Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Automatizované testování softwaru

David Pražák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

David Pražák

Informatika

Název práce

Automatizované testování softwaru

Název anglicky

Automated software testing

Cíle práce

Bakalářská práce je zaměřena na řešení automatizovaného testování softwaru v podnikovém prostředí. Hlavním cílem práce je vytvoření návrhu řešení automatizovaného testování. Dílčí cíle bakalářské práce jsou:

- deskripce problematiky automatizovaného testování software,
- analýza testovaného softwaru,
- vyhodnocení testovacího provozu navrhovaného řešení.

Metodika

Řešení bakalářské práce je založeno na studiu a analýze odborných informačních zdrojů. Vlastní řešení je realizováno formou návrhu a implementace automatizovaných testů v prostředí Selenium v konkrétním podnikovém prostředí. Na základě získaných poznatků budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Testování softwaru, automatizované testování softwaru, Selenium, řízení kvality vývoje softwaru

---

Doporučené zdroje informací

Doug Rosenberg, Matt Stephens. Testování SW řízené návrhem. Přeložil Lukáš Krejčí. Computer Press, 2011. ISBN 978-80-251-3607-2

Elfriede Dustin, Jeff Rashka, John Paul. Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley, 2008. ISBN 0-201-43287-0

Josef Myslín. Scrum Průvodce agilním vývojem softwaru. Computer Press, 2016. ISBN 978-80-251-4650-7

Petr Roudenský, Anna Havlíčková. Řízení kvality softwaru. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 19. 02. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Automatizované testování softwaru" 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 12.03.2019 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Janu Tyrychtrovi, Ph.D. za pomoc a rady při tvorbě mé bakalářské práce.

Automatizované testování software

Abstrakt

Bakalářská práce se zaměřuje na automatizované testování softwaru a způsob jeho řešení v praxi. Teoretická část práce obsahuje definici základních pojmů v oblasti testování softwaru a popis problematiky automatizovaného testování. Praktická část práce je zaměřena na vytvoření návrhu řešení automatizovaného testování v praxi. Řešení tohoto testování je realizováno jako funkční automatizované testování na základě metody Black box, která je aplikována na vlastní vzorovou nativní aplikaci vytvořenou v jazyce C#. Automatizované testy jsou vyvíjeny v nástroji Unified Functional Testing. Jejich správa je řešena na základě vlastního nástroje pro správu automatizovaných testů. Závěrem práce je vyhodnocení výsledků vytvořených automatizovaných testů.

Klíčová slova: Testování softwaru, automatizované testování softwaru, řízení kvality vývoje softwaru, Unified Functional Testing (UFT).

Automated software testing

Abstract

The bachelor thesis focuses on automated software testing and the way of its solution in practice. The theoretical part of the thesis contains a definition of basic concepts in the area of software testing and a description of automated testing. The practical part of the thesis is focused on creating a design solution for automated testing in practice. The solution to this testing is implemented as a functional, automated testing based on the Black Box method, which is applied to a custom native application created in C # language. Automated tests are developed in Unified Functional Testing tool. Their management is applied to own automated test management tool. The conclusion of the thesis is the evaluation of the results of automated tests.

Keywords: Software testing, automated software testing, software quality management development, Unified Functional Testing (UFT).

Obsah

Seznam obrázků

Seznam použitých zkratek

UFT – Unified Functional Testing

GUI – Grafické uživatelské rozhraní
MS – Microsoft
HTML - Hypertext Markup Language
CPU - Central Processing Unit
RAM - Random Access Memory
SSD – Solid-state Drive

1 Úvod

Automatizované testování softwaru patří mezi důležitou fázi vývoje softwaru a měla by mu být věnována odpovídající část vývojového procesu. Pokud je tento proces dobře nastaven, může nemalou částí přispět k vyšší výsledné úrovni kvality testované aplikace. V praxi se však můžeme často setkat s tím, že se automatizované testování neřeší vůbec, nebo se řeší pouze částečně vývojáři aplikace. To může být zapříčiněno obavou z vysokých počátečních nákladů, které ale s dlouhodobým využitím výrazně klesají. Automatizované testování je tak přínosem především tam, kde je kvalifikovaně a dlouhodobě začleněno do vývojového procesu.

Téma bakalářské práce si vybral autor z toho důvodu, protože již v současnosti řeší problematiku automatizovaného testování v IT (informační technologie) společnosti zaměřené na vývoj softwaru, a v budoucnosti se chce této problematice nadále věnovat.

Úvodem práce autor obeznámí čtenáře se základními pojmy v oblasti testování softwaru a problematikou testování softwaru. V průběhu práce se autor zaměřuje především na problematiku funkčního automatizovaného testování. Dále jsou zde řešeny otázky volby testovacích nástrojů, problematika budování testovacích prostředí a správa testů.

Praktická část je zaměřena na vytvoření návrhu řešení automatizovaného testování a jeho následné vyhodnocení. V této části je řešeno postupné zavádění funkčního automatizovaného testování nad vlastní nativní aplikací. Je vytvořeno testovací prostředí, vybrán nástroj pro správu automatizovaných testů, je provedena analýza testovaného softwaru, předvedena tvorba testovacích scénářů a vývoj automatizovaných testů. Závěr práce je zaměřen na vyhodnocení výsledků automatizovaných testů, kterými je otestována konkrétní vzorová aplikace.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce má za cíl seznámit s problematikou automatizovaného testování softwaru. Zaměřuje se na jednotlivé oblasti této činnosti a jejím hlavním cílem je vytvoření návrhu řešení automatizovaného testování.

Cílem teoretické části práce je na základě studia a analýzy odborných zdrojů definovat základní pojmy v oblasti problematiky automatizovaného testování.

Cílem praktické části práce je vytvoření návrhu řešení automatizovaného testování. Jako základ poslouží analýza testovaného softwaru, která se stane podkladem pro realizaci návrhu. Na základě praktického využití výsledné realizace budou získány podklady pro vyhodnocení testovacího provozu navrhovaného řešení.

2.2 Metodika

Teoretická část práce bude zpracována na základě studia a analýzy odborných zdrojů. Její obsah bude rozdělen na definici základního rozdělení testovacích metod dle jejich zaměření. Následně budou uvedeny jednotlivé způsoby testování. Dále bude řešena problematika automatizovaného testování a analýza testovaného softwaru. Na závěr teoretické části bude řešena správa a vyhodnocení testů.

Praktická část se bude zabývat testováním vlastní nativní aplikací vytvořené v objektově orientovaném programovacím jazyce C# [1]. Tato aplikace byla vytvořena pro testovací účely a řeší vzorový případ evidence sázek a výher v loterii. Aplikace byla vytvořena ve vývojovém prostředí MS (Microsoft) Visual Studio. Data aplikace jsou uložena v relační databázi Access [2], která byla vytvořena pomocí nástroje MS Access. K databázi se přistupuje pomocí nástroje MS Access Database Engine a data se zpracovávají pomocí SQL příkazů.

Pro otestování aplikace bude využito funkční automatizované testování metodou Black box (černá skříňka), které simuluje práci uživatele s aplikací. Pro vzorovou aplikaci budou vytvořeny jednoduché automatizované testy v prostředí UFT (Unified Functional Testing), které otestují grafickou část aplikace, nazývanou také jako GUI (grafické uživatelské rozhraní). Algoritmy automatizovaných testů budou rovněž pro lepší názornost promítnuty do grafické podoby jako vývojové diagramy. Diagramy budou vytvořeny pomocí online nástroje draw.io [13]. Vytvořené automatizované testy budou nakonec

použity pro otestování požadované funkčnosti vzorové aplikace. Na konci praktické části budou uvedeny výsledky vykonaných automatizovaných testů. Ty poslouží jako základ pro závěry bakalářské práce.

2.2.1 Black box

Black box je metoda testování, která navrhuje testovací případy založené na informacích ze specifikace požadavků (množina funkcí, které má aplikace poskytovat). Jedná se o metodu, ve které tester aplikace nemá přístup ke zdrojovému kódu testované aplikace. Aplikace je pak pro něj jakási skříňka, kterou může zkoumat pouze na povrchu. Ten je reprezentován GUI. K obsahu skříňky nemá přístup. Místo toho má k dispozici informace, že do černé skříňky lze vložit nějaké vstupy a černá skříňka vrátí nějaké výstupy zpět. Tyto informace lze získat analýzou specifikace požadavků. Z nich tester získá informace o tom, jaké funkce má testovaná aplikace poskytovat. Následně může do černé skříňky vkládat očekávané vstupy a testovat, že černá skříňka vrací požadované výstupy.

Testování černé skříňky se provádí od začátku životního cyklu softwarového projektu. Všichni testeři musí být zapojeni do tohoto cyklu od samotného začátku projektu. Při testování černé skříňky je potřeba, aby byli testeři důkladně obeznámeni s analýzou a s požadavky zákazníků (specifikace požadavků).

Pro testování metodou Black box je vhodné předem vytvořit testovací scénáře a připravit testovací data.

Testovací scénář je sada kroků popisujících akce, které mají být prostřednictvím testovacího skriptu provedeny v testované aplikaci. Dobrý testovací scénář musí být přehledný a srozumitelný. Pokud nebude tyto požadavky splňovat, může dojít k jeho nepochopení, což může zapříčinit chybné otestování aplikace. Chybným otestováním aplikace lze rozumět takový stav, po kterém nebude správně otestována požadovaná funkčnost aplikace.

Testovací data jsou množina hodnot požadovaného typu a struktury, která budou vkládána do aplikace při jejím testování. [3]

2.2.2 Nástroj Unified Functional Testing (UFT)

UFT je komerční nástroj pro tvorbu automatizovaných testů vyvíjený společností Micro Focus. Nástroj je především zaměřen na funkční a regresní testování aplikací. Nástroj umožňuje práci s technologiemi .NET, ActiveX, Java, Mobile, Visual Basic, Web, WPF a Silverlight. Nástroj je dostupný ve třech typech licencí Concurrent, Seat a Trial.

Hlavní výhodou nástroje je to, že pro tvorbu testů využívá skriptovací jazyk VBScript (Visual Basic Scripting). Vývoj automatizovaných testů je v tomto jazyce v porovnání s ostatními programovacími jazyky poměrně snadný. Dalším kladem nástroje UFT je jeho uživatelské prostředí, které je řešeno velmi jednoduše, přehledně a nevyžaduje žádné složité nastavování. Výsledky automatizovaných testů zobrazuje v přehledných reportech, které lze získat v několika podobách a formátech. Aplikace mohou být testovány jak ve standardních počítačích jako nativní či webové aplikace, tak i na mobilních přístrojích, jako jsou chytré telefony a tablety.

Nevýhodou UFT je, že ho lze provozovat pouze na počítačích s operačním systémem MS Windows. Dalšími nevýhodami jsou jeho poměrně vysoké nároky na hardwarové prostředky nutné k jeho běhu (což je ale u většiny nástrojů podobného zaměření dnes většinou obvyklé) a jeho vysoká pořizovací cena za licence.

Díky všem jeho výhodám a navzdory uvedeným nevýhodám je ale UFT jedním z nejpopulárnějších komerčních testovacích nástrojů současnosti.

