Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vytváření automatizovaných testovacích scénářů pro bankovní sektor

Ondřej David

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Ondřej David

Systémové inženýrství a informatika
Informatika

Název práce

Vytváření automatizovaných testovacích scénářů pro bankovní sektor

Název anglicky

Creating of automated test scripts for software testing in banking

---

Cíle práce

Bakalářská práce je zaměřena na problematiku návrhu a následné realizaci testovacích scénářů pro potřeby bankovního sektoru. Smyslem a náplní této BP bude:
a) objasnit teoretické principy problematiky návrhu a vlastní realizace testovacích scénářů pro potřeby bankovního sektoru,
b) zmapovat momentální stav této problematiky a vymezit její relevantnost včetně požadavků na ni kladených,
c) navrhnout akceptovatelnou podobu těchto testovacích scénářů v souladu s identifikovanými požadavky,
d) ověřit funkčnost navržených záležitostí v praktických podmínkách bankovní instituce,
e) ověřené záležitosti zobecnit pro další možná uplatnění.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a kritické analýze dostupných informačních zdrojů, zkušeností jejího autora a existujících řešení v dané oblasti. Stěžejní pro vypracování této závěrečné práce budou metody a techniky testování SW v kontextu se zadanou problematikou. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s řešenou záležitostí. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Závazný harmonogram:

Teoretické principy, literární rešerše – předmět 1. zápočtu z BP: do 5.9.2019

Zmapování momentální situace řešené problematiky, identifikace požadavků na ni kladených: do 31.11.2019

Navržení možného řešení se zřetelem na identifikované požadavky – předmět 2. zápočtu z BP: do 31.1.2020

Ověření a zobecnění navrhovaných záležitostí – předmět 3. zápočtu z BP: do 10.3.2020

Doporučený rozsah práce

45-55 stran

Klíčová slova

Testovací scénář, hodnocení kvality SW, model kvality, bankovní sektor, účinnost testování

Doporučené zdroje informací

BUREŠ, M. RENDA, M. DOLEŽEL, M. Efektivní testování softwaru. Praha: Grada Publishing a.s., 2016. ISBN 978-80-247-5594-6

HAVLÍČKOVÁ, A. ROUDENSKÝ,P. Řízení kvality softwaru: Průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8

MAJCHRZAK, A., M. Improving Software Testing: Technical and Organizational Developments. Berlin: Springer-Verlag, 2012. ISBN 978-3-642-27463-3

STEPHENS, M. ROSENBERG, D. Testování softwaru řízené návrhem. Brno: CPress, 2011. ISBN 978-80-251-3607-2

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 23. 03. 2020

Čestné prohlášení

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

V Praze dne 23.03.2020 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Václavovi Vostrovskému, Ph.D. za ochotnou pomoc a vedení při zpracování této bakalářské práce. Také bych rád poděkoval Ing. Davidu Vítečkovi, Ph.D. za umožnění zpracování této bakalářské práce v Českomoravské stavební spořitelně.

Vytváření automatizovaných testovacích scénářů pro bankovní sektor

Abstrakt

Tato Bakalářská práce se zabývá problematikou testování softwaru a jeho automatizací. V teoretické části jsou popsány modely životního cyklu softwarového vývoje a role testování v těchto modelech. V další části jsou popsány úrovně provádění testů způsoby testování a testové analýza. V praktické části je vylíčen momentální stav problematiky automatizovaného testování softwaru. Hlavním tématem praktické části je vytvoření konkrétního automatizovaného testu v Českomoravské stavební spořitelně. Vytvoření tohoto testu se skládá z testové analýzy, vytvoření testovacího scénáře, vytvoření skriptu a na konec vytvoření monitorování běhů testu. Na závěr jsou popsány výsledky vytvoření testu a konečné shrnutí.

Klíčová slova: automatizované testování, bankovní sektor, monitorování testů, vývoj softwaru, testování softwaru, testová analýza

Creating of automated test scripts for software testing in banking

Abstract

This bachelor thesis deals with software testing and with processes of automating these tests. The theory section of this thesis describes the types of software development life cycles and what role does software testing in those models have. In next part there are described levels of testing, methods of testing and test analysis. The practical part consists of description of current state of software testing automation. Main theme of practical part is the process of creating automated test in Českomoravska stavební spořitelna. Creation of this automated test contains test analysis, creation of test scenario, creation of test script and the creation of test monitoring. In the end the results are presented with the summary following.

Keywords: automated software testing, banking, software development, software testing, test analysis, test monitoring

Obsah

Seznam obrázků

Seznam grafů

Seznam použitých zkratek

ČMSS – Českomoravská stavební spořitelna

KZ – Klientská zóna

Nel – Nová eLiška

UAT – Uživatelské akceptační testy

SIT – Systémové integrační testy

1 Úvod

V dnešní společnosti je již takřka nemožné představit si život bez elektronických zařízení, které lidem usnadňují život. Všechna tato elektronika je řízena softwarem. Nároky na kvalitní a bezchybné fungování tohoto softwaru vzrůstají s rostoucí důležitostí těchto zařízení v lidském životě. Proto je nutné při vývoji tohoto softwaru provádět důkladné testování.

Chyba, která není objevena ve vývojových fázích projektu, je mnohonásobně nákladnější na opravu než chyba, která je nalezena při testování. Chyba, která se dostane do produkční verze může také významný způsobem poškodit reputaci společnosti, snížit její tržní hodnotu a v nejhorších případech ohrozit zdraví, či lidské životy. V bankovním sektoru je bezchybný chod aplikace klíčový, jelikož například špatně otestované zabezpečení může způsobit finanční škodu klientům a tím vytvořit nepříznivou situaci pro bankovní instituci.

Testování může být v závislosti na velikosti projektu nákladné a zdlouhavé. Proces testování se skládá z mnoha různých činností od vytváření testovacích dat, exekuci testovacích případů až po vyhodnocování testů a zajišťování opravy zjištěných defektů. Manuální exekuce testovacích případů může skýtat řadu problémů. Tento proces je zejména u komplexních testů zdlouhavý a klade vysoké nároky na pozornost testera, kterému může chyba po dlouhých hodinách repetitivní práce uniknout. Aby se tento proces zefektivnil, je žádoucí s pomocí vhodných nástrojů alespoň část těchto testů zautomatizovat.

Zautomatizovat však všechny testy není vhodné kvůli omezením které automatizované testování má a kvůli vysoké počáteční časové investici nutné k vytvoření těchto testů. K automatizaci jsou vhodné testy, které se často opakují.

Autor si vybral téma automatizace z důvodů jeho zaměstnání v Českomoravské stavební spořitelně, kde se věnuje testování softwaru. K automatizování bylo vybráno testování založení klientské zóny. Jedná se o jednu z klíčových funkcionalit, a proto je nutné ji spolehlivě otestovat. Tento regresní test se opakuje vždy při nasazení nové aktualizace softwaru. Jelikož se v ČMSS software stále vyvíjí, jsou tyto aktualizace velice časté. Časté opakování testu jej tak dělá vhodným kandidátem na automatizaci.

Při testování není jedinou náplní práce postupovat podle napsaného scénáře, ale také vytváření testovacích dat, která jsou k provedení testů potřebné. Příprava těchto dat

může být v závislosti na testech zdlouhavá. Při použití automatizace v přípravě testovacích dat se ušetří velké množství času testera, který místo toho může věnovat čas dalším úkolům. Autorem vytvořené řešení bude tuto problematiku zohledňovat a vytvořený automatizovaný test bude při každém běhu generovat testovací data ve formátu textového souboru, ve kterém budou přihlašovací údaje do klientské zóny. Tyto údaje budou sloužit k dalšímu testování.

Ověření funkčnosti navrhnutého automatizovaného testu je velice důležité. Na vytvořeném testu závisí úspěšnost nasazení aktualizace, kvůli které testování probíhá. Na test, který funguje nestabilně není možné spoléhat a pozbývá tak svého účelu. Je tedy nutné ověřit, že test probíhá bez chyb skriptu, které by ovlivňovali jeho úspěšné dokončení. K tomuto účelu slouží monitorování běhu testů. V případě že nastane chyba aplikace, z monitorování průběhu to musí být jasně patrné. Vytvořený automatizovaný test bude ověřen v prostředí Českomoravské stavební spořitelny pomocí nástrojů k monitorování testů.

V teoretické části je čtenáři představena problematika testování softwaru od role testování ve vývoji softwaru, úrovní a způsobů testování až po testovou analýzu. Současný stav využití automatizovaného testování a vývoj zastoupení testování v rozpočtu projektu je vylíčen v závěrečné kapitole teoretické části této bakalářské práce.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je navržení akceptovatelné podoby automatizovaného testovacího scénáře pro potřeby bankovního sektoru v souladu s předem identifikovanými požadavky.

Dílčími cíli práce jsou generování testovacích dat pomocí automatizovaného testovacího skriptu, objasnění teoretických principů návrhu a vlastní realizace testovacích scénářů, včetně zmapování momentálního stavu řešené problematiky, vymezení její relevantnosti i s požadavky na ni kladenými.

Součástí práce je ověření funkčnosti navrženého testovacího scénáře v praktických podmínkách bankovní instituce a zobecnění jeho podoby pro další využití.

2.2 Metodika

Vlastní realizaci testovacího scénáře pro potřeby konkrétní společnosti v bankovním sektoru předchází literární rešerše zaměřená na danou problematiku. Tato teoretická východiska jsou získávána zejména za pomocí studia odborné literatury ve formě českých i zahraničních knih zabývajících se testováním softwaru a také internetových zdrojů. Takto nastudované informace jsou následně podrobeny kritické analýze k vytřídění relevantních informací a za pomoci metody deskripce jsou tyto informace interpretovány v teoretické části práce.

V praktické části jsou využity znalosti získané v teoretické části a zkušeností autora s vytvářením automatizovaných testovacích scénářů. Zkušenosti zužitkované v této práci jsou výsledkem autorova zaměstnání v Českomoravské stavební spořitelně, pro jejíž účely regresního testování a vytváření testovacích dat je automatizovaný test v této práci popsaný, vytvořen.

Reporty a statistiky popisující podíl testování na rozpočtu projektu a podílu automatizace na procesu testování slouží k vytvoření závěrů popsaných v kapitole momentální stav řešené problematiky.

Jako podklad pro vytvoření testu je vytvořen testovací scénář pro manuální testování, na jehož základech je postaven test automatizovaný. Tento testovací scénář je vytvořen pomocí analýzy funkční specifikace testované aplikace. Poznatky získané z této analýzy slouží k určení požadovaných vstupů a očekávaných výstupů u jednotlivých kroků

průchodu aplikací. Tato analýza nevysvětluje technické náležitosti aplikace. Testování probíhá metodou černé skříňky, jelikož není k dispozici přístup k zdrojovému kódu aplikace a její vnitřní struktura je neznámá. Testování je provedeno jako test splněním. Testovací data jsou vybrána tak, aby byla aplikací akceptovaná a průchod aplikací byl splněn.

Ve vytvořeném testovacím scénáři se předpokládá určitá znalost testera dané aplikace. Každý jednotlivý krok testovacího scénáře je převeden na řadu událostí, které daný krok provedou. Skript je tvořen po sobě jdoucími událostmi akcí a mezi ně vloženými kontrolními událostmi. Dílčí skripty, které zajišťují dynamické generování dat a ukládání záznamů jsou napsány v jazyku python. Skripty jsou psány procedurálním paradigmatem.

Vyhodnocení a ověření navrhnutého řešení spočívá ve správně nastaveném monitoringu průběhu simulačního skriptu. Jsou vybrány klíčové události, které při úspěšném průchodu odešlou zprávu. Jsou měřeny hodnoty délky běhu a úspěšnost. Doba mezi měřenými událostmi je udávána v milisekundách. Úspěšnost je udávána v procentech. Tyto hodnoty jsou měřeny po určený časový úsek a je z nich vytvořen průměr.

Jednotlivé metodické kroky budou následující:

- Shromáždění informací z knižních a informačních zdrojů
- Prostudování a analyzování těchto zdrojů
- Vymezení role testování ve vývoji softwaru
- Popsání testovacích úrovní a procesů
- Zmapování momentálního stavu
- Analýza testované aplikace
- Vytvoření testovacího scénáře
- Vytvoření automatizovaného testu na základě testovacího scénáře
- Vytvoření monitorování průběhu testů
- Ověření navrhnutého řešení
- Shrnutí práce

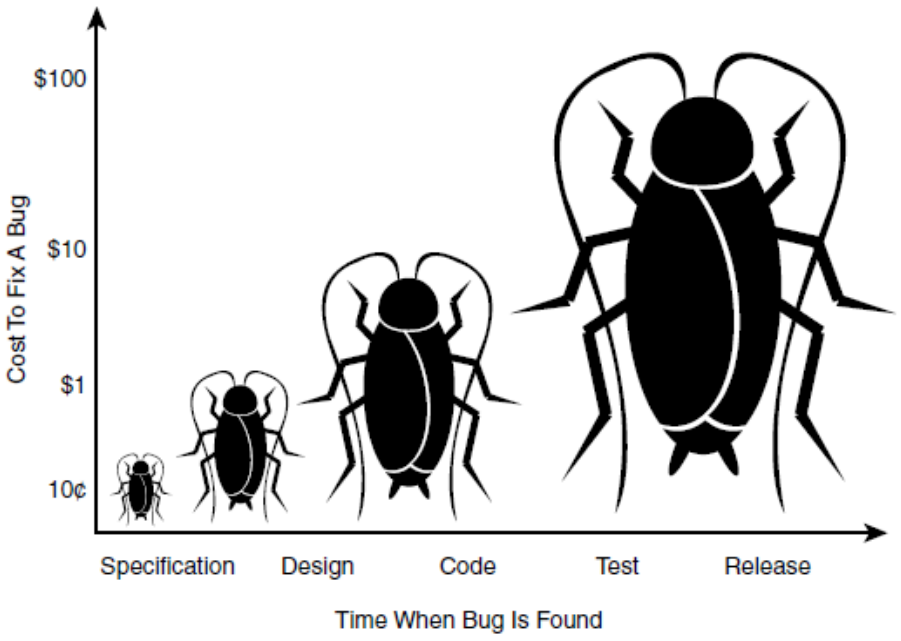
3 Testování softwaru

Software dnes řídí téměř vše, s čím se člověk denně setkává. Od semaforů na křižovatce, až po telefony a počítače, které používá k práci či internetové bankovnictví z kterého platí své účty. V závislosti na odvětví se lze setkat s různě závažnými chybami od v podstatě neškodných grafických odchylek až po chyby na lékařských přístrojích, či leteckých systémech, které mohou mít nedozírné následky. V tomto je celý smysl testování, které má za cíl co největšímu množství chyb předejít a tím zabránit vážnějším následkům, které v nejlepším případě budou mít pouze finanční dopad a v nejhorším dopad na lidské zdraví a životy. [1]

Testování se neskládá pouze hledání chyb v testovaném softwaru, ale zahrnuje například i testovací analýzu, vytváření testovacích dat a reportování výsledků testů.

