalení defektů, které vznikly během vývoje softwaru ve všech předchozích fázích. Testování umí prokázat, že v softwaru jsou chyby, naopak prokázat nelze, že program je bez chyb. Opomíjet testování může mít neblahý vliv na kvalitu vyvíjeného softwaru, a tím i na spokojenost koncového uživatele. Naopak správná aplikace metodiky testování přináší úspory během celého životního cyklu vývoje. Testování může být rozděleno podle různých kritérií. Jedním z mnohých rozdělení je podle toho, jestli je testování prováděno manuálně nebo automaticky. Vzhledem k robustnosti a rozsáhlosti aplikací, které jsou v dnešní době vyvíjeny, hraje právě automatizace v procesu testování důležitou roli. Při jejím využití je testování prováděno rychleji a efektivněji, nicméně manuální testování plně nahradit nemůže.

V teoretické části práce jsou popsány základy testování softwaru a rozdělení testování podle různých kategorií. Následně je provedena rešerše možností automatického testování softwaru a nástrojů užívaných v praxi. V další kapitole jsou představeny nástroje z rodiny Selenium, především pak Selenium WebDriver, pomocí kterého je provedena částečná automatizace uživatelského rozhraní webové aplikace Worklio.

V kapitolách praktické části je představena webová aplikace Worklio a způsoby jejího testování. Následně je vypracován návrh automatických testů uživatelského rozhraní, popsány vybrané testovací scénáře a možnosti jejich spouštění. Na závěr je provedeno celkové zhodnocení vypracované automatizace a její další možný vývoj.

I. TEORETICKÁ ČÁST

1 TESTOVÁNÍ SOFTWARE

Definovat pojem testování softwaru není úplně snadné. Lze říci, že co publikace a autor, to odlišný názor na to, co testování je a co není. Jedna z možných definic říká, že testování je proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatele. V širším úhlu pohledu je možné také říct, že testování je nejenom proces řízeného spouštění, ale zahrnuje také metody statického testování. Zde patří testování požadavků, zdrojových kódů a jiných dokumentací. [1]

Definice kvality softwaru podle vydané normy ISO/IEC 25010 říká, že kvalita softwaru je míra, do jaké softwarový produkt splňuje stanovené a implicitní potřeby, je-li používán za stanovených podmínek. Tato norma definuje model kvality produktu a model kvality užití. [1] Kvalita softwaru se skládá z osmi charakteristik, kterými jsou:

- funkčnost,
- účinnost,
- kompatibilita,
- použitelnost,
- bezporuchovost,
- bezpečnost,
- udržitelnost,
- přenositelnost.

Dalším známým modelem popisujícím kvalitu softwaru je model FURPS vytvořený společností Hewlett-Packard. Název modelu je akronym složený z prvních písmen pěti atributů popisujících [1]:

- functionality (funkčnost),
- usability (použitelnost),
- reliability (spolehlivost),
- performance (výkonnost),
- supportability (rozšiřitelnost).

1.1.1 Znamé historické chyby v testování

V dnešní době se software stal součástí našeho každodenního života. Pro spoustu lidí je denní samozřejmostí navštívit Internet nebo přečíst email. Jelikož je každý software psán lidmi, je zřejmé, že nemůže být dokonalý. [2]

1.1.2 Disney, Lví král

Společnosti Disney v roce 1994 vydala první multimediální hru na CD Lví král. Jak se ukázalo, software nebyl dostatečně otestován na různých modelech osobních počítačů, které byly v té době dostupné. Výsledkem bylo, že software fungoval pouze na některých konfiguracích PC. [2]

1.1.3 Chyba dělení s pohyblivou čárkou

Jednalo se o softwarovou chybu způsobenou dělením s pohyblivou čárkou na procesoru Intel Pentium. U této chyby byl zajímavý především postoj společnosti Intel. Chyba dělení byla objevena ještě před vydáním čipu na trh. Nicméně vedení firmy usoudilo, že se nejedná o závažný problém, a tudíž nestojí za to jej ani opravovat, ani publikovat. Pod nátlakem veřejnosti byl Intel nakonec nucen se veřejně omluvit a na náklady na náhradu vadných čipů vynaložit 400 milionů dolarů. [2]

1.1.4 Nosná raketa Ariane 5 z roku 1996

Chyba vznikla v letovém počítači, jehož systém se osvědčil v předchozí raketě Ariane 4. Spočívala v převodu 64 bitového čísla na číslo 16 bitové, jehož maximální hodnota je 32 767, což bylo u Ariane 4 dostačující, nikoliv pak u Ariane 5. Tato klasická chyba přetečení způsobila selhání dalších systémů. Raketa musela být necelou minutu po startu zničena, neboť se odchýlila od svého kurzu. [1]

1.2 Základní pojmy

V praxi bývají poměrně často zaměňovány pojmy, jakými jsou chyba, defekt, selhání. Proto je dobré pro začátek jednotlivé pojmy a jejich vzájemnou souvislost vysvětlit.

- **Defekt** neboli neformálně bug, je následek pochybení člověka a je příčinou chyby.
- **Chyba** je stav systému, který může vést k selhání.
- **Selhání** je nesoulad mezi aktuálním a očekávaným chováním systému. [1]

Defekt nemusí být pouze ve zdrojovém kódu, ale také v dokumentaci popisující vyvíjený produkt. Tuto dokumentaci můžou představovat různé specifikace, případy užití či jiné diagramy. Podle Rona Pattona [2] můžeme o softwarovém defektu mluvit v případě, pokud je splněna alespoň jedna z níže uvedených podmínek.

- Software nedělá něco, co by měl podle specifikace produktu dělat.
- Software dělá něco, co by neměl podle specifikace produktu dělat.
- Software dělá něco, co není ve specifikaci produktu uvedeno.
- Software nedělá něco, o čem se specifikace produktu nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, pomalý anebo pokud jej uživatel nebude považovat za správný (podle názoru testera softwaru).

1.3 Základní axiomy testování softwaru

Axiomem nazýváme takové tvrzení, jehož pravdivost je obecně přijímána a není potřeba jej více dokazovat. Tak jako u jiných oborů, existuje u testování softwaru několik axiomů, které jsou představeny níže. [1]

1.3.1 Žádný netriviální software nemůže být kompletně otestován

Žádný netriviální software nemůže být kompletně otestován, a to hned z několika důvodů:

- příliš velký počet vstupů,
- příliš velký počet výstupů,
- počet možných cest v softwaru je příliš velký,
- specifikace softwaru je subjektivní věc, tudíž lze vždy říci, že chyba je pouze v očích pozorovatele.

Jelikož nemohou být testovány všechny scénáře, je testování vždy položeno na riziku. Proto při testování musí být zúžena podmnožina scénářů. [1] [2]

1.3.2 Testování je věcí odhadu rizika

Z předchozího odstavce je zřejmé, že není možné otestovat kompletně celý software. Pro testování jsou vybrány testovací scénáře podle uvážení testera nebo analytika. Tím je podstoupeno riziko, že možný defekt nebude možné pomocí vybraných scénářů odhalit.

Správně provedená analýza rizik a jejich minimalizace je tudíž nezbytná dovednost každého testera. [1] [2]

1.3.3 Testování může prokázat jen přítomnost defektů, nikoli jejich absenci

Testováním lze prokázat přítomnost defektů v softwaru, nikoliv jeho bezchybnost (už jen z důvodu prvního axiomu), a to bez ohledu na množství času, který je testování softwaru věnován. [1] [2]

1.3.4 Čím více defektů v softwaru je nalezeno, tím více jich v produktu je

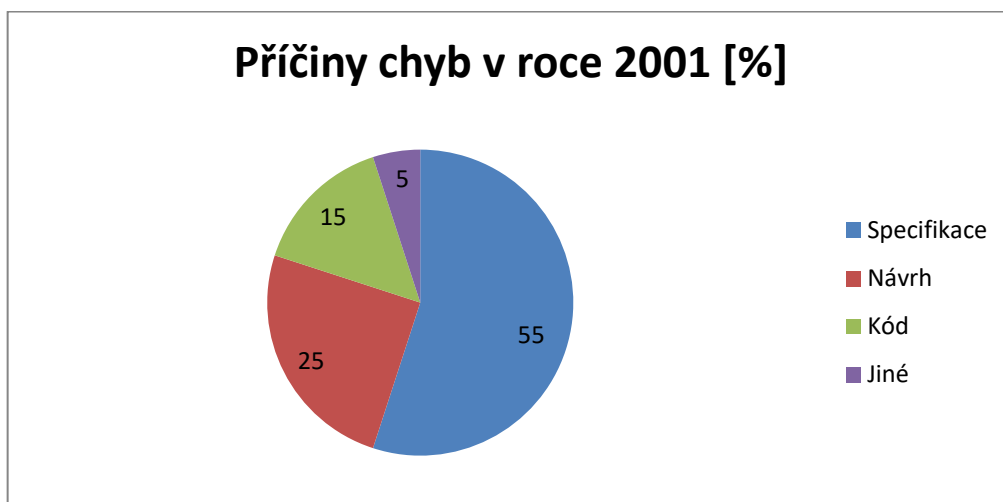
Tento axiom vychází z faktu, že se defekty v softwaru vyskytují ve skupinách, jelikož lidé často dělají stejné chyby. Pokud je tedy v určité části systému nalezeno více defektů, je velká pravděpodobnost, že se zde budou nacházet i další. [1] [2]

1.4 Příčiny defektů

Defekty vznikají ve všech fázích vývoje softwaru od specifikací požadavků až po samotný vývoj. Příčinou defektů může být jeden nebo také kombinace více bodů, které jsou popsány níže [3]:

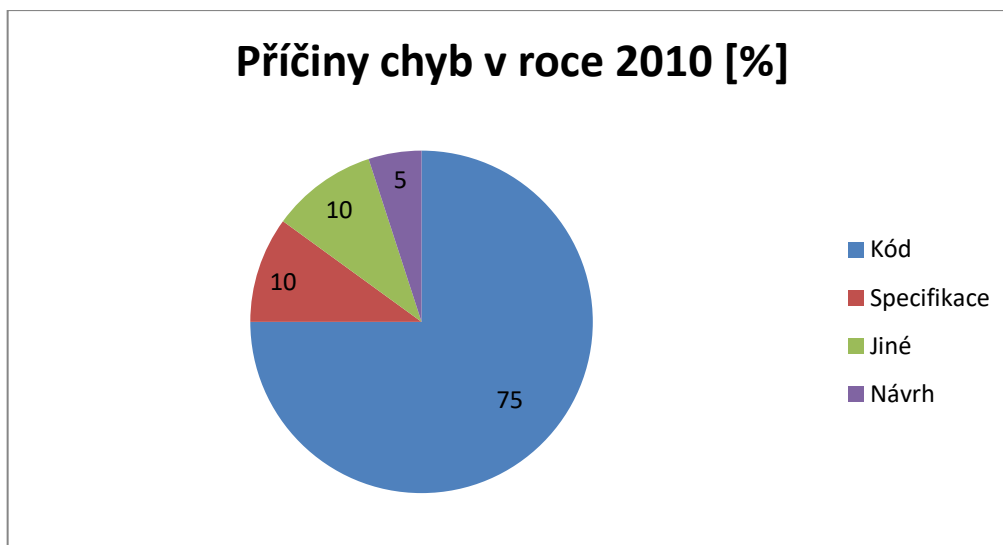
- nepochopení potřeb uživatele, které vede k chybné specifikaci požadavků,
- částečně nebo špatně definované požadavky,
- špatná interpretace požadavků do funkčního designu,
- špatné pochopení funkčního designu a tím pádem nesprávný technický design,
- špatně pochopený technický design a tím špatně naprogramovaný software,
- nedostatečný výkon, špatná ovladatelnost, uživatelské nepohodlí softwaru,
- špatně pochopené požadavky a funkční design a tím pádem defekty v testovacích scénářích a skriptech.

Podle Rona Pattona [2] byla při vývoji softwaru nejčastější příčinou chyb specifikace nebo návrh produktu, což je zdůvodňováno absencí či neúplností psané specifikace, jejími změnami nebo nepromyšleným návrhem.



Obrázek 1: Příčiny vzniku chyb v softwaru v roce 2001 [1]

Aktuálnější materiály však uvádí čísla odlišná. Zde jednoznačně převládají defekty s původem ve zdrojovém kódu, kterých je kolem 75 %. Z toho vyplývá, že oproti předchozím letem, jsou pro konkrétní projekty využívány vhodnější metody a lepší nástroje během analýzy a návrhu systému. [1]

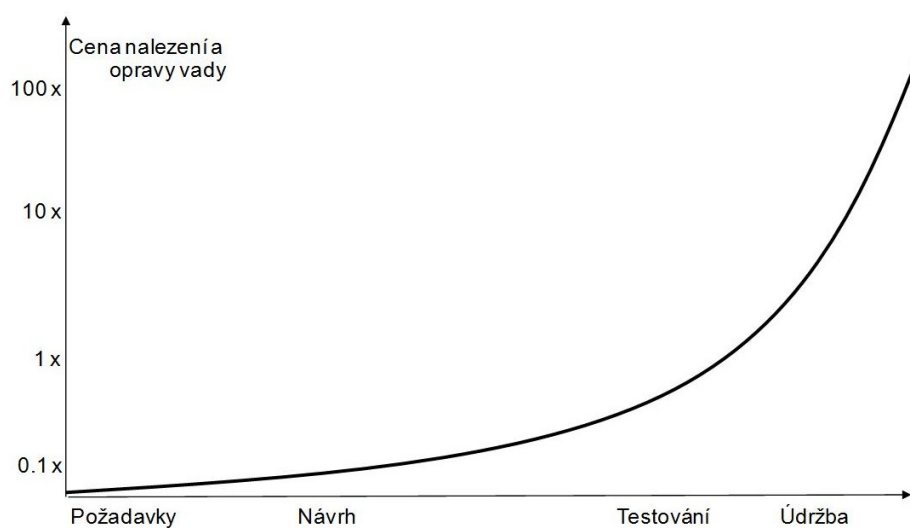


Obrázek 2: Příčiny vzniku chyb v softwaru v roce 2010 [1]

1.5 Náklady na chyby

Chyby v softwaru vznikají od samého začátku psaní specifikací, návrhu, vývoje, testování až po okamžik, kdy je software uveden na trh. Oprava každé chyby má svoje náklady. Výše těchto nákladů v průběhu času roste logaritmickou stupnicí. To znamená, že oprava chyby, která je objevena již ve fázi psaní specifikace, je stonásobně i více levnější, než

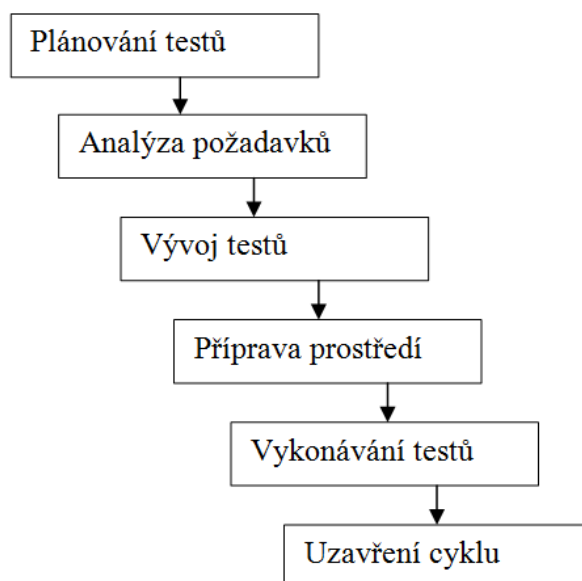
stejná chyba, která je nalezena až ve fázi testování. Obecně platí, že čím dříve je chyba odhalena, tím levnější je její oprava. [2]



Obrázek 3: Náklady na opravu chyb v průběhu času [49]

1.6 Cyklus testování

Životní cyklus testování je posloupnost aktivit, které jsou vykonávány testovacím týmem od začátku až do konce projektu. Posloupnost aktivit se skládá z mnoha různých aktivit, které jsou vykonávány, aby bylo dosaženo kvality produktu. [4]



Obrázek 4: Cyklus testovacího procesu

1.6.1 Analýza požadavků

Analýza požadavků je první fází celého cyklu. V této fázi jsou analyzovány dokumenty s funkčními i nefunkčními požadavky. V případě nejasností jsou požadavky konzultovány s jinými odděleními či přímo s klientem. Činnosti prováděné v této fázi jsou [4]:

- analýza požadavků z hlediska testerů,
- identifikace metod a typů testování,
- určení priorit testování,
- analýza proveditelnosti automatizace.

1.6.2 Plánování testů

Tato fáze začíná hned po fázi analýzy požadavků. Ve fázi je připraven dokument s testovacím plánem a testovací strategií. Dle těchto dokumentů je také odhadnuto úsilí spojené s nadcházejícím testováním. [4]

1.6.3 Vývoj testovacích případů

V této fázi QA tým píše testovací případy a testovací skripty. Psány jsou také testovací případy pro automatické testy, jestliže jsou požadovány. Dále jsou připravována testovací data. Probíhá také ověřování testovacích případů. [4]

1.6.4 Příprava prostředí

Tato fáze zahrnuje instalační proces softwaru a přípravu hardware, na kterém má být testování vykonáváno. Jakmile je instalace provedena, mohou být vygenerována testovací data. Tato fáze může probíhat paralelně s fází vývoje testovacích případů. Bývají zde prováděny také tzv. Smoke testy. [4]

1.6.5 Vykonání testů

Ve fázi je vykonáváno spouštění testů na testovacím prostředí. Nalezené defekty jsou reportovány a odeslány vývojářům, kteří je po opravě posílají zpět k přetestování. Opravené defekty jsou uzavírány. [4]

1.6.6 Uzavření testovacího cyklu

Fáze slouží ke zhodnocení testovacího procesu. Může probíhat formou diskuze, jejíž záměrem je poučit se z chyb, které při testovacím procesu nastaly. To je přínosné pro budoucí testování. [4]

1.7 Testovací dokumentace

Testovací dokumentace by měla být napsána před anebo v průběhu testování softwaru. Dokumentace pomáhá v odhadu testovacího úsilí, pokrytí testů a sledování požadavků. Mezi základní typy dokumentace patří: testovací plán, testovací scénáře a testovací případy. [5]

1.7.1 Testovací plán

Testovací plán je stěžejní dokument, ve kterém je navrhnut postup celého procesu testování softwaru. V plánu je stanovena strategie, která bude použita pro testování, dále prostředky, které budou pro testování využity a testovací prostředí, ve kterém budou testy prováděny. Obsahuje také rozsah postupu testování softwaru a stanovuje časový rozvrh prací. Testovací plán by měl obsahovat tyto části [5]:

- předpoklady testování,
- seznam testovacích případů,
- seznam funkcionalit k testování,
- zvolit přístup k testování,
- definice rizik pro testování,
- plán úkolů, kterých má být dosaženo.