Zájemci mohou UFT vyzkoušet zdarma po dobu 30 dní ve verzi trial. [4]

2.2.3 Vývojový diagram

Vývojový diagram slouží pro grafické znázornění algoritmu. Skládá se z pěti základních symbolů [5]:

- obdélník se zaoblenými rohy symbolizuje počátek nebo ukončení zpracování
- šipka určuje směr zpracování
- obdélník s popisem definuje dílčí krok zpracování
- kosočtverec značí větvení rozhodování na základě vyhodnocení podmínky
- kruh je spojkou několika šipek

Každý vývojový diagram obsahuje vždy jeden začátek a alespoň jeden konec. Pro vývojové diagramy existují tři typy struktur. Jsou to sekvence, větvení a cyklus.

Sekvence je řada kroků, které se vykonávají v bezprostřední návaznosti jeden po druhém. Každý krok může navazovat na jeden krok předchozí a může na něj navazovat jeden krok následující.

Větvení je závislé na podmínce, která větví algoritmus na několik částí. V případě, že nebude podmínka splněna, vykoná se dál jiná část algoritmu, než kdyby podmínka splněna byla.

Poslední strukturou je cyklus, kdy se provádí konkrétní akce algoritmu pořád dokola, dokud není splněna podmínka.

Výhody vývojových diagramů jsou v jejich přehlednosti a názornosti, ale algoritmus nesmí být moc složitý, jinak uvedené výhody neplatí. Mezi nevýhody vývojových diagramů patří jejich obtížná upravitelnost i neaktuálnost, pokud se algoritmus změní. [6]

3 Literární rešerše

3.1 Testování softwaru

Testování softwaru je proces, při kterém dochází k ověřování správnosti chování testované aplikace z různých pohledů, aby se ověřilo splnění všech požadovaných parametrů pro její bezproblémový rutinní provoz. Jde především o následující oblasti testování:

- Jednotkové testování;
- Funkční testování;
- Integrační testování;
- Akceptační testování;
- Testování spolehlivosti;
- Testování výkonu;
- Testování použitelnosti;
- Bezpečnostní testování.

3.1.1 Jednotkové testování

Jednotkové (unit) testování softwaru slouží pro otestování zdrojového kódu aplikace. Jedná se o jednu z nejdůležitějších částí vývoje softwaru. Toto testování je prováděno vývojáři, kteří mají za cíl ověřit korektní chování jednotlivých komponent aplikace jako jsou funkce a objekty. [14]

3.1.2 Funkční testování

Funkční testování softwaru slouží převážně pro eliminaci chyb, které vznikly neúmyslně při samotném vývoji softwaru. Samotné testování softwaru probíhá na základě podrobně sepsaných testovacích scénářů, které obsahují podrobné informace o tom, co a jak je potřeba otestovat. Testovací scénáře jsou nejčastěji uspořádány do kroků, které na sebe navazují a tester nebo automatizovaný test podle těchto kroků postupuje. Na základě výsledků testování jsme schopni říci, zda je otestovaný software připravený k předání do rutinního provozu, nebo je třeba pokračovat ve vývoji a testování. [11]

3.1.3 Integrační testování

Hlavním cílem integračního testování je otestování integračního rozhraní mezi moduly nebo spolupracujícími systémy testované aplikace. Toto testování probíhá při samotném vývoji testované aplikace. Cílem tohoto testování je odhalit chyby v testované aplikaci co nejdříve, aby mohly být opraveny před samotným finálním nasazením testované aplikace do ostrého provozu. [15]

3.1.4 Akceptační testování

Akceptační testování navazuje na integrační testování tím, že obě tyto metody mají stejný cíl a to, prokázat, že otestovaná aplikace splňuje požadavky zadané zadavatelem a je možné jí nasadit do ostrého provozu. Celé testování probíhá na základě domluvených podmínek se zadavatelem. Zadavatel je při tomto testování obeznámen s výslednou aplikací, která může být nasazena do ostrého provozu. [4]

3.1.5 Testování spolehlivosti

Testování spolehlivosti je proces, kterým se testuje spolehlivost testované aplikace. Spolehlivostí testované aplikace se rozumí, jakým způsobem se bude aplikace chovat v krizových situacích, jako jsou výskyty chyb, vytížení databáze, vytížení sítě nebo celkový výpadek systému ať už na straně běžného klienta nebo serveru. [9]

3.1.6 Testování výkonu

Testování výkonu, nazývané také jako zátěžové testování, je proces, pomocí kterého se testuje rychlost a efektivnost testované aplikace. Na základě tohoto testování jsme schopni zjistit stabilitu různých aplikací při standardní nebo velké zátěži. Testováním výkonu jsme také schopni prověřit vytíženost aplikace při běžné práci uživatele a na základě tohoto výstupu jsme schopni určit, zda by měla být aplikace lépe optimalizována, nebo zda není nutné výkonově posílit prostředí, na němž se bude aplikace provozovat. [9]

3.1.7 Testování použitelnosti

Testování použitelnosti je proces, kterým se testuje aplikace z pohledu uživatele a zadaných požadavků. Mezi hlavní cíle tohoto testování patří uspokojení požadavků uživatele na práci v testované aplikaci. Pro tento typ testování se využívá manuálního testování, které mají na starosti vybraní uživatelé. Na základě jejich poznatků a zhodnocení

jsou vypracovány výsledky testování. Mezi další cíle patří testování dle zadaných požadavků, zda aplikace ve výsledku splňuje to, pro co byla navržena a také zda je pro uživatele dobře ovladatelná (zda je uživatelsky přívětivá). [11]

3.1.8 Bezpečnostní testování

Bezpečnostní testování slouží pro ověření bezpečnosti testované aplikace. Mezi bezpečností testování patří kontrola různých nastavení sítě a zabezpečení kódu. Bezpečností testování se využívá převážně pro aplikace, ke kterým se uživatel přihlašuje za pomocí uživatelských účtů s heslem. [11]

3.2 Rozdělení podle znalosti kódu

Z pohledu znalosti kódu testované aplikace se využívají následující metody testování [7]:

- White box;
- Black box (kapitola 2.2.1);
- Grey box.

3.2.1 White box

White box je termín pro testování, které se provádí během vlastního vývoje aplikace. Toto testování provádí většinou přímo vývojář aplikace, protože je k němu zapotřebí znalost a přístup k programovému kódu a vývojovému prostředí. Těmto testům se říká jednotkové (unit) testy. Jednotkové testy se většinou vykonávají jako automatizované testy a není zapotřebí, aby testovaná aplikace byla jako celek funkční. Také není nutné, aby ten, kdo testy vykonává, znal fungování aplikace jako celku. [3]

3.2.2 Grey box

Grey box testy jsou kombinací obou předešlých způsobů, tedy způsobu White box a způsobu Black box. [7]

3.3 Způsob provádění testů

Z pohledu provádění testů existují následující způsoby [7]:

- Manuální testování;
- Automatizované testování;

- Semiautomatické testování.

3.3.1 Manuální testování

Manuální testování patří mezi jednu z nejčastěji využívaných metod testování softwaru z důvodu nízkých nákladů pro jeho zavedení. Dlouhodobé manuální testování může ale náklady na testování výrazně navyšovat, a proto je vždy dobré zvážit, zda po ukončení základního vývoje nevyužít rovněž automatizované testování. Pro rozhodnutí, zda a v jakém rozsahu testování automatizovat, je ale důležité zvážit mnoho okolností, především možnosti společnosti pro zavedení automatizovaného testování a vhodnost aplikace k automatizovanému testování. Manuální testování je prováděno přímo konkrétními testery, kteří na základě testovacích scénářů, analytických dokumentů a dokumentace defektů přímo v grafickém uživatelském prostředí aplikace ověřují její bezchybnost a požadovanou funkcionalitu. K tomuto způsobu testování je nutná funkční testovaná aplikace a testující osoby musí mít alespoň základní znalosti o jejím fungování. Manuální testování se uplatňuje především v období dynamického vývoje testované aplikace pro testování nových funkcionalit a následně pro udržení kvality vývoje při kontrole dalších nových funkcionalit a ověřování funkčnosti opravených defektů. Výhodou je jeho značná flexibilita a rychlost zahájení. Nevýhodou je jeho pomalost a u větších aplikací i personální náročnost. [8]

3.3.2 Automatizované testování

Automatizované testování je proces, při kterém se testovaná aplikace testuje pomocí tzv. skriptů, které byly vytvořeny za pomoci nástroje pro tvorbu automatizovaných testů. Tvorba takovýchto skriptů je závislá na testovacích scénářích, které obsahují konkrétní postupy pro otestování požadované funkčnosti testované aplikace. Úkolem těchto skriptů je otestování určité části testované aplikace bez nutnosti zásahu člověka. Jelikož je automatizované testování závislé na životním cyklu vývoje testované aplikace, je vhodné jej začít využívat až po dokončení základního vývoje testované aplikace. Důvodem je především to, aby se testovací skripty nemusely stále přizpůsobovat častým změnám testované aplikace. Při zavádění automatizovaného testování je potřeba si uvědomit, že počáteční náklady na zavedení jsou poměrně značné a jejich ekonomický přínos se projeví často až po několika letech. Z tohoto důvodu se v některých případech, například u krátkodobých projektů, nevyplatí automatizované testování zavádět vůbec. Hlavní

výhodou automatizovaného testování je jeho rychlost a neomezená opakovatelnost. V dlouhodobém horizontu může být přínosem i ekonomická úspora. Nevýhodou jsou vysoké počáteční náklady a čas pro vývoj testovacích skriptů. [8]

3.3.3 Semiautomatické testování

Semiautomatické testování je proces, při kterém se testovaná aplikace testuje kombinací manuálního a automatizovaného testování. Tím, že je software otestován oběma způsoby, dochází k možnosti odhalení maximálního možného výskytu problémů v daném software. Tento způsob je v praxi nejčastěji využívaným způsobem funkčního testování. Nevýhodou semiautomatického testování jsou vysoké náklady vynaložené na tým pro manuální a automatizované testování. [7]

3.4 Problematika automatizovaného testování

Jednou ze základních otázek automatizovaného testování je „Co všechno automatizovat“. Při testování softwaru se musíme rozhodnout, ve kterých případech je výhodné využití automatizovaného testování, a ve kterých je naopak vhodnější využít manuálního testování softwaru. Automatizované testy jsou vytvářeny především pro otestování takových částí testované aplikace, kde se již vývoj aplikace usadil a které se budou testovat opakovaně. U částí aplikace, kde se vývoj ještě nepřiblížil cílovému stavu, je vhodné s automatizovaným testováním počkat. V takovýchto případech je vhodné využít manuálního testera, který na základě testovacího scénáře dosud „živé“ části testované aplikace otestuje manuálně. Dále se můžeme setkat s případy, které pomocí automatizovaných testů otestovat nelze, nebo by bylo testování softwaru pomocí takovýchto testů příliš složité.