Dalším důvodem testování je finanční hledisko. Testování je poměrně náročná činnost jak na personální, tak na finanční úrovni. Obzvlášť u větších projektů. Důležité ale brát v potaz, že každá nalezená a odstraněná chyba ve fázi vývoje, dokáže ušetřit mnohonásobně více peněz, než kdy se taková chyba dostane do produkce a musí se opravit zpětně. Při testování platí pravidlo, že čím později je chyba objevena, tím budou náklady na její opravení větší, jak je znázorněno na obrázku 1.[1]

Obrázek 1: Cena chyby



Zdroj: [1]

3.1 Základní pojmy v testování

Testovací scénář (případ)

Testovací scénář, často také označován anglickým názvem test case nebo zkratkou „TC“, popisuje postup, pomocí kterého testujeme požadovanou funkčnost daného softwaru. Každý test case se skládá z jednotlivých kroků, které mají být postupně provedeny. Obvykle je zadán vstup a očekávaný výstup, který je porovnán s realitou. [2]

Chyba

Chyba je způsobena lidským faktorem při psaní kódu, nebo analýze. Může se jednat o syntaktickou, sémantickou, logickou nebo chybu v samotném návrhu. O chybu se jedná, když je splněno jedno nebo více následujících pravidel:

- Výsledek interakce se softwarem je jiný, než by dle specifikace měl být.
- Výsledek interakce se softwarem je takový, jaký by dle specifikace být neměl.
- Výsledek interakce se softwarem je takový, o jakém se specifikace nezmiňuje.
- Výsledek interakce se softwarem je takový, o jakém se specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo – podle názoru testera – jej koncový uživatel nebude považovat za správný. [1]

Závažnost

Závažnost (Severity) určuje, jaký bude mít defekt dopad na systém. Závažnost nebere v potaz to, kolikrát se chyba objeví, či kolik uživatelů bude chybou zasaženo. Stupně závažnosti jsou obvykle následující:

1. *Kritická (Critical)* - Může se jednat o ztracená data, poškození hardwaru, porušení bezpečnosti nebo ukončení celého systému. Chybná funkce je nepoužitelná a neexistuje obejití problému.

2. *Vysoká (Major)* - Chyba, která způsobuje nefunkčnost jedné nebo více složek systému nebo dokonce celého systému. Může také poškozovat data. Chybná funkce se nedá použít, ale existuje možnost obejití chyby a dosáhnout tím požadovaných výsledků.

3. *Nízká (Minor)* - Neovlivňuje celkové fungování systému, ale omezuje některé funkce.

4. *Kosmetická (Cosmetic)* - Vada, která má obvykle dopad pouze na vzhled systému. [1][3]

Priorita

Priorita specifikuje, s jakou důležitostí se máme věnovat dané chybě. Určuje se podle toho, jaký vliv má na bezproblémový chod softwaru a jakým způsobem by ovlivnila koncového uživatelé v případě, že by nedošlo k opravě. Metodika, na základě které se určuje priorita, je vytvořena projektovým a testovým manažerem na začátku projektu. V některých případech není priorita stanovena pevně a rozhoduje o ni tester, který defekt nalezl. Stupně priority jsou obvykle následující:

1. *Vysoká (High)* - Nejvyšší priorita chyby, která si žádá okamžité řešení a opravu. Následkem chyby s touto prioritou je nefunkčnost softwaru.

2. *Střední (Medium)* - Chyby s touto priorito se často objevují při nasazování nových verzí softwaru. Omezuje určité funkce softwaru, ale nikdy ne takovým způsobem, aby zapříčinila nefunkčnost.

3. *Nízká (Low)* - Takováto chyba nemá vliv na funkčnost softwaru a není tedy nutné řešit ji okamžitě. Řeší se až po vyřešení chyb s vyšší prioritou. [1][3]

Incident

Incident je jakákoliv událost, která vyžaduje prozkoumání. Obvykle se ale v reálném životě používá incident jako označení chyby. Incident je tedy myšlena chyba, na kterou se přišlo během testování.[1]

3.2 Model životního cyklu softwarového vývoje

Model životního cyklu softwarového vývoje popisuje vzájemné vztahy mezi jednotlivými fázemi životního cyklu softwaru. Takovýchto modelů existuje celá řada a některé z nich jsou již staré desítky let. Většina z nich má kořeny v původních definicích vodopádového a spirálovitého modelu. Mohou být velmi pečlivě strukturované nebo naopak velice flexibilní. Pro různé projekty jsou vhodné různé modely v závislosti na požadavcích a výběr toho správného je důležité pro úspěch projektu.[4][5]

V závislosti na velikosti projektu se může vývojový tým skládat z několika málo vývojářů až po stovky a tisíce členů, kde každý člen týmu má svou roli, ale všichni se řídí daným modelem. Tester se musí orientovat v obvyklých modelech vývoje, musí znát model, jež se používá v jeho týmu a v jaké fázi vývoje se nachází, aby mohl vhodně navrhovat testy a přispět tím k co nejvyšší kvalitě výsledného produktu.[4][5]

3.2.1 Vodopádový model

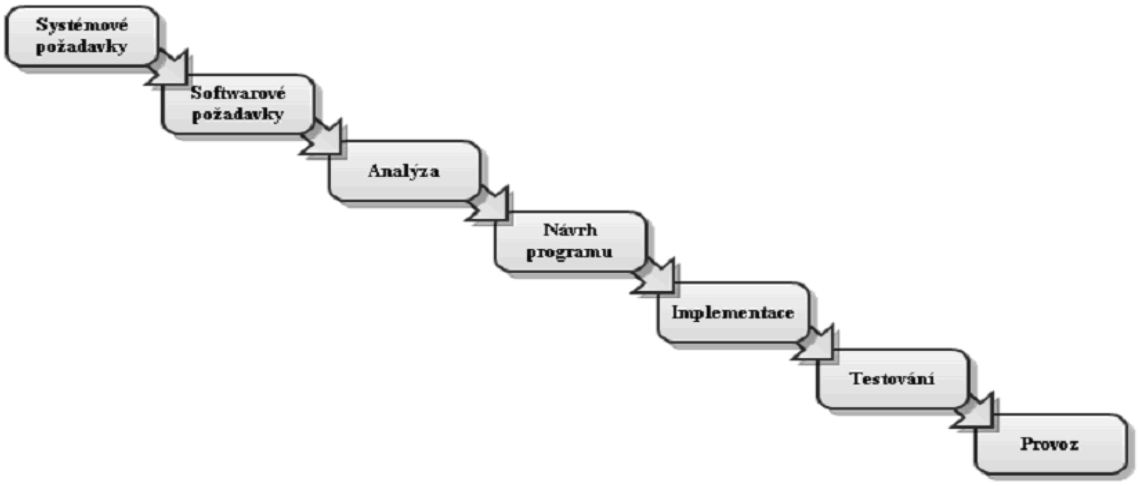
Vodopádový model spočívá v tom, že projekt přejde do další fáze až poté, co je úspěšně dokončena fáze předchozí. Například návrh programu nikdy nezačne dříve, než jsou specifikovány všechny požadavky. Tento model nutí k velkému zamýšlení se nad požadavky, jelikož v pozdějších fázích vývoje již není možné je změnit.[4][5]

Výhoda a zároveň slabinou tohoto řešení spočívá v tom, že jakmile je započata další fáze projektu, všechna práce z přechozí fáze je dokončena. Proto, když se během testování objeví nedostatky v návrhu, není možné je opravit. .[4][5]

V praxi se již tento model kvůli svým nedostatkům příliš nepoužívá. Je vhodný nanejvýše pro drobné projekty, kde nejsou vysoké nároky na funkcionality a

nepočítá se s jejich změnami. Využití má tento model k výukovým účelům, jelikož se na něm snadno vysvětluje životní cyklus softwaru. [4][5]

Obrázek 2: Vodopádový model



Zdroj: [4]

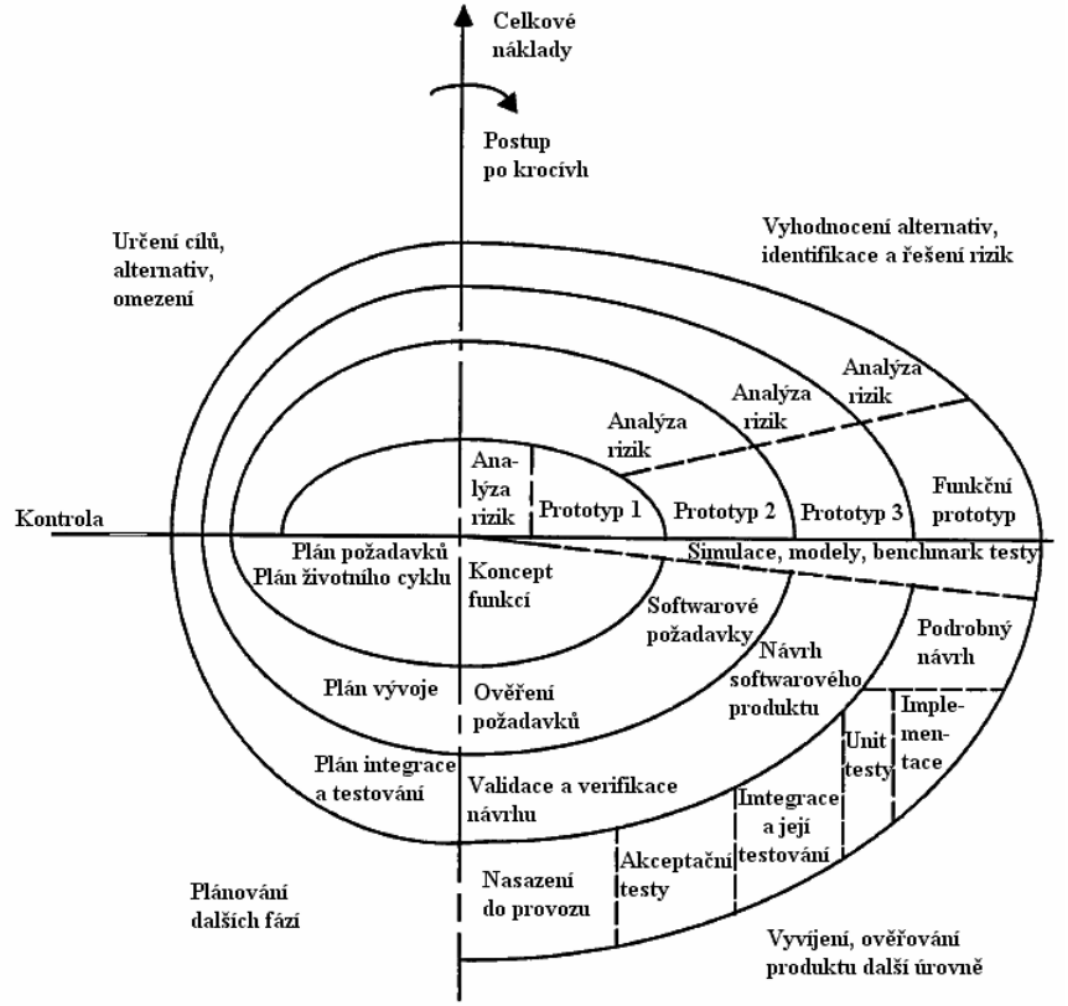
3.2.2 Spirálový model

Spirálový model funguje na principu opakování čtyř hlavních fází: určení cílů, vyhodnocování rizik, vývoj a plánování další iterace. Postup do další fáze je závislý na pečlivé analýze všech rizik.[4][5]

Na počátku je vytvořen prototyp nebo okleštěný model, který se každým průchodem spirálou upřesňuje a zdokonaluje. Po každém průchodu se provádí testování a poté se zjišťuje zpětná vazba zákazníka. Výsledný produkt je tedy pečlivě hlídán již od počátečních fází. Díky hojnému testování dochází k včasnému odhalování chyb a jejich opravě, případně při velkém množství chyb k úpravě

analýzy projektu. Tento přístup je vhodný k nasazení automatizovaných testů, kdy je pouze nutné dle současné verze upravit testovací případy a scénáře. [4][5]

Obrázek 3: Spirálový model



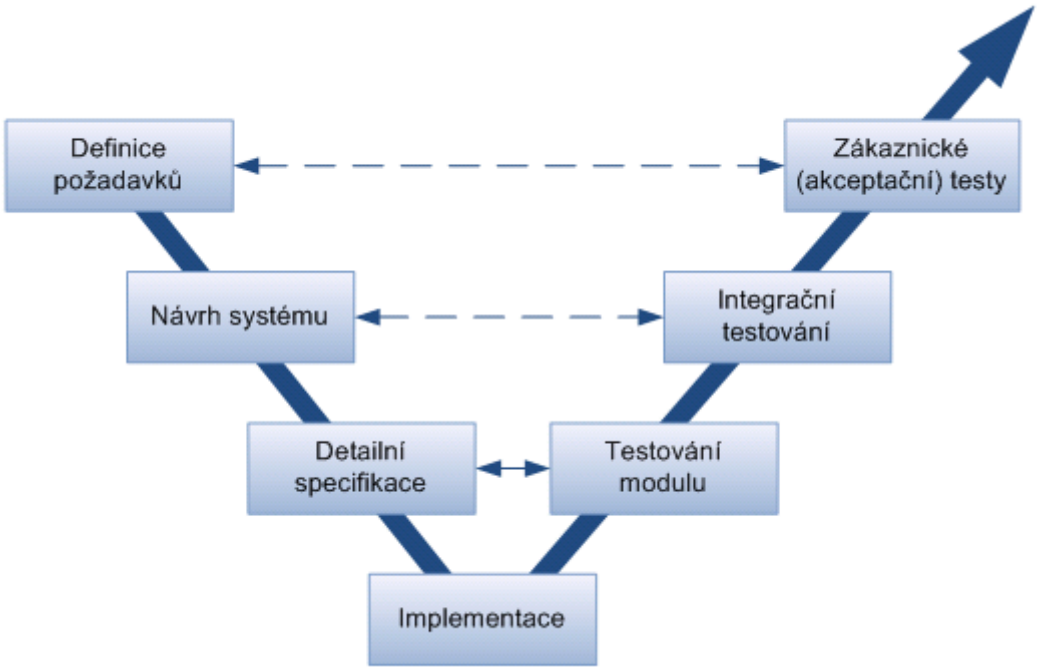
Zdroj: [4]

3.2.3 V-model

V-model představuje testování, jako proces, který prochází každým krokem životního cyklu softwaru. Na rozdíl tedy od vodopádového modelu, kde probíhá testování až na konci procesu vývoje, je zde testování prováděno po každém kroku vývoje. Ke každému kroku v životním cyklu softwaru je odpovídající fáze testování. Běžné se podle V modelu testuje ve čtyřech fázích a je vyobrazen do tvaru písmene V. Může mít ale i více fází.[6]

Na rozdíl od vodopádového modelu se při aplikaci V modelu zjistí chyba již v ranné fázi vývoje a je tedy levnější ji opravit, než kdyby se chyba objevila až na konci vývojového cyklu. [6]

Obrázek 4: V-model



Zdroj: [7]

3.3 Úrovně provádění testů

Testy se řadí do několika kategorií, dle podrobnosti a doby od napsání kódu.[1]

3.3.1 Testování programátorem

Testování programátorem probíhá obvykle hned po napsání kódu programátorem. Program je testován na úrovni zdrojového kódu. Správně by si programátor neměl testovat svůj vlastní kód, ale mě by ho nechat zkontrolovat jiným programátorem. Tomuto postupu se říká test čtyř očí. Je vhodné tyto testy provádět důkladně, neboť v této fázi je oprava chyby nejméně nákladná. V praxi se tyto testy označují jako assembly tests. [4]

3.3.2 Testování jednotek

Testování jednotek testuje jednotky, což jsou nejmenší samostatně testovatelné části programu. Můžeme je chápat jako stavební bloky softwaru. V objektově orientovaném programování se jedná o testy jednotlivých tříd a metod. V procedurálním programování můžeme brát jednotku jako jednotku program, funkci, proměnnou atd.[4][8]

Účelem je otestování jednotky separátně, tak aby nebyla žádným způsobem ovlivňována ostatními jednotkami. To znamená, že se k otestování nesmí používat, žádné další třídy, programy, databáze a jiné externí zdroje dat. Často je tedy nutné tyto externí prvky simulovat, jinak není možné test provést.[4][8]

Samotné testování funguje na jednoduchém principu. Kód testu ověřuje poskytnuté vstupy, které porovnává s očekávanými výstupními hodnotami. V případě nesouladu mezi těmito hodnotami oznámí chybu. Z principu tyto testy zachytí pouze chyby v dané jednotce, neodhalí však chyby při integraci jednotky do systému.[4][8]

Tato fáze testování je obvykle prováděna přímo pisateli kódu nebo jejich kolegy. Testy se zapisují ve formě programového kódu a k jejich vytváření se používají nástroje na bázi frameworků.[4][8]

3.3.3 Integrační testy

Po úspěšném jednotkovém testování jsou jednotky dány dohromady. Tímto spojením vzniká systém. Předchozím testováním ověřená funkčnost jednotek neznamená,

že výsledný systém bude také bezproblémově fungovat. Proto přichází na řadu *integrační testy*. [9]

Na této úrovni jsou jednotky zkombinovány dohromady a testují se jako skupina. Účelem těchto testů je objevit chyby v komunikaci jednotek a mezi jednotkami a operačním systémem a hardwarem. Postupně se testuje integrace mezi dvěma jednotkami a postupně se přidávají další a další. Tester se poté snaží najít chyby vzniklé spojením a interakcí jednotlivých jednotek. Tyto testy lze provádět manuálně, ale jsou vhodné i pro automatizaci. [9]

Integrace jednotek lze provádět různými způsoby:

- *Velký třesk*

  Všechny moduly jsou integrovány najednou v jednom kroku. Výhoda toho způsobu je jednoduché provedení. Velkou nevýhodou ovšem je z toho vyplívající obtížnost nalezení příčiny problému, který se může skrývat kdekoliv. Tento způsob je vhodný pro méně složité systémy.