1.7.2 Testovací scénáře

Testovací scénář je sada několika testovacích případů, která je určená k otestování určité oblasti v aplikaci. Tyto případy na sebe navazují a každý následující případ je závislý na výstupu předchozího testovacího případu. Každá oblast aplikace může mít od jednoho přes stovky testovacích scénářů. [5]

1.7.3 Testovací případy

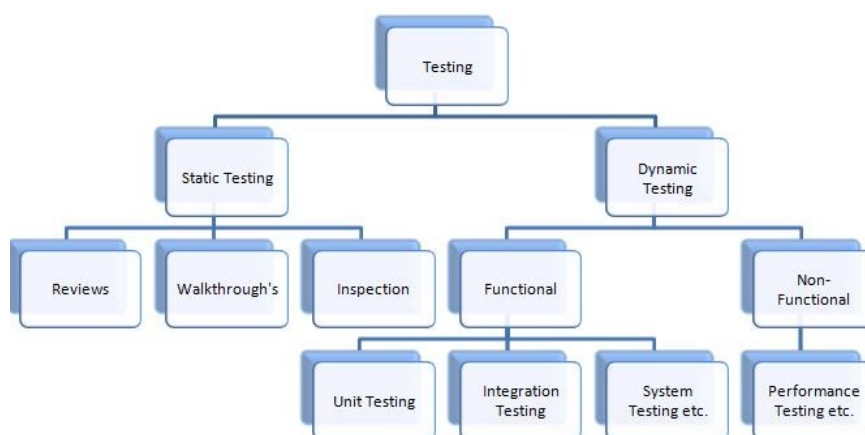
Testovací případy popisují akce prováděné s určitou softwarovou komponentou. Testovací případy je možné vytvořit jak pro manuální, tak pro automatické testy. Pro manuální testy

se jedná o seznam prováděných kroků, vstupů, podmínek a očekávaných výsledků. Automatizované testy jsou tvořeny sadou instrukcí, které by samy měly rozpoznat, zda uspěly či selhaly. Testovací případ by měl obsahovat [5]:

- identifikátor,
- účel,
- podmínky,
- specifikace kroků a vstupů,
- očekávaný výstup,
- aktuální výstup.

2 ROZDĚLENÍ TESTOVÁNÍ SOFTWARE

Testování softwaru lze rozdělit podle různých kritérií do několika kategorií. Základní schéma rozdělení je zobrazeno obrázku níže.



Obrázek 5 : Základní rozdělení testování [6]

2.1 Statické a dynamické testování

Statické testování je testování softwaru bez jeho spuštění. Při testování jsou kontrolovány dokumenty jako specifikace požadavků, dokumenty týkající se designu, zdrojové kódy nebo testovací plány. Výhodou statického testování je, že může být prováděno již v počátečních fázích vývoje. Dynamické testování je testování softwaru za jeho běhu. Funguje na základě zadávání vstupů a následné kontrole výstupů s očekávanými výsledky. Dynamické testování se dále dělí na funkcionální a nefunkcionální. Testování funkcionalit se provádí na všech úrovních testování – jednotkové, integrační, systémové. Mezi nefunkcionální testování se řadí testy spojené s výkonem nebo bezpečností aplikace. [7]

2.2 Testování podle úrovně

Testování může probíhat na několika úrovních. Podle typu úrovně může být testování rozděleno na jednotkové, integrační, systémové, akceptační, alfa, beta a regresní testování.

2.2.1 Jednotkové testování

Jednotkové testování je typ testování, kdy jsou testovány nejmenší části aplikace odděleně od zbytku funkcionality. Mezi testované části aplikace patří třídy, funkce, metody, interface. Jednotkové testování je obecně prováděno vývojáři před integrací softwaru.

Výhodou jednotkového testování je odhalení a oprava chyb v rané fázi vývoje, kdy chyba nemá ještě žádný dopad na zbytek programu. [8]

2.2.2 Integrovní testování

Integrovní testování je testování větších celků tzn. modulů daného softwaru. Typický softwarový projekt se skládá z více modulů, které jsou programovány různými vývojáři. Integrovní testování se zaměřuje na testování datové komunikace mezi těmito moduly. [9]

2.2.3 Systémové testování

Systémové testování je prováděno za účelem zjištění chování celého systému jako jednoho celku. Tyto testy bývají prováděny v pozdějších fázích vývoje softwaru. V této fázi jsou testovány specifikace požadavků softwaru, business procesy, případy užití, interakce s operačním systémem a systémovými zdroji. [10]

2.2.4 Akceptační testování

Akceptační testování je považováno za jedno z nejdůležitějších testování. Je prováděno před uvedením softwaru do provozu, a to přímo klientem nebo jím pověřeným testovacím týmem. Cílem testu je ověřit, jestli je software validní, tzn. jestli splňuje všechny požadavky a je připraven do provozu. Jedná se o důležitý milník vývoje softwaru, kdy klient projekt schválí nebo zamítne. [11] [12]

2.2.5 Alfa a beta testování

Alfa testování je prováděno za účelem identifikace možných chyb a problémů před vydáním softwaru veřejnosti. Testováním je simulován typický uživatel a jeho činnost v aplikaci. Alfa testování je prováděno interními zaměstnanci společnosti, většinou testery v simulovaném prostředí. Beta testování je prováděno reálnými uživateli v reálném prostředí. Beta verze softwaru je vydána pro limitovaný počet koncových uživatelů této aplikace za účelem získání zpětné vazby na kvalitu produktu. Beta testování snižuje rizika selhání produktu a zvyšuje jeho kvalitu. Jedná se o poslední prováděný typ testování před vydáním softwaru na trh. [13]

2.2.6 Regresní testování

Regresní testování je prováděno za účelem ověření, zda změny ve zdrojovém kódu neměly negativní vliv na stávající funkcionalitu softwaru. Testování je prováděno v případech, jestliže:

- jsou změněny požadavky a kód je nutné upravit,
- je přidána jakákoliv nová funkcionalita,
- je opravena jakákoliv chyba v softwaru,
- je opraven problém s výkonem softwaru.

Regresní testování může být zdlouhavé a nákladné. V praxi se jedná o jednu z oblastí testování softwaru, která bývá často automatizována. [14]

2.3 Nefunkční testování

Nefunkční testování zahrnuje veškeré testování systémů, které se netýká přímo jejich funkcionalit, nicméně jsou nezbytné pro jejich správné fungování. Mezi základní typy nefunkčního testování patří výkonnostní, zátěžové, stresové, bezpečnostní testování, testování přenositelnosti a použitelnosti.

2.3.1 Výkonnostní testování

Výkonnostní testování slouží k identifikaci problémů s výkonem aplikace. Provádí se za účelem zjištění, jak rychlé jsou některé aspekty systému pod konkrétním zatížením. Testováním se pomáhá určit, zda systém splňuje výkonnostní požadavky softwaru. Snížení výkonu může být způsobeno [15]:

- prodloužením sítě,
- zpracováním na straně klienta,
- zpracováním transakcí v databázi,
- renderováním dat.

2.3.2 Zátěžové testování

Zátěžovým testováním je simulována práce více uživatelů na systému najednou, čímž se zvyšují nároky na hardware. Testování probíhá za normálních i za hraničních podmínek. Cílem zátěžového testování je ověřit chování systému při tomto zatížení, což pomáhá určit

maximální provozní kapacitu a identifikovat slabá místa v systému. Zátěžové testování je většinou prováděno pomocí automatických nástrojů. [15] [16]

2.3.3 Stresové testování

Stresové testování zahrnuje testování systému za abnormálních podmínek. Testování se používá k zajištění větší stability systému, který pracuje v podmínkách nedostatečných výpočetních zdrojů. Testování může být prováděno například [15] [16]:

- vypnutím nebo restartem síťových portů,
- vypnutím a zapnutím databáze,
- spouštěním různých procesů, které spotřebovávají zdroje jako CPU, paměť.

2.3.4 Testování použitelnosti

Testování použitelnosti je technika testování černé skříňky uživatelského prostředí. POUŽITELNOST může být definována z hlediska pěti faktorů, kterými jsou:

- účinnost využívání,
- jednoduchost na naučení,
- jednoduchost na zapamatování,
- spolehlivost a bezpečnost,
- spokojenost.

Testováním je zajištěna vyšší kvalita softwaru, jeho snadnější použití, uživatelé jsou ochotnější software používat a jeho učení trvá kratší dobu. [15] [16]

2.3.5 Bezpečnostní testování

Bezpečnostní testování zahrnuje testování softwaru k identifikaci mezer v bezpečnosti a zranitelnosti systému. Při bezpečnostním testování by měly být zajištěny následující aspekty [15] [16]:

- integrita,
- autorizace,
- data softwaru jsou bezpečná,
- software splňuje bezpečnostní regulace,
- kontrola vstupů a validace,
- bezpečnost proti SQL Injection,
- bezpečnosti proti Cross-site-scripting,

- bezpečnost řízení SESSION.

2.3.6 Testování přenositelnosti

Testováním přenositelnosti je ověřováno, do jaké míry je daná aplikace schopna pracovat na jiné konfiguraci hardware a operačním systému. [15] [16]

2.4 Černá a bílá skříňka

Z hlediska přístupu ke zdrojovému kódu a znalosti implementace softwaru, může být testování rozděleno na testování černé, bílé a šedé skřínky.

2.4.1 Černá skříňka

Testování černou skříňkou představuje testování softwaru bez znalosti zdrojového kódu, struktur a implementačních detailů. Tento typ testování je založen především na testování podle požadavků a specifikací. Testování je zaměřeno pouze na vstupy a výstupy softwaru, které jsou srovnány s očekávanými výstupy. Nicméně produkt samotný je při testování černou skříňkou, do které není vidět. Z toho vyplývá, že tester nemusí mít pro testování znalosti programování. [17]

2.4.2 Bílá skříňka

Testování bílou skříňkou je způsob testování, kdy tester má k dispozici přístup ke zdrojovému kódu a vidí tedy, jak je software implementován a jak pracuje. Díky této detailní znalosti je schopen software lépe otestovat s využitím neočekávaných vstupních hodnot. Tato technika je zaměřena zejména na bezpečnost softwaru, tok vstupů a výstupů dat, zlepšení designu a použitelnosti. Technika je používána na všech úrovních vývoje (jednotkové, integrační a systémové testování). Pro testování bílou skříňkou musí tester kromě znalostí testování nutně disponovat také znalostí programování, aby byl schopen orientace ve zdrojovém kódu softwaru. [17]

2.4.3 Šedá skříňka

Šedá skříňka je způsob testování, který je kombinací obou předešlých technik. Tester má částečný přístup ke zdrojovému kódu. Nejedná se však o takový přístup, aby bylo testování považováno za testování bílé skřínky. [17]

2.5 Pozitivní a negativní testování

Pozitivní testování je proces ověřování softwaru a kontrola funkcionalit, zda se chovají podle očekávání, kdy jsou na vstupu použity validní data. Validní vstup je množina dat, která musí být aplikací vždy akceptována. Například vstupní pole ve formuláři přijme pouze čísla, žádné jiné vstupy nejsou akceptovány. Pozitivní test je proveden, pokud je dané pole testováno při vstupní hodnotě, která je číselná. Negativní testování je proces ověřování softwaru při použití dat, která nejsou validní. Tímto testováním se ověřuje, jestli se software chová podle očekávání při použití s negativními vstupy. Například chování softwaru po vložení textu do pole, které očekává pouze číselný vstup. [18]



Obrázek 6: Ukázka pozitivního a negativního testování [18]

2.6 Manuální a automatické testování

Manuální testování je testování softwaru, které je vykonáváno člověkem. Tento způsob testování je vhodné použít v případech, kde je k vyhodnocení testu potřeba lidského úsudku a v procesech, které nejsou pravidelně opakovány. Automatické testování je jakékoliv testování, k jehož provedení je využíván software. Některé úlohy jsou tak vykonávány bez lidských testerů, kteří jsou do určité míry nahrazeni. Nicméně plné nahrazení lidského faktoru není reálné. Obecně je v současnosti automatickému testování věnováno stále více pozornosti, jelikož s rostoucími nároky na kvalitu softwaru se vyvíjí také schopnosti nástrojů pro automatizaci. Důvodem k zavedení automatického testování je zvýšení jeho efektivity, jelikož provádění testů a analýza jejich výsledků manuální cestou bývá často časově náročná. [1]

2.6.1 Rozdíl manuálního a automatického testování

Hlavním rozdílem manuálního a automatického testování je, kým je testování prováděno. Mezi další rozdíly patří:

- Automat neumí sám řešit situace, pokud se změní chování systému od očekávaného chování v testu, například malou změnou v uživatelském rozhraní.

- Automat má omezenou možnost analýzy chybného chování testovaného systému. Automat například není schopen dobře vyhodnotit rozvržení prvků na stránce. Tohle je možné vyřešit například snímkováním jednotlivých kroků stránek, které pak tester projde rychleji, než kdyby měl vykonávat celý test manuálně.
- Pokud automatický test selže, výsledky testu musí vyhodnotit ještě tester. Chyba může být buď na straně testovaného systému nebo na straně testu.
- Rozdíl je také v kvalifikaci. K provedení automatizace je potřeba umět programovat skripty testů, znát potřebné nástroje, metody a specifikace v oblasti. [3]

2.6.2 Výhody automatického testování

Mezi hlavní výhody automatického testování softwaru patří:

- Automat vykonává testy "zdarma" a relativně rychle. Testování může být opakováno častěji, s různými vstupními parametry.
- Z dlouhodobého hlediska značné snížení nákladů na údržbu.
- Vysoké pokrytí regresními testy v důsledku stále se zvyšujícího počtu automatizovaných testovacích případů.
- Testy mohou být spouštěny na více platformách pro více vstupních dat. Tím je dosaženo provedení vyššího počtu kombinací testů, které by manuálním testováním nebylo možné dosáhnout.
- Provedené kroky zopakuje automat vždy stejně bez chyb podle toho, jak je naprogramován. Na druhou stranu automat nezkusí jinou variantu, než jaká je mu naprogramována. Tudíž je zde postrádána kreativita manuálního testování.
- Systém je testován již při vývoji automatických skriptů.
- Zvýšení výkonnosti testerů a analytiků, kteří se podílí na automatizaci. Vychází z faktu, že většina lidí vítá možnost rozšířit své dovednosti a vykonávat práci jiným, zajímavým způsobem. [1] [3]

2.6.3 Mýty automatického testování

O automatizovaném testování koluje také řada mýtů. Někdy je také považováno za univerzální prostředek, který v testování vyřeší všechny problémy. To samozřejmě není pravda. Některé z těchto mýtů jsou zmíněny níže:

- Zavedením automatizace je zredukován počet testerů a jsou sníženy náklady na testování. To je pravda pouze z dlouhodobého hlediska testování. Nicméně na začátku vývoje náklady na testování jsou zvýšeny, jelikož pro takový vývoj je potřeba větší počet pracovníků. Výhoda však spočívá ve vyšší efektivitě a kvalitě provádění testů.
- Je možná automatizace 100 % testovacích případů. Automatizace všech testovacích případů je nereálný a nepraktický cíl. Některé testy se z důvodů nákladů, důležitosti a četnosti jejich provádění automatizovat nevyplatí.
- Automatizace se ihned vyplatí. Očekávání, že se náklady spojené s automatizací testování rychle vrátí, je nesprávné. Většinou se jedná o návratnost v řádu měsíců až let. [1]

3 MOŽNOSTI POUŽITÍ AUTOMATICKÉHO TESTOVÁNÍ

Automatické testování je při vývoji softwaru využíváno pro usnadnění a zefektivnění testování. Existuje celá řada volně dostupných nebo placených nástrojů umožňující testování automatizovat. Podle typu prováděného testování je nástroje možné rozdělit do několika skupin. Jedná se o:

- jednotkové testování,
- integrační testování,
- testování uživatelského rozhraní,
- zátěžové testování.

3.1 Jednotkové testování

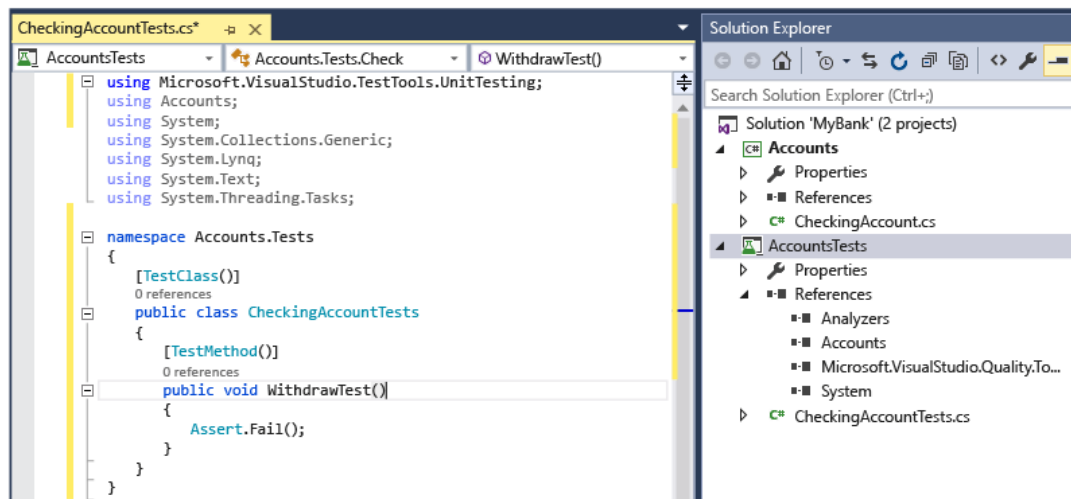
Jednotkové testování zahrnuje testování nejmenších jednotek softwaru. Při testování jsou volány jednotlivé funkce a je kontrolováno, jestli vrátí stejný výsledek jako je ten očekávaný. Výhodou testování jednotek je, že testy mohou být prováděny na úrovni samotného kódu ještě před tím, než existuje integrační nebo uživatelské rozhraní vyvíjeného softwaru. Pro testování jednotek jsou používány frameworky, s jejichž pomocí je testování efektivnější. Většina těchto frameworků vychází z rodiny xUnit architektury. Pro každý rozšířenější jazyk existuje konkrétní framework. Soubor jednotkového testu se skládá z několika částí. Těmi jsou tzn. set-up, tear-down a testovací metody. Set-up metoda je metoda, která se vykoná vždy před samotným testem a slouží k inicializaci prostředí před testem. Tear-down metoda je vykonána vždy po skončení testu a slouží k jakémusi úklidu prostředí po testu. Některé z používaných frameworků jsou JUnit, NUnit nebo MSTest. [3]

3.1.1 MSTest

Jedná se o framework pro jednotkové testování na platformě .NET od společnosti Microsoft. Pomocí Visual Studio Test Explorer je možné spouštění testů a zobrazení jejich výsledků přímo ve Visual Studiu. S verzí Visual Studio Enterprise mohou být testy spouštěny automaticky po každém buildu. Základní atributy metod a tříd v MSTest jsou [50]:

- TestClass: Slouží jako identifikátor třídy, která obsahuje testy.
- TestMethod: Slouží k identifikaci metod, které obsahují test.

- **TestInitialize:** Metoda označená tímto atributem je provedena před každým testem.
- **TestCleanUp:** Metoda označená tímto atributem je provedena po každém testu.
- **Assert:** Obsahuje metody pro porovnání očekávaného a aktuálního výsledku.



Obrázek 7: Ukázka testovací metody v MS Test [50]

3.1.2 JUnit

JUnit je framework pro jednotkové testování v jazyce Java. Na vývoji se podíleli Kent Beck a Erich Gamma. Framework vychází z rodiny xUnit frameworků. Třídy a metody testů jsou označovány atributy. Mezi hlavní atributy JUnit patří [19]:

- **@Test:** Atribut označuje testovací metody.
- **@Before:** Metoda s atributem je vykonána před každým testem.
- **@After:** Metoda s atributem je vykonána po každém testu.
- **@BeforeClass:** Metoda je vykonaná jednou před spuštěním všech testů.
- **@AfterClass:** Metoda je vykonaná jednou po provedení všech testů.