Další otázkou bývá „Jakým nástrojem se bude automatizovaně testovat“. Na trhu se vyskytuje několik nástrojů, které se dají využít pro vytváření automatizovaných testů. Při výběru konkrétního nástroje pro tvorbu automatizovaných testů se musíme zaměřit hlavně na to, aby testovací nástroj podporoval technologie, které testovaná aplikace využívá. Při výběru konkrétního nástroje pro tvorbu automatizovaných testů musíme předem zanalyzovat testovanou aplikaci a na základě zjištěných skutečností vybrat konkrétní nástroj pro tvorbu automatizovaných testů. Důležitou informací při výběru nástroje může být například skutečnost, zda je testovaná aplikace nativní nebo webová. Pro tvorbu automatizovaných testů jsou na trhu k dispozici jak nástroje určené pro nativní aplikace,

tak i nástroje určené pro webové případně mobilní aplikace. Mezi nástroje, které se využívají pro tvorbu automatizovaných testů, patří například [4]:

Placené:

- Unified Functional Testing (Micro Focus);
- Rational Functional Tester (IBM);
- Visual Studio Test Professional (Microsoft);
- TestComplete (SmartBear Software);
- Silk Test (Micro Focus);
- RIAtest (Cogitek).

Neplacené:

- Selenium;
- Katalon Studio;
- Watir;
- MonkeyTalk (zaměřeno především na automatizaci mobilních aplikací).

3.5 Analýza testovaného softwaru

Na začátku testování každé aplikace je vždy nutné se s aplikací dobře seznámit a provést její důkladnou analýzu. Zjistit její architekturu, použité vývojové nástroje, její požadavky na provoz apod. Dále se seznámit s jejím ovládáním a pochopit vlastnosti jejího chování, kterým je zapotřebí se při testování přizpůsobit. Tato analýza by měla vyřešit především následující otázky:

- Jaké jsou požadavky pro provoz aplikace;
- Jak aplikaci testovat.

Na základě těchto skutečností se následně navrhuje testovací prostředí a testovací strategie. Testovací prostředí se může skládat z jednoho nebo několika počítačů či serverů, síťové infrastruktury, datového úložiště apod. V dnešní době se velmi často pro testování využívají virtualizované prostředí. Testovací strategie navrhuje způsob, rozsah, dobu a další parametry pro testování.

3.5.1 Požadavky na provoz aplikace

Zjišťují se potřebné prostředky pro provozování testované aplikace, jako jsou požadavky na software a hardware, požadavky na licence, požadavky na potřebné konfigurace apod.

Software požadavky, lze rozdělit na dvě základní skupiny.

První zahrnuje software, jako jsou operační systémy (Windows, Linux, IOS, Android apod.), případný software pro virtualizaci (VMware, Virtual Box, Hyper-V apod.), aplikační servery (JBoss, WebLogic, Apache Tomcat apod.) webové servery (Apache, IIS apod.), databázové servery (Oracle, SQL server, MySQL apod.).

Druhá skupina pokrývá software potřebný pro vlastní testování. Sem patří např. webové prohlížeče (Google Chrome, Mozilla Firefox, Opera, Internet Explorer apod.), různé runtime potřebné pro běh aplikace (JRE, .Net Runtime apod.).

Požadavky na hardware musí řešit především dostatečný výkon, který musí zvládnout bezproblémový běh jak testované aplikace, tak případně všech dalších potřebných aplikací a služeb jako např. webového prohlížeče, webového, aplikačního nebo databázového serveru, síťových, monitorovacích, virtualizačních a jiných služeb apod. Speciální kapitolou je pak hardware v případě zátěžových testů, kdy je pro vygenerování potřebné zátěže často nutné počítat s dostatečně vysokým výkonem testovacího prostředí. [10]

3.5.2 Testování aplikace

Na základě zjištěných informací o aplikaci (velikost aplikace, použitá technologie, stupeň dokončenosti, předpokládaný další vývoj, životnost aplikace apod.) lze rozhodnout o vhodném způsobu testování.

V případě volby manuálního testování by mělo být testovací prostředí ještě rozšířeno o nástroje pro management testování. Jde především o nástroje pro evidenci scénářů (Testlink, Confuence apod.), nebo nástroje na evidenci požadavků a řešení incidentů (Jira apod.). Tyto nástroje je nutné přičíst k hardware a software požadavkům testovacího prostředí, pokud již nejsou zavedeny a používány i pro jiné činnosti (např. vývoj, technická podpora apod.).

V případě volby automatizovaného testování, musí být na základě analýzy testovaného software vyřešeny následující otázky:

- Jaký testovací nástroj bude použit,

- Jak se budou v automatizovaných testech identifikovat jednotlivé komponenty.

Zvážení požadavků na testovací prostředí pro automatizované testování s přihlédnutím na další náklady spojené s vývojem a údržbou automatizovaných testů by mělo nastínit případnou ekonomickou výhodnost použití automatizovaných testů.

Součástí kvalitního testovacího prostředí musí být kvalitní testovací nástroj. Pro výběr správného nástroje pro tvorbu automatizovaných testů je potřeba identifikovat typ testované aplikace. V praxi se můžeme setkat s nástroji pro tvorbu automatizovaných testů k nativním aplikacím, webovým aplikacím, případně s nástroji, které si umí poradit s oběma typy (UFT, Katalon, Selenium, Protractor apod.). Tyto nástroje se většinou doplňují nástroji pro evidenci testů a testovacích scénářů, automatizované spouštění testů a zobrazování testovacích reportů (Jenkins, Micro Focus Quality Center apod.). Se všemi těmito nástroji je rovněž nutné počítat při zjišťování hardware a software požadavků testovacího prostředí (samozřejmě opět pokud některé z nich již nejsou zavedeny pro jiné účely). Zvláště u testovacího nástroje může jít o nemalé výkonové požadavky.

Velmi důležitým úkolem při použití automatizovaného testování je ověření možnosti identifikace použitých komponent tvořících GUI testované aplikace. Zde se zjišťuje, jak jednotlivé komponenty reagují na své ovládání pomocí testovacího nástroje a jaké identifikátory bude nejvhodnější pro jednotlivé komponenty používat, případně se na jejich zavedení domluvit s vývojem. Rovněž je vhodné určit oblasti aplikace, pro které bude vhodné vytvořit funkce. Funkce jsou většinou nevelké části testovacího kódu, který řeší zpracování určitých informací, které se v aplikaci vícekrát na různých místech opakují. Příkladem může být například otestování vkládání osoby, adresy, bankovního spojení apod. V testu se pak taková funkce může jednou použít např. pro vložení žadatele, příště jako adresáta, nebo kontaktní osoby apod. [4]

3.6 Správa testů

Celý testovací proces je poměrně složitý mechanismus, který je třeba dobře nastavit, organizovat a kontrolovat. Všechny činnosti okolo tvorby, úprav, evidence, plánování, spouštění a vyhodnocování testů a testovacích reportů nazýváme management testování. V rámci něj se evidují testovací projekty, testovací požadavky, testovací případy, testovací scénáře a skripty, testovací plány a verze, výsledky testů atd. V případě automatizovaných

testů je součástí managementu rovněž jejich automatické spouštění a evidence testovacích reportů (výsledků proběhlých testů).

Pro management testování se používají specializované aplikace, které jsou pro tuto činnost speciálně vyvinuty. Tyto aplikace existují jak v komerčních placených verzích, tak i v open source neplacených a nabízí různou funkčnost. V zásadě lze říci, že pro manuální funkční testování si lze velmi dobře vystačit s neplacenými aplikacemi. V případě automatizovaného testování je vhodnější pořídit některou z placených verzí, případně si vyvinout vlastní firemní řešení. Aplikace pro test management se z pohledu nasazení dají rozdělit do dvou skupin.

3.6.1 Samostatné aplikace

Jedná se často o webové aplikace, jejich součástí je webový server, databáze a uživatel s nimi pracuje prostřednictvím webového rozhraní. Aplikaci si tedy pouští ve webovém prohlížeči. Celé toto řešení bývá postaveno na open source komponentech, a proto nebývá finančně příliš náročné. Často je celé řešení zcela zdarma, nebo se platí pouze aplikace, která pro svůj běh využívá open source software. Příkladem takového řešení pro manuální testování, které lze provozovat zcela zdarma je například aplikace Testlink. Pro automatizované testování je k dispozici třeba Jenkins, který je také zdarma jako open source. Mezi komerční aplikace patří například Quality Center od společnosti Micro Focus. [12]

3.6.2 Pluginy do jiných systémů

Jedná se o programový doplněk do jiné aplikace, který rozšiřuje její funkčnost o funkci managementu testování. Příkladem jsou například pluginy Kanoah Tests, nebo PractiTest for JIRA jako rozšíření aplikace Jira od společnosti Atlassian.

Druhý pohled na tyto aplikace, který je rovněž dělí do dvou skupin, je dle jejich funkčnosti:

3.6.3 Aplikace pro evidenci testovacího procesu

Tyto aplikace obsahují méně nebo více podrobné testovací workflow a slouží převážně k evidenci testovacího procesu. K těmto aplikacím patří již zmiňovaný Testlink.

Tyto aplikace nemají podporu pro automatizované testování. Neumí sami spouštět automatizované testy.

3.6.4 Aplikace pro kompletní test management

Tyto aplikace většinou poskytují obdobnou funkčnost jako aplikace v předešlém bodu, ale navíc mají funkce pro evidenci, spouštění a vyhodnocování automatizovaných testů. Mezi tyto aplikace patří zmiňovaný Jenkins nebo Quality Center od společnosti Micro Focus.

Kvalitní nástroj pro management testovacího procesu je nedílnou součástí testování software. Proto je nutné počítat s případnými náklady při jeho pořizování a to především, rozhodneme-li se testovat automatizovaně, protože nástroje s touto funkčností jsou převážně komerční a poměrně nákladné.

3.7 Vyhodnocení testů

Výsledkem automatizovaného testu je výstup, který poskytuje konkrétní informace o průběhu testu a stavu testované aplikace. Tento výstup se nazývá testovací report, a kromě různě podrobných informací o průběhu testu obsahuje především jeho výsledek.

Žádaným, a nakonec i požadovaným výsledkem je stav, kdy jsou všechny příkazy testu vykonány bez chyby. V takovémto případě je celý test považován za úspěšný a testovaná část aplikace za ověřenou a funkční.

Naproti tomu nežádoucím výsledkem skončeného testu je generovaná chyba. Ta může mít několik projevů a příčin, které mají různé způsoby řešení.

Z hlediska projevů můžeme chyby rozdělit na trvalé a náhodné.

- Náhodné chyby vznikají například tím, že testovací nástroj nedokázal identifikovat objekt GUI, nebo že došlo ke krátkodobému výpadku síťového spojení apod. Pokud tedy neexistuje zjevná příčina chyby testu, je nutné test s případnými identickými testovacími daty spustit znovu a pokud se již chyba neopakuje, lze test považovat za úspěšný.
- Trvalé chyby testu jsou takové, které jsou generovány stejným příkazem testu a se stejnou příčinou i při opakovaném spuštění automatizovaného testu.

Trvalé chyby mohou mít dvě příčiny, a to chybu aplikace a chybu testu.