- *Přístup shora-dolů*

  Moduly se integrují dle hierarchie v pořadí od nejvyššího po nejnižší. Systém může fungovat až po přidání určitého počtu modulů.

- *Přístup zdola-nahoru*

  Tento přístup je opačný od přístupu shora-dolů. Začíná se s integrací nejnižšího modulu až po ten nejvyšší. V tomto případě nebude systém fungovat jako celek do doby, než se integruje nejvyšší modul.

- *Kombinovaný přístup*

  V tomto případě kombinujeme přístup shoda-dolů a zdola-nahoru na třech úrovních. Při integraci nejvyšší úrovně jsou integrovány hlavní moduly přístupem shoda-dolů. Poté se integruje spodní úroveň přístupem zdola-nahoru a ostatní moduly jsou v prostřední úrovni. Testování lze provádět současně na horní a dolní úrovni směrem do středu. Díky tomu se tento přístup označuje také jako sendvičoví. [9]

U menších projektů se systémovým testům nepřisuzuje příliš velká důležitost. Tato úroveň testů lze totiž teoreticky úplně vynechat, aniž by byla ovlivněna výsledná kvalita softwaru, za předpokladu, že bude správě provedena následující úroveň testování. Chyba,

která by se objevila při integračních testech, se objeví v dalších úrovních. Jelikož je ale cena za opravu chyby tím menší, v čím ranější fázi se objeví, neměly by se integrační testy podceňovat.[4][9]

3.3.4 Systémové testy

Systémové testy jsou známé také jako SIT (System integration test). V této úrovni testování se testuje aplikace jako fungující komplet. Toto testování se provádí ve finálních fázích vývoje, kdy je již software v podstatě kompletně vyvinut.[4]

Software se testuje stylem, kterým ho bude používat zákazník. Na začátku testování se připraví testovací scénáře, které co nejvěrněji kopírují předpokládané chování uživatele. Při objevení chyby, se tato chyba opraví a test se prochází znovu.
Používají se funkční i nefunkční testy.[4]

Tato úroveň testování je poslední na straně softwarové firmy, která produkt vytváří, před jeho uživateli. Je tedy nesmírně důležitá, jelikož bez těchto testů by testování jako takové nemělo žádný význam. Mohli bychom sice provést jednotkové testy i integrační testy, ale bez celkového otestování aplikace bychom neměli záruku, že funguje, jak má. Tyto testy tedy fungují jako výstupní kontrola kvality softwaru.[4]

3.3.5 Uživatelské akceptační testy

Uživatelské akceptační testy (User acceptance tests) se již provádějí na straně zákazníka. Akceptační testy se provádí, aby zákazník zjistil, zda software funguje podle jeho představ a bez chyb, které by narušovaly kvalitu produktu. Při těchto testech se zákazník také ujišťuje, zda původní požadavky na funkce aplikace byly správně nastavené, nebo zdali je nutné tyto požadavky pozměnit.[4][9]

Aplikace se testuje jako už finální hotový produkt, který je připraven na nasazení. Zákazník provádí testy podle předem připravených scénářů, na kterých se dohodne s dodavatelem a které by měli pokrýt veškeré možné využití dodaného softwaru. Například, když bychom testovali aplikaci klientské zóny banky, testovací scénář se může skládat z kroků přihlášení do aplikace, zkontrolování zůstatku na účtu, změnu kontaktní adresy a podobné úkony. Všechny tyto testy se provádějí způsobem, jakým by postupoval koncový uživatel (zákazník banky, obchodní zástupce atd.) .[4][9]

Případná chyba nalezena v této fázi se posílá zpět dodavateli, který tuto chybu opraví. V případě, že nalezená chyba neohrožuje základní funkce, aplikace se většinou nasadí bez opravy a oprava se nasadí v dodatečném balíčku později.[4][9]

3.4 Způsoby testování

3.4.1 Manuální testování

Manuální testování se provádí ručně bez pomoci softwaru pro automatizaci. Tester provádí kroky podle předem připraveného testovacího scénáře a hlásí případné nalezené incidenty. Tento způsob testování je vhodný k testovacím případům, u kterých by byla automatizace velmi obtížná nebo dokonce nemožná.[4]

Manuální testy na rozdíl od automatizovaných testů nevyžadují velkou časovou investici při přípravě testů, ale časová náročnost při jejich exekuci je o mnoho vyšší. Proto se manuálně testují převážně testy, které se neopakují. Další nevýhoda manuálního provedení je možná chybovost testera, jelikož velký počet testů k provedení, únava, nemoc a další vlivy mohou výrazně ovlivnit výsledky testů. [4]

3.4.2 Automatizované testování

K *automatizovanému testování* se používají nástroje, pomocí kterých se vytvoří testovací skript. Ten poté projde samostatně krok po kroku testovací scénář podle předem nastavených podmínek. Takové testy mohou být, za předpokladu správného vytvoření, velice efektivní.[9]

Důvodem k automatizaci je především zvýšení efektivity. Ta se zvýší díky časové úspoře, jelikož simulační skript provede testovací případ mnohem rychleji, než když testuje manuálně tester. Takto ušetřený čas je pak možné využít jinde.[9]

Automatizace všech testovacích případů není žádoucí ani výhodná. Pro automatizaci je nutné vybrat takové případy, které se opakují. Vytvořit automatizovaný test, který se provede jednou, je zbytečné plýtvání finančními a časovými zdroji. Aby se automatizace vyplatila, může to trvat i několik týdnů až měsíců, podle náročnosti vytváření automatizovaného testu. Proto je nutné dobře promyslet, zda se takový test vyplatí.[9]

3.4.2.1 Techniky automatizace

Vytvoření automatizovaných testů je možné dosáhnout pomocí různých technik. Tyto techniky se liší ve snadnosti použití, komplexnosti skriptů a nutnosti tyto skripty udržovat. Tyto techniky jsou následující: [9]

- *Zachycení a přehrávání aktivity uživatele*

  Tato technika funguje pomocí nástroje, který nahrává jednotlivé akce uživatele, který provádí jednotlivé kroky testovacího případu v rozhraní testované aplikace. Tyto nahrané vstupy jsou poté převedeny na jednotlivé kroky skriptu. Výhodou této techniky je jednoduchost. Pro vytvoření takového skriptu není zapotřebí znalostí programovacího jazyka. Stačí pouze zapnout nahrávání a nahrát test. Nevýhodou je velká citlivost takto vytvořeného skriptu na změnu grafického rozhraní. Při změně aplikace je často nutné odpovídajícím způsobem upravit skript. [9]

- *Modifikace vygenerovaného skriptu*

  Tato technika je založena na předchozí. Pouze rozšiřuje její možnosti. Pomocí skriptovacích jazyků lze generovat vstupní data, uložit hodnoty a tak dále. Takto vytvořený testovací skript je tedy komplexnější a lze automatizovat složitější testovací případy. Pro vytvoření testovacích skriptů touto technikou musí mít již tester určité znalosti skriptovacích jazyků. [9]

- *Testování řízené daty*

  Při užití této techniky využívá vytvořený testovací skript jako data z tabulky jako vstupní data a poté porovnává výstupní data s očekávanými výstupními daty, která jsou rovněž uložena v tabulce. Tato technika již kladně vyšší nároky na znalost skriptovacích jazyků. .[9]

3.4.3 Testy splněním a selháním

Při *testech splněním* jsou zadána vstupní data taková, která jsou aplikací vždy akceptována. Poté je prováděna kontrola, zda výstup aplikace odpovídá návrhu.

Naopak při *testech selháním* jsou zadána vstupní data, která jsou nestandartní. Tester se snaží způsobit ukončení běhu aplikace. [4]

3.4.4 Progresní testy a regresní testy

Progresní testy se provádějí při uvedení nových funkcí a jejich kontrole. Pro správné provedení je třeba dokumentace, která popisuje nově implementované oblasti. Používají se ve všech fázích testování. Tyto testy nejsou v praxi příliš využívané.

Při opravě chybného kódu nebo implementaci nových funkcí je vždy určitá pravděpodobnost, že tato změna mohla nějakým způsobem ovlivnit části aplikace, které s danou úpravou aplikace nemají žádnou spojitost.[1][9]

Regresní testy slouží k opakovanému testování funkčností aplikace. Používají se k ověřování, že nově implementované funkce neměli negativní vliv na bezproblémový chod všech funkcí aplikace. Při těchto testech se tester soustředí na testování oblastí, které nebyly předmětem úprav, jelikož upravované oblasti by měly již být otestovány v rámci funkčních testů. Pro potřebu regresních testů nejsou vytvářený nové testovací scénáře, ale provádí se opětovný průchod již dříve vytvořenými a exekuovanými testy.

Regresní testování se provádí ve všech fázích testování.[1][9]

3.4.5 Smoke testy

Smoke testy fungují k rychlému zjištění, je-li možné přistoupit k další fázi testování. Ověřujeme si základní průchodnost systémem. Pomocí jednoduchých testů se zjišťuje, zda jsou nejdůležitější funkce aplikace funkční. Když fungují, je aplikace připravená na důkladné testování, nefungují-li, další testování je pozastaveno.
Díky rozsahu, který pokrývá výrazně menší část aplikace než ostatní testy, můžou být smoke testy dobře automatizované. [1][4]

3.4.6 Testování bílé a černé skříňky

Při metodě *bílé skříňky* máme k dispozici zdrojový kód softwaru. Je to tedy metoda založená na důkladné analýze vnitřní struktury softwaru. Díky tomu můžeme provést všechny průchody systémem. Znalost zdrojového kódu umožňuje testerovi otestovat chování na validní i nevalidní vstupy a srovnat reálný výstup s očekávaným. [1][8]

Výhodou této metody je možnost započít testování v brzkých fázích, jelikož není nutné čekat na grafické prostředí softwaru. Testování je také velmi důkladné a pokryje s velkou pravděpodobností maximální množství možných průchodů. [1][8]

Nevýhodou může být velká komplexita testů a z toho vyplívající požadavky na podrobnou znalost programovacího jazyka. [1][8]

Naopak při metodě *černé skříňky* není pro testera vnitřní struktura softwaru, který testuje, známá. Software je v tomto případě něco jako černá skříňka, do které tester nevidí. Zadává pouze vstupy podle scénáře a vyhodnocuje, zda jsou správné výstupy. Testování se provádí v podstatě z pohledu uživatele a může tím i tedy pomoci odhalit nesrovnalosti v návrhu. [1][8]

Výhodou je, že osoba, která testuje, nemusí mít znalosti programovacích jazyků. Tyto testy je možné vytvořit hned, jakmile jsou vytvořeny specifikace softwaru.

Nevýhodou naopak může být to, že je pouze limitované množství vstupů, a značné množství softwaru takto otestovat nelze. Není tedy vhodné testovat pouze tímto způsobem. [1][8]

3.5 Testová analýza

Testová analýza je proces, při kterém na základě dokumentací (funkční specifikace, byznysová specifikace) k softwaru stanovíme podmínky testů. Definujeme tím, co má být testováno a jaké podmínky mají být nastaveny. Výsledkem testové analýzy jsou tedy podmínky, způsoby a strategie testování daného softwaru. Testovou analýzu můžeme rozdělit do následujících kroků:

1. Analýza dokumentace – Toto je první a nejdůležitější krok. Pečlivé prozkoumání dokumentace slouží k poznání vlastností, funkcí, uživatelského prostředí atd. To nám zajistí dobré porozumění systému a jeho struktury.
2. Identifikace testovacích podmínek – Dalším důležitým krokem je stanovení vstupních podmínek, které zajistí otestování všech aspektů softwaru.
3. Vytvoření testovacích případů – Po stanovení podmínek je nutné vytvořit testovací případ a připravit testovací data, na základě kterých se poté bude provádět testování. V ČMSS testovacími daty rozumíme například vytvoření testovacího klienta, který bude mít potřebné produkty k otestování výpočtu stavebního spoření.
4. Zhodnocení výstupů – po vykonání testů je provedeno porovnání, zda se očekávaný výsledek shoduje s reálným výsledkem. [10][11]

4 Vlastní práce

Praktická část se skládá ze zmapování momentálního stavu testování softwaru a využití automatizovaného testování softwaru. Dále se praktická část zabývá vytvořením automatizovaného testovacího scénáře. Vytváření tohoto scénáře probíhalo v Českomoravské stavební spořitelně a předmětem testování byla aplikace Nová eLiška (dále Nel), která slouží síti obchodních zástupců.

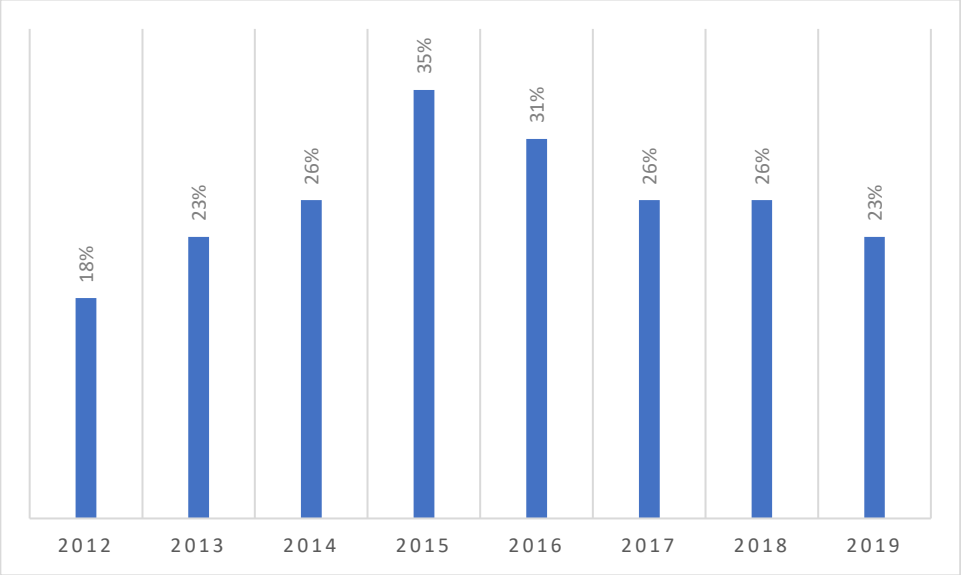
4.1 Momentální stav řešené problematiky

Testování softwaru má ve vývoji softwaru důležité místo. Jak bylo zmíněno v kapitole Testování softwaru, cena chyby velmi závisí na tom, v jaké fázi je objevena. Čím dříve je objevena tím je levnější na opravu. Cena chyby, která se dostane do produkce se může vyšplhat na miliardy dolarů. Jako příklad je možné uvést japonskou automobilku Toyota. Ta měla mezi lety 2009–2011 sérii svolávacích akcí. Na vině byly mimo jiné i softwarové chyby v modulech ABS a elektronickém ovládání plynového pedálu. Tyto svolávací akce stáli Toyotu mezi 2-3 miliardami dolarů.