3.1.3 NUnit

NUnit je jednotkový open-source framework patřící do rodiny programů xUnit, který je vytvořen pro platformu .NET. NUnit byl původně přenesen z frameworku JUnit, nicméně aktuální verze NUnit 3.0 byla kompletně přepsána. Přidána byla podpora pro širokou škálu .NET platforem a mnoho nových funkcí. Pomocí tohoto frameworku a adaptéru pro něj určeného, je možné testy zobrazit a spouštět přímo ve Visual Studiu. [39] Struktura testovacích tříd a metod s atributy NUnit, která byla použita v projektu je zobrazena níže.

```
namespace WorkliouUnitTest
{
    [TestFixture]
    0 references | 0 changes | 0 authors, 0 changes
    public class Setup
    {
        [OneTimeSetUp]
        0 references | 0 changes | 0 authors, 0 changes
        public void BeforeAllTests()
        {
        }

        [OneTimeTearDown]
        0 references | 0 changes | 0 authors, 0 changes
        public void AfterAllTests()
        {
        }
    }

    [TestFixture]
    0 references | 0 changes | 0 authors, 0 changes
    public class Tests
    {
        [Setup]
        0 references | 0 changes | 0 authors, 0 changes
        public void BeforeEveryTest()
        {
        }

        [Test]
        0 references | 0 changes | 0 authors, 0 changes
        public void Test1()
        {
        }

        [Test]
        0 references | 0 changes | 0 authors, 0 changes
        public void Test2()
        {
        }

        [TearDown]
        0 references | 0 changes | 0 authors, 0 changes
        public void AfterEveryTest()
        {
        }
    }
}
```

Obrázek 8: Struktura testů v NUnit

Mezi základní atributy patří:

- **SetUpFixture**: Atribut třídy, která obsahuje metody, které jsou vykonány před a po vykonání všech testovacích případů.
- **OneTimeSetUp**: Metoda je provedena pouze jednou před vykonáním všech testů.
- **OneTimeTearDown**: Metoda je provedena pouze jednou po vykonání všech testů.
- **TestFixture**: Atribut označuje třídu, která obsahuje testy v NUnit.
- **Setup**: Metoda označená tímto atributem je vykonána před každým testem.
- **Test**: Značení pro metodu, která je testem.
- **TearDown**: Metoda označená tímto atributem je vykonána po každém testu.

3.2 Integrační testování

Integračním testováním je ověřována správnost dat při přenosu mezi jednotlivými moduly softwaru přes integrační rozhraní. Pro využití automatických testů jsou uvažovány dva typy integračních rozhraní. Jedná se o:

- Lokální API: slouží pro komunikaci mezi moduly jednoho systému.
- Vzdálené API: slouží pro komunikaci se svým okolím pomocí rozhraní REST nebo jiné webové služby.

Pro integrační testování je možné použít jednotkový framework nebo nějaký samostatný nástroj pro integrační testování, například SoapUI. [3]

3.2.1 SoapUI

SoapUI je multiplatformní nástroj od společnosti SmartBear Software. SoapUI byl vyvinut v roce 2006. Jedná se o velmi rozšířený open-source nástroj s širokou komunitou. Nástroj slouží k funkčnímu i nefunkčnímu testování softwaru. SoapUI podporuje webové služby jakými jsou například SOAP a REST. [20]

3.3 Testování uživatelského prostředí

Další možností využití automatických testů je testování uživatelského rozhraní. Existují dva základní způsoby tvorby testovacích scénářů UI. První možností je použití nástroje umožňující nahrávat aktivitu uživatele v testovaném systému a následně tuto aktivitu přehrát. Druhou možností je programování testovacích skriptů, které jsou následně automaticky spouštěny. Testování uživatelského rozhraní je možné rozdělit podle platformy, na které jsou testy prováděny. Jedná se tedy o desktopové, webové a mobilní aplikace. Mezi nejpoužívanější nástroje pro testování uživatelské rozhraní patří HP Unified Functional Testing, IBM Rational Functional Tester, SilkTest, Watir, Ranorex, TestComplete, Telerik TestStudio, Selenium.

3.3.1 HP Unified Functional Testing

Unified Functional Testing (UFT) je komplexní placený nástroj pro automatické testování softwaru od společnosti Hewlett-Packard. Dříve se jmenoval QuickTest Professional. Pomocí nástroje můžou být testovány desktopové, webové a mobilní aplikace. Testy je možné nahrávat pomocí GUI anebo vytvářet skripty v jazyce Visual Basic Script. [24]

3.3.2 Rational Functional Tester

Rational Functional Tester je nástroj pro automatické testování funkčnosti, regresní testování, testování grafického uživatelského rozhraní a testování řízené daty od společnosti IBM. Rational Function Tester podporuje široké spektrum aplikací, např. webových aplikací, aplikací v prostředí .Net, Java, Siebel, SAP. [43]

3.3.3 SilkTest

SilkTest je nástroj pro automatizaci funkcionálního a regresního testování softwarových aplikací. Testy je možné vytvářet pomocí nahrávání nebo psaním skriptů. Mezi základní podporované technologie patří AJAX, web, mobile web, Java, .NET, SAP. Testování je možné provádět v prohlížečích Internet Explorer, Edge, Firefox, Safari. Mobilní testování je dostupné pro platformy iOS a Android. [44]

3.3.4 Watir

Watir (Web Application Testing in Ruby) je open-source sada knihoven v Ruby sloužící pro automatické testování webových aplikací. Watir neumožňuje nahrávat automatické testy, lze pouze psát automatizované skripty v jazyce Ruby. Podporováno je použití prohlížečů Internet Explorer, Google Chrome, Mozilla Firefox, Opera a Safari. Jedná se o účinný a na použití jednoduchý nástroj. [21]

3.3.5 Ranorex

Ranorex je placený nástroj pro automatické testování desktopových, webových a mobilních aplikací, který podporuje spoustu technologií. Mezi některé z nich patří:

- Desktopové: .NET, WinForms, WPF, Java, Delphi, Oracle Forms.
- Webové: HTML5, Javascript, Ajax, Silverlight, Flash, ASP.NET, jQuery.
- Mobilní: nativní iOS, nativní Android, MonoTouch, PhoneGap.

Testy je možné vytvářet v GUI, dále přímo pomocí skriptů za použití jazyků C# nebo VB.NET. Dostupnost Ranorexu je pouze pro operační systém Windows. [22]

3.3.6 TestComplete

TestComplete je nástroj pro tvorbu funkcionálních automatických testů desktopových, webových a mobilních aplikací od společnosti SmartBear Software. Testy je možné

nahrávat nebo vytvářet pomocí skriptů. TestComplete umožňuje vytvářet výkonnostní testy zkonvertováním funkcionálních testů. [23]

3.3.7 TestStudio

TestStudio je rozsáhlý nástroj pro automatizaci testování od společnosti Telerik. Nástroj je určený pro automatizaci desktopových, webových, mobilních aplikací uživatelského rozhraní, zátěžových a výkonnostních testů. Mezi podporované technologie patří HTML, AJAX, Silverlight, WPF and ASP.NET MVC. TestStudio umožňuje testy spouštět v Internet Explorer, Mozilla Firefox, Google Chrome a Safari. [25]

3.4 Nástroje pro průběžnou integraci

Průběžná integrace je soubor nástrojů a praktik, které jsou využívány při vývoji komplexních systémů různorodými týmy. Při vývoji softwaru v mnohačetných týmech bývají zdrojové kódy uloženy na centrálním úložišti. Tyto úložiště obvykle disponují funkcionalitou, jakou je verzování, historie souborů nebo informace o změnách. Cílem průběžné integrace je průběžné ověřování dílčích dodávek funkcionalit. Při tzn. commitu provedených změn v kódu jsou změny nahrány z lokálního úložiště programátora na centrální úložiště. Zde je následně automaticky provedeno sestavení kontrolovaného systému. Jestliže sestavení selže, programátor o tomto selhání dostane oznámení. V případě, že je sestavování v pořádku, následuje proces průběžné integrace provedením testů systému, kterými mohou být např. jednotkové, integrační nebo UI testy. Výhodou průběžné integrace je urychlení vývoje softwaru a urychlení nalezení chyb při jeho vývoji. Mezi známé a často používané nástroje pro integraci patří Jenkins, Jet Brains TeamCity, Atlassian Bamboo nebo Visual Studio Team Services.

3.4.1 Jenkins

Jedním z používaných nástrojů pro průběžnou integraci je Jenkins. Jedná se o open-source nástroj napsaný v jazyce Java. Proto má Jenkins silnou podporu nástrojů určených pro projekty napsané v tomto jazyce. Nicméně jej lze použít i pro projekty napsané v jiných jazycích. Jenkins obsahuje nástroje pro verzování jako jsou Git, CVS nebo Mercurial. Sestavení může být spouštěno různými způsoby, např. commitem ve verzovacím systému, nebo lze sestavení naplánovat. [45]

3.4.2 TeamCity

Jedná se o placený nástroj pro průběžnou integraci od společnosti JetBrains, vydán v roce 2006. Nástroj je napsaný v jazyce Java. Mezi podporované platformy patří Java, .Net a Ruby. Nástroj je možné integrovat s Eclipse, Visual Studio, IntelliJ IDEA. Podporované verzovací systémy jsou Subversion, Perforce, CVS, Git, Mercurial, Visual studio Team Services. [46]

3.4.3 Bamboo

Bamboo je server průběžné integrace vyvinutý společností Atlassian v roce 2007. Jedná se o placený nástroj. Bamboo umí provést sestavení pro nástroje, kterými jsou Ant, Maven nebo MSBuild. Podporuje testovací nástroje JUnit, PHPUnit nebo Selenium. Některými podporovanými verzovacími systémy jsou Perforce, CVS, Git, Mercurial, Apache Subversion. Bamboo je úzce propojeno s nástrojem pro trackování defektů JIRA. [47]

3.4.4 Visual Studio Team Services

Komplexní nástroj od společnosti Microsoft, vytvořený jako cloudové řešení. Jedná se o placený produkt, nicméně pro malé týmy do pěti uživatelů je zdarma. Některé z funkcionalit Visual Studio Team Services jsou:

- Úložiště: podpora TFVC a Git.
- Agilní nástroje: nástroje jako kanban, backlogy, scrum panel, propojení změn kódu se scénáři, defekty.
- Průběžná integrace: sestavení je možné na platformách Windows, Mac a Linux.
- Manuální testování: správa testovacích případů. Možnost spouštět a zaznamenávat chyby přímo v systému.
- Průběžné testování: podpora frameworků a technologií jakými jsou NUnit, MSTest, JUnit, Maven, NodeJS, Java, xUnit.net. [51]

3.5 Výkonnostní testování

Další oblastí, která může být automatizována, je výkonnostní testování. Cílem je monitorování výkonu během scénářů s různou mírou zátěže za účelem ověření, zda splňuje očekávání zákazníka. Pro automatizaci zátěžových testů je možné použít nástroje, jakými jsou JMeter, HP Load Runner nebo Visual Studio.

3.5.1 HP Load Runner

HP Load Runner je oblíbeným nástrojem pro testování výkonu. Nástroj umí simulovat práci tisíce uživatelů na testované aplikaci, nahrávat a později analyzovat výkon klíčových komponentů aplikace. [26]

3.5.2 Apache JMeter

Apache JMeter je volně dostupný nástroj vyvinutý v jazyce Java. Je možné jej použít pro výkonnostní testy webových aplikací nebo k simulaci zatížení na serverech. Mezi výhody JMeteru patří rozšiřitelnost, škálovatelnost, přenositelnost a přehledné uživatelské rozhraní. [1] [27]

3.5.3 Visual Studio

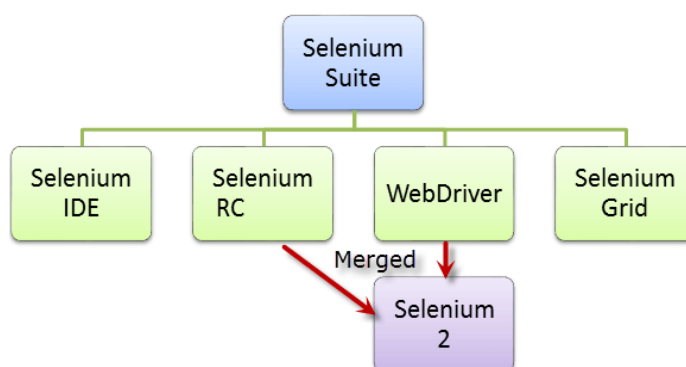
Vytvářet a spouštět výkonnostní testy lze přímo v nástroji Visual Studio. Pro vytvoření testu je využit projekt Web Performance and Load Testing Project. Jednotlivé kroky testu jsou nahrávány při činnosti uživatele na testovaném webu. K provedení výkonnostního testu je možné použít vlastní počítač nebo je testy možné spouštět na virtuálních strojích na cloudu, za použití služby Visual Studio Team Services. [48]

4 NÁSTROJE SELENIUM

Selenium je sada testovacích nástrojů, která slouží k automatizaci testování uživatelského rozhraní webových aplikací. Sada je tvořena celkem ze čtyř komponent, kterými jsou:

- Selenium Integrated Development Environment (IDE),
- Selenium Remote Control (RC),
- Selenium WebDriver,
- Selenium Grid.

Komponenty Selenium RC a WebDriver byly ve verzi Selenium 2 sloučeny do jednoho frameworku. [28]



Obrázek 9: Rozdělení nástrojů Selenium [29]

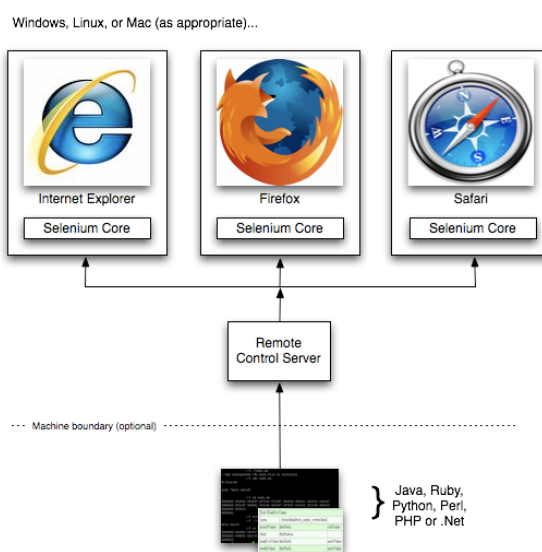
Selenium bylo vytvořeno v roce 2004 Jasonem Hugginsem jako nástroj pro testování interních aplikací ve společnosti ThoughtWorks. Pro usnadnění byla vyvinuta JavaScriptová knihovna, která umožňovala automatické spouštění testů s více různými webovými prohlížeči. Tato knihovna se stala základem všech funkcionalit Selenia RC a Selenia IDE. Velkou výhodou Selenia RC bylo, že umožňoval použití více programovacích jazyků. Nicméně vzhledem k bezpečnostním limitům prohlížečů vztahujícím se k JavaScriptu, nebylo možné některé akce provádět. [28]

V roce 2006 začal Simon Stewart pracovat na projektu, který se jmenoval WebDriver. Jeho záměrem bylo vyvinout testovací nástroj, který by komunikoval přímo s prohlížečem. Díky tomu se vyhnul omezení JavaScriptového prostředí, jako tomu bylo v Seleniu RC. V roce 2009 došlo ke spojení těchto dvou nástrojů, čímž vznikl Selenium Webdriver, známý také jako Selenium 2. [28]

4.1 Selenium Remote Control

Selenium Remote Control (RC) je testovací nástroj umožňující psát automatizované testy uživatelského rozhraní webové aplikace. Testy je možné psát v různých programovacích jazycích a také s využitím více prohlížečů. Selenium RC je tvořen dvěma částmi [30]:

- Server, který automaticky spouští a ukončuje prohlížeč, zachycuje a interpretuje http zprávy z testovacího skriptu.
- Klientské knihovny, které poskytují rozhraní mezi jedním z programovacích jazyků (Java, C#, Python, Perl, PHP) a Selenium RC serverem.

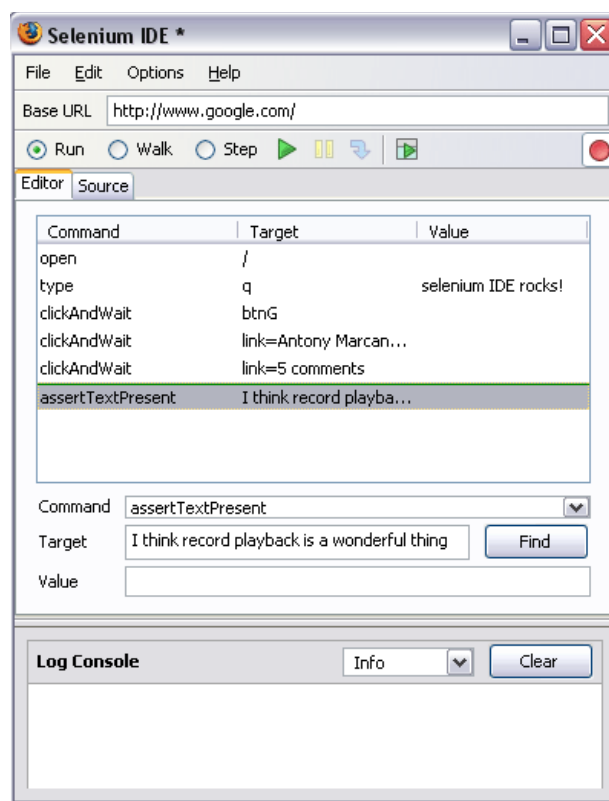


Obrázek 10: Architektura Selenium RC [30]

4.2 Selenium IDE

Selenium IDE (Integrated Development Environment) je plugin umožňující nahrát, editovat a spouštět automatické testy. Je založen na Selenium Core a je implementován jako rozšíření webového prohlížeče Mozilla Firefox. Z toho také vyplývá fakt, že tento prohlížeč je jediný, ve kterém testy mohou být vykonávány. Jedná se o nejjednodušší způsob vytváření a spouštění automatických testů pomocí Selenia. Testovací kód je vytvářen po stisknutí tlačítka Record tak, že Selenium nahrává aktivity, které uživatel provádí. Jako kroky můžou být nahrány otevření nové URL adresy, vyplnění pole ve formuláři či kliknutí na nějaké tlačítko. Nahrané testy je poté možné spouštět jednotlivě anebo všechny najednou. IDE je vhodné pro uživatele, kteří nemají žádné zkušenosti s psaním automatických testů a znalosti programovacích jazyků. Naopak tento způsob

vytváření testů není vhodný pro rozsáhlé projekty a vytváření komplexních testovacích případů. [31]



Obrázek 11: Selenium IDE [32]

4.3 Selenium Grid

Selenium Grid je nástroj, který umožňuje distribuovat automatizované testy napříč více fyzických či virtuálních PC. Testy pak mohou být prováděny paralelně, což výrazně urychlí proces testování s různými prohlížeči. Obecně existují dva důvody, proč použít Selenium Grid [32]:

- možnost spouštění testů proti více prohlížečům, více verzím prohlížečů a prohlížečům běžících na různých platformách,
- sníží se testovací čas.

4.4 Selenium Webdriver

Selenium WebDriver, znám také pod názvem Selenium 2, je framework pro automatické testování webových aplikací, který vznikl spojením Selenium 1 a WebDriver. Selenium Webdriver umožňuje provádět automatické testy s různými prohlížeči, přičemž skripty mohou být psány v programovacích jazycích Java, .NET, PHP, Python, Perl, Ruby. Nástroj

komunikuje přímo s prohlížečem, tudíž není potřeba žádná další komponenta jako je tomu u Selenia RC. Proto je také WebDriver rychlejší než Selenium RC. [33][34]



Obrázek 12: Architektura Selenium WebDriver [34]

Dalšími výhodami nástroje jsou jednodušší API WebDriveru, které neobsahuje redundantní a někdy matoucí příkazy, jako tomu bylo v Seleniu RC. Pomocí WebDriveru je možná práce s dialogovými a pop-up okny, čímž se zvyšuje pokrytí funkčních testů. V neposlední řadě WebDriver podporuje testování na platformách iPhone a Android. [33][34]

5 SELENIUM WEBDRIVER V C#

Pro psaní skriptů pomocí nástroje Selenium WebDriver v jazyce C# jsou využívány především dvě základní rozhraní, kterými jsou `IWebDriver` a `IWebElement`. Rozhraní `IWebDriver` slouží pro práci s webovým prohlížečem, `IWebElement` reprezentuje prvky na webové stránce.