- Pokud se jedná o chybu aplikace, je nutné chybu podrobně zdokumentovat (v jaké části aplikace chyba vzniká, jak lze chybu navodit, jaká byla použita testovací data, jaký error log byl generován apod.). Následně je nutné tyto informace předat do vývoje. Po opravě chyby a nasazení nové verze aplikace se provede nové otestování.
- V případě, že je chyba způsobena testem (došlo k takové změně aplikace, která si vyžaduje úpravu automatizovaného testu), upraví se chybná část testu. Novým testem se poté provede nové otestování.

V obou případech popisovaných chyb je pro testera důležitým pomocníkem testovací report. V něm nalezne místo vzniku chyby a příkaz, který chybu vyvolal. Následně může test spustit ve vývojovém nástroji a zaměřit se na problematické místo testované aplikace. V případě chyby aplikace tak získá podklady, které zdokumentuje a předá vývoji. V případě chyby testu získá její přesné místo a může test přímo opravit.

4 Vlastní práce

Vlastní práce je zaměřena na vytvoření návrhu řešení automatizovaného testování, který je realizován na základě vlastní nativní aplikace. Analýza GUI a testování této aplikace je řešeno pomocí nástroje UFT. Závěrem vlastní práce je vyhodnocení testovacího provozu navrhovaného řešení formou vyhodnocení reportů provedených automatizovaných testů.

4.1 Testovací prostředí

Na úvod řešení automatizovaného testování je potřeba připravit testovací prostředí. To bylo navrženo tak, aby minimalizovalo náklady na jeho vybudování a provoz, a zároveň plně vyhovělo požadavkům na jeho výkon. Jako mateřský systém byl použit standardní počítač s operačním systémem Windows 10.

- CPU – Quad Core 3,5GHz;
- RAM – 8 GB
- SSD – 250 GB.

Na tomto počítači byl nainstalován virtualizační nástroj Oracle VM VirtualBox. Pomocí tohoto nástroje byl vytvořen virtuální počítač s operačním systémem Windows 10 s parametry:

- CPU - Dual Core;
- RAM – 4 GB;
- HDD – 50 GB.

Virtuální počítač byl umístěn na SSD mateřského systému. Po instalaci operačního systému byly provedeny veškeré jeho aktualizace. Následně byla v tomto virtuálním počítači nainstalována testovaná aplikace, testovací nástroj UFT a nástroj pro správu testů OKtesting, jehož součástí je web server Apache a databáze MySQL. Celé testovací prostředí se bude spouštět vždy pouze v okamžiku testování a po jeho ukončení se opět vypne.

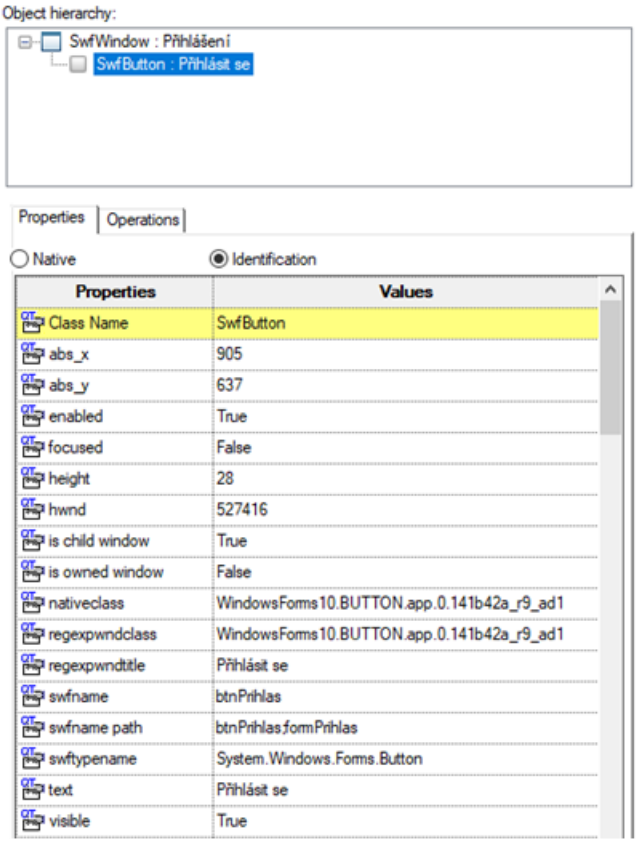
4.2 Správa testů

Pro správu testů byl využit vlastní nástroj nazvaný OKtesting. Tento nástroj umožňuje kompletní evidenci testovacích scénářů a testovacích skriptů. Dále nabízí manuální spouštění automatizovaných testů jednotlivě i hromadně (po sadách testů), nebo automaticky na základě časového plánu. Výsledky testů jsou zobrazeny přehledně ve frontě testů, kde je možné sledovat proběhlé i dosud čekající testy. Po ukončení každého testu je možné zobrazit protokol o průběhu testu.

4.3 Analýza testovaného softwaru

Před zahájením vývoje automatizovaných testů je nutné provést analýzu testované aplikace. Hlavním cílem analýzy testovaného softwaru je seznámit se s celkovou funkčností aplikace. Jakmile je tester obeznámen s funkcemi aplikace, je možné se zaměřit na řešení identifikace GUI objektů testované aplikace a zvolení nejvhodnějšího způsobu jejich využívání.

Pro identifikaci GUI objektů je v UFT obsažen nástroj Object Spy, pomocí kterého lze zobrazit jednotlivé identifikační údaje konkrétního objektu testované aplikace. Za pomoci těchto identifikačních údajů jsou následně jednotlivé objekty v testech ovládány. Při použití nástroje Object Spy na objekt tlačítko Přihlásit testované aplikace se zobrazí tabulka s identifikačními údaji tohoto objektu (Obrázek č. 1).



Obrázek č. 1: Nástroj Object Spy (vlastní zpracování)

Každý objekt obsahuje několik identifikačních údajů jako jsou, velikost, název a index objektu, nebo vlastnosti typu enabled, visible apod. Před zahájením tvorby automatizovaných testů je zapotřebí identifikovat veškeré objekty, které budou v automatizovaném testu použity a zvolit nejvhodnější identifikační informaci pro daný objekt.

V testované aplikaci byly nalezeny tyto objekty:

- SwFWindow – okno aplikace
- SwFEdit – pole pro vložení textu
- SwFButton – tlačítko aplikace
- WinButton – tlačítko obecného hlášení
- SwFComboBox – roletové menu
- Dialog – dialog

Pro identifikaci objektů SwFWindow, Dialog a WinButton byl použit index objektu. Identifikace ostatních aplikačních objektů byla provedena dle informace swfname.

4.4 Testovací scénáře

Vývoj automatizovaných testů je závislý na testovacích scénářích, které obsahují konkrétní postup pro otestování požadované funkčnosti testované aplikace.

Seznámením se s celkovou funkčností aplikace bylo zjištěno, že aplikace byla navržena tak, že práce v ní je rozdělena na dvě role, a to role běžného uživatele a role administrátora. Administrátorem je člověk, který má možnost evidence nového losování, vkladů a zakládání nových uživatelů. Od administrátora se na základě tohoto oprávnění předpokládá, že je člověkem, který se stará o veškeré finance v rámci sázení. Běžný uživatel má přístup do aplikace pouze v režimu prohlížení. Na základě těchto poznatků je patrné, že mezi základní funkčnosti aplikace patří:

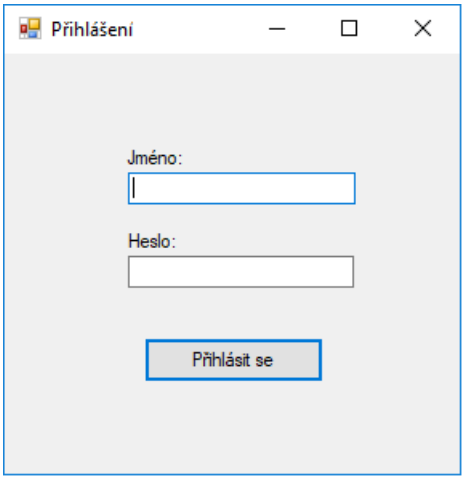
- Přihlášení do aplikace
- Založení nového losování
- Založení nového vkladu
- Založení nového uživatele

Jelikož se jedná o základní funkce aplikace, je v případě automatizovaného testování nezbytně nutné vytvořit pro tuto funkčnost automatizované testy.

4.4.1 Přihlášení do aplikace

První testovací scénář je zaměřen na otestování formuláře pro přihlášení (Obrázek č. 2). Pro přihlášení do aplikace jsou potřeba údaje jméno a heslo. Testovací scénář pro otestování takového formuláře by mohl vypadat následovně:

1. Spustit aplikaci.
2. Přihlásit se do aplikace:
    - Jméno: Lukas
    - Heslo: lukas
3. Stisknout tlačítko Přihlásit se.
4. Ukončit aplikaci.

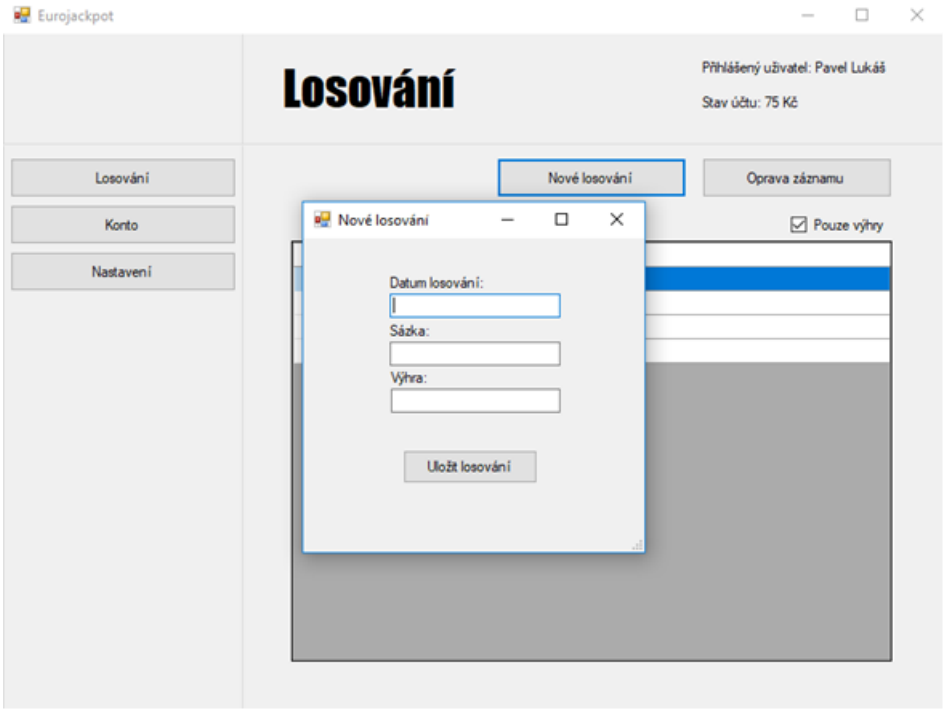


Obrázek č. 2: Přihlašovací formulář aplikace (vlastní zpracování)

4.4.2 Založení nového losování

Druhý testovací scénář je zaměřen na evidenci nového losování (Obrázek č. 3). Pro otestování funkčnosti založení nového losování jsou nutné údaje datum losování, sázka a výhra. Testovací scénář pro takovouto funkčnost bude vypadat následovně:

1. Spustit aplikaci.
2. Přihlásit se do aplikace pod uživatelem:
   - Jméno: Lukas
   - Heslo: lukas
3. Stisknout tlačítko Přihlásit se.
4. Otevřít si modul Losování stisknutím tlačítka Losování.
5. Stisknout tlačítko Nové losování.
6. Na dialogu Nové losování zadat:
   - Datum losování: 1.1.2019
   - Sázka: 500
   - Výhra: 0
7. Uložit losování.
8. Potvrdit hlášení o úspěšném založení losování.
9. Ukončit aplikaci.