Takováto kritická chyba vpuštěná do produkce má také velký vliv na celkovou hodnotu firmy. Softwarové chyby, o kterých se psalo na titulních stránkách novin, způsobily průměrně v roce 2015 pokles hodnoty firmy o 4,06 procent.

Graf 1 ukazuje vývoj podílu kontroly kvality a testování na celkovém rozpočtu projektu. V roce 2019 byly na zajištění jakosti a testování vynaloženy prostředky v rozsahu 23 procent celkového rozpočtu projektu. To je 12 procent pokles oproti roku 2015, ale stále o 5 procent vyšší hodnota než v roce 2012. Tento graf také ukazuje, že zajištění kvality a testování softwaru má velký podíl na vývoji softwaru.

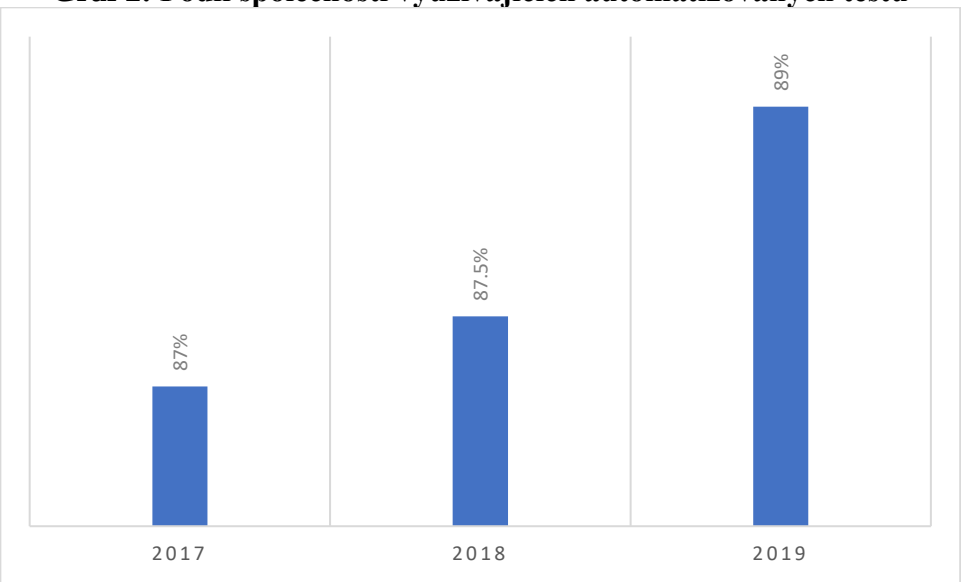
Graf 1: Podíl kontroly kvality softwaru a testovaní na rozpočtu projektu



Zdroj: Vlastní zpracování, [12]

Automatizaci testování využívá velké procento firem. K regresnímu testování používá automatizaci 76 procent, k jednotkovému testování 45 procent a ke generování testovacích dat 29 procent dotázaných firem. Graf 2 ukazuje podíl firem, které využívají nějakou formu automatizace testování. Je zde vidět vzrůstající podíl automatizace na testování. Z tohoto vyplývájí rostoucí požadavky na automatizaci testování softwaru.

Graf 2: Podíl společností využívajících automatizovaných testů



Zdroj: Vlastní zpracování, [13]

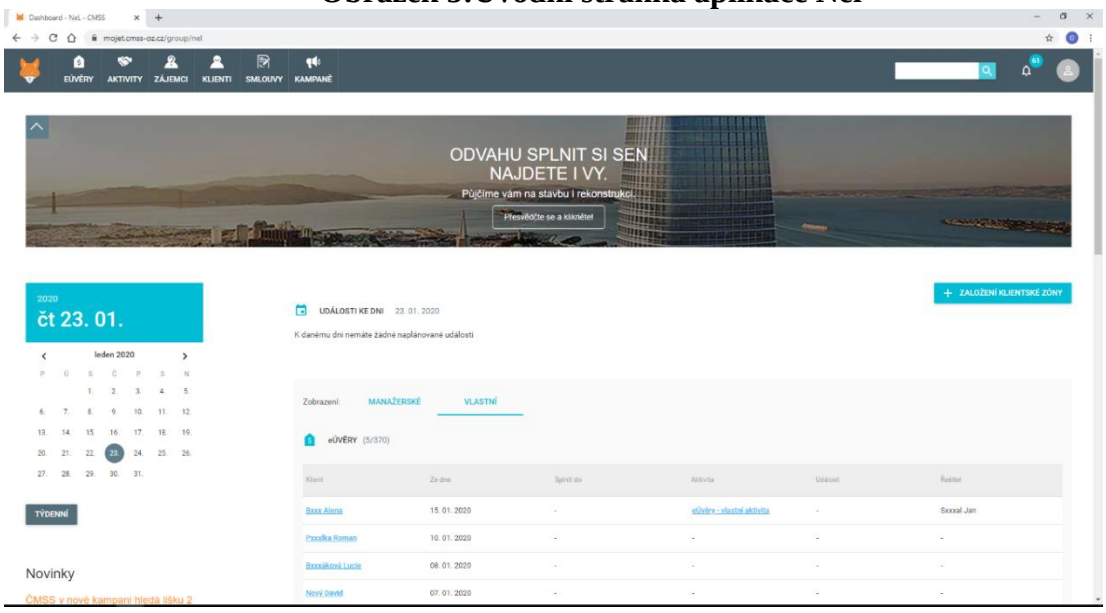
4.2 Identifikace požadavků

V této práci je řešena problematika vytváření klientské zóny v aplikaci Nel. Je to jedna z důležitých funkcionalit této aplikace. Z důvodů častého nasazování aktualizací a přidávání funkcí do této aplikace je velice důležité provádět časté regresní testy. Těmito testy je možné ověřit, zda integrace nových funkcí a aktualizací proběhla v pořádku. Dalším požadavkem pro tuto práci je vytváření testovacích dat. Klientské zóny jsou v testech často využívány a jejich manuální vytváření testerem je zdlouhavé, a ne vždy žádoucí. Proto bude tento simulační skript generovat testovací data ve formě přihlašovacích jmen a hesel do nově vytvořených klientských zón. Poslední důležitým prvkem je monitorování průběhu testů. Tento monitoring slouží k rychlému a přehlednému zjištění, jestli daná funkcionalita funguje tak, jak má.

4.3 Analýza testované aplikace

Založení klientské zóny zle provést dvěma způsoby. Prvním způsobem je záložka „KLIENT“, kde se nám zobrazí přehled klientů. Zde je možné vyhledat stávajícího klienta a vytvořit klientskou zónu. Je možné také vytvořit klientskou zónu pro osobu, která není klientem. To lze provést tlačítkem „ZALOŽENÍ KLINTSKÉ ZÓNY“. Druhá možnost je stisknutí tlačítka „ZALOŽENÍ KLIENTSKÉ ZÓNY“ přímo z úvodní obrazovky aplikace Nel.

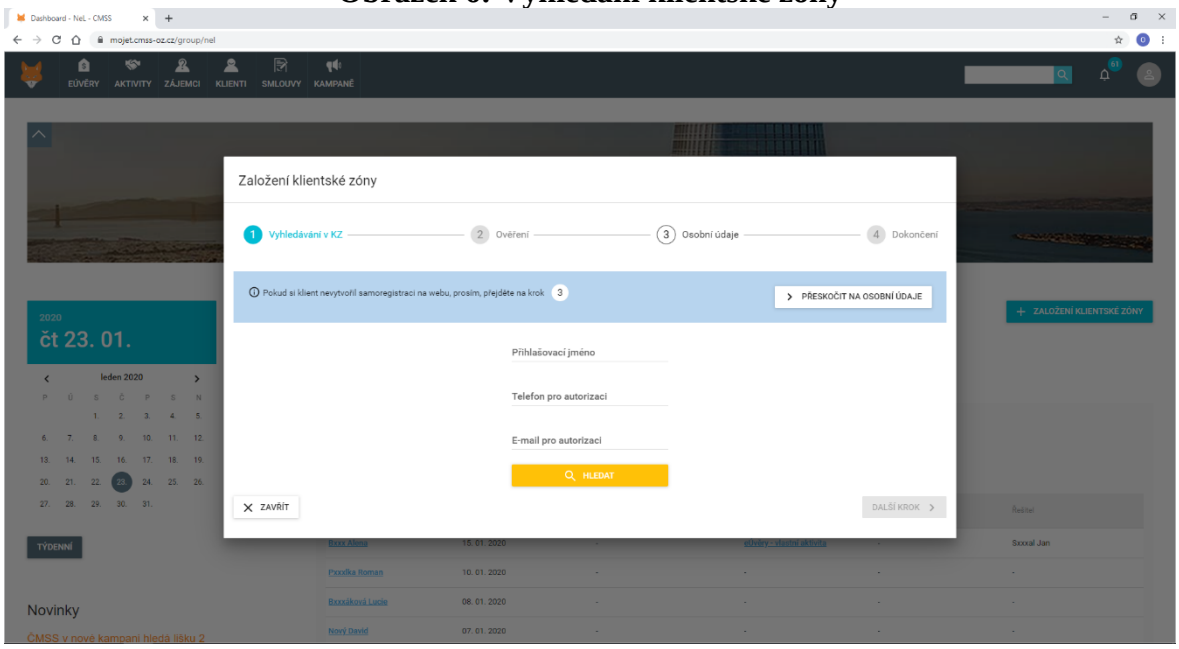
Obrázek 5:Úvodní stránka aplikace Nel



Zdroj: Vlastní zpracování, ČMSS, 2020

Po kliknutí na tlačítko „ZALOŽENÍ KLIENTSKÉ ZÓNY“ se objeví okno pro vyhledání klientské zóny. Tato možnost slouží pro uživatele, kteří si vytvořili „samoregistraci“ na webu. Pro založení nové klientské zóny zmáčkneme tlačítko „PŘESKOČIT NA OSOBNÍ ÚDAJE“.

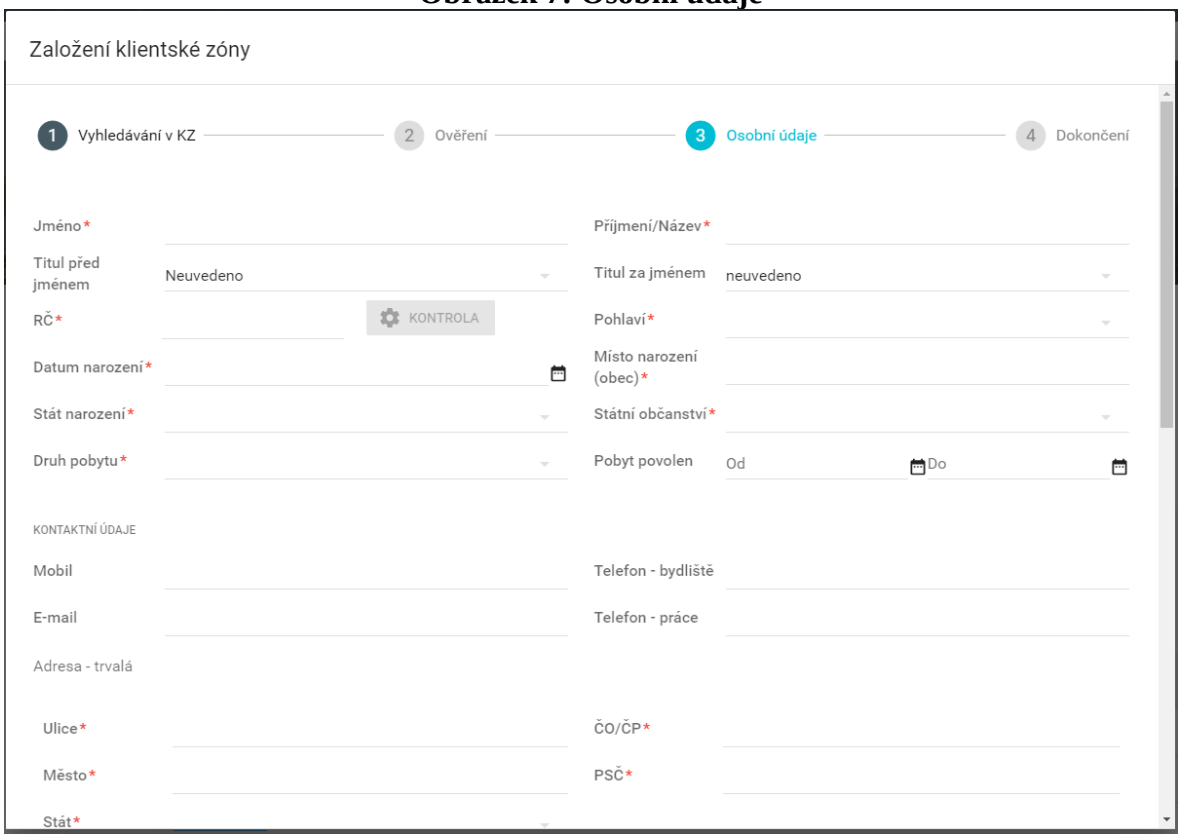
Obrázek 6: Vyhledání klientské zóny



Zdroj: Vlastní zpracování, ČMSS, 2020

Po přechodu na stránku osobních údajů se zobrazí formulář, který je nutné vyplnit. Po vyplnění povinných polí je nutné tyto údaje zkontrolovat. To lze provést stiskem tlačítka „ZKONTROLOVAT“. Při stisku tohoto tlačítka dojde k vyhodnocení, zda jsou všechna pole validně vyplněna a zároveň dojde k uložení dat v CRM (customer relationship management). Pokud dojde po kontrole ke změně údajů, musí se kontrola provézt znovu. Po kontrole se zaktivní tlačítko „DALŠÍ KROK“.

Obrázek 7: Osobní údaje



Zdroj: Vlastní zpracování, ČMSS, 2020

V dalším a zároveň posledním kroku vytvoření klientské zóny je nutné zadat údaje pro přístup a ověřování v KZ. Jsou zde 3 pole:

- Přihlašovací jméno

  Toto pole je defaultně vyplněno a je tvořeno prvním písmenem křestního jména, příjmením a případně číslicí v případě, že stejně přihlašovací jméno již bylo vygenerováno jinému uživateli.