5.1 Rozhraní `IWebDriver`

Rozhraní `IWebDriver` obsahuje metody a vlastnosti, které slouží pro ovládání webového prohlížeče. Aby bylo možno přistoupit k metodám a vlastnostem rozhraní, musí být vytvořena jeho instance. Při vytváření instance se také rozhoduje o tom, ve kterém webovém prohlížeči bude automatický test proveden. Vytvoření instance driveru nejpoužívanějších internetových prohlížečů je zobrazena v tabulce níže. [35]

Tabulka 1: Vytvoření instance driveru pro nejpoužívanější prohlížeče

Prohlížeč	Vytvoření instance driveru prohlížeče
Mozilla Firefox	<code>IWebDriver driver = new FirefoxDriver();</code>
Google Chrome	<code>IWebDriver driver = new ChromeDriver();</code>
Internet Explorer	<code>IWebDriver driver = new InternetExplorerDriver();</code>
Edge	<code>IWebDriver driver = new EdgeDriver();</code>

Proto, aby bylo možné driver použít, musí být nainstalovány knihovny driveru Selenium a driveru prohlížeče, ve kterém mají být testy spouštěny. Všechny drivery je možné vyhledat a nainstalovat přímo v prostředí Visual Studio pomocí služby správce balíků NuGet. Často používané metody rozhraní jsou zobrazeny v tabulce níže.

Tabulka 2: Výběr používaných metod rozhraní `IWebDriver` [35]

Metoda	Popis
<code>FindElement(By by)</code>	Metoda nalezne první element na stránce, na který narazí.
<code>FindElements(By by)</code>	Metoda nalezne všechny elementy na dané stránce.
<code>Close()</code>	Zavře aktuální okno prohlížeče.
<code>Quit()</code>	Ukončí driver a s ním všechna asociovaná okna.

Tabulka 3: Metody používané pro navigaci webového prohlížeče [36]

Metoda	Popis
<i>GoToUrl(string url)</i>	Metoda otevře požadovanou webovou stránku do aktuálního okna prohlížeče.
<i>Refresh()</i>	Metoda provede refresh aktuálně zobrazené webové stránky.
<i>Back()</i>	Metoda zobrazí předchozí navštívené webové stránky podle historie prohlížeče.
<i>Forward()</i>	Metoda zobrazí následující zobrazenou stránku podle historie prohlížeče.

5.2 Rozhraní IWebElement

Rozhraní IWebElement obsahuje metody a vlastnosti, které slouží pro ovládání jednotlivých elementů na webové stránce. Těmito elementy mohou být tlačítka, textová pole, radiobuttony, checkboxy, odkazy, tabulky atd. Mezi nejpoužívanější metody a vlastnosti patří [37]:

Tabulka 4: Nejpoužívanější metody rozhraní IWebElement [37]

Metoda	Popis
<i>Clear()</i>	Metoda smaže obsah elementu.
<i>Click()</i>	Metoda simuluje kliknutí na daný element.
<i>FindElement(By by)</i>	Metoda nalezne první element v kontextu, na který narazí.
<i>FindElements(By by)</i>	Metoda nalezne všechny elementy v kontextu.
<i>GetAttribute(string attributeName)</i>	Metoda vrací hodnotu atributu elementu.
<i>SendKeys(string text)</i>	Metoda simuluje psaní textu do zvoleného elementu.
<i>Submit()</i>	Metoda slouží pro odeslání formuláře na server.

Tabulka 5: Nejpoužívanější vlastnosti rozhraní IWebElement [37]

Vlastnost	Popis
<i>Displayed</i>	Vrací hodnotu uvádějící, jestli je element zobrazen.
<i>Enabled</i>	Vrací hodnotu uvádějící, jestli je element povolen.
<i>Selected</i>	Vrací hodnotu uvádějící, jestli je element vybrán.
<i>Text</i>	Vrací textový řetězec elementu.

5.3 Vyhledávání elementů ve WebDriver

K tomu, aby mohly být prováděny jakékoliv operace s elementy na stránce, musí být tyto elementy nejprve nalezeny. K jejich vyhledávání slouží nástroje, které jsou zabudované ve webových prohlížečích nebo je možné je do prohlížeče stáhnout jako doplněk. Některé z těchto nástrojů jsou Firebug, Firepath nebo Chrome Developer Tools. Elementy jsou identifikovány pomocí tzv. lokátorů, kterých je ve WebDriveru celkem osm. [38]

5.3.1 Id

Vyhledávání elementů pomocí atributu Id je nejjednodušší a nejbezpečnější způsob, jak najít hledaný element na stránce. Každý atribut Id by měl být v rámci jedné webové stránky unikátní. U atributu Id je méně pravděpodobné, že bude v programovém kódu změněn, než je tomu u jiných atributů. [38]

5.3.2 Name

Vyhledávání podle atributu Name je velmi podobné vyhledávání podle atributu Id. Atribut Name bývá často použit ve formulářích v prvcích, jako jsou textová pole nebo radiobuttony. Jeho hodnota je po potvrzení formuláře poslána na server. [38]

5.3.3 LinkText

Vyhledávání podle atributu LinkText slouží pouze pro vyhledávání hypertextových odkazů za použití textu tohoto hyperlinku jako parametru. [38]

5.3.4 PartialLinkText

V případě lokátoru PartialLinkText stačí jako parametrem lokátoru poskytnout část textu hledaného odkazu. Tenhle způsob vyhledávání je výhodný v situacích, kdy je potřeba vyhledat hyperlink, který je dynamicky měněn za chodu aplikace. [38]

5.3.5 XPath

XPath je zkratka XML path jazyka. Jedná se o dotazovací jazyk pro výběr uzlu z dokumentu XML. Jedná se o velmi efektivní způsob, jak najít daný element na webové stránce. Výhodou tohoto lokátoru je, že může být použit téměř ve všech případech, dokonce když není možné přistoupit k elementu pomocí Id, Name, Class atd. Na druhou stranu se pro svou komplexnost a množství pravidel jedná o nejkomplikovanější způsob

vyhledání elementu. Nicméně existují nástroje, které generování XPath usnadní, jedná se například o nástroj Firebug. [38]

5.3.6 ClassName

Atribut Class bývá využíván pro stylování elementu. Mimoto může být použit také jako lokátor k identifikaci elementu. [38]

5.3.7 CSS

Pro vyhledání elementů může sloužit také CSS selektor. Vyhledávání je rychlejší než za použití XPath. [38]

5.3.8 TagName

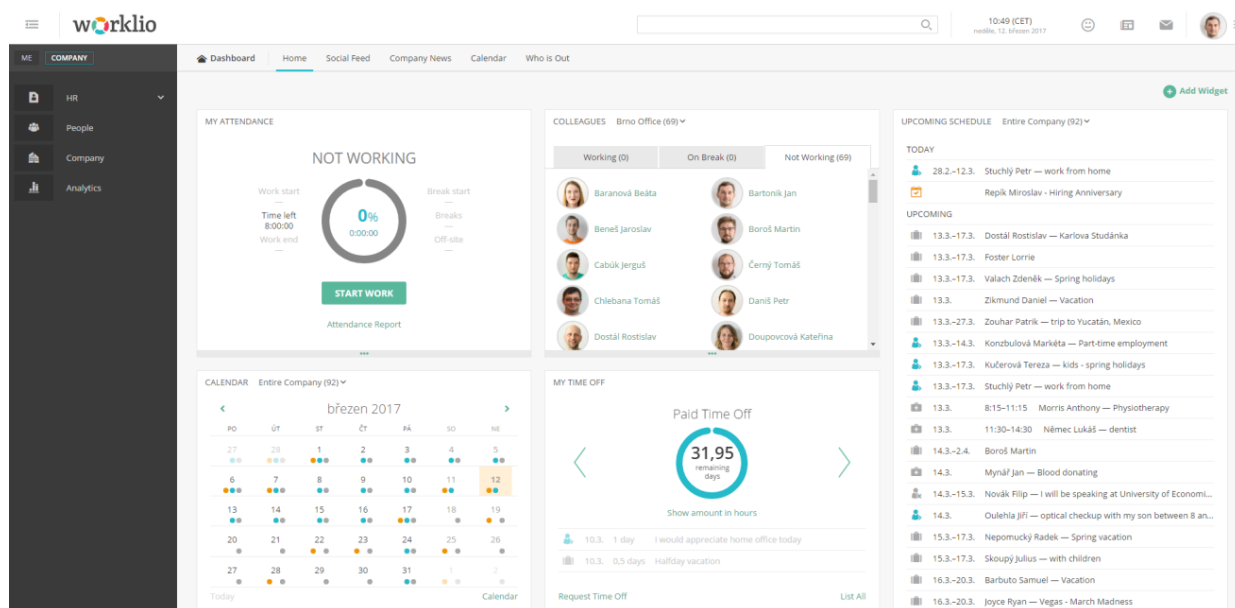
Pro vyhledání elementů slouží také HTML Tagy. Stejný Tag může být na stránce obsažen vícekrát. Příklad vhodného použití TagName je pro vyhledávání tabulek. [38]

I. PRAKTICKÁ ČÁST

6 POPIS APLIKACE WORKLIO A JEJÍ TESTOVÁNÍ

Worklio je mezinárodní společnost s hlavním sídlem na Floridě v USA. Jedna z poboček společnosti sídlí v ČR v Brně. Stěžejním produktem firmy je webová aplikace Worklio vyvíjená na platformě ASP.NET. Jedná se o HR informační systém určený primárně pro trh v USA. Mezi základní moduly systému patří:

- dashboard,
- správa dovolené,
- správa docházky,
- správa mezd,
- správa pojištění,
- správa úkolů,
- správa majetku,
- 5-15 reporty.



Obrázek 13: Dashboard webové aplikace Worklio

Vedle webové aplikace je vyvíjena také mobilní aplikace tohoto systému. Pro její vývoj je použita technologie Xamarin. Mobilní aplikace je podporuje tři nejpoužívanější mobilní platformy, kterými jsou iOS, Android a Windows Phone.

6.1 Testování webové aplikace Worklio

Testování aplikace je prováděno týmem testerů, který je složen z pěti členů. Úkolem týmu je zajištění kvality vyvíjené webové aplikace, mobilní aplikace a marketingových stránek společnosti. Nová funkcionality aplikace je vydávána v pravidelných cyklech, tzv. release, které probíhají přibližně každé dva až tři měsíce. Kromě těchto hlavních release jsou vydávány ještě menší release a tzv. hotfixy, pokud je tomu třeba. Testování systému je prováděno jako testování černé skříňky, což znamená, že testéři nemají přístup k databázi ani ke zdrojovým kódům. Systém je tedy testován na základě vstupů, očekávaných výsledků a dosaženého výstupu. Než je nová funkcionality vydána na produkční prostředí, musí nejprve projít přes všechny testovací prostředí, na kterých je testována. Za účelem ověření, jestli tato funkcionality nenarušila některou ze stávajících funkcionalit, musí být otestován celý systém. Jednotlivá prostředí využívána při vývoji jsou:

- Build, Trunk: Prostor, na kterých pracují zejména vývojáři. Ve výjimečných případech zde probíhá i proces testování.
- Test-demo: Ve většině případů se jedná o první prostředí, na kterém se začíná s testováním aplikace. Jakmile je aplikace s novou funkcionalitou na tomto prostředí otestována a všechny nalezené defekty opraveny, může být verze posunuta na další testovací prostředí.
- Live-demo: Testovací prostředí, na které je nová verze vydána po Test-demo. Je zde opakováno veškeré testování, které bylo provedeno na předchozím prostředí. Nutné je také přetestovat, jestli jsou vyřešeny všechny defekty nalezené na předchozím prostředí. Jedná se o poslední prostředí, na kterém je aplikace testována předtím, než je vydána na produkční prostředí.
- Live: Jedná se o produkční prostředí, na kterém by se už z principu neměly vyskytovat žádné defekty. Nicméně aplikaci je nutné opět přetestovat a ověřit, že se žádný defekt na tomto prostředí neprojevil a že nevznikl žádný další problém.
- Live-test: Na toto prostředí probíhá release současně s prostředím Live. Kromě dalšího testování probíhá na prostředí spouštění automatických testů namísto prostředí Live, kde spouštění není možné kvůli ověřování identity.

6.1.1 Testování nových funkcionalit

Dokumentace každé nové funkcionality je vedena v modulu správa úkolů Tasks aplikace Worklio. Jednotlivé úkoly jsou přidávány do struktury projektů a podprojektů. Mezi vlastnosti každého úkolu patří:

- název a popis úkolu,
- autor úkolu,
- přiřazená osoba + osoby na kopii,
- priorita úkolu,
- uzávěrka,
- status.

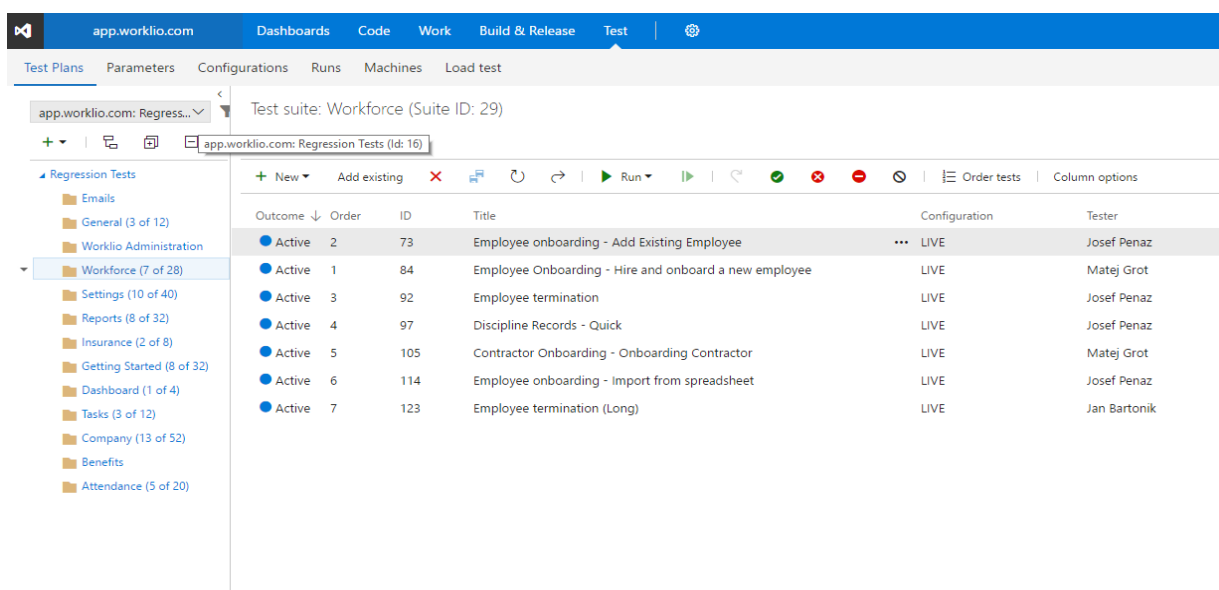
V úkolech je možné vést diskuzi a komentovat vložené příspěvky. K příspěvkům lze přikládat soubory, kterými bývají specifikace požadavků, popis chování nebo návody, jak danou funkcionalitu testovat. Správa úkolů funguje také jako výkaz práce a času stráveného na daném úkolu.

The screenshot displays the Worklio application interface. At the top, the 'worklio' logo is visible on the left, and the current time '20:14 (CET)' and date 'Wednesday, 15. March 2017' are on the right. Below the header, a navigation bar contains 'ME', 'CO', and a 'Tasks' button. The main content area shows a task titled 'Add conditional signing of ACH authorization document by EE after saving of bank account data'. The task details include a comment section with two comments from 'Řondík Ivo' and 'Křišťanová Jana', a list of participants including 'Nepomucký Radek', 'Pletenkov Dmitrij', 'Krause III Raymond', 'Křišťanová Jana', 'Repik Miroslav', and 'Řondík Ivo', and a list of attachments including 'Specification for testers.docx' and 'wireframes.png'.

Obrázek 14: Ukázka vedení dokumentace v modulu Task

6.1.2 Regresní testování

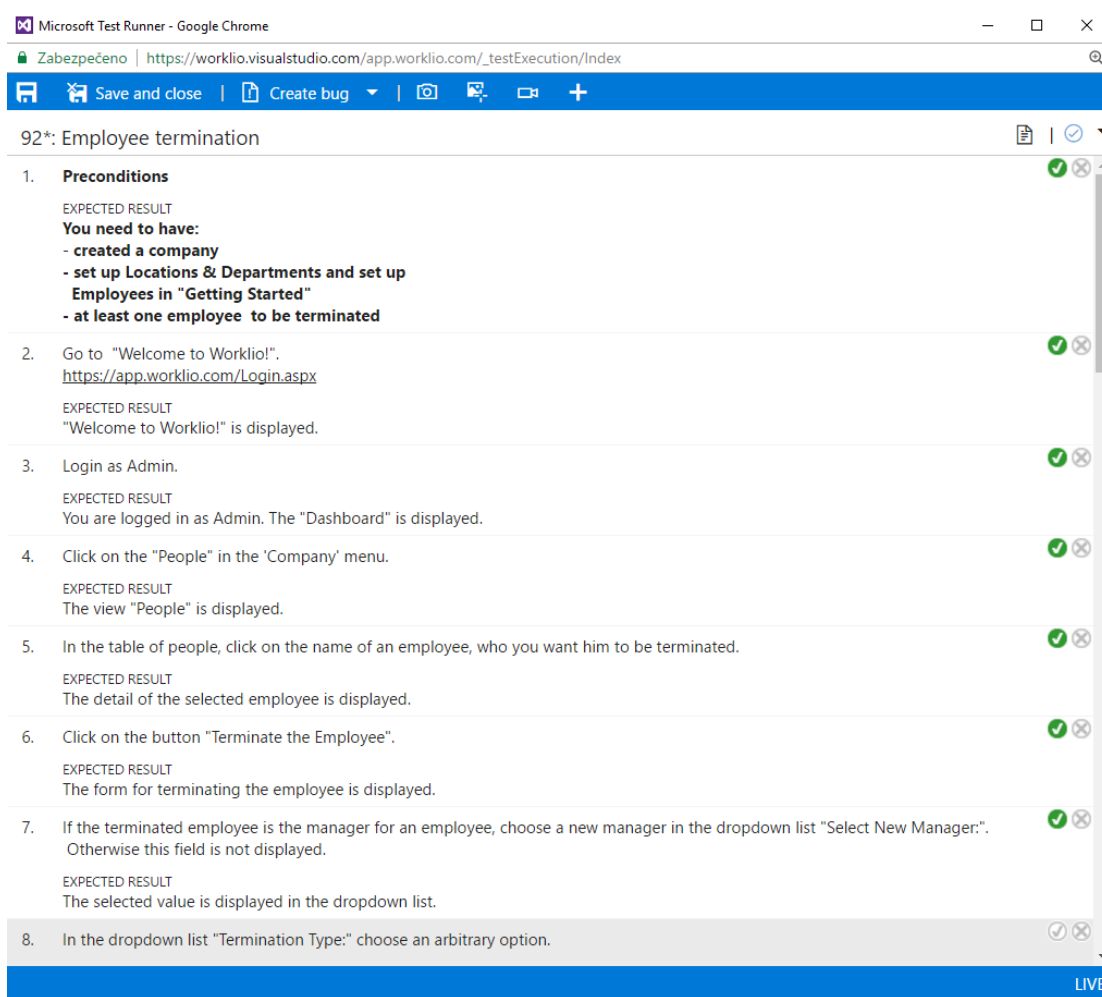
Regresní testování je prováděno především za použití testovacích případů. Testovací případy jsou napsány a prováděny ve webové aplikaci Visual Studio Team Services. Ty jsou podle jednotlivých modulů aplikace rozděleny do několika kategorií. Při regresním testování je pro každý release vytvořen nový testovací plán. Do plánu jsou přiřazeny jednotlivé testovací případy, které mají být v tomto release vykonány. Ke každému testovacímu případu je přiděleno prostředí, na kterém má být případ vykonán a tester, který má daný případ provést.