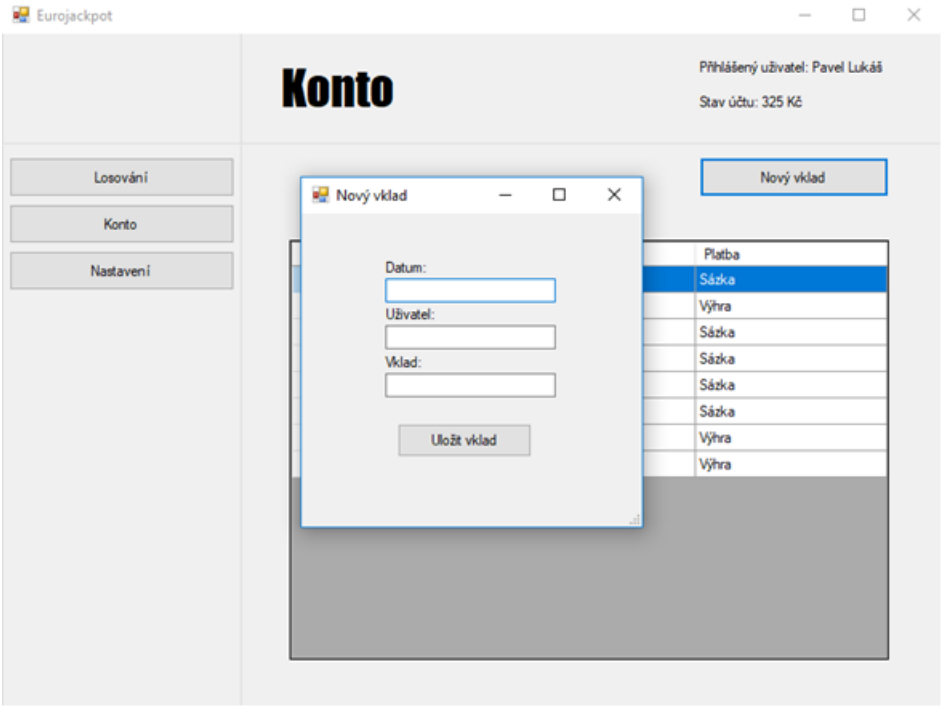
Obrázek č. 3: Formulář Nové Losování (vlastní zpracování)

4.4.3 Založení nového vkladu

Třetí testovací scénář je zaměřen na evidenci vkladu uživatele (Obrázek č. 4). Veškeré peníze, které obdrží administrátor od uživatelů na sázení, eviduje pomocí nových vkladů v aplikaci. K zaevidování vkladu jsou potřeba údaje: datum, kdy administrátor obdržel peníze na účet. Uživatel, který peníze zaslal. A výši vkladu, kterou reprezentuje částku zaslaná administrátorovi na účet. Testovací scénář pro založení nového vkladu by vypadal následovně:

1. Spustit aplikaci.
2. Přihlásit se do aplikace pod uživatelem:
   - Jméno: Lukas
   - Heslo: lukas
1. Stisknout tlačítko Přihlásit se.
2. Otevřít si modul Konto stisknutím tlačítka Konto.
3. Stisknout tlačítko Nový vklad.
4. Na dialogu Nový vklad zadat:
   - Datum: 1.1.2019
   - Uživatel: Pavel
   - Vklad: 200

5. Stisknout tlačítko Uložit vklad.
6. Potvrdit hlášení o úspěšném založení nového vkladu.
7. Ukončit aplikaci.



Obrázek č. 4: Formulář Nový vklad (vlastní zpracování)

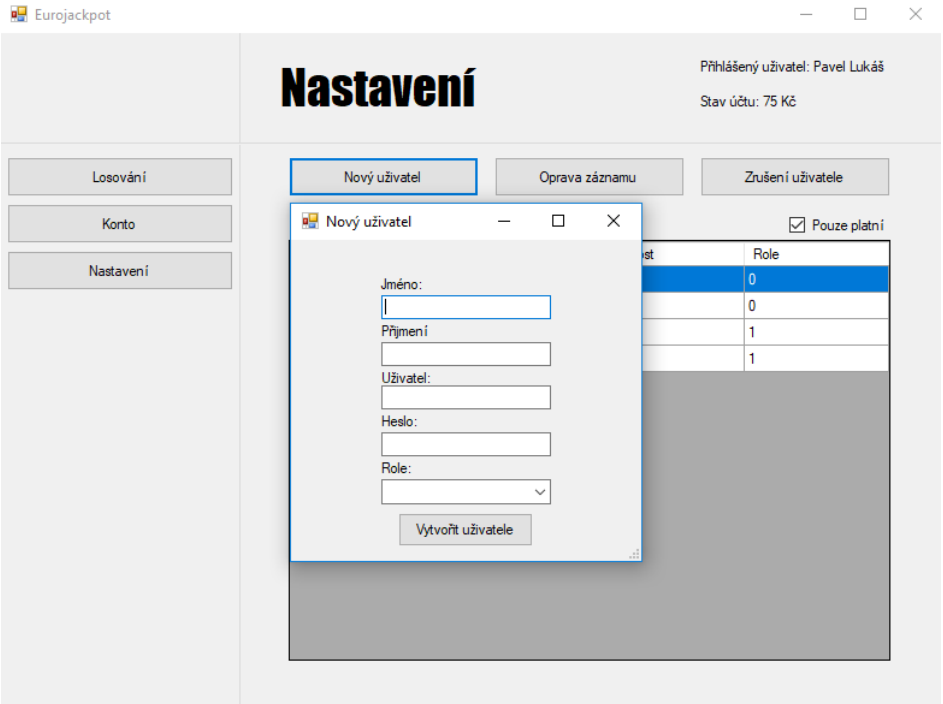
4.4.4 Založení nového uživatele

Poslední funkčností aplikace je evidence uživatelů aplikace (Obrázek č. 5). Pro zaevidování uživatele do aplikace je vyžadováno: jméno uživatele, příjmení uživatele, uživatelské jméno, uživatelské heslo a role uživatele, která je rozdělena na běžného uživatele nebo administrátora. Testovací scénář na založení Administrátora vypadá následovně:

1. Spustit aplikaci.
2. Přihlásit se do aplikace pod uživatelem:
   - Jméno: Lukas
   - Heslo: lukas
3. Stisknout tlačítko Přihlásit se.
4. Otevřít si modul Nastavení stisknutím tlačítka Nastavení.
5. Stisknout tlačítko Nový uživatel.
6. Na dialogu Nový uživatel zadat:
   - Jméno: Petr

- Příjmení: Novák
- Uživatelské jméno: Petrn
- Heslo: petr123
- Role: Administrátor

7. Stisknout tlačítko Vytvořit uživatele.
8. Potvrdit hlášení o úspěšném založení uživatele.
9. Ukončit aplikaci.



Obrázek č. 5: Formulář Nový uživatel (vlastní zpracování)

4.5 Vývoj automatizovaných testů

V této části je předpokladem znalost GUI testované aplikace, která je součástí analýzy testovaného softwaru. Tester by tak měl být seznámen s jednotlivými objekty testované aplikace, které se budou při vývoji automatizovaných testů využívat. Dalším předpokladem je existence testovacích scénářů.

Na úvod vývoje automatizovaných testů je potřeba vyřešit způsob spouštění aplikace z konkrétních testů. Pro spuštění testované aplikace slouží parametrizovaný příkaz SystemUtil.Run, kterému se v parametru předává cesta k požadované aplikaci. Použití tohoto příkazu pro spuštění testované aplikace, která je umístěna ve složce AT na disku C:\ vypadá následovně:

```
SystemUtil.Run "C://AT/Eurojackpot.exe"
```

Po vyřešení spouštění aplikace se začíná pracovat s dostupnými GUI objekty. Prvním objektem, je objekt SwFWindow (standardní aplikační formulář). Tento objekt slouží k identifikaci formuláře nebo dialogu, na kterém jsou umístěny objekty, které budou dále v testu využívány. Úvodní formulář pro přihlášení obsahuje pole pro vložení jména uživatele. Prvním krokem tedy bude zjistit identifikaci formuláře pro přihlášení. Následně se budou hledat identifikace pro další potřebné objekty na formuláři. V části analýzy testovaného softwaru bylo uvedeno, že pro identifikaci objektu SwFWindow a Dialog bude použita indexace a pro ostatní objekty swfname. Výsledný kód vypadá následovně:

```
SwfWindow("index:=0").SwfEdit("swfname:=txtJmeno")
```

Aby nemusela být identifikace objektu SwFWindow uvedena u každé akce s každým objektem na formuláři, použije se příkaz With. Akce s objekty budou následně uvedeny v rámci tohoto příkazu.

```
With SwfWindow("index:=0")
     .SwfEdit("swfname:=txtJmeno")
     .SwfEdit("swfname:=txtHeslo")
End With
```

Pro vývoj automatizovaných testů budou u jednotlivých GUI objektů použity následující příkazy:

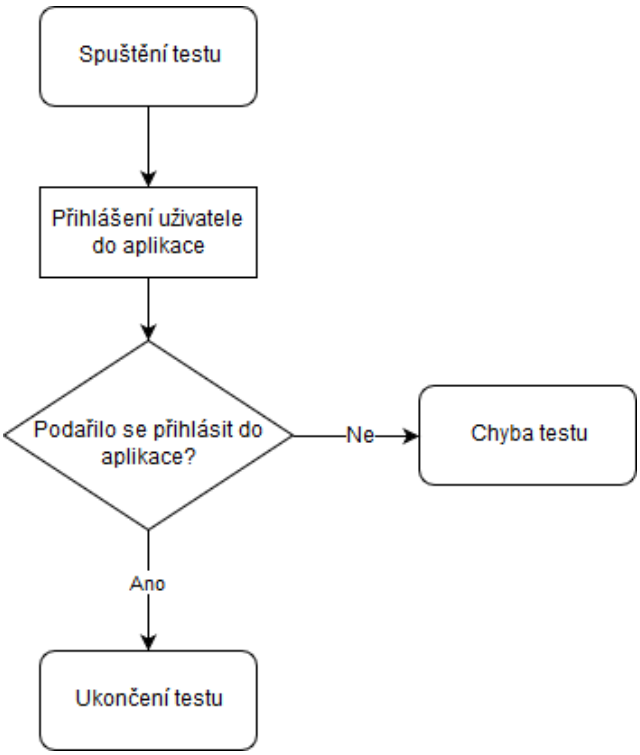
- .SwfEdit("swfname:= ").Set – vložení konkrétní hodnoty do pole
- .SwfButton("swfname:= ").Click – kliknutí na tlačítko
- .WinButton("index:= ").Click – kliknutí na tlačítko obecného hlášení
- .SwfComboBox("swfname:= ").Select – výběr hodnoty z roletového menu

4.5.1 Automatizovaný test pro přihlášení uživatele

Na základě vytvořeného testovacího scénáře „Přihlášení do aplikace“ byl vytvořen první automatizovaný test. Zdrojový kód testu je součástí přílohy (Příloha č. 1). Vytvořený test byl rovněž převeden do grafické podoby, vyjádřené pomocí vývojového diagramu (Obrázek č. 6), které znázorňuje průchod testovanou aplikací. Automatizovaný test může skončit dvěma způsoby, úspěšně a neúspěšně.