- Telefon pro autorizaci

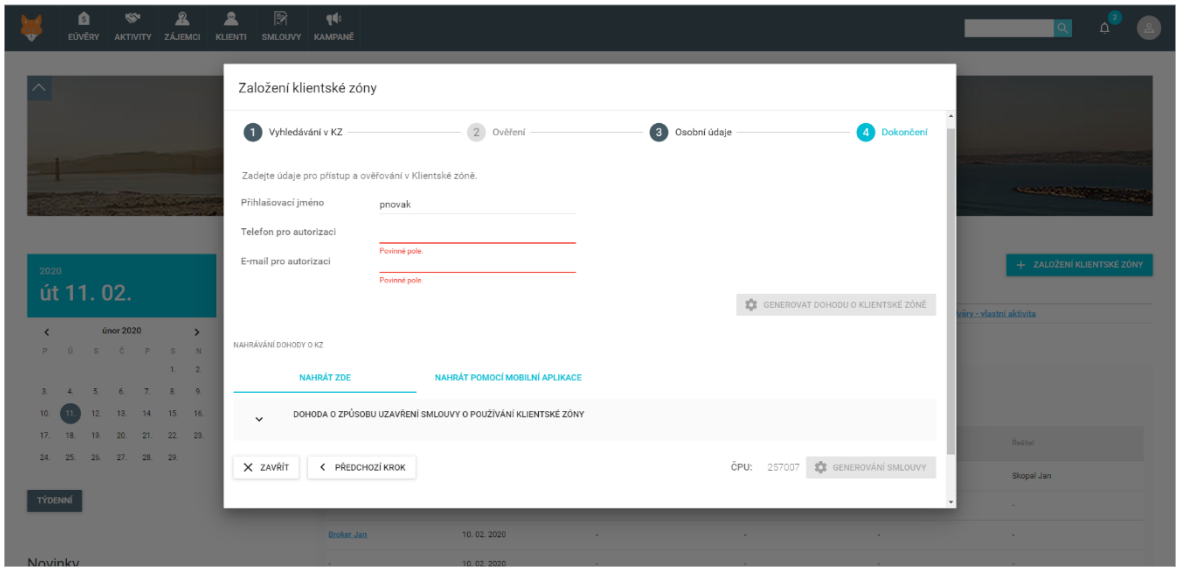
  Telefon je vyplněn defaultně v případě, že byl zadán v předchozím kroku při vyplňování osobních údajů. Pokud není vyplněn, je potřeba zadat telefonní číslo ve formátu devíti číslic.

- E-mail pro autorizaci

  Email je vyplněn defaultně v případě, že byl zadán v předchozím kroku při vyplňování osobních údajů. Pokud není vyplněn, je potřeba jej zadat ve formátu emailové adresy (např. ondra.david1@seznam.cz).

Po zadání těchto údajů se zpřístupní tlačítko „GENEROVAT DOHODU O KLIENTSKÉ ZÓNĚ“. Po stisknutí tohoto tlačítka dojde ke kontrole duplicity zadaných údajů, která se provede v příslušné databázi. Po úspěšné kontrole duplicity dojde k vygenerování dohody o způsobu uzavření klientské zóny a zobrazení tlačítka „OTEVŘÍT“, po jehož stisknutí dojde jejímu stažení ve formátu PDF.

Obrázek 8: Dokončení



Zdroj: Vlastní zpracování, ČMSS, 2020

Poté se zpřístupní pole k nahrání dokumentu. Toto pole se musí nejdříve rozbalit pomocí ikony ve tvaru šipky. Po rozbalení je k dispozici možnost nahrát dokument. Zde by se měla nahrát výše vygenerovaná smlouva podepsaná klientem a převedená do digitální podoby ve formátu PDF nebo JPG ale pro účely testování je možné nahrát jakýkoliv dokument PDF nebo obrázek JPG. Po vybrání dokumentu je potřeba stisknout tlačítko „SLOUČIT A NAHRÁT“, čímž dojde ke sloučení všech nahraných dokumentů do jednoho, a pokud je některý dokument nahrán ve formátu JPG, dojde k jeho konverzi na PDF. Po nahrání dokumentu je možné tento dokument odstranit a nahrát jiný nebo nahrání potvrdit. Potvrzením se zpřístupní tlačítko „GENEROVÁNÍ SMLOUVY“. Po stisknutí

tohoto tlačítka dojde k vygenerování dokumentu. Ten se musí pomocí tlačítka „OTEVŘÍT“ otevřít a poté je možné jej elektronicky podepsat.

Po stisknutí tlačítka „ELEKTRONICKY PODEPSAT“ dojde k nastavení všech obrazovek do stavu READ-ONLY (není je již možné měnit) a dojde k uložení dat o klientovy do CRM a příslušných databází. Tímto je proces vytvoření klientské zóny hotov.

4.4 Vytvoření testovacího scénáře

Na základě analýzy procesu vytváření KZ je sestaven návrh testovacího scénáře s jednotlivými kroky testovacího případu. Každý krok obsahuje instrukce pro testera a případná vstupní data. Ke každému kroku je popsán očekávaný výsledek, který říká, jaké výstupy by měla testerova interakce s aplikací přinést. Tyto kroky jsou definovány následovně:

1. Krok: Přihlášení uživatele do Nel.
   Očekávaný výsledek: Uživatel je přihlášen.
2. Krok: Uživatel klikne na tlačítko „ZALOŽIT KLIENTSKOU ZÓNU“.
   Očekávaný výsledek: Uživateli se zobrazí obrazovka: „Založení klientské zóny“
3. Krok: Uživatel přejde na osobní údaje, zadá rodné číslo a stiskne tlačítko „ZKONTROLOVAT“.
   Očekávaný výsledek: Po stisknutí tlačítka „ZKONTROLOVAT“ se automaticky vyplní datum narození a pohlaví.
4. Krok: Uživatel vyplní všechny povinné údaje o klientovi. Po vyplnění stiskne tlačítko „ZKONTROLOVAT“. Uživatel poté přejde na další krok.
   Očekáváný výsledek: Objeví se hláška: „Všechna pole jsou v pořádku. Pokračujte prosím na doplnění údajů ke klientské zóně.“. Uživatel je přesměrován na obrazovku „Dokončení“.
5. Krok: Uživatel zadá údaje pro přístup do KZ – tel. číslo a emailovou adresu. Poté stiskne tlačítko „GENEROVAT DOHODU O KLIENTSKÉ ZÓNĚ“.
   Očekávaný výsledek: Objeví se hláška „Dokument byl vygenerován“ a zobrazí se pole pro nahrání dokumentu.

6. Krok: Uživatel nahraje dokument ve formátu PDF nebo JPG a potvrdí ji.
   Očekávaný výsledek: Dokument je nahrán a zaktivní se tlačítko „GENEROVAT SMLOUVU“.
7. Krok: Uživatel klikne na tlačítko „GENEROVAT SMLOUVU“.
   Očekávaný výsledek: Je vygenerována smlouva o používání klientské zóny.
8. Krok: Uživatel stiskne tlačítko „OTEVŘÍT“ a poté „ELEKTRONICKY PODEPSAT“.

Očekávaný výsledek: Objeví se okno s hláškou „Smlouva o KZ byla úspěšně podepsána.“.

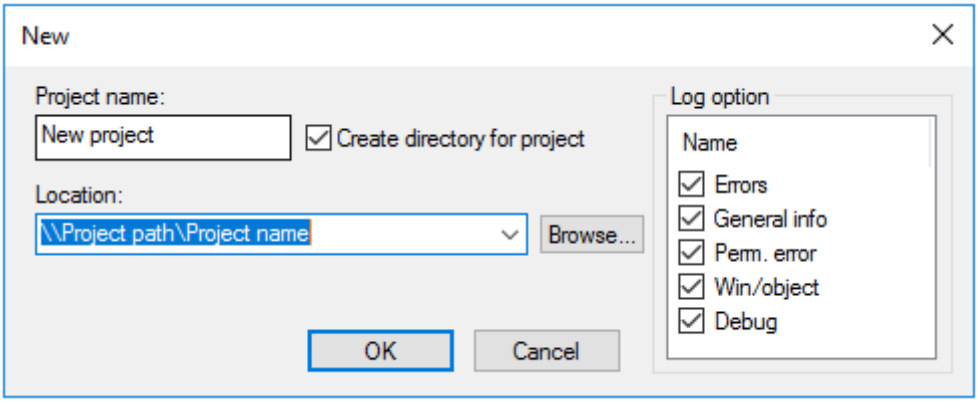
4.5 Vytvoření skriptu

Skript bude vytvořen na základě jednotlivých kroků testovacího scénáře popsaného v přechozí kapitole. K jeho vytvoření je použita aplikaci WinRobot z balíčků Simsuite od společnosti Stringdata. Tento nástroj je používán ve společnosti ČMSS, pro jejíž potřeby jsem tento test vytvářel. Skripty se pomocí tohoto nástroje dají vytvářet dvěma způsoby. První způsobem je nahrávání. Po spuštění nahrávání zaznamenává WinRobot každou jednotlivou událost (pohyb myší, kliknutí, atd). Tyto zaznamenané události poté převede do jednotlivých kroků, které je možno dále upravovat. V praxi je však tento způsob poněkud nespolehlivý a takto nahraný skript vyžaduje velké množství úprav, aby fungoval tak, jak se očekává. Druhý způsob je pomocí nástroje zvaného „Analyzér“. Pomocí tohoto nástroje je možné analyzovat jednotlivé prvky grafického rozhraní testované aplikace a jejich identifikátory. Poté lze přidat novou událost (event), které jsou přiřazeny identifikátory analyzovaného prvku a určen typ události (kliknutí myši, zadání textu, stisknutí klávesy)

4.5.1 Vytvoření nového projektu

Pro vytvoření simulačního skriptu je nutné nejdříve vytvořit nový projekt pomocí tlačítka „New“ na horní liště nástrojů.

Obrázek 9: Vytvoření nového projektu



Zdroj: Vlastní zpracování, ČMSS, 2020

Poté je nutné vybrat umístění souboru a možnosti logování a stisknout tlačítko „OK“. Pro potřeby testování jsou důležité zejména logy o chybách průběhu simulačního skriptu, které jsou pojmenovány „Errors“.

4.5.2 Vytvoření událostí (eventů)

Každý krok simulačního skriptu se skládá z událostí (eventů). Každá událost odpovídá jednomu kliknutí myši, či stisknutí tlačítka.

Rozlišujeme 3 typy událostí:

- Keybord – Tato událost simuluje stisknutí nastavené klávesy. Zmáčknutí klávesy je zaznamenáno jako 2 události „Key down“ a „Key up“.
- Mouse – Událost typu mouse simuluje stisk tlačítka na myši. Je možné nastavit stisk levého, pravého či prostředního tlačítka nebo posun myši o určenou vzdálenost a tak dále. Je také možné nastavit stisknutí tlačítka staticky podle souřadnic x a y, nebo dynamicky do centra MSAA objektu. V druhém případě bude souřadnice x a y dopočítána do středu objektu MSAA.
- Text – Slouží k zadávání textových řetězců. Textový řetězec může být buď události přiřazen, nebo může být dynamicky generován pomocí modulu DGSim.

Při nastavování událostí jsou využity tyto prvky:

- TP window (Top parent window) – V tomto kontrolním modulu probíhá nastavování kontrolních identifikátorů hlavního (nejvyššího) okna testované aplikace. Nalezení objektů specifikovaných v dalších kontrolních modulech závisí na nalezení TP window. Nastavují se zde tyto identifikátory:
    - Class name – Název třídy TP window.
    - Text – Název okna TP window
- Window – Tento kontrolní modul slouží k identifikaci Window dané události. Window může být různý prvek, například tlačítko, nástrojová lišta, editační pole atd. Nastavují se zde tyto identifikátory:
    - Class name – název třídy window
    - Text – název okna window
- MSAA – Tento modul je doplňkový (není nutné ho nastavovat), ale velkým způsobem přispívá k funkčnosti skriptu. S jeho pomocí se nastavují identifikátory MSAA objektu dané události. Nastavují se zde obvykle tyto identifikátory:
    - Name – Název daného MSAA objektu.
    - Value – Hodnota daného MSAA objektu.
    - Role – Role daného MSAA objektu. Objekt může mít roli grafiky, tlačítka, editačního pole a další.
    - State – Stav daného MSAA objektu. Může nabývat hodnot normální, dostupný, nedostupný a další.
- DGSim – Dynamické generování (DG) umožňuje řízení skriptů pomocí jazyka python. Modul lze spustit před, po nebo při startu události, po nalezení určitého objektu nebo naopak když takový objekt nebude nalezen. Tuto funkci je možné využít například při periodických změnách hesla, když je potřeba zadat do textového pole aktuální datum a tak dále. V tomto projektu je python využíván například k vytvoření jednoduchého generátoru rodného čísla tak aby odpovídalo požadavkům na validaci.
- Time control – Zde se je možné nastavit časový limit události (event-time out) nebo zpoždění události (event-delay). Po vypršení časového limitu ohlásí aplikace chybu. Zpoždění události znamená, že událost se spustí o určený časový interval po předchozí události. Tato hodnota se nastavuje v milisekundách.

- Control – v této sekci lze nastavit spouštění skriptů napsaných v jazyce python. Je možné je spustit před, při a po události, nebo po vypršení časového limitu události. Lze zde také manuálně nastavit přeskok na událost jinou než následující. Po nastavení možnosti „*No simulation*“ dojde pouze k vyhledání objektu dle identifikátorů, ale nedojde k žádné akci (kliknutí myši, zmáčknutí klávesy atd.)
- Mouse / Keybord / Text – Tato sekce se mění v závislosti na typu události. Specifikují se zde typy kliknutí, zmáčknutí klávesy nebo textový řetězec.

Při vytvoření skriptu lze využít události ke dvěma účelům. První účel je posunutí se dále v průchodu aplikací. To znamená například zadat text nebo stisknout tlačítko. Druhým účelem je kontrola. Kontrolní událost slouží k verifikaci správného průchodu předchozími kroky. Tímto je rozuměno například načtení všech prvků po přechodu na další stránku. Obvykle probíhá kontrola tak, že se hledá prvek, který se nezobrazoval na předchozí stránce, ale na nové stránce se zobrazuje. Tento prvek v události je v události nalezen, ale po nalezení není provedena žádná akce. Tímto je zajištěno to, že další událost neproběhne dříve, než se najde kontrolní prvek. Při zanedbání vytváření kontrolních událostí může nastat situace, kdy simulační skript bude provádět akci na místě, kde se prvek ještě nenačetl a způsobí tím chybu průběhu skriptu. Takový skript bude nestabilní a nebude přinášet požadované výsledky.

Všechny události nebudou podrobně rozepisovány, jelikož simulační skript jich má velké množství a události se často opakují. Proto je zde popsáno několik typů událostí, které slouží jako modelové události, podle kterých lze poté vytvořit celý skript.

Na začátku je vytvořena první událost, která má pořadové číslo 0 a bude tedy dále označována jako nultá událost. V této události je nutné nastavit spuštění aplikace Google Chrome. Událost je vytvořena stisknutím pravého tlačítka v hlavním okně a zvolením možností „Event(s)“ poté „Add“ a nakonec „*Keybord*“. Do nastavení události se lze dostat dvojnásobným stisknutím levého tlačítka myši. V sekci „Control“ je vybráno „Use DGSim on event start“. Po stisknutím tlačítka „DGsim“ se zobrazí okno, ve kterém je možné psát a upravovat skript. V tomto skriptu je nutné specifikovat adresu, se kterou se prohlížeč spustí. Do proměnných je uloženo přihlašovací jméno a heslo pro přihlášení do aplikace Nel. Nastavení cesty k spuštění prohlížeče a příznaky, se kterými se prohlížeč spustí jsou specifikovány ve spodní části skriptu. Skript vypadá takto (uživatelské jméno a heslo jsou pouze ilustrační):

url = https://mojet.cmss-oz.cz
uzivatel = "číslo obchodního zástupce"
heslo = "heslo"

dgsim.Exec('C:\Program Files (x86)\Google\Chrome\Application\chrome.exe --force-renderer-accessibility --disable-session-crashed-bubble --disable-infobars --ignore-certificate-errors --incognito --start-maximized ' + url)

Dgsim.Exec je příkaz pro spuštění programu, jehož cesta je specifikována v závorce. Dvěma pomlčkami jsou poté odděleny jednotlivé příznaky pro spuštění Google Chrome. Příznaky mají tento význam:

- Force-renderer-accessibility – Slouží k tomu, aby bylo možné rozlišovat jednotlivé prvky okna prohlížeče. Bez aktivování tohoto příznaku vidí WinRobot celé okno jako jeden prvek a není schopen rozlišovat jednotlivé objekty.
- Disable-session-crashed-bubble --disable-infobars --ignore-certificate-errors – Tato skupina příznaků slouží k tomu, aby se při startu prohlížeče neobjevovali žádné hlášky.
- Incognito – Spustí prohlížeč v soukromém režimu.
- Start-maximized – Spustí prohlížeč v maximalizovaném okně.
- Disable-tranlate – Zabrání vyskakovacímu oknu nabídnout překlad do jiného jazyka. Bez tohoto nastavení může vyskakovací okno zakrývat některé objekty a tím zamezovat běhu simulačního skriptu.