Outcome	Order	ID	Title	Configuration	Tester
Active	2	73	Employee onboarding - Add Existing Employee	LIVE	Josef Penaz
Active	1	84	Employee Onboarding - Hire and onboard a new employee	LIVE	Matej Grot
Active	3	92	Employee termination	LIVE	Josef Penaz
Active	4	97	Discipline Records - Quick	LIVE	Josef Penaz
Active	5	105	Contractor Onboarding - Onboarding Contractor	LIVE	Matej Grot
Active	6	114	Employee onboarding - Import from spreadsheet	LIVE	Josef Penaz
Active	7	123	Employee termination (Long)	LIVE	Jan Bartonik

Obrázek 15: Seznam testovacích případů

Testovací případy jsou psány jako posloupnost jednotlivých kroků. Každý krok obsahuje činnost, která má být provedena a očekávaný výsledek této činnosti. U každého kroku musí být jednoznačně rozhodnutelné, zda byl daný krok splněn či nikoliv. V prvním kroku každého testovacího případu jsou popsány předběžné podmínky a také doporučeno nastavení, aby bylo možné daný testovací případ dokončit. U kroků, které se v testovacích případech často opakují, je vhodné použít tzv. sdílený krok. Z testovacího případu, který selhal, je možné přímo vytvořit bug. Tím je testovací případ a vytvořený bug navzájem propojen.



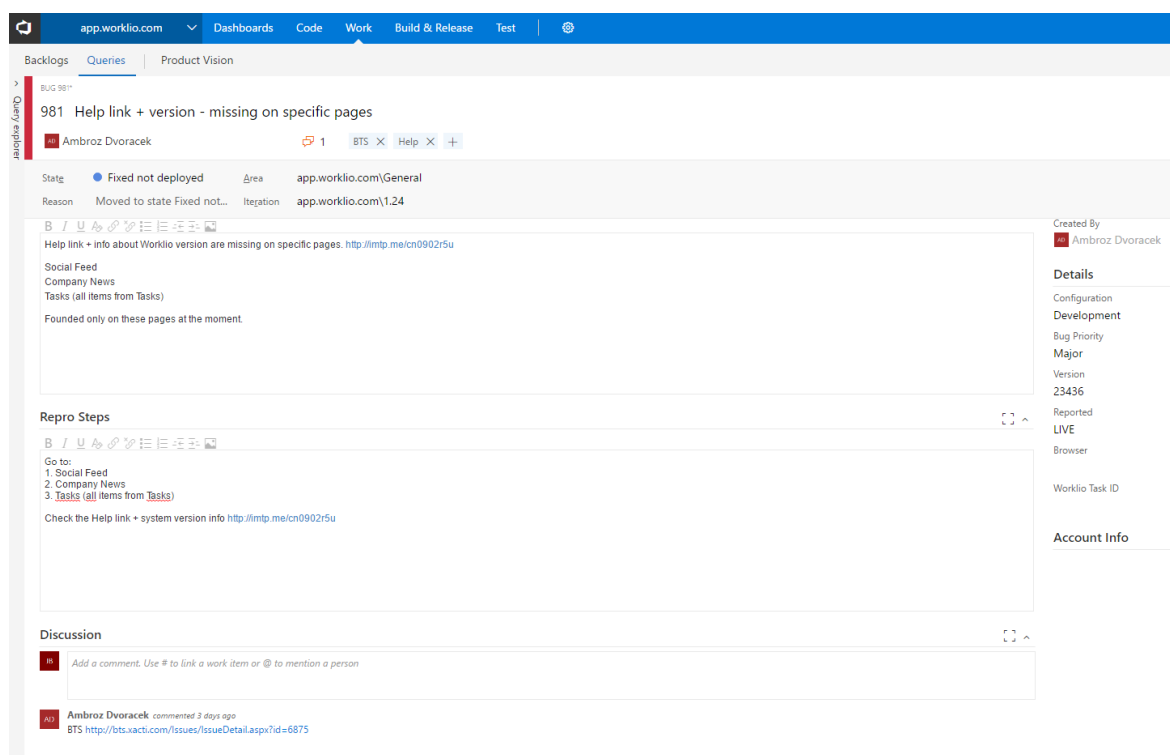
Obrázek 16: Ukázka spuštěného testovacího případu

6.1.3 Správa defektů

Defekty nalezené v testované aplikaci jsou zaznamenávány v systému Visual Studio Team Services (VSTS), jehož součástí je také modul pro správu defektů. Výhodou použití tohoto systému je naprostá variabilita a také přizpůsobení atributů a stavů defektu. Pro třídění a vyhledávání defektů v systému jsou využívány dotazy, které je možné ukládat jako vlastní nebo sdílené. Každý defekt obsahuje následující atributy:

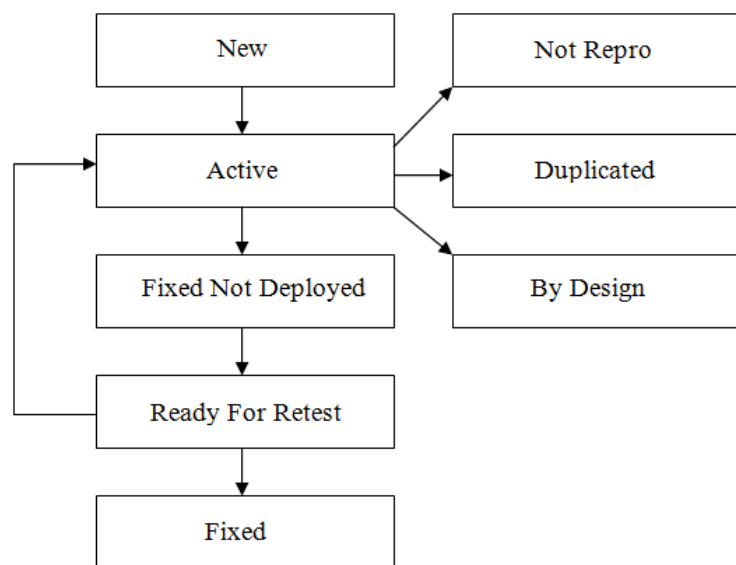
- Id: Identifikační číslo slouží pro jednoznačné určení defektu.
- Date: Date podává informaci o tom, kdy byl defekt reportován.
- Author: Author udává informaci o tom, kdo defekt reportoval.
- Assigned: Podává informaci o tom, na koho je defekt přiřazen.
- Description: Description je textové pole, ve kterém je defekt popsán.
- Repro Steps: V Repro Steps je popsán detailní postup k reprodukci defektu.

- State: State popisuje stav, ve kterém se defekt nachází.
- Area: Area podává informaci o tom, ve které části systému byl defekt nalezen.
- Bug Priority: Bug priority určuje závažnost defektu.
- Configuration: Configuration podává informaci, na kterém prostředí byl defekt nalezen.
- Browser: Browser specifikuje, na kterém prohlížeči byl defekt nalezen.
- Version: Version udává, v jaké verzi produktu byl defekt reportován.
- Worklio Task Id: Identifikátor úkolu v systému Worklio, ve kterém je defekt řešen.
- Tag: Slouží pro rychlé vyhledávání defektů.



Obrázek 17: Ukázka detailu defektu v systému VSTS

Jakmile je defekt nalezen a zaznamenán, začíná jeho životní cyklus. Ten je tvořen jednotlivými stavy, ve kterých se defekt nachází od jeho nalezení až po uzavření. V každém okamžiku je nutné vědět, v jakém stavu se všechny defekty nachází. Životní cyklus defektu používaný ve společnosti Worklio je zobrazen na obrázku níže.



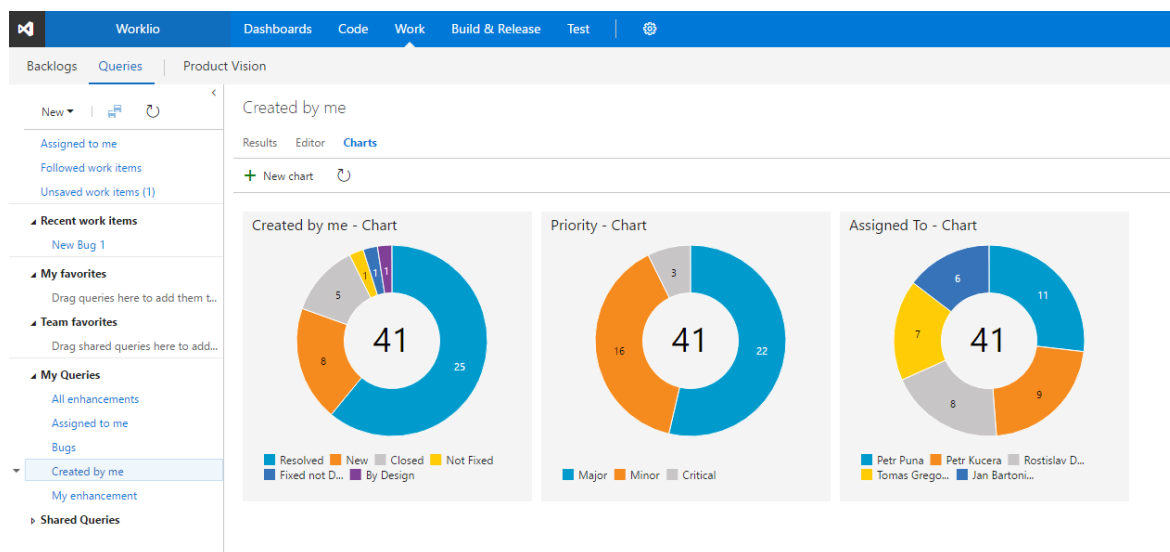
Obrázek 18: Životní cyklus defektu používaný ve společnosti

Stavy životního cyklu defektu od jeho vytvoření až po uzavření jsou:

- New: Při nalezení a zaznamenání defektu je nastaven stav na New.
- Active: Ve stavu Active probíhají opravy defektu.
- Not Repro: V případě, že defekt není možné zreprodukovat, dostává se záznam do stavu Not Repro.
- Duplicated: Stav, do kterého se defekt dostane, pokud je vytvořený defekt duplikát.
- By design: Stav, ve kterém se defekt nachází, pokud dané chování není považováno za defekt.
- Fixed Not deployed: Defekt v tomto stavu je opraven a čeká na vydání.
- Ready For Retest: Opravený defekt je připraven k přetestování.
- Fixed: Defekt je opraven.

Za zmínku také stojí funkcionality systému VSTS jakými jsou:

- propojování defektů, které spolu nějakým způsobem souvisí,
- historie defektů,
- grafické znázornění stavů, ve kterých se defekt nacházel,
- vizualizace defektů pomocí grafů podle různých kritérií.



Obrázek 19: Ukázka grafů s defekty podle různých kritérií

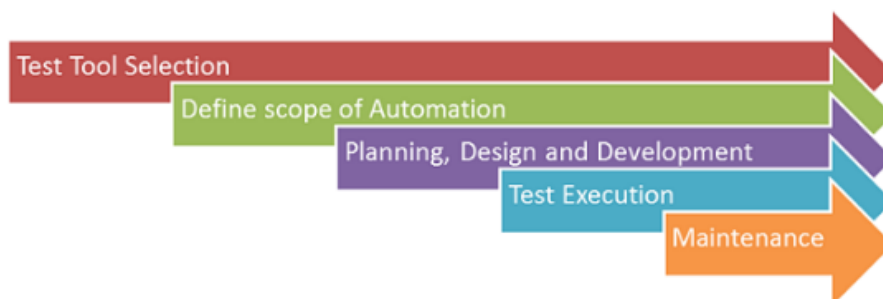
7 NÁVRH AUTOMATIZACE

Cílem této části práce je navrhnout a vytvořit sadu automatických testovacích případů uživatelského rozhraní webové aplikace Worklio. Od automatického testování je očekáváno:

- snížení počtu testovacích případů, které jsou vykonávány manuálně,
- zvýšení efektivity testování,
- úspora času z dlouhodobého hlediska.

Proces návrhu automatizace je složen z několika kroků, které na sebe navazují. Těmito kroky jsou:

- výběr testovacího nástroje,
- určení rozsahu automatizace,
- plán, návrh a vývoj,
- spouštění testů,
- údržba testů. [42]



Obrázek 20: Proces automatizace [42]

7.1 Výběr nástrojů

Jedno z prvních rozhodnutí při návrhu automatizace, na kterém záleží úspěšnost automatizace, je výběr správného nástroje. Pro automatizaci testů uživatelského rozhraní byl vybrán nástroj Selenium WebDriver, jelikož splňoval všechny vytyčené požadavky:

- použitý nástroj je open-source,
- pro psaní testovacích skriptů je možné použít jazyk C#,
- možné pořízení snímků z průběhu testů,
- při testování je používán reálný prohlížeč,

- rozšířenost, dobrá dokumentace a široká komunita,
- testování je možné provádět ve více prohlížečích.

Vedle testovacího nástroje Selenium WebDriver byl také použit jednotkový framework NUnit 3, který byl popsán v kapitole 3.1.3. Výhody jeho použití jsou:

- možnost nastavovat konfigurační hodnoty ze souboru default.runsettings,
- možnost parametrického testování,
- možnost paralelního spouštění testů,
- možnost spouštění testů přímo z Visual Studio,
- možná integrace se systémem Visual Studio Test Services.

7.2 Výběr testovacích případů

Důležitým krokem zavedení automatizace je výběr testovacích případů určených k automatizaci. Jelikož ne všechny testovací případy jsou vhodné k automatizaci, měly by se při výběru uvažovat určité aspekty. Případy vhodné k automatizaci by měly mít následující vlastnosti:

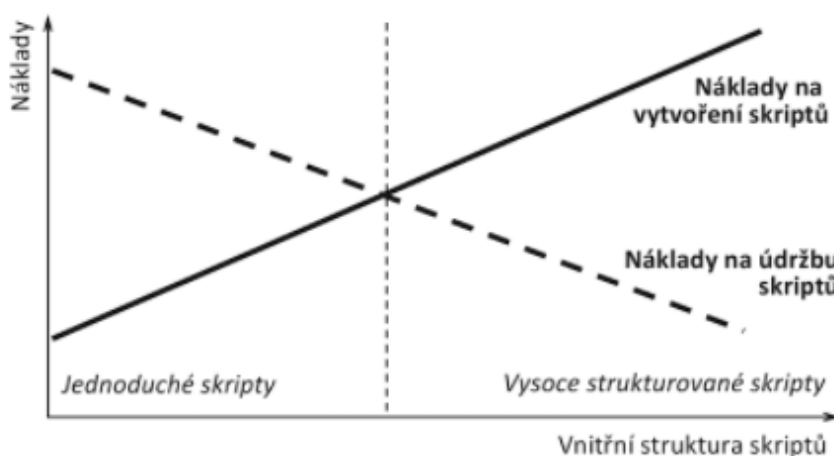
- Stabilita: Testy modulů, které se nemění. Nemá smysl automatizovat oblasti, které jsou teprve vyvíjeny nebo se často mění.
- Četnost provádění: Vhodné jsou testy, které se provádějí často a pravidelně.
- Obtížnost provádění: Testy, které je složité testovat manuálně, například z důvodů složitých výpočtů nebo velkého množství kroků.
- Důležitost: Automatizovat by se měly testy, jejichž funkcionality je zásadní.
- Časová náročnost: Vhodné je automatizovat testy, které sice nejsou vykonávány pravidelně, nicméně jejich vykonávání zabírá spoustu času.
- Složitost případné automatizace: Měly by být automatizovány případy, u kterých je automatizace proveditelná.

Na základě výše zmíněných vlastností byly k automatizaci vybrány regresní testy, které jsou vykonávány manuálně při každém release. V testech je kontrolována základní funkcionality a procesy všech modulů aplikace. Jelikož proces regresního testování zabírá testerům spoustu času, je vhodné jej automatizovat. Jedná se navíc o poměrně monotónní činnost, při kterém jsou opakovaně procházeny stejné testovací případy. Z tohoto důvodu může vzniknout i potenciální riziko přehlédnutí chyby, jelikož tester může při testování stejných případů ztrácet koncentraci. Výběr testovacích případů k automatizaci probíhal

na základě konzultace s produktovým a vývojovým oddělením. Tím byl zajištěn výběr testovacích případů v oblastech systému, které již nejsou v plánu v budoucnosti měnit.

7.3 Návrh projektu

Při návrhu projektu byl kladen důraz na vysokou strukturu a odolnost testů. Se zvyšujícím se počtem testovacích scénářů zákonitě přibývá i programového kódu. Volba správné struktury při vytváření testů proto ušetří spoustu času a práce spojené s údržbou testovacích případů v pozdějších fázích vývoje. Pojmem odolnost testů se rozumí šance na odolnost vůči změnám prováděných v systému. V ideálním případě je odolnost zajištěna výběrem testovacích případů v částech systému, které jsou stabilní (nejsou již nadále měněny). Nicméně pokud k úpravám automatizovaných částí systému dojde, je možné zvýšit odolnost testů správným způsobem vyhledávání elementů na stránce. Z toho důvodu je vhodné k vyhledávání elementů používat identifikátory, které se i při změně struktury stránky mění minimálně. Mezi takové identifikátory patří například Id nebo Name. Vhodné se ukázalo také využití identifikátoru XPath v relativním tvaru. Naopak velmi náchylné na změny v rozvržení stránek je využití identifikátoru CSS.



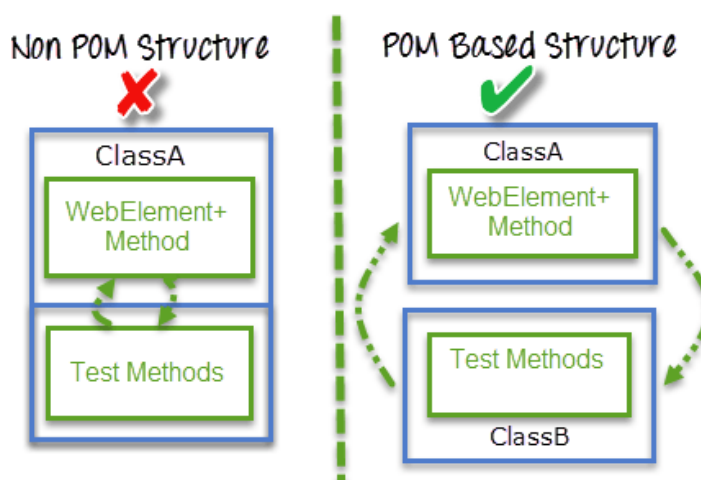
Obrázek 21: Náklady na údržbu v závislosti na strukturu [3]

7.3.1 Návrhový vzor Page Object Model

Pro vývoj testovacích scénářů byl použit návrhový vzor Page Object Model (POM). Využití tohoto návrhového vzoru je vhodné pro automatizaci komplexnějších projektů, jakým je například systém Worklio. V modelu POM je každá stránka definována třídou. Tyto třídy obsahují elementy, které se nacházejí na dané stránce a metody, které s těmito elementy provádějí různé akce, například vyplní pole textem nebo kliknou na tlačítko.