V případě úspěšného průchodu testovanou aplikací, test v grafickém vyjádření začíná symbolem Spuštění testu a po vykonání všech příkazů končí symbolem Ukončení testu.

Pokud test skončí neúspěšně z důvodu chyby, začíná takový test v grafickém vyjádření opět symbolem Spuštění testu a v okamžiku výskytu chyby končí symbolem Chyba testu.



Obrázek č. 6: Vývojový diagram testu pro přihlášení uživatele (vlastní zpracování)

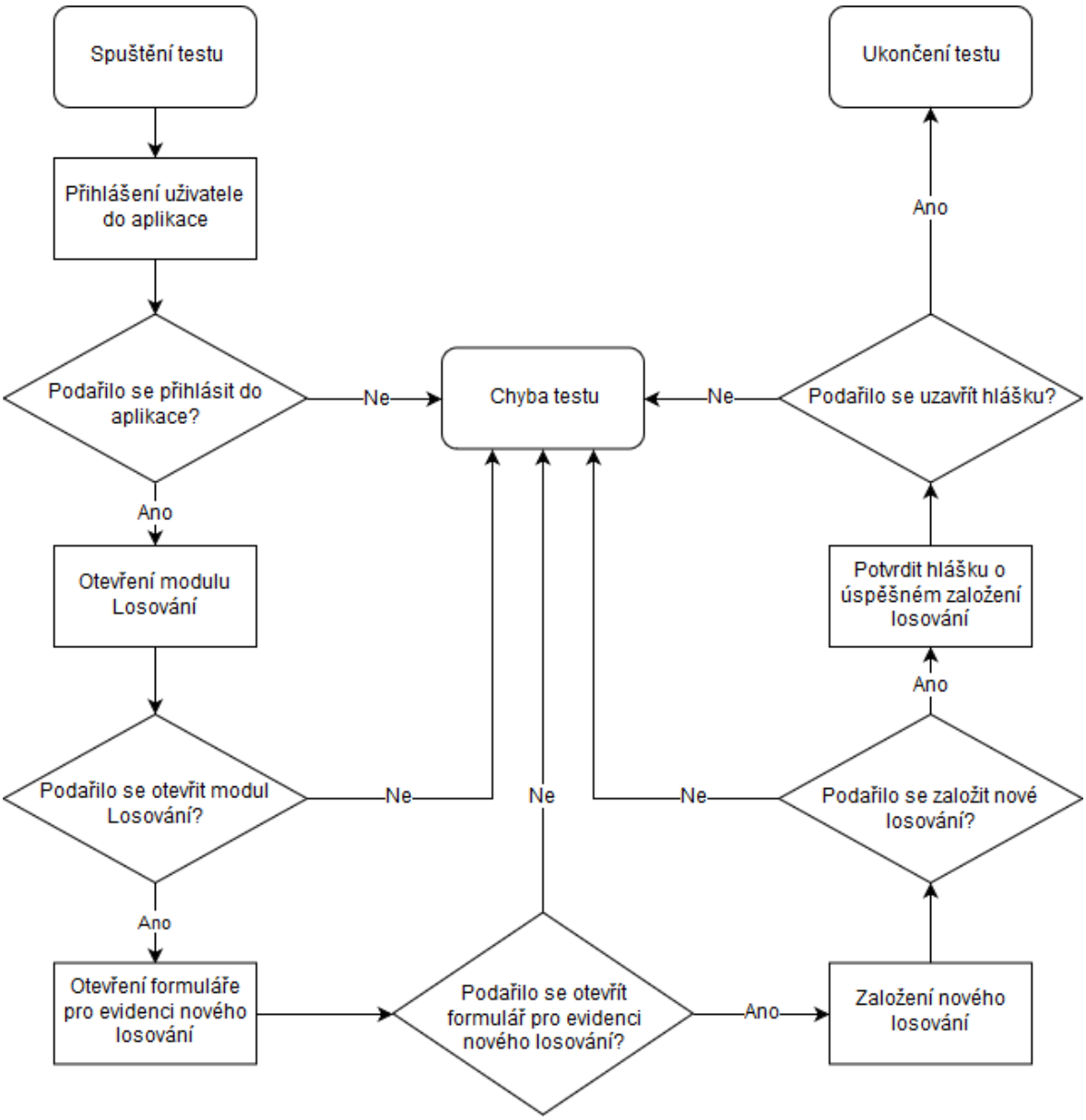
Při vývoji automatizovaných testů je výhodné v aplikaci identifikovat takové činnosti, které se opakují a vykonávají stejnou činnost. Protože vykonávají stejnou činnost, je možné ji otestovat stejným testem. Příkladem takové opakující se činnosti může být výše uvedený test na přihlášení uživatele do aplikace. Jelikož se na začátku každého testu musí uživatel přihlásit do aplikace, bude se tento test opakovat ve všech ostatních testech. Z tohoto důvodu bude výhodné z něj vytvořit funkci, která se bude v rámci testů spouštět vždy, když bude zapotřebí v testu provést přihlášení. Protože je možné se přihlašovat pod různými uživateli, funkce bude mít dva parametry, jméno uživatele a heslo uživatele. Zdrojový kód výsledné funkce pro přihlášení je součástí přílohy (Příloha č. 2).

Využití takovéto funkce v testu by pro uživatele Lukas s heslem lukas vypadalo následovně:

```
fnc_prihlaseni "Lukas", "lukas"
```

4.5.2 Automatizovaný test pro založení nového losování:

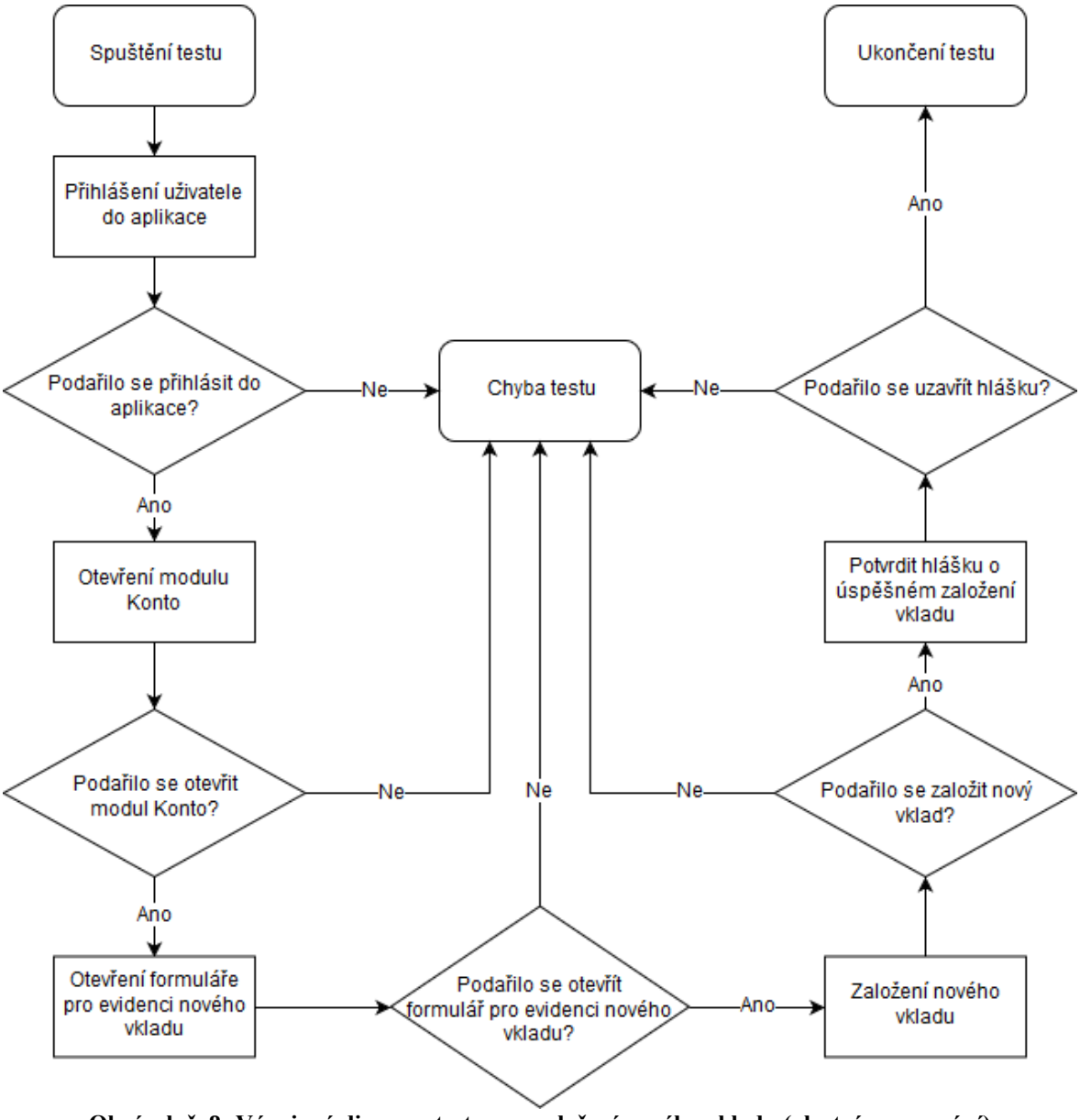
Na základě vytvořeného testovacího scénáře „Založení nového losování“ byl vytvořen automatizovaný test. Zdrojový kód testu je součástí přílohy (Příloha č. 3). Grafické znázornění vytvořeného testu pomocí vývojového diagramu (Obrázek č. 7).