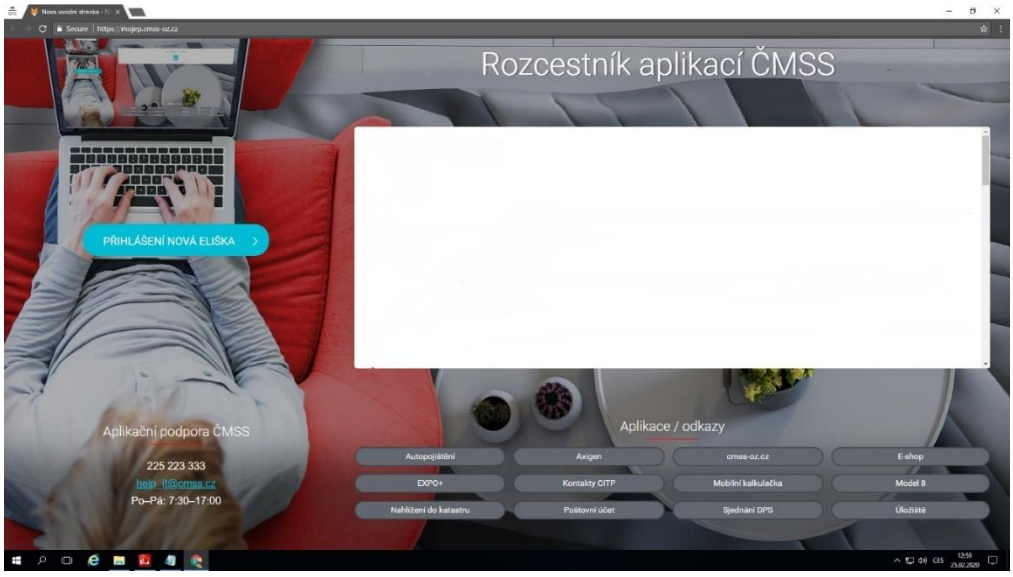
Skript je spuštěn tlačítkem „Run“. Spustí se Google Chrome v konfiguraci, jaká byla stanovena. Nyní se nástrojem „Analyzér“ označí okno, kliknutím myši se označí nultá událost a v sekci „Tools“ nacházející se v pravém panelu nástrojů je zvolena možnost „Update events control attributes“. Tímto jsou do nulté události vloženy identifikátory okna označeného „Analyzérem“. Poté je třeba v nastavení události vybrat v sekci „Control“ možnost „No simulation“. To znamená, že v tomto kroku nedojde k žádné akci stisknutí tlačítka nebo klávesy, ale pouze k provedení skriptu a poté k hledání okna dle identifikátorů. Nastavení identifikátorů je následující:

- TP window class name – Hodnota třídy je v prohlížeči Google Chrome „Chrome_WidgetWin_1“. Kontrola tohoto identifikátoru může být nastavena na částečnou shodu (partial match) nebo přesnou shodu (exact match). Na průběh skriptu to nebude mít žádný vliv, jelikož se hodnota této třídy v prohlížeči Google Chrome nemění.
- TP window text – Tato hodnota je „ČMSS – Google Chrome“. Vytvářený skript má sloužit na testovacím a před produkčním prostředí a tato hodnota se na obou prostředích liší. Aby fungoval na obou těchto prostředích bez problému, je tato hodnota změněna pouze na „Google Chrome“ a je vybrána možnost „partial match“.
- Window class name – Hodnota třídy bude v prohlížeči Google Chrome „Chrome_RenderWidgetHostHWND“. Kontrola tohoto identifikátoru může být nastavena na částečnou shodu (partial match) nebo přesnou shodu (exact match). Na průběh skriptu to nebude mít žádný vliv, jelikož se hodnota této třídy v prohlížeči Google Chrome nemění.

Jelikož má nultá událost za úkol pouze zapnout prohlížeč s danou URL adresou, je v sekci „Control“ nastavena možnost „No simulation“ a proto v sekci „Mouse“ není nutné nic nastavovat, jelikož nedojde k žádné akci.

Další modelová událost je událost první s číselným označením 1. Tato událost je kontrolní. V této události je vyhledán kontrolní prvek, který potvrdí, že se stránka načetla a je možné pokračovat. K tomuto účelu je vybráno tlačítko „PŘIHLÁŠENÍ NOVÁ ELIŠKA“, na které je v následující události číslo 2 kliknuto.

Obrázek 10: Kontrolní událost



Zdroj: Vlastní zpracování, ČMSS, 2020

Nastavení identifikátorů je v kontrolních modulech „TP Window“ a „Window“ stejné jako v předchozí události. K identifikaci je však přidán ještě doplňkový kontrolní modul „MSAA“. Nastavení těchto identifikátorů tohoto modulu je následující:

- Name – Hodnota tohoto identifikátoru je název vybraného MSAA objektu „PŘIHLÁŠENÍ NOVÁ ELIŠKA“. Zvolená možnost kontroly je „exact match“.
- Role – Hodnota tohoto identifikátoru je „push button“ neboli zmáčknutí tlačítka. Zvolená možnost kontroly je opět „exact match“.

Aby byla stabilita testu větší, je nutné v kontrolním modulu „MSAA“ v sekci „Search settings“ nastavit rozsah, ve kterém WinRobot objekt hledá. Tento rozsah se udává v krocích a jejich velikost se určuje v pixelech. Určuje se hodnotu na ose x a y. Jako ideální se osvědčila velikost kroků jako polovina výšky a šířky objektu. Počet kroků je poté nutné vyladit tak, aby skript fungoval a zároveň nehledal daný objekt příliš dlouho. V sekci „Control“ je znovu zvolena možnost „No simulation“

Událost číslo 2 má za úkol kliknout na tlačítko, které bylo v předchozí události kontrolováno. Tato událost je vytvořena zkopírováním události číslo a pouhou změnou možnosti „No simulation“ v sekci „Control“. V tomto testu je hojně využívána událost pro psaní textu do textových polí. Poprvé je využita v události číslo 4, která slouží k zadání uživatelského jména. Analyzérem je označeno pole pro uživatelské jméno a identifikátory v modulech „TP Window“, „Window“ a „MSAA“. Poté je v sekci „Text“ specifikován textoví řetězec, který bude zapsán do vybraného pole. Text je možné nastavit buď staticky

nebo dynamicky pomocí modulu „DGSim“. V tomto případě je využita možnost „Use DGSim“. Skript pro vypsání uživatelského jména je následovný:

dgsim.SetReturn(1, uzivatel)

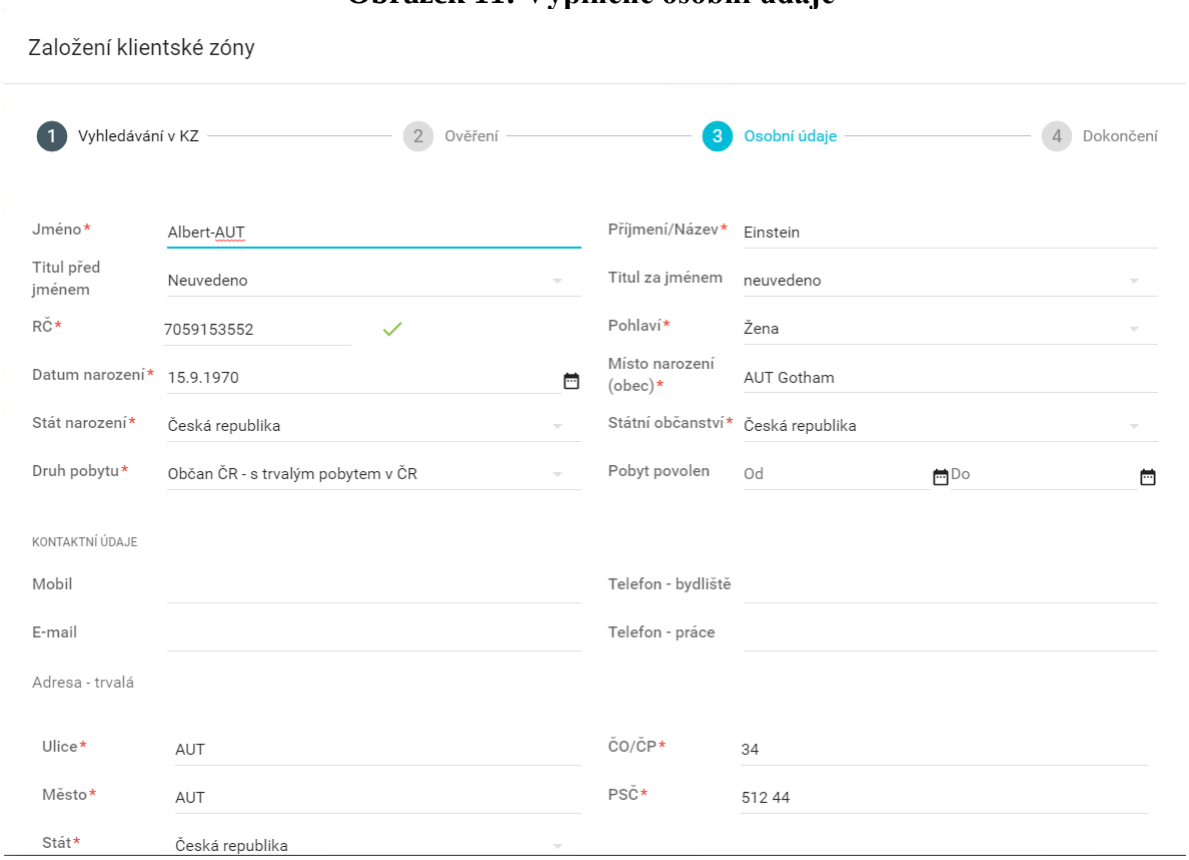
Metoda „dgsim.SetReturn“ vrátí hodnotu podle zadaného parametru. V tomto případě vrátí hodnotu proměnné „uzivatel“ do které bylo v předchozím skriptu uloženo uživatelské jméno. Hodnota parametru 1 znamená pokračování simulačního skriptu. Při zadání hodnoty 0 dojde k ukončení skriptu.

Vytváření simulačního skriptu pokračuje postupným vytvářením dalších událostí typu akce a typu kontroly podle kroků testovacího scénáře rozepsaného v kapitole 4.3. Jednotlivé události jsou vytvářeny podle principu popsaného na příkladech předchozích událostí. Problém nastává v části, ve které je nutné zadat rodné číslo do textového pole. Rodné číslo musí mít přesný formát rodného čísla. To znamená, že prvních šest číslic určuje rok, měsíc (pro ženy se k měsíci připočte padesát) a den narození. Další tři čísla jsou libovolná doplňková čísla a poslední číslice je taková, aby celé rodné číslo bylo dělitelné číslicí 11. Rodné číslo je po zadání nutné zkontrolovat, zda je validní. To je provedeno stisknutím tlačítka „KONTROLA“. V případě, že rodné číslo není validní nebo je již použité pro jinou klientskou zónu, skript se vrátí zpět a rodné číslo zadá znovu. Pro potřeby generování rodného čísla jsem vytvořil jednoduchý skript:

```
import random
#Rok narození
xx = random.randint (55,99)*10000000
#Muž nebo žena
mz = random.randint (0,1)
#Měsíc narození
#Muž
if mz == 0:
  yy = random.randint (1,12)*100000
#žena
else:
  yy = (random.randint (1,12)+50)*100000
```

```
#Den narození
if yy == 2 or yy == 52:
  zz = random.randint(1,28)*1000
else:
  zz = random.randint(1,30)*1000
#Doplňkové číslo
ddd = random.randint(000,999)
#RČ bez kontrolního čísla
rc = xx+yy+zz+ddd
#RČ ve tvaru pro přičtení kontrolního čísla
rc_k = rc*10
print (rc_k)
#Kontrolní číslo
k = rc%11
rodne_cislo = rc_k+k
dgsim.SetReturn(1,rodne_cislo)
```

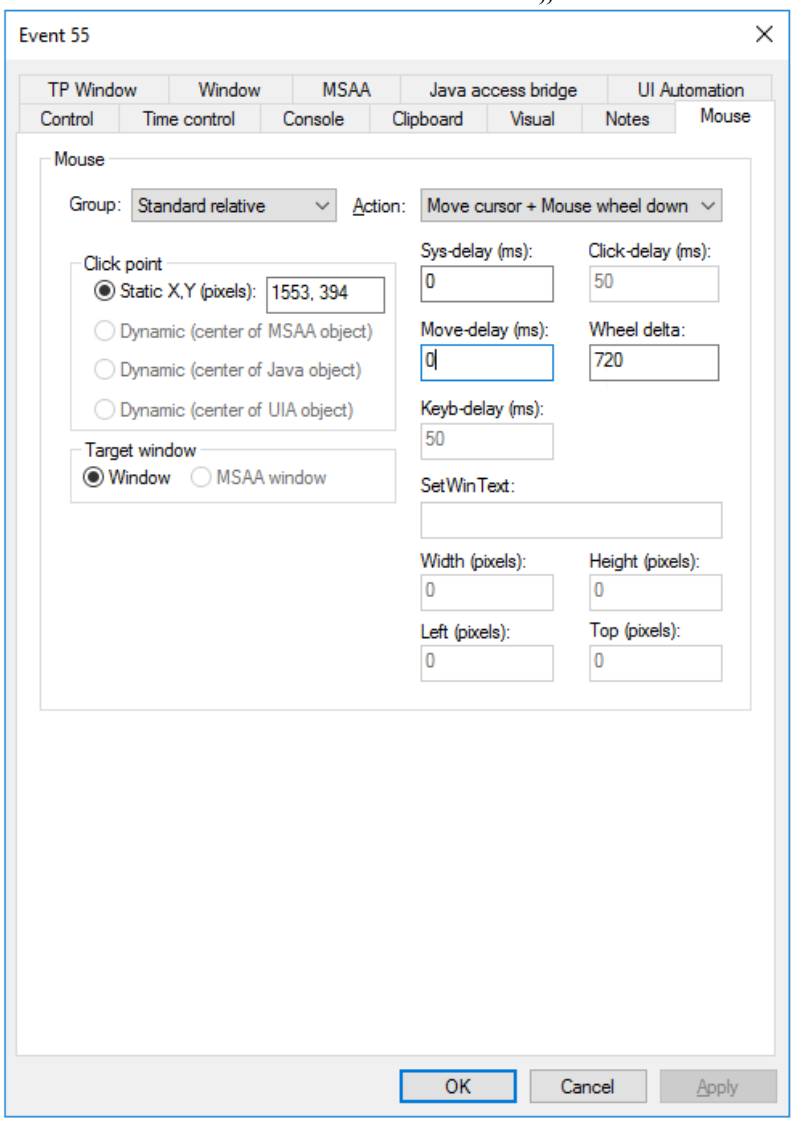
Obrázek 11: Vyplněné osobní údaje



Zdroj: Vlastní zpracování, ČMSS, 2020

Následuje vypsání všech povinných údajů. Po zadání PSČ je potřeba posunu níže na stránku. To lze provést pomocí události, které je v sekci „Mouse“ v poli „Group“ nastavena možnost „Standart relative“ a v poli „Action“ možnost „Move cursor + Mouse wheel down“. Do pole „Wheel delta“ je zvolena požadovaná hodnota posunu v pixelech. V tomto případě je to 720 pixelů. Moduly „TP Window“ a „Window“ budou nastaveny obvykle. Modul „MSAA“ zde není použit.