Díky modelu POM je zajištěno, že každý element je v projektu nadefinován právě v jedné třídě reprezentující danou stránku. Tím je zamezeno duplicitnímu použití lokátorů. Tyto elementy a metody jsou pak volány v testovacích scénářích. V případě, že v budoucnu dojde ke změně některého z elementů, stačí provést změnu vždy jen na jednom místě. Výhody tohoto vzoru jsou následující:

- lepší čitelnost kódu,
- lepší udržitelnost kódu,
- kód je znovupoužitelný,
- oddělení elementů a metod od testovacích scénářů. [40]



Obrázek 22: Srovnání struktury kódu bez a s modelem POM [40]

7.3.2 Page Factory

Jedním ze způsobů, jak vyhledávat elementy na webové stránce, je pomocí Page Factory. Elementy stránky jsou vyhledávány pomocí atributu `FindBy()`, který je přidán nad proměnnou zastupující daný element stránky. V závorkách jsou předávány parametry, kterými jsou způsob vyhledání elementu a konkrétní hledaný řetězec. Příklad použití je vidět na příkladu níže:

```
//Ddl Offender  
[FindBy(How = How.Id, Using = "ct100_ct100_cphContent_cphAppContent_ddlOffender")]  
2 references | Jan Bartonik, 43 days ago | 1 author, 1 change  
public IWebElement ddlOffender { get; set; }
```

Obrázek 23: Vyhledání dropdown listu pomocí lokátoru Id

7.3.3 Pořadí testů

Při spouštění testovacích případů je důležité vykonávání testů v určitém pořadí. Typickým případem je požadavek, aby první spouštěný test byl vytvoření nové společnosti nebo aby přidání nového zaměstnance bylo provedeno vždy před jeho smazáním. Toho je možné docílit použitím atributu Order v Nunit. V atributu je jako parametr předáno číslo typu integer, které určuje pořadí testu.

```
[Order(9)]
[TestCase("DepartmentAutomate")]
0 references | Jan Bartoník, 4 hours ago | 1 author, 7 changes
public void TestDepartment(string name)
{
    try
    {
        new TestDepartments(PropertiesCollection._driver).TestDepartment(name);
    }
    catch (Exception e)
    {
        TakePicture();
        FailedMessageWithUnit("Department Test", e);
    }
}
```

Obrázek 24: Ukázka použití atributu Order

7.3.4 Parametrické testování

Framework NUnit 3 umožňuje označení testovacích metod atributem TestCase. Pomocí tohoto parametru jsou testu předány vstupní hodnoty, se kterými má být test vykonán. Test je pak možné vykonat pro více vstupních sad. Na příkladu níže je zobrazena testovací metoda, která volá funkci pro přihlášení. Metoda je při spouštění testů provedena třikrát pro tři různé sady dat.

```
[Test]
[TestCase("bartonik@xacti.com", "1")]
[TestCase("bartonik200@lendea.com", "1")]
[TestCase("bartonik@lendea.com", "1")]
0 | 0 references | Jan Bartoník, 12 days ago | 1 author, 3 changes
public void TestLoginPOM(string email, string password)
{
    try
    {
        new LoginPage(PropertiesCollection._driver).Login(email, password);
    }
    catch (Exception e)
    {
        TakePicture();
        FailedMessageWithUnit("Login Test", e);
    }
}
```

Obrázek 25: Ukázka použití atributu TestCase

7.3.5 Kontrola dat

V testovacích případech, ve kterých jsou přidávána nebo editována data, probíhá také jejich kontrola. K této kontrole je využíváno funkcí třídy Assert frameworku NUnit. V podstatě se jedná o funkce, které srovnávají očekávaný a aktuální výstup dat. Jestliže se tyto hodnoty liší, test v tomto kroku selže a vypíše o vzniklém problému chybovou hlášku.

```
//Kontrola názvu firmy, časové zóny a emailu
Assert.AreEqual(basic.companyName, dd[0].Text);
Assert.AreEqual(basic.timeZone, dd[1].Text);
Assert.AreEqual(basic.companyEmail, dd[2].Text);
```

Obrázek 26: Ukázka použití kontroly dat

7.3.6 Nahrávání souborů do systému

Ve vytvořených testovacích případech jsou některé z prováděných kroků nahrání dokumentů nebo obrázků do systému. Pro provedení těchto kroků je nutné ovládání dialogových oken, což není možné přímo pomocí Selenium WebDriver. Problém byl vyřešen použitím volně stažitelného skriptovacího jazyka AutoIt. Postup nahrání souboru pomocí AutoIt je následovný:

- dialogové okno je aktivováno,
- je vyplněna cesta k nahrávanému souboru,
- je provedeno potvrzení dialogového okna.

```
//Funkce zajišťující nahrání dokumentu
4 references | Jan Bartonik, 48 days ago | 1 author, 1 change
public void UploadPDF(string nameOfFile)
{
    //Otevře dialogové okno
    btnChooseFile.Clicks();
    //Rozpozná a aktivuje dialogové okno
    var autoIt = HelperMethods.RecogniseBrowserByDialog(driver);
    //Do textového pole napíše cestu ke vkládanému souboru
    autoIt.Send(Path.Combine(Path.GetDirectoryName(nameOfFile)));
    //Potvrdí nahrání souboru
    autoIt.Send("{ENTER}");
}
```

Obrázek 27: Ukázka kódu zajišťujícího vkládání dokumentů

7.3.7 Testování s různými prohlížeči

Použití nástroje Selenium WebDriver umožňuje provádět testovací případy ve více internetových prohlížečích. V návrhu projektu bylo uvažováno vykonávání testů

v prohlížečích Google Chrome, Mozilla Firefox, Internet Explorer a Microsoft Edge. Výběr prohlížeče pro testování je nastavován v souboru default.runsettings. Část kódu, ve kterém je rozhodováno o výběru prohlížeče, je zobrazen na obrázku níže.

```
switch (PropertiesCollection.browser)
{
    case "firefox":
        PropertiesCollection._driver = new FirefoxDriver();
        break;

    case "chrome":
        PropertiesCollection._driver = new ChromeDriver();
        break;

    case "ie":
        InternetExplorerOptions options = new InternetExplorerOptions();
        options.IgnoreZoomLevel = true;
        options.PageLoadStrategy = InternetExplorerPageLoadStrategy.Eager;
        PropertiesCollection._driver = new InternetExplorerDriver(options);
        break;

    case "edge":
        EdgeOptions option = new EdgeOptions();
        option.PageLoadStrategy = EdgePageLoadStrategy.Eager;
        PropertiesCollection._driver = new EdgeDriver(path, option);
        break;

    default:
        PropertiesCollection._driver = new ChromeDriver();
        break;
}
```

Obrázek 28: Kód s výběrem prohlížeče k testování

7.3.8 Vytvoření chybových hlášení

V případě, že test selže, je o tomto selhání zobrazena chybová zpráva. Chybové hlášení obsahují informace, které mohou pomoci identifikovat příčiny pádu testu. Zprávy použité v projektu obsahují následující informace:

- název testu,
- název internetové stránky, na které se test při selhání nacházel,
- URL stránky, na které se test nacházel při selhání,
- administrátor testované společnosti,
- přihlášený uživatel (nemusí být vždy totožný s administrátorem),
- chybová hláška popisující vzniklou situaci.

Vhodné je také využití snímkování testů. Snímky z testů jsou pořizovány na základě nastavení parametru screen v souboru default.runsettings.

7.3.9 Využité třídy a metody

Součástí projektu jsou pomocné třídy, které jsou při vytváření testů uživatelského rozhraní používány. Mezi tyto třídy patří:

- **UITests:** Třída, ve které jsou nastavovány Set-Up, Tear-Down metody a jednotlivé automatizované testy.
- **HelperMethods:** Třída obsahuje metody, které jsou opakovaně používány napříč všemi testy. Jedná se například o ověření zobrazení elementu, vytvoření elektronického podpisu nebo nastavení dialogového okna pro vkládání dokumentů.
- **Menu:** Třída, která obsahuje elementy menu a další elementy, které se v systému opakují.
- **TableMethods:** Třída obsahuje metody sloužící pro obsluhu tabulek v systému.
- **SeleniumGetMethods:** Třída, která obsahuje metody pro získávání elementů stránek.
- **SeleniumSetMethods:** Třída, která obsahuje metody pro nastavování hodnot polí, zaškrťovacích polí a jiných elementů.
- **Screenshots:** Třída umožňuje vyhotovit snímky obrazovky během testu.

7.4 Vytvoření testů

Cílem vykonávání testů je ověření funkcionality základních procesů v systému a omezení přítomnosti kritických chyb. Kritickou chybou je myšlena taková chyba, která znemožňuje další práci v systému. Takovou chybou je například zhroucení systému nebo nekonzistenci zpracovávaných dat. Naopak smyslem testů není ověřit všechny kombinace vložených dat ve formulářích a jejich validace. Testy, které byly automatizovány, jsou popsány níže.

7.4.1 TestCreateCompany

V testovacím případě je ověřována funkcionality procesu vytvoření nové společnosti v aplikaci Worklio. Po dokončení tohoto procesu je pokračováno přidáním hlavní lokality a jednotlivých oddělení společnosti. V dalším kroku je vytvořen profil administrátora. Zde je na výběr přidat:

- interního administrátora – kromě administrátorské role disponuje rolí zaměstnaneckou,

- externího administrátora – bez zaměstnanecké role, slouží pouze k nastavování procesů ve vytvářené společnosti.

Jedná se o test, který je při automatických testech prováděn vždy jako první. Na jeho úspěšném dokončení závisí všechny následující testy.

7.4.2 TestLocations

V testu je kontrolována správa lokalit společnosti. Proces přidání nové lokality je složen ze dvou kroků. V prvním kroku je vyplněn název lokality, její správce a časová zóna. Na druhé stránce je pak vyplněna adresa lokality. Po dokončení vytvoření jsou vyplněná data kontrolována v tabulce všech lokalit. Test pokračuje editací nově vytvořené lokality, při které jsou procházeny stejné kroky jako v případě vytváření. Po editaci probíhá opět kontrola dat. Na závěr je lokalita smazána. V testu jsou dále kontrolovány textové zprávy vykonaných akcí a názvy stránek. Test je v současném stavu spouštěn se třemi vstupními sadami dat – s adresou pro společnost v USA, Kanadě a České Republice.

7.4.3 TestDepartments

V testovacím případě je kontrolována funkcionality týkající se modulu oddělení společnosti. V testu jsou postupně kontrovány procesy vytváření, editace, přiřazení zaměstnance do oddělení a smazání oddělení. Po každé z těchto akcí je na stránce seznamu všech oddělení provedena kontrola změněných dat. V testu jsou dále kontrolovány textové zprávy vykonaných akcí a názvy stránek.

7.4.4 TestAddExistingEmployee

Ve scénáři je testována funkcionality přidání nového zaměstnance. Jelikož je ve formuláři pro přidání nového zaměstnance možné nastavit velký počet vstupů, je tento test prováděn pro více sad vstupních dat. V první části formuláře jsou vyplněna data související s kontakty zaměstnance. V druhé části jsou vkládány informace týkající se pracovního poměru. Po dokončení a odeslání formuláře jsou data uložena. Nově vytvořený zaměstnanec je nalezen v tabulce zaměstnanců a je otevřen detail jeho profilu, kde jsou kontrolována všechna vložená data. U testu je možné nastavit, jestli má být provedeno dokončení registrace profilu ze strany vytvářeného zaměstnance.

7.4.5 TestTerminateEmployee

Testovací scénář je zaměřen na vykonání procesu propuštění zaměstnance a smazání jeho profilu. V tabulce všech aktivních uživatelů je vybrán libovolný zaměstnanec a následně je vyplněn formulář pro ukončení jeho pracovního poměru. Při prvním potvrzení formuláře je nastaveno datum ukončení pracovního poměru do budoucnosti, a je ověřeno, zda zaměstnanec zůstal v tabulce aktivních uživatelů. Následuje navigace zpět do formuláře terminace tohoto zaměstnance. Předchozí vložená data jsou zkontrolována a následně změněna – datum ukončení pracovního poměru je nastaveno do minulosti. Po potvrzení formuláře je zaměstnanec přesunut do tabulky propuštěných zaměstnanců. Zaměstnanec je v tabulce nalezen, po otevření detailu jeho profilu je provedena kontrola dat týkajících se propuštění. Detail profilu je následně smazán. V testovacím případě je ověřováno zobrazení všech zpráv o vykonání dané činnosti.

7.4.6 TestAddingFullAccessAdmin

V systému Worklio je možné měnit uživatelská práva z předem nadefinovaných rolí nebo práva pro přístup do modulů nastavovat jednotlivě. Testovací případ ověřuje funkcionality, kdy administrátor přidá uživateli s právy zaměstnance práva administrátorská a stává se z něj tudíž administrátor s plným přístupem do všech modulů. Po dokončení tohoto procesu proběhne přehlášení na uživatele se změněnými právy. Následně jsou postupně procházeny a kontrolovány všechny stránky, ke kterým by měl mít uživatel přístup. Na konci testu je uživatel vrácen zpět do zaměstnanecké role.

7.4.7 TestCompanyIdea

Cílem tohoto testovacího případu je průchod základního procesu v modulu Company Ideas. Modul umožňuje zaměstnancům společnosti navrhovat svoje myšlenky pro zlepšení chodu společnosti. Kroky testovacího případu jsou následující:

1. Přihlášení administrátorem a nastavení modulu Company Ideas v sekci Settings.
2. Vytvoření 4 nových nápadů na stránce Dashboard.
3. Změna stavů vytvořených nápadů – 2 zamítnuty, 2 poslány k hlasování.
4. Přehlášení libovolným zaměstnancem a hlasování – 1 podpořen, 1 nepodpořen.
5. Přehlášení zpět na roli administrátora a změna stavu nápadů, které byly určeny k hlasování – 1 akceptován, 1 zamítnut.

6. Kontrola stavů všech 4 nápadů na stránce Archived Ideas a jejich následné smazání.

7.4.8 TestDisciplineRecords

Testovací případ je zaměřen na procesy týkající se disciplinárních záznamů, které je možné zaměstnancům v systému přidávat. Případ je prováděn ve dvou verzích shodujících se v krocích vytvoření a editaci záznamu. V první verzi je test zakončen smazáním disciplinárního záznamu po jeho editaci. V druhé verzi po editaci záznamu následuje přehlášení a podpis ze strany uživatele, pro kterého byl záznam vytvořen.

7.4.9 Test515Report

Jeden z modulů systému jsou 5-15 reporty. Reporty slouží k sebehodnocení, zaměstnanci zde mají prostor popsat úspěchy, neúspěchy své práce nebo vyjádřit názory. Test vytváření reportu je prováděn ve dvou verzích pro týdenní a měsíční reporty. Všechna vložená data při vytváření reportů jsou na závěr testu ověřována a report je smazán.

7.4.10 TestAddContractor

Cílem testovacího případu je vytvoření nového kontraktora ve společnosti. Přidat kontraktora je možné s použitím šablon dokumentů. V případě, že nastavení šablon není provedeno, je toto vykonáno v rámci testu. Druhou možností je přidání kontraktora bez použití dokumentů. V obou případech je možné nastavit požadavek na vložení osobních informací kontraktora při dokončení profilu. Jakmile je kontraktor vytvořen, následuje fáze dokončení profilu ze strany nově přidaného kontraktora.

7.4.11 TestSetupPTO

Modul PTO je modul správy dovolené v systému. Proto, aby bylo možné tento modul používat, musí být nejprve provedeno jeho nastavení. Automatický test postupně projde a nastaví jednotlivé formuláře, kterými jsou:

- placené volno,
- nemocenská,
- jiné volno,
- přiřazení zaměstnanců do vytvářené PTO,
- nastavení kategorie a svátků.

Po dokončení PTO nastavení jsou data zkontrolována v tabulce a PTO je smazána.

7.4.12 TestCompanyInfo

V testovacím případě jsou kontrolovány informace týkající se společnosti. Postupně jsou procházeny a editovány data ve všech oddílech společnosti. Mezi jednotlivé oddíly patří:

- základní informace,
- adresa společnosti,
- bankovní informace,
- informace o daních,
- kontakty.

Jakmile je editace všech stránek dokončená, je provedena kontrola všech změněných dat.

7.5 Spouštění testů

Dalším z důležitých kroků automatizace je samotné spouštění vytvořených testů. Testy mohou být prováděny s různými vstupními parametry. Ty jsou nastavovány v souboru default.runsettings.

```
<RunSettings>
  <TestRunParameters>
    <Parameter name="browser" value="edge" />
    <Parameter name="envi" value="demo" />
    <Parameter name="login" value="bartonik10000@lendea.test" />
    <Parameter name="password" value="1" />
    <Parameter name="screen" value="no" />
  </TestRunParameters>
</RunSettings>
```

Obrázek 29: Nastavení souboru default.runsettings

Parametr pro nastavení přihlašovacího emailu je tvořen příjmením přihlašovaného uživatele a libovolným číslem. Doména .test slouží k vypnutí emailové kampaně během provádění testů. Přihlašovací email by měl být pro každý běh testů unikátní. Jako parametr pro heslo se z důvodu jednoduchosti používá číslo 1. Možnosti nastavení dalších parametrů, kterými jsou nastavení prohlížeče, dále prostředí a snímání testů, je popsáno v tabulkách níže:

Tabulka 6: Možnosti nastavení parametru browser

Parametr	Popis
chrome	Spouštění testů v prohlížeči Google Chrome.
firefox	Spouštění testů v prohlížeči Mozilla Firefox.
edge	Spouštění testů v prohlížeči Microsoft Edge.
ie	Spouštění testů v prohlížeči Internet Explorer.

Tabulka 7: Možnosti nastavení parametru envi

Parametr	Popis
demo	Spouštění testů na prostředí demo.
livedemo	Spouštění testů na prostředí livedemo.
livetest	Spouštění testů na prostředí livetest.

Tabulka 8: Možnosti nastavení parametru screen

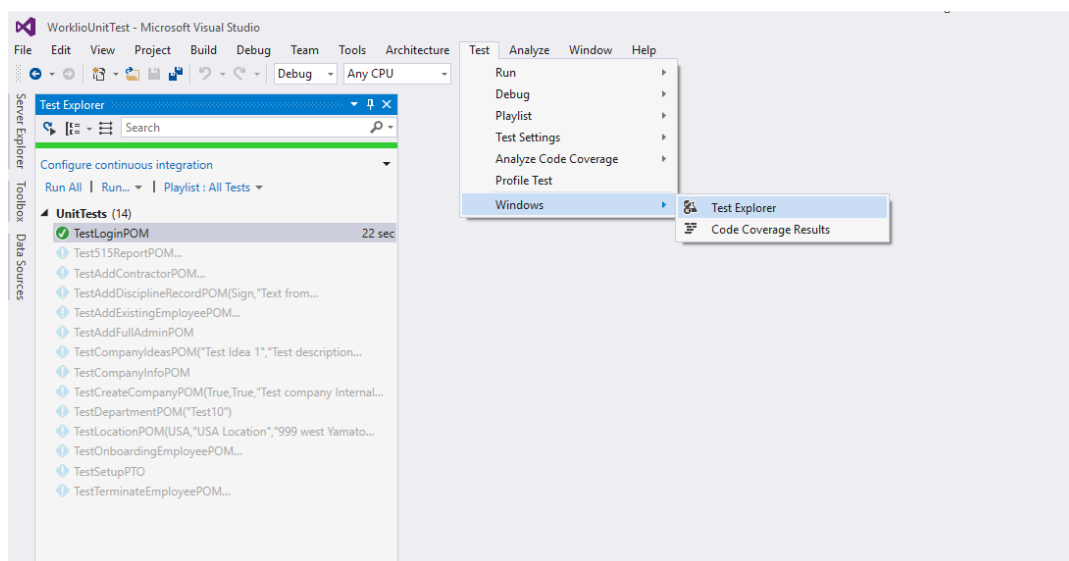
Parametr	Popis
no	Při nastavení nejsou při průchodu testů pořizovány žádné snímky.
aftererror	Při nastavení je snímek vyfocen pouze v případě pádu testu.
afterclick	Nastavení, při kterém jsou pořizovány snímky po každém kliknutí.