Obrázek č. 7: Vývojový diagram testu pro založení nového losování (vlastní zpracování)

4.5.3 Automatizovaný test pro založení nového vkladu

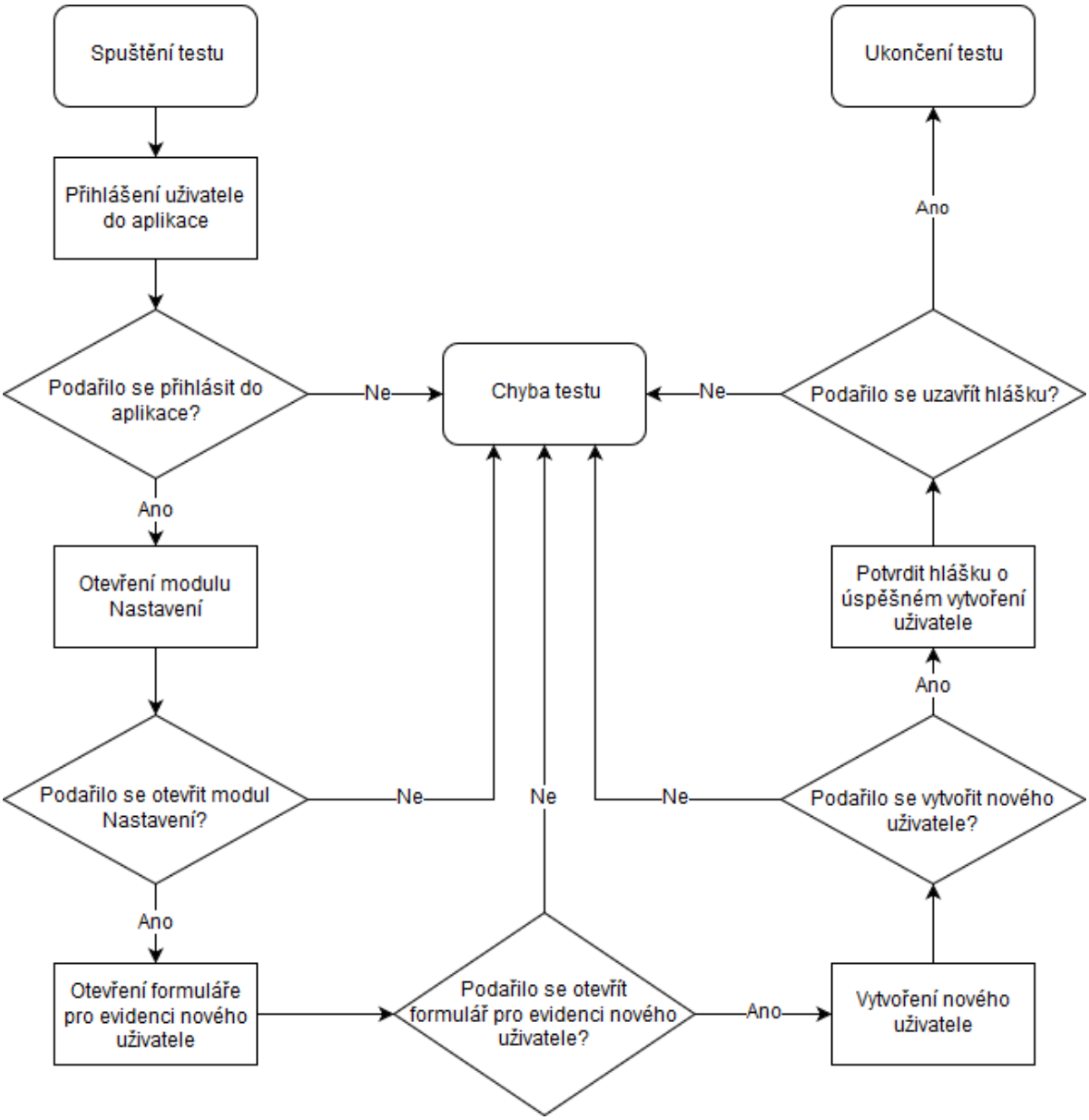
Na základě vytvořeného testovacího scénáře „Založení nového vkladu“ byl vytvořen automatizovaný test. Zdrojový kód testu je součástí přílohy (Příloha č. 4). Grafické znázornění vytvořeného testu pomocí vývojového diagramu (Obrázek č. 8).



Obrázek č. 8: Vývojový diagram testu pro založení nového vkladu (vlastní zpracování)

4.5.4 Automatizovaný test pro založení nového uživatele

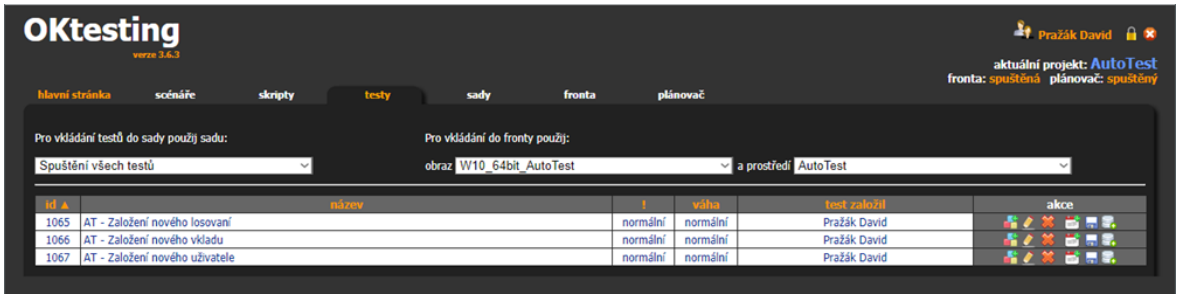
Na základě vytvořeného testovacího scénáře „Založení nového uživatele“ byl vytvořen automatizovaný test. Zdrojový kód testu je součástí přílohy (Příloha č. 5). Grafické znázornění vytvořeného testu pomocí vývojového diagramu (Obrázek č. 9).



Obrázek č. 9: Vývojový diagram testu pro založení nového uživatele (vlastní zpracování)

4.6 Evidence testovacích scénářů a testovacích skriptů

Vytvořené testy a testovací scénáře byly zavedeny do nástroje pro správu automatizovaných testů OKtesting. Testy jsou zde evidovány v rámci daného projektu v tabulce seznam testů, odkud je možné je editovat, přidávat do fronty testů, do sady testů, do plánovače testů nebo je odstranit (Obrázek č. 10).

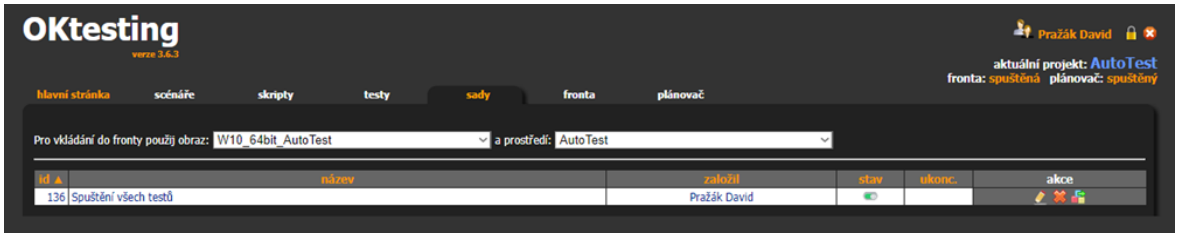


Obrázek č. 10: Přehled automatizovaných testů (vlastní zpracování)

V detailu testu je možné provádět jeho editaci a zároveň je zde dostupný scénář testu (Obrázek č. 11).

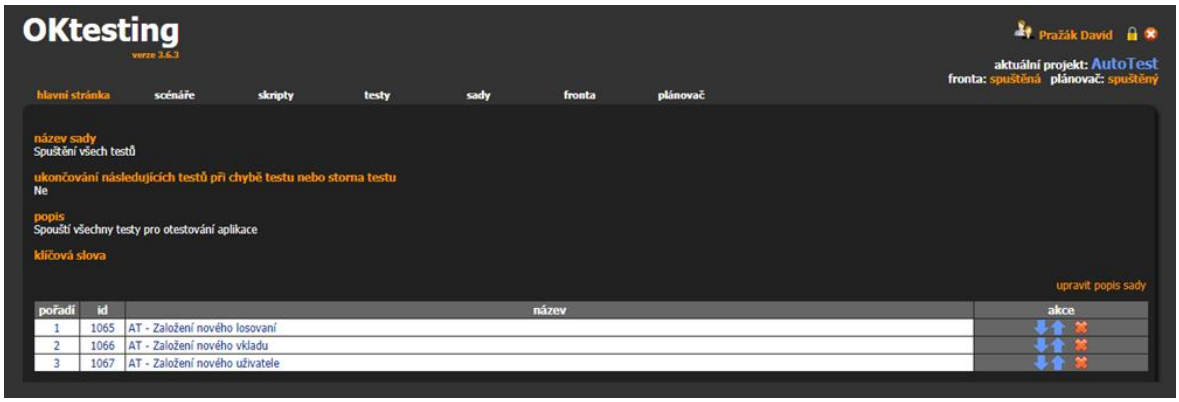
Obrázek č. 11: Evidence scénářů a skriptů (vlastní zpracování)

Jak již bylo výše ukázáno, testy je možné spouštět jednotlivě ze seznamu testů, nebo je možné je vložit do sady. Sadu pak lze vložit do fronty, čímž se do fronty vloží celý její obsah (Obrázek č. 12).



Obrázek č. 12: Evidence testovacích sad (vlastní zpracování)

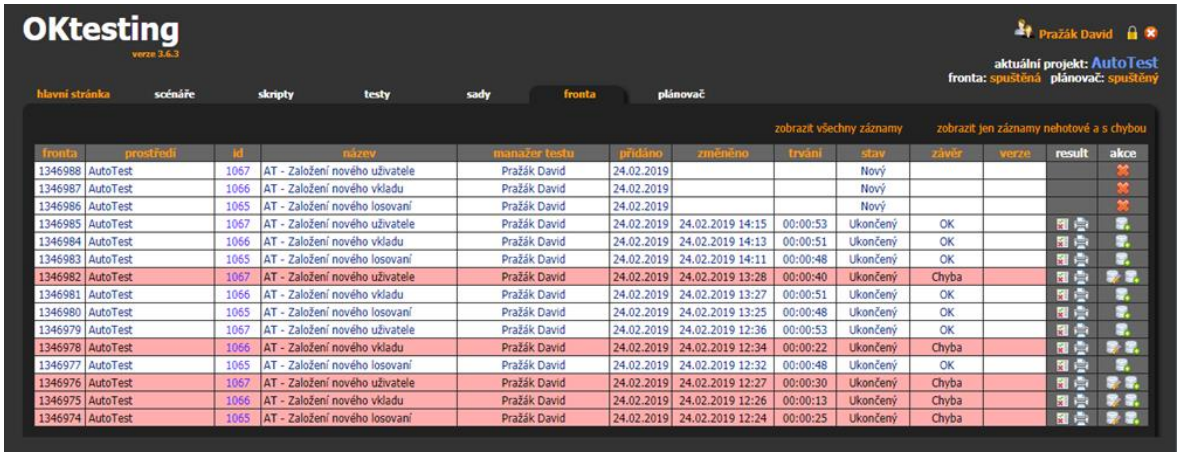
Ze stránky se seznamem sad je možné předat sadu do fronty, zrušit sadu, nebo přejít do obsahu sady, kde je možné vidět testy, které obsahuje, odstranit je ze sady, nebo změnit pořadí testů v sadě (Obrázek č. 13).



Obrázek č. 13: Evidence testů v sadě (vlastní zpracování)

Fronta testů obsahuje všechny testy daného prostředí spuštěné za posledních pět dní. Na vzorovém obrázku (Obrázek č. 14) je vidět, jak byla sada testů přidána pětkrát do fronty (první testy jsou dole, poslední nahoře).

- První kolo – všechny testy skončily s chybou.
- Druhé kolo (po opravě aplikace) – druhý test s chybou, zbytek v pořádku.
- Třetí kolo (po další opravě) – první a druhý test v pořádku, do třetího byla opravou zavlečena chyba.
- Čtvrté kolo (po další opravě) – všechny testy v pořádku.
- Zbylé testy čekají na spuštění.