Obrázek 12: Nastavení sekce „Mouse“



Zdroj: Vlastní zpracování, ČMSS, 2020

Na stránce „Dokončení“ je nutné zadat telefonní číslo a email. Telefonní číslo je zadáno pomocí jednoduchého skriptu, který generuje náhodná devítimístná čísla. Email je důležitý pro pozdější použití, proto ho po vygenerování skript uloží do proměnné „mail“. Email musí být pro každého uživatele unikátní, a proto je k jeho vygenerování využito přidělené uživatelské jméno. Skript vypadá takto:

```
import random
x = random.randint (0,3)
domena = ["seznam.cz","gmail.com","centrum.cz"]
mail = "AUT"+login+"@"+domena[x]
```

dgsim.SetReturn (1,mail)

Proměnná „login“ je získána pomocí kontrolní události, která kontroluje pole, kde je defaultně vyplněno přihlašovací jméno. V modulu „MSAA“ je nastavena kontrola na „name“ a „role“ a rozsah hledání MSAA objektu. Dále je zvolena možnost „Use DGSim if found“. Hodnota „value“ se bude měnit podle aktuálního uživatelského jména. Tato hodnota bude získána a uložena do proměnné „login“. Toho je docíleno skrze tento skript:

```
import msaa
login = msaa.GetValue()
```

Metoda „mssaa.GetValue“ uloží do proměnné „login“ aktuální hodnotu identifikátoru „value“.

Vytvoření klientské zóny končí po stisknutí tlačítka „ELEKTRONICKY PODEPSAT“. Poté se objeví hláška „Smlouva o KZ byla úspěšně podepsána“. Následují události k odhlášení z aplikace Nel. Tímto končí část simulačního skriptu, která slouží k regresnímu testování. Pokračování slouží již pouze k získání přístupových údajů do nově vytvořených klientských zón a jejich uložení do souboru. Tyto slouží testerům při provádění různých typů testů.

Údaje budou z emailové služby, ve které lze přistupovat k veškerým emailům odeslaným aplikacemi ČMSS. Pro přístup k této službě je vytvořena událost, která je podobná jako nultá událost. Slouží pouze k otevření prohlížeče Google Chrome s nastavenou adresou. Použijeme tedy stejný postup jako u nulté události. Rozdíl je pouze v jiném URL a v tom, že není potřeba uložit žádné uživatelské jméno a heslo, jelikož přístup do této služby není podmíněn přihlášením.

Po načtení aplikace následuje událost pro kliknutí na tlačítko „Sent“. Poté vyhledáme emailovou adresu, kterou jsme si uložili do proměnné „mail“. Vyhledávací okno lze spustit stiskem jakékoliv klávesy. Stisknuté písmeno, se objeví ve vyhledávacím okně, a proto je nutné ho poté smazat. K tomuto účelu je vytvořena událost typu „Key“ pro stisknutí klávesy u které jsou nastaveny identifikátory modulů „TP window“ a „Window“ stejně jako v předchozích případech. V sekci „Keybord“ lze nastavit klávesu určeno ke stisknutí. To lze provést kliknutím na pole „Press new key“ a stisknutím požadované klávesy. Tím se změní číselný kód klávesy v políčku „Virtual key“ na stisknutou klávesu.

Následuje událost pro zdvihnutí stisknuté klávesy, jinak by klávesa zůstala stisknuta. Ta je vytvořena zkopírováním události pro stisknutí, pouze v sekci „Keybord“ je vybrána možnost „Key up“. Smazání napsaného písmene je zajištěno zkopírováním předchozích dvou událostí a pouhou změnou stisknuté klávesy na klávesu „Backspace“.

Po označení textového pole „Analyzérem“ je vytvořena událost typu „Text“ a pomocí dynamického generování je určen obsah textu. Skript pro generování textu je tento:

dgsim.SetReturn (1, mail)

K otevření hledaného odeslaného emailu je použita událost typu „Mouse“ a v sekci „Mouse“ v poli „Group“ je nastavena možnost „Standart relative right“ a v poli „Action“ možnost „Right“. Tato událost otevře nabídku akcí pro daný email. Možnost „Open“ otevře email. Proto je vytvořena událost pro stisknutí tohoto tlačítka.

Po otevření je nutné vytvořit kontrolní událost, zda se stránka korektně zobrazila. Poté je vytvořena kontrolní události s číslem 148, ve které bude označeno textové pole „Uživatelské jméno: uživatelské jméno“ a další kontrolní událost s číslem 149 pro označení textového pole „Heslo pro první přihlášení: heslo“. Zde uvedené uživatelské jméno a heslo jsou pouze ilustrační
V těchto kontrolních událostech jsou nastaveny moduly „TP Window“ a „Window“ jako v předchozích událostech a modul „MSAA“ bude má tyto nastavení:

- Name – Defaultní hodnoty budou „Uživatelské jméno: „uživatelské jméno““ a „Heslo pro první přihlášení: „heslo““. Ty je nutné změnit pouze na tyto hodnoty „Uživatelské jméno:“ a „Heslo pro první přihlášení:“. Možnost kontroly je nastavena „partial match“. Tento postup je nutný, protože uživatelské jméno a heslo se mění s každým během simulačního skriptu a není tedy možné mít pevně nastavenou hodnotu na jedno uživatelské jméno a heslo.
- Role – Hodnota role je „text“ a možnost kontroly „exact match“
- Je zvolena možnost „Use DGSim if found“

Pomocí následujícího skriptu jsou hodnoty z události číslo 148 uloženy do textového souboru:

import msaa
import datetime

```
#Spočítá počet řádků
count=len(open("//“Cesta k
souboru/Prihlasovaci_udaje/prihlasovaci_udaje_prep.txt").readlines())
#Datum a čas zápisu
date = datetime.datetime.now()
date = date.strftime("%d.%m.%Y - %X")
date = str(date)
#Údaje o klientovi
rodne_cislo = str(rodne_cislo)
#Zapíše hodnotu name  MSAA objektu do proměnné login
login = msaa.GetName()
#Přemazávání starých záznamů
if count > 4000:
  with
open("//sydeskprod2/SD_Script/PREP_NEL_ZALOZENI_KZ/Prihlasovaci_udaje/prihlaso
vaci_udaje_prep.txt") as f:
    lines = f.readlines()
  with
open("//sydeskprod2/SD_Script/PREP_NEL_ZALOZENI_KZ/Prihlasovaci_udaje/prihlaso
vaci_udaje_prep.txt","w") as f:
    f.writelines(lines[1000:])
#Zapsání nových záznamů
with
open("//sydeskprod2/SD_Script/PREP_NEL_ZALOZENI_KZ/Prihlasovaci_udaje/prihlaso
vaci_udaje_prep.txt","a") as f:
    f.write(date+"\n")
    f.write("Rodné číslo:"+rodne_cislo+"\n")
    f.write(login+ "\n")
```

Skript pro zapsání hodnot z události číslo 149 bude následující:

```
import msaa
```

```
#Zapíše hodnotu name MSAA objektu do proměnné passwd
passwd = msaa.GetName()
#Připsání hesla do záznamů
with
open("//sydeskprod2/SD_Script/PREP_NEL_ZALOZENI_KZ/Prihlasovaci_udaje/prihlaso
vaci_udaje_prep.txt","a") as f:
  f.write(passwd+"\n"+"\n")
```

Na konec skriptu je vytvořena událost pro zavření okna. „Analyzérem“ je označeno celé okno prohlížeče Google Chrome. Jako u předchozích událostí jsou nastaveny moduly „TP Window“ a „Window“ a v sekci „Mouse“ v poli „Group“ je zvolena možnost „Window message“ a v poli „Action“ možnost „Close“. Tímto je celý simulační skript ukončen a zbývá už pouze nastavit měření průběhu testů.

4.6 Monitoring průběhu testů

Nastavení monitorování průběhu simulačního skriptu je velice důležité k vyhodnocení, zda test funguje správně a zejména poté k monitorování funkčnosti stěžejních aplikací v ČMSS. K monitoringu je v ČMSS využita aplikace Sydesk z balíčku Simsuite. Tato aplikace slouží k vytváření monitoringu a následnému sledování průběhu v přehledném formátu.

Obrázek 13: Aplikace Sydesk

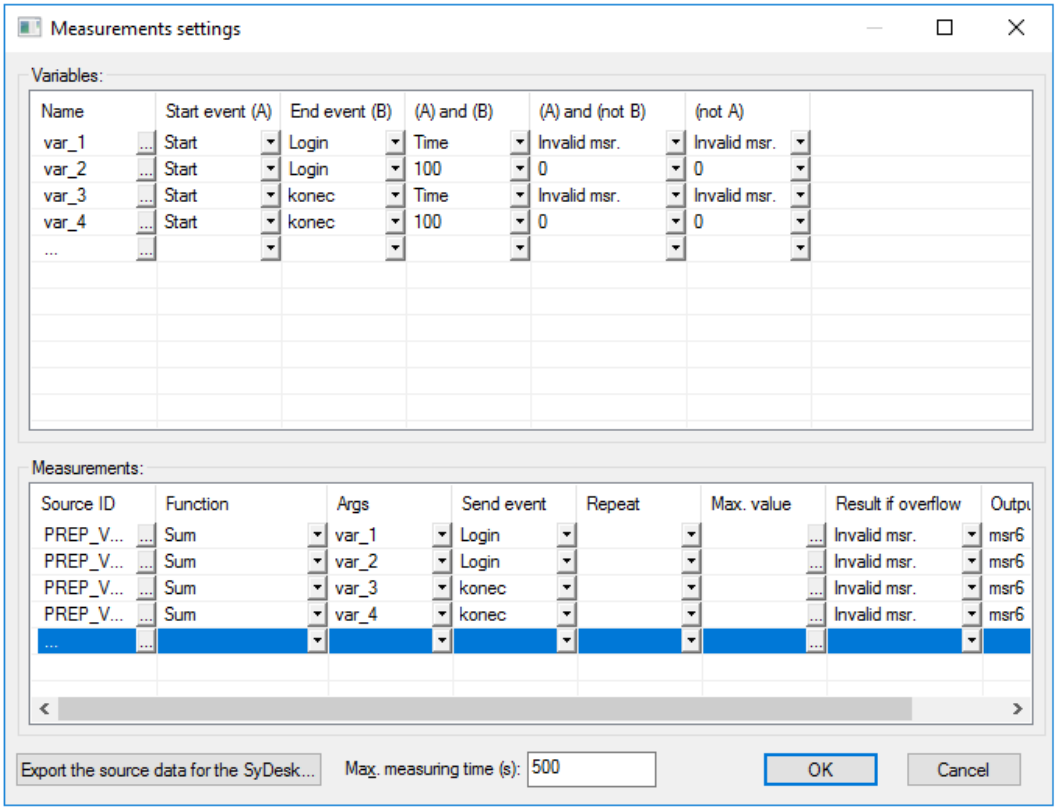
Zdroj: Vlastní zpracování, ČMSS, 2020

Pro vytvoření monitoringu je nutné nejdříve v aplikaci WinRobot nastavit měření. Aby aplikace věděla, co má přesně měřit, musí se pojmenovat počáteční a koncové události měření. Pojmenování je vytvořeno stisknutím pravého tlačítka na požadované

události a vybráním možnosti „Edit label event“. V je případě žádoucí měřit průběh od začátku skriptu po přihlašování a od začátku skriptu po konec skriptu. Je nutné mít tyto tři události pojmenované. Nultá událost je pojmenována „Start“, jelikož zde skript začíná. Událost, které značí úspěšné přihlášení je událost číslo 9 a je pojmenována „Login“. Poslední událost se nazývá „End“.

Samotné měření je vytvořeno v sekci „Measeruments“ v nabídce „Data“ na horní liště aplikace WinRobot. Jsou vytvořeny proměnné, u kterých se určí počáteční a koncové události. V části „(A) and (B)“ se určím, jakou hodnotu měření vrátí při úspěšném projití událostí „A“ i událostí „B“. Když je zvolena možnost „Time“, vrátí se čas, který uplynul mezi počáteční a koncovou událostí. Když je zvolena možnost „100“ vrátí se100 procent. Část „(A) not (B) vrací hodnotu, když nedojde k úspěšnému projití událostí „B“. Možnost „0“ znamená 0 procent. Část „(not A)“ určí hodnotu pro případ, že nedojde ani k projití události „A“. Volba „Invalid msr.“ znamená, že nedošlo k měření, jelikož nelze změřit čas mezi dvěma událostmi, když druhá událost nenastala. Nastavené hodnot jsou ukázány na obrázku 15.

Obrázek 14: Nastavení měření



Zdroj: Vlastní zpracování, ČMSS, 2020

V části „Measeruments“ je nutné nastavit „Source ID“ které slouží k pozdějšímu vyhledání měření v aplikaci Sydesk. V kolonce „Function“ je zvolena možnost „Sum“ neboli součet. Sloupec „Args“ obsahuje výše vytvořené proměnné. Sloupec „Send event“ určí událost, po které dojde k odeslání měření. Nastavení těchto hodnot je zobrazeno na obrázku číslo 15.

Po nastavení všech hodnot měření je nutné exportovat data ve formátu XML. Tento XML soubor vytvoří WinRobot po stisknutí tlačítka export. XML soubor se poté importuje do aplikace Sydesk, kde slouží jako zdroj dat. Struktura XML je ukázána zde:

```
<?xml version=“1.0“?>
<measurements>
    <measurement id =“0“>
        <Identifier>PREP_VYTVORENI_KZ_LOGIN_D</Identifier>
        <Name>< PREP_VYTVORENI_KZ_LOGIN_D</Name>
        <Plugin_type>winrobot</Plugin_type>
        <Plugin_version>4.0.0.1</Plugin_version>
        <Location/>
        <Interval>0</Interval>
        </measurement>
</measurements>
```

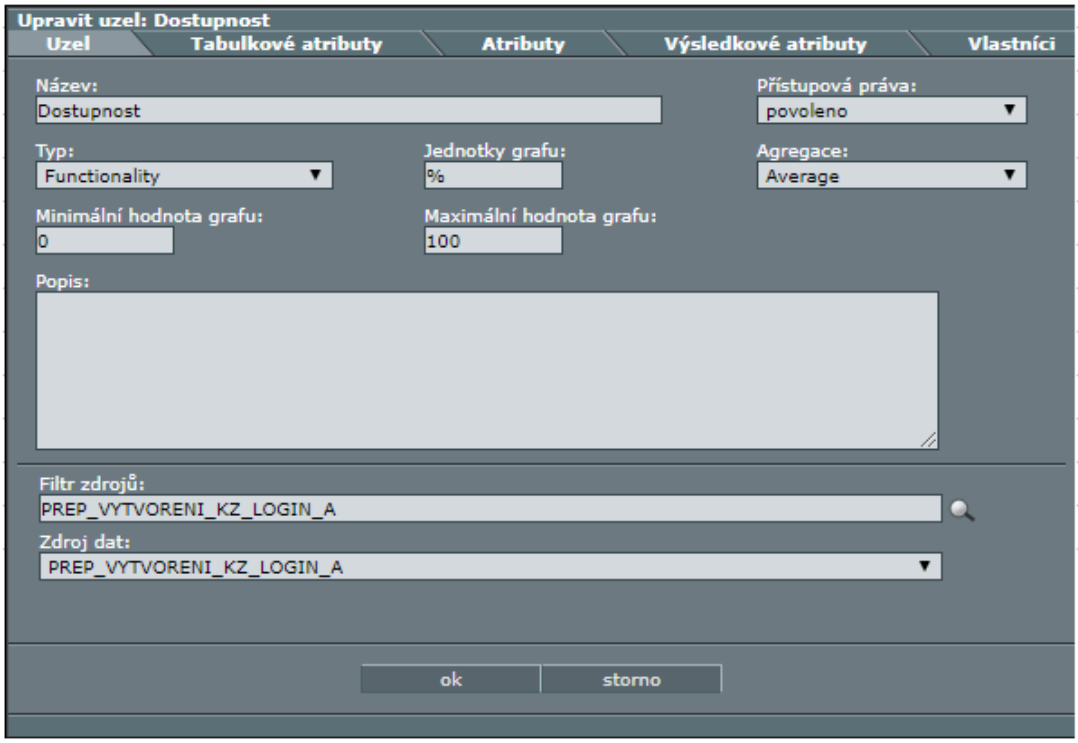
Definiční strom aplikace Sydesk se skládá z uzlů a listů. Uzly jsou nadřazeny listům a uzel může obsahovat další uzly. Do uzlu lze vložit list, ale naopak ne. List slouží k samotnému měření. K vytvoření monitoringu je tedy potřeba vytvořit uzel, do kterého jsou vloženy dva vnitřní uzly. Jeden uzel je pro monitorování úspěšnosti přihlášení a druhý pro monitorování úspěšnosti celého skriptu. Každý z vnitřních uzlů obsahuje dva listy. Jeden list slouží k monitorování úspěšnosti průchodu a druhý k měření doby průchodu. Struktura je zobrazena na obrázku číslo 17.