Vytvořené testovací případy je možné spouštět pomocí:

- průzkumníku testů ve Visual Studio,
- Visual Studio Team Services.

7.5.1 Visual Studio

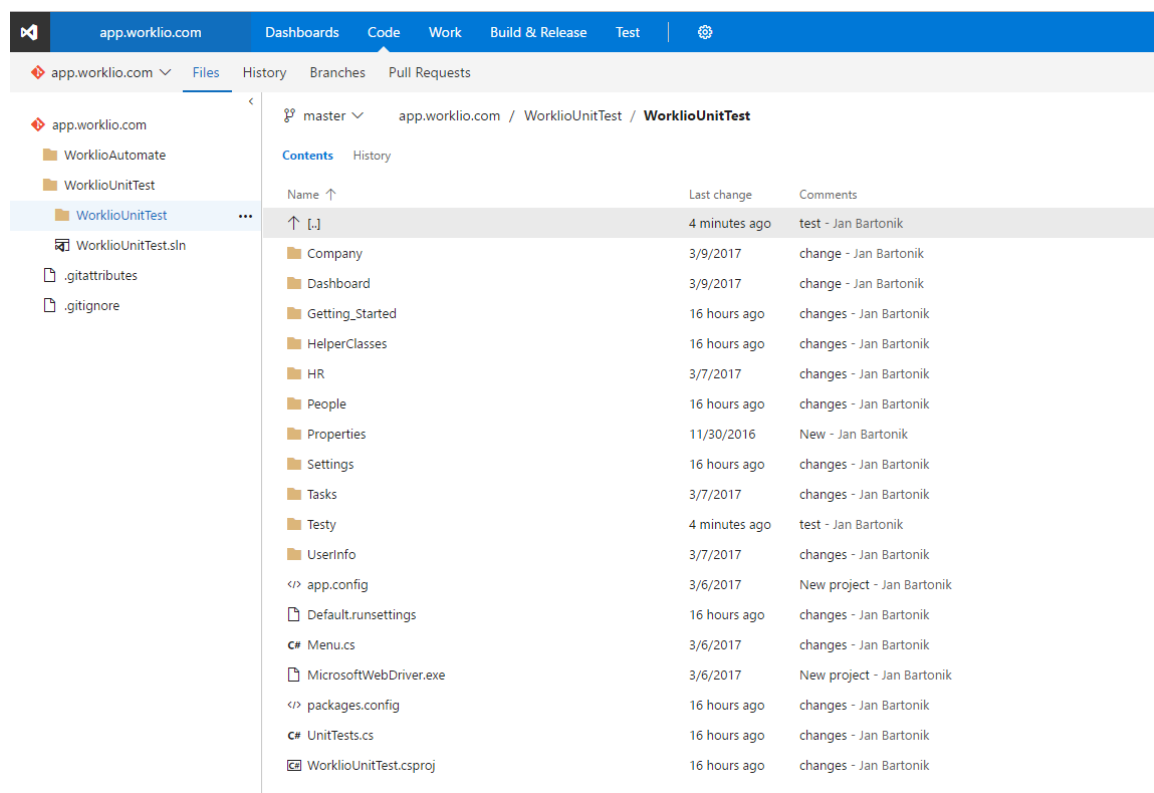
Jedním ze způsobů, jak spouštět testy, je přímo z programu Visual Studio pomocí průzkumníku testů. Aby bylo vykonávání testů v průzkumníku možné, musí být provedena instalace vybraného testovacího frameworku a adaptéru v závislosti na vybraném frameworku. Při návrhu tohoto projektu byl vybrán framework NUnit 3 s testovacím adaptérem NUnit3TestAdapter. Jedná se o rychlý způsob, jak vytvořené testy spouštět. Proto byla tato možnost použita při vývoji nových testů a při ladění testů, které při provádění selhaly.



Obrázek 30: Průzkumník testů v programu Visual Studio 2015

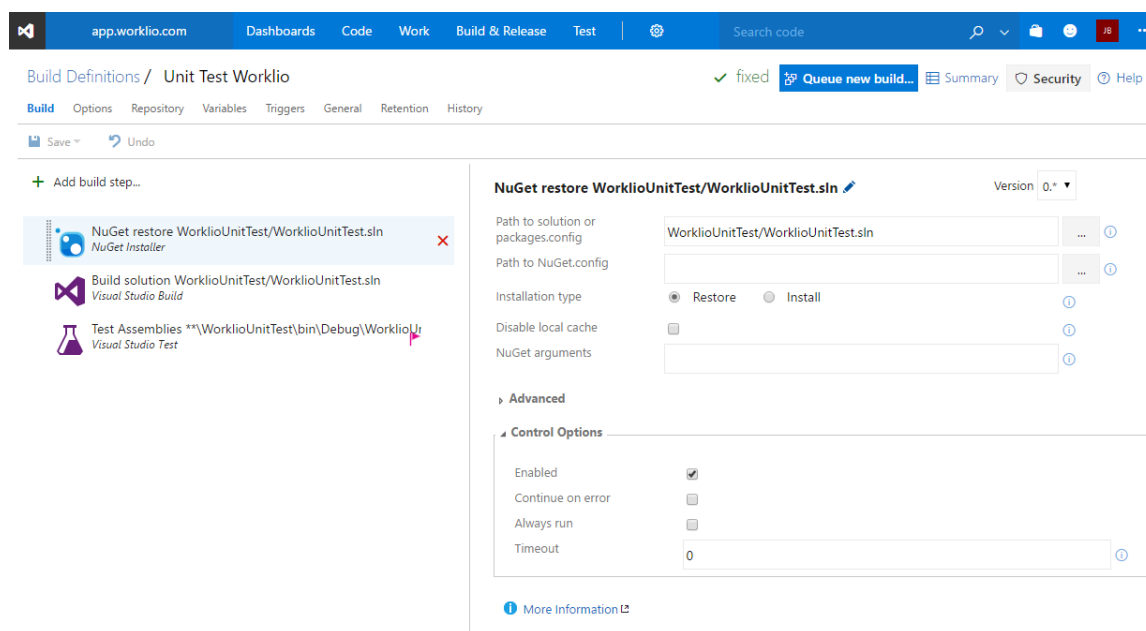
7.5.2 Visual Studio Team Services

Automatické spouštění testů je prováděno pomocí nástroje Visual Studio Team Services. Zdrojové kódy automatických testů jsou nahrány do systému pro verzování Git na server VSTS do projektu app.worklio.com.



Obrázek 31: Projekt automatických testů ve VSTS

Před prováděním sestavení a vykonáváním testů musí být na zařízení, na kterém mají úkoly běžet nainstalován tzv. agent. Agent je software, který slouží ke komunikaci s testovacím kontrolérem. Všechny zprávy mezi agentem a VSTS probíhají pomocí protokolu http nebo https v závislosti na tom, jak je agent nakonfigurován. Na jednom agentovi může probíhat v jeden okamžik právě jeden úkol (sestavení, test atd.). V systému VSTS je možné vybrat mezi hostovaným nebo soukromým agentem. Výhodou použití hostovaného agenta je v tom, že není potřeba žádná údržba a upgrade softwaru. Naopak privátní agent nabízí větší kontrolu nad instalací softwaru, který je potřebný pro sestavení a testování. Agent může být nainstalován na zařízení s operačním systémem Windows, Linux nebo OSX. [41]



Obrázek 32: Nastavení sestavovací definice v systému VSTS

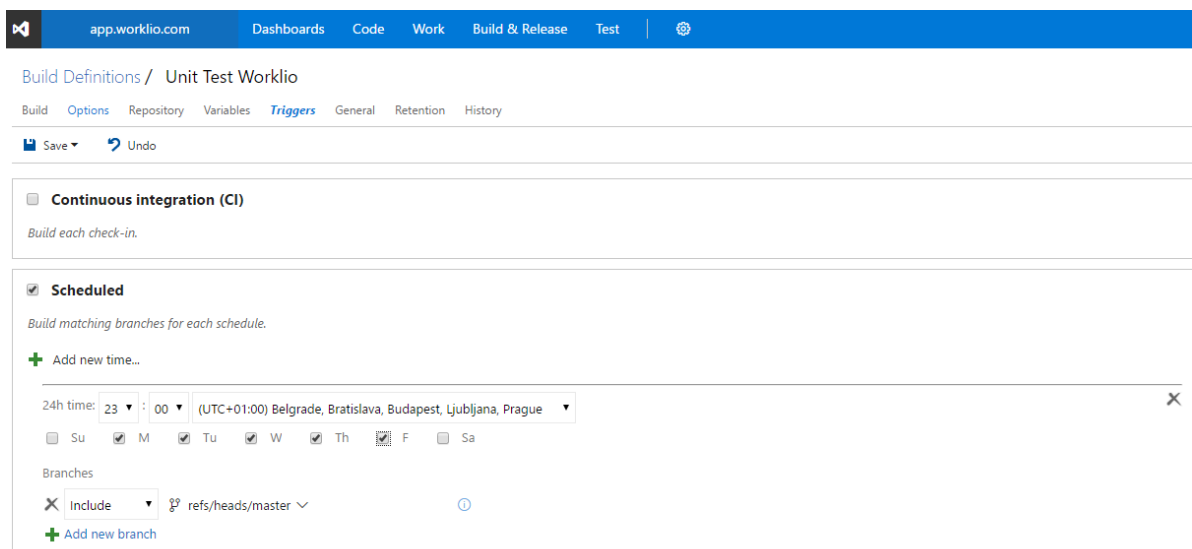
Testovací případy popsané v této práci jsou spouštěny na firemním serveru s privátním agentem. Dalším krokem je nastavení sestavovací definice, jejíž součástí je také vykonávání automatických testů. Ta je možná nastavit v menu v položce s názvem Build & Release. V záložce build jsou nastavovány všechny kroky, které jsou po spuštění sestavovací definice postupně vykonávány. Kroky, které byly pro spuštění testů nastaveny, jsou:

1. NuGet restore: Před sestavením projektu je provedena instalace balíčků, na které je v projektu odkazováno. Balíčky jsou definovány v souboru packages.config. Do parametru Path to solution or packages.config byla přidána cesta k souboru s příponou .sln v projektu.

2. Build solution: Druhým krokem je sestavení projektu. Do parametru solution byla přidána cesta ke stejnému souboru jako v předchozím kroku.
3. Test assemblies: Třetím krokem definice je provedení testovacích případů v projektu. V parametru Test Assembly je předána cesta k souboru WorklioUnitTest.dll.

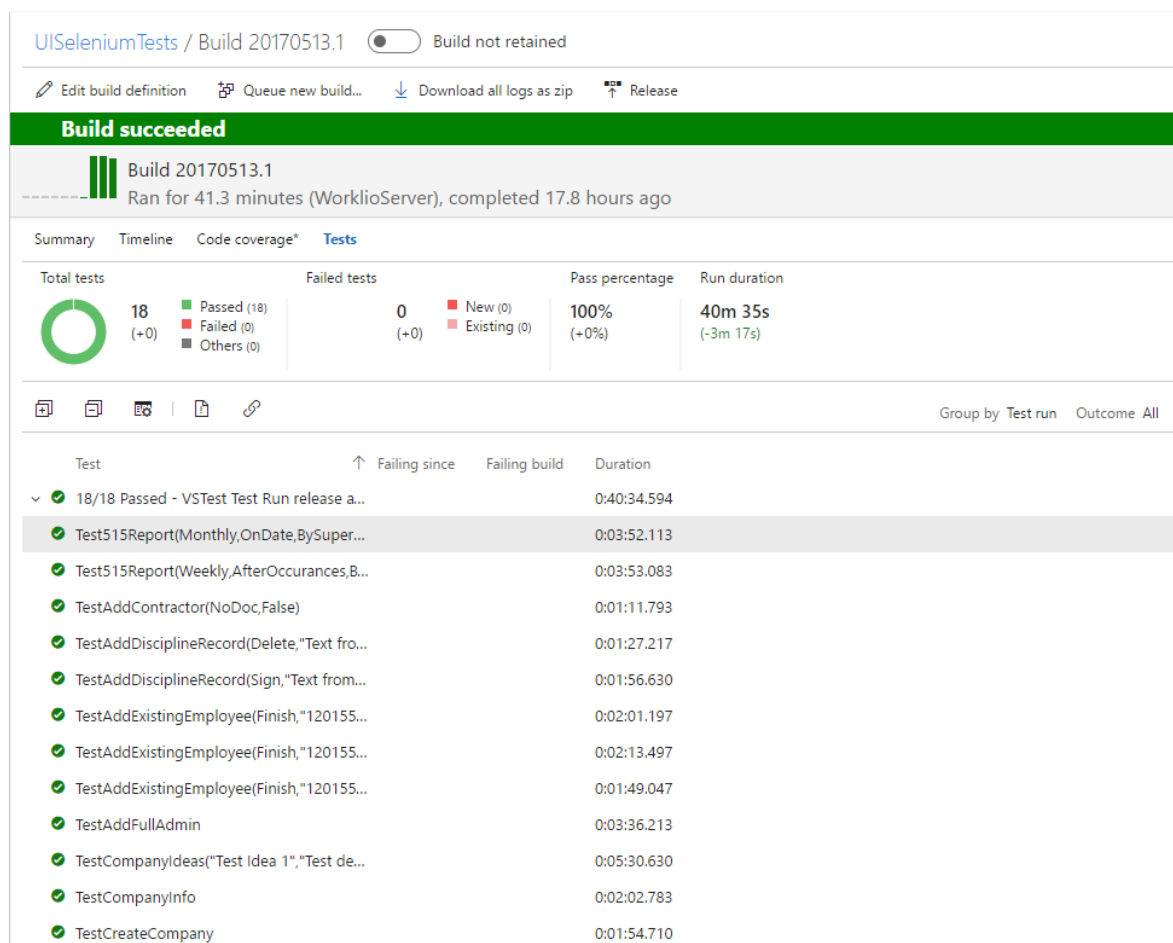
Testovací případy na firemním serveru jsou v současném stavu spouštěny pouze na prohlížeči Google Chrome. V případě požadavku firmy může být spouštění testů rozšířeno o další prohlížeče, se kterými bylo v návrhu počítáno. Automatické spouštění je možné nastavit v záložce Triggers ve vytvořené definici. Zde je možné nastavit spouštění dvěma způsoby:

- Průběžná integrace – možnost spouštění testů při jakékoliv změně zdrojového kódu.
- Plánované spouštění – umožňuje nastavit čas a dny v týdnu, ve kterých má být testování prováděno.



Obrázek 33: Nastavení spouští automatického spouštění testů

Posledním krokem je analýza výsledků provedených testů. Po otevření záznamu o provedeném sestavení je zobrazen přehled o vykonaných činnostech. V horní části je zobrazen souhrn informací jako je počet vykonaných testů, počet testů, které selhaly, procento úspěšnosti a doba trvání. Pod těmito informacemi je uveden seznam všech testů. Kliknutím na jednotlivé testy je možné zobrazit detail daného testu, popřípadě zprávu chyby, která při vykonávání nastala. U všech testů, jenž selžou, je nutné provést analýzu jejich selhání.



Obrázek 34: Zobrazení výsledků automatických testů ve VSTS

7.6 Údržba automatických testů

Při nasazení automatických testů do procesu testování musí být počítáno s tím, že vytvořené skripty bude nutné udržívat. Proto byla automatizace navržena tak, aby údržba zabírala co nejméně práce a času. Nicméně údržba je nutná minimálně v případě, když je provedena změna v testované oblasti systému, která testy naruší. K eliminaci tohoto problému byl při návrhu automatizace kladen důraz na správný výběr testovacích případů. Redukovat čas vynaložený na opravu testu pomáhá také zvolený návrhový vzor Page Object Model, který definuje elementy stránky na jednom místě. V poslední řadě závisí odolnost testů na správně zvoleném lokátoru při vyhledávání elementů. Údržba je také nutná v případě, když automatický test selže. Pokud takhle situace nastane, je nutné vyšetřit příčiny selhání, což může být časově náročná činnost. Při této činnosti je test, který selhal spouštěn manuálně, přičemž musí být pozorováno, ve kterém kroku chyba nastala. Selhání testu může být způsobeno ze dvou příčin. První příčinou je chyba zanesená v testovacím scénáři. V tomto případě musí být testovací scénář opraven

tak, aby se chyba v příštím běhu neopakovala. Druhou příčinou pádu testu je chyba v testovaném systému. Při tomto závěru musí být o této chybě vytvořen záznam.

8 VYHODNOCENÍ PROVEDENÉ AUTOMATIZACE

Návrh automatizace je složen z několika kroků, kterými jsou výběr testovacího nástroje, výběr testovacích případů, návrh projektu, vytvoření testovacích případů, jejich spouštění a údržba. Pro automatizaci byly vybrány regresní testy, jejichž provádění je pravidelně opakováno při každém release. Jedná se tím pádem o testy, jejichž vykonávání zabírá týmu spoustu času. Jako nástroje použité pro automatizaci byly zvoleny Selenium WebDriver a Nunit 3. Díky jejich použití je možné:

- psát testy v jazyce C#,
- provádět testy s různými prohlížeči,
- testovat s reálnými prohlížeči,
- použití testovacích parametrů,
- spouštět testy paralelně,
- nastavovat pořadí testů.

Jako další nástroj pro ovládání dialogových oken při vkládání dokumentů do systému byl využit skriptovací jazyk AutoIt. Při návrhu testů byl použit návrhový vzor Page Object Model. Model definuje jednotlivé webové stránky jako třídy. Díky tomuto přístupu je zajištěno, že každý element stránky je definován právě jednou v dané třídě. Použitím modelu POM je projekt přehlednější a jednodušší je také jeho údržba. Spouštět testy je možné manuálně nebo automaticky pomocí služby Visual Studio Team Services. Testy, které selhaly, je nutné ověřovat ještě manuálně. Časově poměrně náročným úkolem se ukázalo psát testovací případy s kompatibilitou pro více různých prohlížečů. A to z důvodu, že každý prohlížeč má jiné chování, jednotlivé kroky provádí jinak rychle a pro lokalizaci elementů stránky musí být vybírány metody, které jsou podporovány všemi prohlížeči. Z těchto důvodů bude do budoucna určité důležité rozhodnutí, jestli je potřeba psát skripty a testovat se všemi zmíněnými prohlížeči nebo bude zvolen nějaký kompromis, například testovat pouze pro dva nebo jen jeden prohlížeč. Během vývoje testů také nastala situace, kdy byl kompletně změněn design hlavního menu testované aplikace, kvůli kterému bylo nutné již všechny napsané testy přepracovat. Nehledě na fakt, že této změně nešlo předejít, se pouze potvrdilo, jak důležitým krokem je správný výběr testovacích případů.

8.1 Omezení provádění testů

Komplikací se do jisté míry ukázalo být použití reálných internetových prohlížečů. Proto, aby bylo možné testy provádět, je nutné udržovat verze prohlížečů kompatibilní s ovladači použitými v projektu. Jedním z důležitých kroků je vypnutí automatických aktualizací použitých internetových prohlížečů. Tím je zamezeno možnosti nefunkčnosti testů tím, že by aktualizovaná verze prohlížeče nebyla s ovladačem Selenia kompatibilní. Automatické testy je možné spouštět v prohlížečích Google Chrome, Mozilla Firefox, Microsoft Edge a Internet Explorer a to ve verzích, které jsou uvedeny v tabulce níže.

Tabulka 9: Verze použitých prohlížečů

Prohlížeč	Verze
Google Chrome	57.0.2987.133
Mozilla Firefox	52.0.2
Microsoft Edge	38.14393.1066.0
Internet Explorer	11.1066.14393.0

Vykonávání testů s sebou nese některá další omezení, která se vztahují k typu použitého internetového prohlížeče a jeho ovladače. Tyto omezení se týkají prohlížečů Mozilla Firefox, Microsoft Edge a Internet Explorer.

8.1.1 Mozilla Firefox

Ovladač v současném stavu neumožňuje použití metod Selenia z třídy Actions, které byly použity pro vytváření elektronického podpisu při popisování dokumentů v některých testovacích případech. Proto tyto testovací případy nemohou být v prohlížeči Mozilla Firefox provedeny. Jedná se o testy:

- TestAddDisciplineRecord,
- TestAddContractor.

8.1.2 Microsoft Edge

Při použití prohlížeče Microsoft Edge nefunguje vkládání souborů do systému. Po otevření dialogového okna v prohlížeči Edge test zamrzne a posléze selže. Bohužel nebylo nalezeno žádné jiné řešení, jak se s tímto problémem vypořádat. Proto jsou v současnosti testovací

případy v tomto prohlížeči prováděny bez nahrávání jakýchkoliv dokumentů. Jedná se o testovací případy:

- TestCompanyInfo,
- TestTerminateEmployees.

8.1.3 Internet Explorer

Provádění automatických testů v prohlížeči Internet Explorer je výrazně pomalejší než v ostatních prohlížečích. Dalším omezením je, že testy v tomto prohlížeči není možné vykonávat na prostředí "demo" z důvodu použití protokolu http namísto https, jako tomu je u zbylých dvou prostředích. Řešení, se kterým je počítáno do budoucna, je sjednocení komunikace pomocí protokolu https na všech prostředích.

8.2 Ekonomické zhodnocení

Při zavádění automatizace testování je nutné počítat s vyššími počátečními náklady. A to minimálně z důvodu času investovaného do návrhu a vývoje testovacích případů. Nicméně z dlouhodobého hlediska jsou tyto náklady vráceny. Návrh investice zavedení automatizace se liší projekt od projektu. Mezi další faktory, které souvisí s návratností investice spojené s automatizací, patří:

- cena použité technologie a nástrojů,
- velikost testovacího týmu a jeho cena,
- počet plánovaných release,
- testovací parametry – počet konfigurací, testovacích případů,
- rozsah automatizace.

Obecně platí fakt, že se návratnost nachází v rozmezí 2-4 let. [52]

8.3 Další vývoj

Budoucí vývoj automatických testů spočívá především v automatizaci dalších testovacích případů. Cílem je pokrytí systému maximálním počtem automatických testů. S jejich navyšujícím se množstvím bude pravděpodobně řešena možnost jejich paralelního spouštění. Jedno z řešení nabízí přímo framework NUnit změnou struktury třídy obsahující testovací případy a použitím parametru Parallelizable. Testovací metody, které mají být prováděny paralelně, musí být obsaženy v různých třídách. Tyto třídy jsou následně

označeny atributem `Parallelizable`. Příklad takové struktury frameworku NUnit je zobrazen na obrázku níže.

```
using NUnit.Framework;

namespace WorklioUnitTest
{
    [Parallelizable]
    [TestFixture]
    0 references | 0 changes | 0 authors, 0 changes
    public class TestsParalel1
    {
        [Test]
        0 references | 0 changes | 0 authors, 0 changes
        public void Test1Par()
        {
        }
    }

    [Parallelizable]
    [TestFixture]
    0 references | 0 changes | 0 authors, 0 changes
    public class TestsParalel2
    {
        [Test]
        0 references | 0 changes | 0 authors, 0 changes
        public void Test2Par()
        {
        }
    }
}
```

Obrázek 35: Příklad paralelního spouštění

ZÁVĚR

Práce se zabývá automatickým testováním softwaru. Hlavním cílem byl návrh a vytvoření sady automatických testů uživatelského rozhraní webové aplikace Worklio. V teoretické části jsou popsány základy testování softwaru a jeho rozdělení podle různých kritérií. Další část je věnována automatickému testování. Zde jsou popsány vlastnosti, výhody a mýty automatického testování. Následně jsou uvedeny oblasti testování, ve kterých může být automatizace aplikována. Ke každé z těchto oblastí je vypracován přehled nástrojů, jenž jsou k automatizaci v praxi používány. V další kapitole jsou představeny nástroje z rodiny Selenium. Popsán je především Selenium Webdriver, který byl využit při vytváření sady testovacích případů.

V praktické části práce je představena webová aplikace Worklio, způsoby jejího testování a proces vedení defektů nalezených při testování této aplikace. Další kapitola je věnována návrhu automatických testů. Ten je složen z několika kroků, kterými jsou výběr nástrojů a testovacích případů, návrh projektu, vytvoření testovacích případů, možnosti spouštění automatických testů a jejich údržba. Na závěr je provedeno zhodnocení návrhu automatizace. Pro vývoj testovacích případů byly vybrány open-source nástroje Selenium WebDriver a jednotkový framework Nunit 3. Jako programovací jazyk byl použit C#. Automatické spouštění bylo vyřešeno integrací s cloudovým nástrojem Visual Studio Team Services. Ten umožňuje sestavování projektu a provedení vytvořených testů na privátním serveru.

Přínosem práce je vytvořená sada automatických testů uživatelského rozhraní. Při jejich zapojení je testování aplikace efektivnější. Namísto manuálního provádění testů, které byly automatizovány, se testéři mohou věnovat jiným činnostem testování. Jedná se například o manuální testování nových funkcionalit, psaní testovacích případů nebo jejich následná automatizace. Dalším rozšířením práce by mohlo být vyšší pokrytí automatickými testy. Další možností je zapojení automatizace do jiných oblastí testování. Uvažováno může být například testování výkonu aplikace.

ZÁVĚR V ANGLIČTINĚ

The thesis deals with automated software testing. The main goal was to design and create a set of user interface automated tests for the Worklio web application. The theoretical part describes the basics of the software testing and its distribution according to different criteria. The next part is dedicated to automated testing. Features, advantages, and the myths of automated testing are described. Next, the areas of testing where automation can be applied are mentioned. For each of these areas, there is an overview of the tools that are used for automation in practice. In another chapter, the tools of Selenium framework are introduced. Selenium WebDriver is also described; it was used to design and create a set of test cases.

In the practical part of the work, there is an introduction of the Worklio web application, the methods of its testing, and the process of tracking defects when they are found. Another chapter is devoted to the design of the automated tests. This part consists of a few steps: the selection of tools and test cases, the design of the project and the creation of test cases, and, finally, the possibilities for running automated tests and their maintenance. In the last chapter, an evaluation of the designed automation is performed. For the development of the tests, open-source tools Selenium WebDriver and unit framework NUnit 3 were used. C# was selected as the programming language. The automatic running of the tests was resolved by integration with the cloud tool Visual Studio Team Services, which allows for the building of projects and runs tests on a private server.

The benefit of this work is to create a set of automatic user interface tests. By using them, application testing is more effective. Instead of executing manual tests that are automated, testers can do other activities for the testing. These include, for example, the manual testing of new functionalities, writing manual test cases, and planning their subsequent automation. The next extension of the work could be increased coverage for the testing by the automated tests. Another possibility is automation in other areas of testing. Performance testing may be considered.

SEZNAM POUŽITÉ LITERATURY

- [1] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality software: průvodce testováním*. Brno: Computer Press, 2013. ISBN 9788025138168.
- [2] Patton, R. *Testování software*. Computer Press, 2002, ISBN 80-7226-636-5
- [3] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování software: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
- [4] *What is Software Testing Life Cycle?* [online]. [cit. 2017-03-31]. Dostupné z: <http://istqbexamcertification.com/what-is-software-testing-life-cycle-stlc/>
- [5] *Software Testing - Documentation* [online]. [cit. 2017-03-31]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_documentation.htm
- [6] *Testing Methodologies* [online]. 2013 [cit. 2017-03-30]. Dostupné z: <https://tutorialshouse1.wordpress.com/2013/06/26/testing-methodologies/>
- [7] *Static Testing Vs Dynamic Testing* [online]. [cit. 2017-03-30]. Dostupné z: <http://www.guru99.com/static-dynamic-testing.html>
- [8] *What is Unit testing?* [online]. [cit. 2017-03-30]. Dostupné z: <http://istqbexamcertification.com/what-is-unit-testing/>
- [9] *Integration Testing* [online]. [cit. 2017-03-30]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/integration_testing.htm
- [10] *What is System testing?* [online]. [cit. 2017-03-30]. Dostupné z: <http://istqbexamcertification.com/what-is-system-testing/>
- [11] *What is Acceptance testing or User Acceptance Testing (UAT)?* [online]. [cit. 2017-03-30]. Dostupné z: <http://istqbexamcertification.com/what-is-acceptance-testing/>
- [12] *What is User Acceptance Testing (UAT)?* [online]. [cit. 2017-03-30]. Dostupné z: <http://www.guru99.com/user-acceptance-testing.html>
- [13] *Best Difference between Alpha and Beta Testing* [online]. [cit. 2017-03-30]. Dostupné z: <http://testingbasicinterviewquestions.blogspot.cz/2015/03/best-difference-between-alpha-and-beta.html>
- [14] *What is Regression testing in software?* [online]. [cit. 2017-03-30]. Dostupné z: <http://istqbexamcertification.com/what-is-regression-testing-in-software/>
- [15] *Software Testing - Levels* [online]. [cit. 2017-03-31]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm

- [16] *What is Non-functional testing* [online]. [cit. 2017-03-30]. Dostupné z: <http://istqbexamcertification.com/what-is-non-functional-testing-testing-of-software-product-characteristics/>
- [17] *Differences Between Black Box Testing and White Box Testing* [online]. [cit. 2017-03-30]. Dostupné z: <http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/>
- [18] *Positive Vs Negative testing* [online]. [cit. 2017-03-30]. Dostupné z: <http://www.guru99.com/positive-vs-negative-testing.html>
- [19] *Unit Testing with JUnit* [online]. 2016 [cit. 2017-03-31]. Dostupné z: <http://www.vogella.com/tutorials/JUnit/article.html>
- [20] *SoapUI Open Source* [online]. [cit. 2017-04-01]. Dostupné z: <https://www.soapui.org/open-source.html>
- [21] *Watir.com* [online]. [cit. 2017-04-01]. Dostupné z: <https://watir.com/>
- [22] *Ranorex* [online]. [cit. 2017-04-01]. Dostupné z: <http://www.ranorex.com/how-test-automation-works.html>
- [23] *TestComplete Platform Features* [online]. [cit. 2017-04-01]. Dostupné z: <https://smartbear.com/product/testcomplete/features/>
- [24] *Unified Functional Testing (UFT)* [online]. [cit. 2017-04-01]. Dostupné z: <https://saas.hpe.com/en-us/software/uft>
- [25] *Progress Test Studio* [online]. [cit. 2017-04-01]. Dostupné z: <http://www.telerik.com/teststudio>
- [26] *LoadRunner* [online]. [cit. 2017-04-01]. Dostupné z: <https://saas.hpe.com/en-us/software/loadrunner>
- [27] *Apache JMeter* [online]. [cit. 2017-04-01]. Dostupné z: <http://jmeter.apache.org/>
- [28] *Introduction* [online]. 2017 [cit. 2017-04-01]. Dostupné z: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp#brief-history-of-the-selenium-project
- [29] *Introduction to Selenium* [online]. 2017 [cit. 2017-04-01]. Dostupné z: <http://www.guru99.com/introduction-to-selenium.html>
- [30] *Selenium Remote Control* [online]. [cit. 2017-04-01]. Dostupné z: <http://www.seleniumhq.org/projects/remote-control/>
- [31] *Selenium IDE* [online]. [cit. 2017-04-01]. Dostupné z: <http://docs.seleniumhq.org/projects/ide/>
- [32] *Selenium-Grid* [online]. 2017 [cit. 2017-04-01]. Dostupné z: http://www.seleniumhq.org/docs/07_selenium_grid.jsp
- [33] *Introduction to WebDriver & Comparison with Selenium RC* [online]. [cit. 2017-04-01]. Dostupné z: <http://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html>
- [34] *Selenium - Webdriver* [online]. [cit. 2017-04-01]. Dostupné z: https://www.tutorialspoint.com/selenium/selenium_webdriver.htm

- [35] *IWebDriver Interface* [online]. [cit. 2017-04-02]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_IWebDriver.htm
- [36] *INavigation Interface* [online]. [cit. 2017-04-02]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_INavigation.htm
- [37] *IWebElement Interface* [online]. [cit. 2017-04-02]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_IWebElement.htm
- [38] *Selenium WebDriver Recipes in C#*. Second Edition. ISBN 978-1-4842-1741-2.
- [39] *What Is NUnit?* [online]. [cit. 2017-04-04]. Dostupné z: <https://www.nunit.org/index.php?p=home>
- [40] *Page Object Model (POM)* [online]. [cit. 2017-04-04]. Dostupné z: <http://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html>
- [41] *Build and Release Agents* [online]. 2017 [cit. 2017-04-11]. Dostupné z: <https://www.visualstudio.com/en-us/docs/build/concepts/agents/agents#install>
- [42] *AUTOMATION TESTING* [online]. [cit. 2017-04-12]. Dostupné z: <http://www.guru99.com/automation-testing.html>
- [43] *Rational Functional Tester* [online]. [cit. 2017-04-22]. Dostupné z: <http://www-03.ibm.com/software/products/cs/functional>
- [44] *Silk Test 17.5* [online]. [cit. 2017-04-22]. Dostupné z: https://www.microfocus.com/media/data-sheet/silk_test_ds.pdf
- [45] *Jenkins* [online]. [cit. 2017-04-22]. Dostupné z: <https://jenkins.io/>
- [46] *TeamCity* [online]. [cit. 2017-04-22]. Dostupné z: <https://en.wikipedia.org/wiki/TeamCity>
- [47] *Bamboo* [online]. [cit. 2017-04-22]. Dostupné z: [https://en.wikipedia.org/wiki/Bamboo_\(software\)](https://en.wikipedia.org/wiki/Bamboo_(software))
- [48] *Performance test your app before release* [online]. 2017 [cit. 2017-04-22]. Dostupné z: <https://www.visualstudio.com/en-us/docs/test/performance-testing/run-performance-tests-app-before-release>
- [49] OTTA, Jiří. *Testování v procesu implementace informačního systému* [online]. 2016 [cit. 2017-04-29]. Dostupné z: <https://www.systemonline.cz/clanky/testovani-v-procesu-implementace-informacniho-systemu.htm>
- [50] *Unit Test Basics* [online]. 2016 [cit. 2017-04-29]. Dostupné z: <https://msdn.microsoft.com/en-us/library/hh694602.aspx>
- [51] *Visual Studio Funkce služby Team Services Bezplatný účet* [online]. 2017 [cit. 2017-04-29]. Dostupné z: <https://www.visualstudio.com/cs/team-services/features/>
- [52] *Return on Investment for Automated Testing* [online]. 2009 [cit. 2017-05-05]. Dostupné z: <https://www.infoq.com/news/2009/04/testing-roi>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

QA	Quality Assurance
VS	Visual Studio
VSTS	Visual Studio Team Services
POM	Page Object Model
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
RC	Remote Control
IDE	Integrated Development Environment
SOAP	Simple Object Access Protocol
REST	Representational state transfer
Watir	Web Application Testing in Ruby
WPF	Windows Presentation Foundation
SAP	Systems Applications - Products in data processing
MVC	Model-View-Controller
CVS	Concurrent Version System
UI	User Interface
GUI	Graphical User Interface
HR	Human Resources
TFVC	Team Foundation Version Control
ASP	Active Server Pages
XPath	XML Path Language
XML	Extensible Markup Language
CSS	Cascading Style Sheets
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML

DLL Dynamic-link library

PTO Paid Time Off

SEZNAM OBRÁZKŮ

Obrázek 1: Příčiny vzniku chyb v softwaru v roce 2001 [1]	17
Obrázek 2: Příčiny vzniku chyb v softwaru v roce 2010 [1]	17
Obrázek 3: Náklady na opravu chyb v průběhu času [49]	18
Obrázek 4: Cyklus testovacího procesu	18
Obrázek 5 : Základní rozdělení testování [6]	22
Obrázek 6: Ukázka pozitivního a negativního testování [18]	27
Obrázek 7: Ukázka testovací metody v MS Test [50]	31
Obrázek 8: Struktura testů v NUnit	32
Obrázek 9: Rozdělení nástrojů Selenium [29]	38
Obrázek 10: Architektura Selenium RC [30]	39
Obrázek 11: Selenium IDE [32]	40
Obrázek 12: Architektura Selenium WebDriver [34]	41
Obrázek 13: Dashboard webové aplikace Worklio	47
Obrázek 14: Ukázka vedení dokumentace v modulu Task	49
Obrázek 15: Seznam testovacích případů	50
Obrázek 16: Ukázka spuštěného testovacího případu	51
Obrázek 17: Ukázka detailu defektu v systému VSTS	52
Obrázek 18: Životní cyklus defektu používaný ve společnosti	53
Obrázek 19: Ukázka grafů s defekty podle různých kritérií	54
Obrázek 20: Proces automatizace [42]	55
Obrázek 21: Náklady na údržbu v závislosti na strukturu [3]	57
Obrázek 22: Srovnání struktury kódu bez a s modelem POM [40]	58
Obrázek 23: Vyhledání dropdown listu pomocí lokátoru Id	58
Obrázek 24: Ukázka použití atributu Order	59
Obrázek 25: Ukázka použití atributu TestCase	59
Obrázek 26: Ukázka použití kontroly dat	60
Obrázek 27: Ukázka kódu zajišťujícího vkládání dokumentů	60
Obrázek 28: Kód s výběrem prohlížeče k testování	61
Obrázek 29: Nastavení souboru default.runsettings	66
Obrázek 30: Průzkumník testů v programu Visual Studio 2015	68
Obrázek 31: Projekt automatických testů ve VSTS	68
Obrázek 32: Nastavení sestavovací definice v systému VSTS	69

Obrázek 33: Nastavení spoušti automatického spouštění testů	70
Obrázek 34: Zobrazení výsledků automatických testů ve VSTS	71
Obrázek 35: Příklad paralelního spouštění	76

SEZNAM TABULEK

Tabulka 1: Vytvoření instance driveru pro nejpoužívanější prohlížeče	42
Tabulka 2: Výběr používaných metod rozhraní IWebDriver [35]	42
Tabulka 3: Metody používané pro navigaci webového prohlížeče [36]	43
Tabulka 4: Nejpoužívanější metody rozhraní IWebElement [37]	43
Tabulka 5: Nejpoužívanější vlastnosti rozhraní IWebElement [37]	43
Tabulka 6: Možnosti nastavení parametru browser	67
Tabulka 7: Možnosti nastavení parametru envi	67
Tabulka 8: Možnosti nastavení parametru screen	67
Tabulka 9: Verze použitých prohlížečů	74

SEZNAM PŘÍLOH

PŘÍLOHA P1: Obsah přiloženého DVD

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO DVD

Zde je uvedena struktura adresářů přiloženého DVD

- Text – obsahuje text diplomové práce
- Source – obsahuje projekt WorklioUnitTest pro Visual Studio 2015
- Manual – návod použití projektu