Obrázek 15: Definiční strom

Zdroj: Vlastní zpracování, ČMSS, 2020

Zbývá nastavit listy. List může být dvojího typu. Buď typu „Functional“ nebo typu „Number of units“. První slouží k monitorování funkčnosti skriptu a udává se v procentech a druhý pro měření času v milisekundách. Pole „Agregace“ nastavuje, jaká hodnota bude ukázána. Možnost „Average“ průměruje hodnoty za daný časový úsek a ty zobrazí. Ve zdroji dat je nutné nalézt měření s názvem který má v souboru exportovaném z aplikace WinRobot. V sekci „Tabulkové hodnoty“ jsou určeny hodnoty „Tresholds“. To jsou hodnoty, které lze chápat jako milníky.

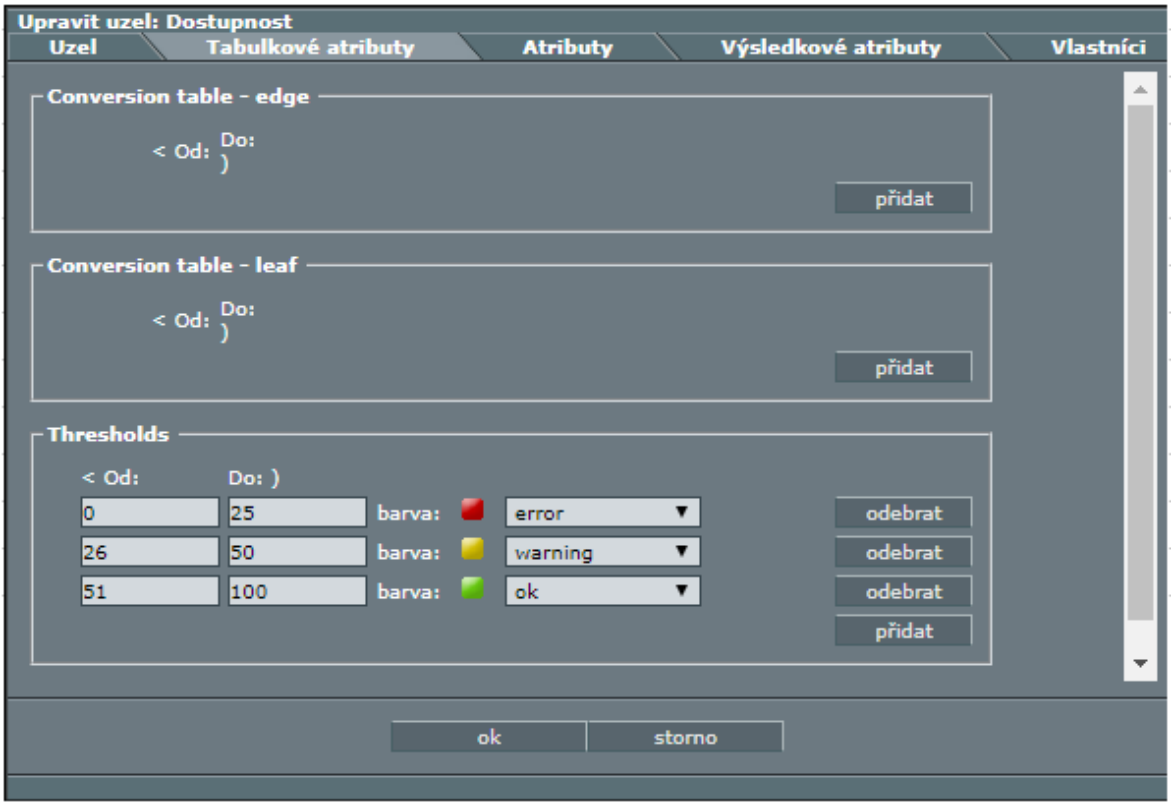
Obrázek 16: Nastavení uzlů



Zdroj: Vlastní zpracování, ČMSS, 2020

Ty slouží k rychlému rozpoznání, jak daný skript probíhá. Když vidíme červenou barvu, nejspíše se objevila fatální chyba. Žlutá může znamenat problém, který ale stále umožňuje průchod testem. Zelená znamená, že je vše v normě, která byla určena. Hodnoty jsou ukázány na obrázku číslo 19.

Obrázek 17: Nastavení "Tresholds"

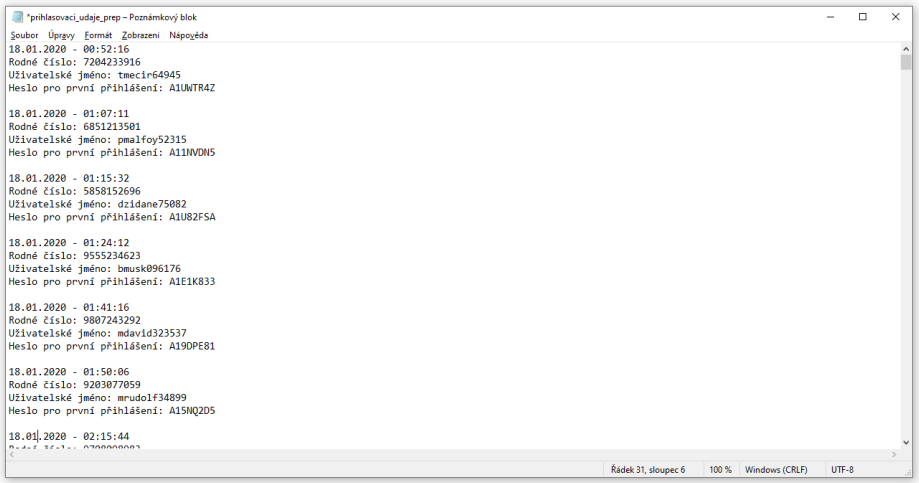


Zdroj: Vlastní zpracování, ČMSS, 2020

4.7 Ověření navrženého řešení

Zda je navržený simulační skript funkční, lze ověřit jednoduše jeho spuštěním a sledováním, zda doběhl úspěšně do konce bez chyb. Následuje přihlášení do klientské zóny pomocí přihlašovacích údajů, které proběhlo v pořádku. Tímto je dokázáno, že tento simulační skript funguje z jednorázového hlediska. Vygenerované přihlašovací údaje jsou ukázány na obrázku 18.

Obrázek 18: Vygenerované přihlašovací údaje



Zdroj: Vlastní zpracování, ČMSS, 2020

Aby se dalo z určitostí říct, že je simulační skript v této podobě užitečný pro účely regresního testování, je nutné sledovat úspěšnost průběhu v aplikaci Sydesk. Na obrázku 19 je vidět úspěšnost 85.62 procent v rozmezí hodin 8-17 a 75,64 procent v průběhu celých 24 hodin. Chybovost přibližně 15 procent, popřípadě 25 procent je způsoben občasnými chybami simulačního skriptu, či drobnými chybami aplikace. Tyto drobné chyby aplikace jsou například nutnost zmáčknout některé tlačítko podruhé, chybné vygenerování smlouvy, které se musí provést znovu a podobné chyby.

Obrázek 19: Průběh simulačního skriptu

Zdroj: Vlastní zpracování, ČMSS, 2020

Tato procentuální úspěšnost simulačního skriptu je dostatečná a lze tedy tento test bezpečně používat k regresnímu testování. Vážná chyba, která by se objevila při nasazení nové verze aplikace, by se projevila prudkým procentuálním poklesem. Při takovémto poklesu je upozorněn test leader a podle chybových hlášení se určí konkrétní příčina.

5 Výsledky a diskuze

V rámci této bakalářské práce byla zkoumána problematika testování softwaru a jeho automatizace.

V teoretické části byly vysvětleny základní pojmy užívané při testování. V další části práce byly popsány modely životního cyklu vývoje softwaru a jakou roli v těchto cyklech zastává testování. Podrobně byly popsány jednotlivé úrovně testů od testování programátorem až po uživatelské akceptační testy. Dalším tématem byly způsoby testování jako například manuální nebo automatizované testování a jejich silné a slabé stránky. Na závěr byla vylíčena důležitá role testové analýzy při vytváření testovacích scénářů.

V praktické části byla provedena testová analýza testované aplikace, na jejímž základě byly sestaveny jednotlivé kroky testovacího scénáře. Tento testovací scénář byl poté přepracován do podoby automatizovaného testu v podobě simulačního skriptu. Byly popsány modelové události, které ukazují, jak probíhá proces vytváření simulačních skriptů. V této práci byly také ukázány skripty v programovacím jazyce python, které slouží například ke generování uživatelského jména nebo rodného čísla. Na závěr praktické části bylo vylíčeno nastavení monitorování testů a jeho důležitá role v automatizovaném testování.

Na podkladě této práce lze vytvořit automatizované testy množství testovacích případů v ČMSS jako například zkoumané automatizované testy pro klientskou zónu, pro proces vytváření úvěrů online, vyhledávání klientů, mazání klientů a další, což je hlavním přínosem této bakalářské práce. Dalším významným přínosem této práce je vytváření testovacích dat v podobě vytvořených fiktivních klientů, jejichž manuální vytváření by bylo časově náročné. Takto vytvoření klienti poté slouží k testování dalších funkcionalit napříč ČMSS.

Testy, vytvořené na principu této práce, je možné využít k regresnímu testování také v ostatních bankovních institucích. Při použití stejného nástroje jako v této práci bude jediným rozdílem jiné grafické rozhraní aplikace. V případě použití jiných nástrojů, které fungují na principu modifikace nahraného skriptu, je nutné nejprve poznat fungování daného nástroje a poté lze k jejich vytvoření použít proces popsaný v této práci. Stejně tak lze tyto testy využívat v jakémkoliv jiném sektoru pro testování na základě grafického rozhraní.

Vytvořením simulačního skriptu ovšem proces automatizace nekončí. Údržba těchto testů je velice důležitá a je nutné ji provádět po celou dobu využívání daných testů. Změny v grafickém rozhraní aplikace mohou ovlivnit fungování simulačního skriptu. Proto je důležité například při změně rozložení grafických prvků, upravit odpovídajícím způsobem simulační skript.

6 Závěr

Testování tvoří důležitou součást vývoje softwaru. S rostoucí velikostí projektu rostou i jeho nároky na časové, lidské a tím i finanční zdroje. Na testování vyvíjeného softwaru se tak průměrně alokuje přibližně pětina rozpočtu projektu. Role testování ve vývoji softwaru závisí na zvoleném modelu vývoje softwaru. Dle zvoleného modelu, může testování probíhat souběžně s psaním kódu aplikace, či až na konci po dokončení všech předchozích fází.

Automatizace všeho druhu je stále probíranější téma a automatizace testování není výjimkou. Nějakou formu automatizace využívá přibližně 90 procent firem, které se zabývají vývojem softwaru. Většinou se jedná o automatizaci regresních testů a jednotkových testů. Své místo má při automatizaci ale také generování testovacích dat. Automatizované testování má oproti manuálnímu spoustu výhod, ale automatizovat veškeré testy není žádoucí. Vysoká časová náročnost na vytváření testů se nemusí u testů, které se neopakují vyplatit. Některé testy také vyžadují lidské uvažování testera.

Významnou část testování tvoří regresní testy, které se opakují v závislosti na aktualizacích a přidávání nových funkcionalit softwaru. Pro časté opakování jsou tyto testy vhodné k automatizaci, jelikož se vysoká časová investice do vytvoření automatizovaného testu rozprostře do velkého počtu opakování takového testu.

V této práci byl vytvořen automatizovaný testovací scénář pro potřeby Českomoravské stavební spořitelny, jehož účelem je testování procesu vytvoření klientské zóny. Navržené řešení usnadňuje a urychluje regresní testování této důležité funkcionality aplikace Nová eLiška. Zároveň tento test generuje testovací data, která jsou využívána při dalších testech ať již manuálních či automatizovaných. Dále bylo v práci vytvořeno monitorování tohoto testu, jež přehledným způsobem informuje, zda daný test běží v pořádku.

Na základě postupů popsaných v této práci by měl každý být schopen vytvořit automatizovaný testovací scénáře s využitím stejného nebo podobného nástroje jako zde využitá aplikace WinRobot. Tyto postupy jsou využitelné jak v bankovním sektoru, tak mimo něj.

7 Seznam použitých zdrojů

[1] PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-635-5.

[2] Cleverandsmart [online]. [cit. 2019-12-20]. Dostupné z: https://www.cleverandsmart.cz/category/testovaci-pripad/

[3] Professionalqa.com/ [online]. [cit. 2019-12-20]. Dostupné z: http://www.professionalqa.com/defect-severity-in-software-testing

[4] HLAVA, Tomáš. Testování softwaru. Testování softwaru [online]. 2011 [cit. 2019-09-16]. Dostupné z: http://www.testovanisoftwaru.cz

[5] PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5.

[6] FEWSTER, Mark a Dorothy GRAHAM. Software Test Automation: Effective use of test execution tools. Velká Británie: ACM Press books, 1999. ISBN 978-0-201-33140-0.

[7] Umel.feec.vutbr.cz: embedded systemy [online]. [cit. 2019-09-23]. Dostupné z: http://www.umel.feec.vutbr.cz/bdts/index.php/embedded-systemy/vyvojove-modely

[8] Softwaretestingfundamentals.com: unit-testing [online]. [cit. 2019-09-22]. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/

[9] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: Průvodce testováním. Praha 4: Albatros Media, 2013. ISBN 978-80-251-3816-8.

[10] Professionalqa.com: test-analysis. Professionalqa [online]. [cit. 2020-01-13]. Dostupné z: http://www.professionalqa.com/test-analysis

[11] Softwaretestingmentor.com: what is test analysis. Softwaretestingmentor [online]. [cit. 2020-01-13]. Dostupné z: https://www.softwaretestingmentor.com/what-is-test-analysis/

[12] Statista.com: Proportion of budget allocated to quality assurance and testing as a percentage of IT spend from 2012 to 2019 [online]. [cit. 2020-03-13]. Dostupné z: https://www.statista.com/statistics/500641/worldwide-qa-budget-allocation-as-percent-it-spend/

[13] State of Testing Report 2019 [online]. [cit. 2019-09-23]. Dostupné z: https://www.practitest.com/resource/state-of-testing-report-2019/