Obrázek č. 14: Fronta testů (vlastní zpracování)

U proběhlých testů lze ve frontě prohlížet protokoly o průběhu testu, které jsou vygenerovány nástrojem UFT, nebo je lze vložit jako nové na začátek fronty. Neúspěšné testy lze také nastavit jako neproběhlé, aby byly znovu spuštěny.

5 Výsledky a diskuse

Nástroj UFT nabízí několik formátů reportování automatizovaných testů. Pro vyhodnocení automatizovaných testů byl vybrán formát HTML Report, který je dle mého názoru nejpřehlednější. Při každém spuštění testu se vytvoří složka RES obsahující report s informacemi o příslušném spuštění testu. Tyto složky se vytváří jako obsah složky, ve které je projekt s příslušným testem uložen. Složka RES je vždy doplněna o pořadí spuštění testu. Pokud se jedná o první spuštění testu, bude se report nabízet ve složce Res1. V této složce je pak k dispozici složka Report, v které se nachází vygenerovaný HTLM report. Soubor má název run_results.html.

Jelikož byl test pro přihlášení do aplikace převeden do funkce, která se využívá ve všech ostatních testech, byly vyhodnoceny pouze samostatné testy obsahující funkci pro přihlášení do aplikace.

5.1 Vyhodnocení testu pro založení nového losování

Na základě vytvořeného automatizovaného testu pro založení nového losování byl vytvořen HTML report, který uvádí celkové shrnutí testu a výsledky jednotlivých kroků. Z vytvořeného reportu je patrné, že test obsahuje 12 akcí s různými objekty včetně samotného spuštění a ukončení aplikace. Celkový výsledek testu je možné vidět u názvu testu (AT-Res1), který je uveden v levém horním rohu report obrazovky. Výsledek testu pro jednotlivé akce je pak uveden u jednotlivých akcí uvedených pod záložkou Test flow, nebo v pravém horním rohu report obrazovky, kde je uvedeno vyhodnocení testu dle počtu kroků, které byly bezchybné, nebo skončily s varováním, případně s chybou. Na základě vytvořeného reportu můžeme konstatovat, že test proběhl bezchybně a jeho délka byla 2 vteřiny. Obrazovka s vyhodnocením testu je součástí přílohy (Příloha č. 6).

5.2 Vyhodnocení testu pro založení nového vkladu

Vyhodnocení testu pro založení nového vkladu je podobné jako u testu pro založení nového losování. Pro test byl vygenerován report obsahující 12 akcí s objekty včetně spuštění a ukončení aplikace. Veškeré akce proběhly v pořádku a délka testu byla 3 vteřiny. Obrazovka s vyhodnocením testu je součástí přílohy (Příloha č. 7).

5.3 Vyhodnocení testu pro založení nového uživatele

Vyhodnocení testu pro založení nového uživatele dopadlo také v pořádku. Z vygenerovaného reportu je patrné, že test obsahuje 14 akcí s objekty včetně spuštění a ukončení aplikace. Celková délka testu byla 3 vteřiny. Obrazovka s vyhodnocením testu je součástí přílohy (Příloha č. 8).

5.4 Shrnutí výsledků

Výsledkem práce je funkční vzorové testovací prostředí, které bylo použito pro otestování vzorové aplikace. Na základě výsledků testů lze konstatovat, že aplikace je v rozsahu provedených automatizovaných testů plně funkční.

6 Závěr

Automatizované testování softwaru patří mezi důležité fáze vývoje softwaru, a nemělo by být v žádném případě opomíjeno. Při rozhodování o jeho využití je zapotřebí vyhodnotit všechny požadavky, které s jeho zavedením souvisí. Automatizované testování není závislé pouze na nástroji pro tvorbu automatizovaných testů a na testované aplikaci. K provozu automatizovaného testování jsou nutné také hardwarové prostředky, které mohou být v podobě fyzických stanic nebo virtuálních strojů. Dále nástroje pro správu automatizovaných testů, které evidují vytvořené testovací scénáře a automatizované testy, které byly na jejich základě vytvořeny. Tvorba samotných automatizovaných testů vyžaduje také určitou znalost testované aplikace a jazyka, ve kterém se automatizované testy vyvíjejí. Při rozhodování o zavedení automatizovaného testování je potřeba zvážit také vysoké počáteční náklady a jejich případnou návratnost v dlouhodobém horizontu.

Cílem teoretické části byla definice základních pojmů v oblasti testování softwaru s následným zaměřením se na automatizované testování. Na úvod teoretické části byla provedena definice základního rozdělení testovacích metod dle jejich zaměření. Následně byly uvedeny jednotlivé způsoby testování. Dále v práci byla rozvedena problematika automatizovaného testování a analýza testovaného softwaru. Na závěr teoretické části byla popsána správa a vyhodnocení testů.

Cílem praktické části byl návrh řešení automatizovaného testování v praxi. Na úvod této části bylo vyřešeno testovací prostředí a volba konkrétního nástroje pro správu automatizovaných testů. Po vyřešení těchto základních otázek byla provedena analýza testovaného softwaru, kde proběhlo seznámení s funkcemi aplikace a objekty jejího GUI. Výsledkem této části byly analyzované GUI objekty a jejich nejvhodnější identifikace, která byla využita pro následný vývoj automatizovaných testů. Po analýze testovaného softwaru byly vytvořeny testovací scénáře, které pokrývají základní funkčnosti testované aplikace. Na základě testovacích scénářů byly vytvořeny automatizované testy, které byly vloženy společně s testovacími scénáři do nástroje pro správu automatizovaných testů. Závěrem praktické části bylo vyhodnocení reportů provedených automatizovaných testů. Výsledkem práce je funkční vzorové testovací prostředí, které bylo použito pro otestování vzorové aplikace. Na základě výsledků testů lze konstatovat, že aplikace je v rozsahu provedených automatizovaných testů plně funkční.

7 Seznam použitých zdrojů

1. ELMASRI, Ramez; NAVATHE, Shamkant. Fundamentals of database systems. Addison-Wesley Publishing Company, 2011. ISBN: 978-0-136-08620-8
2. SHARP, John. Microsoft Visual C# 2013 Step by Step. Pearson Education, 2013. ISBN: 978-0-7356-8183-5.
3. NIDHRA, Srinivas; DONDETI, Jagruthi. Black box and white box testing techniques-a literature review. International Journal of Embedded Systems and Applications (IJESA) [online], 2012 [cit.2018.10.21]. Dostupné z: http://www.academia.edu/download/38077013/Black_Box_and_White_Box_Testing_Techniques_-_A_Literature_Review.pdf
4. ROUDENSKÝ, Petr; HAVLÍČKOVÁ, Anna. Řízení kvality softwaru. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-4519-7.
5. KOŘÍNEK, Ondřej. Srovnání vývojových diagramů a pseudokódu ve výuce algoritmizace. Matematika – fyzika – informatika. Prometheus, 2014, (23), 219-224. ISSN 1805-7705.
6. CHURÝ, Lukáš., 2005: Desktopový vývoj: Vývojové diagramy – díl 2 [online] [cit.2019.02.10]. Dostupné z: http://programujte.com/clanek/1970010171-vyvojove-diagramy-2-dil/
7. SELECKÝ, Matúš. Penetrační testy a exploitace. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3752-9.
8. LEITNER, Andreas, et al. Reconciling manual and automated testing: The autotest experience. In: 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07) [online]. IEEE, 2007 [cit.2018.08.20]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4076909
9. SOMMERVILLE, Ian. Softwarové inženýrství. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3826-7.
10. BUREŠ, Miroslav; RENDA, Miroslav; DOLEŽEL, Michal. Efektivní testování softwaru. Grada, 2016. ISBN: 978-80-247-5594-6.
11. PAGE, Alan; ROLLISON, Bj; JOHNSTON, Ken. Jak testuje software Microsoft. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-2869-5.
12. SETH, Nikita; KHARE, Rishi. ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development. In: 2015 2nd

International Conference on Recent Advances in Engineering & Computational Sciences (RAECS) [online]. IEEE, 2015 [cit.2018.09.11]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7453279

13. JGRAPH LTD. draw.io [software]. [přístup 8. února 2019]. Dostupné z: https://www.draw.io/
14. ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Computer Press, 2015. ISBN: 978-80-251-4573-9
15. STEPHENS, Matt; ROSENBERG, Doug. Testování softwaru řízené návrhem. Computer Press, 2017. ISBN: 978-80-251-3607-2
16. GÁLA, Libor; ŠEDIVÁ, Zuzana; POUR, Jan. Podniková informatika. Grada, 2015. ISBN: 978-80-247-5457-4.

8 Přílohy

Příloha č. 1: Automatizovaný test Přihlášení do aplikace (vlastní zpracování)

```
'Spuštění aplikace
SystemUtil.Run "C://AT/Eurojackpot.exe"
'Identifikace formuláře, na kterém se bude pracovat
With SwfWindow("index:=0")
     'Vyplnění uživatele
     .SwfEdit("swfname:=txtJmeno").Set "Lukas"
     'Vyplnění hesla
     .SwfEdit("swfname:=txtHeslo").Set "lukas"
     'Přihlášení
     .SwfButton("swfname:=btnPrihlas").Click
     'Ukončení aplikace
     .Close
End With
```

Příloha č. 2: Funkce Přihlášení do aplikace (vlastní zpracování)

```
'Funkce pro přihlášení naplněna parametry uživatel a heslo
Public Sub fnc_prihlaseni(uzivatel, heslo)
     'Spuštění aplikace
     SystemUtil.Run "C://AT/Eurojackpot.exe"
     'Identifikace formuláře, na kterém se bude pracovat
     With SwfWindow("index:=0")
          'Vyplnění uživatele
          .SwfEdit("swfname:=txtJmeno").Set uzivatel
          'Vyplnění hesla
          .SwfEdit("swfname:=txtHeslo").Set heslo
          'Přihlášení do aplikace
          .SwfButton("swfname:=btnPrihlas").Click
     End With
End Sub
```

Příloha č. 3: Automatizovaný test Založení nového losování (vlastní zpracování)

```
'Přihlášení uživatele do aplikace
fnc_prihlaseni "Lukas", "lukas"
'Identifikace formuláře, na kterém se bude pracovat
With SwfWindow("index:=0")
     'Otevření modulu Losování
     .SwfButton("swfname:=btnLosovani").Click
     'Založení nového losování
     .SwfButton("swfname:=btnNovylos").Click
     'Identifikace formuláře, na kterém se bude pracovat
     With SwfWindow("index:=0")
          'Vyplnění data losování
          .SwfEdit("swfname:=txtDatumlos").Set "1.1.2019"
          'Vyplnění sázky
          .SwfEdit("swfname:=txtSazka").Set 500
          'Vyplnění výhry
          .SwfEdit("swfname:=txtVyhra").Set 0
          'Uložení losování
          .SwfButton("swfname:=btnUloz").Click
          'Potvrzení hlášky o uložení losování
          .Dialog("index:=0").WinButton("index:=0").Click
     End With
     'Ukončení aplikace
     .Close
End With
```

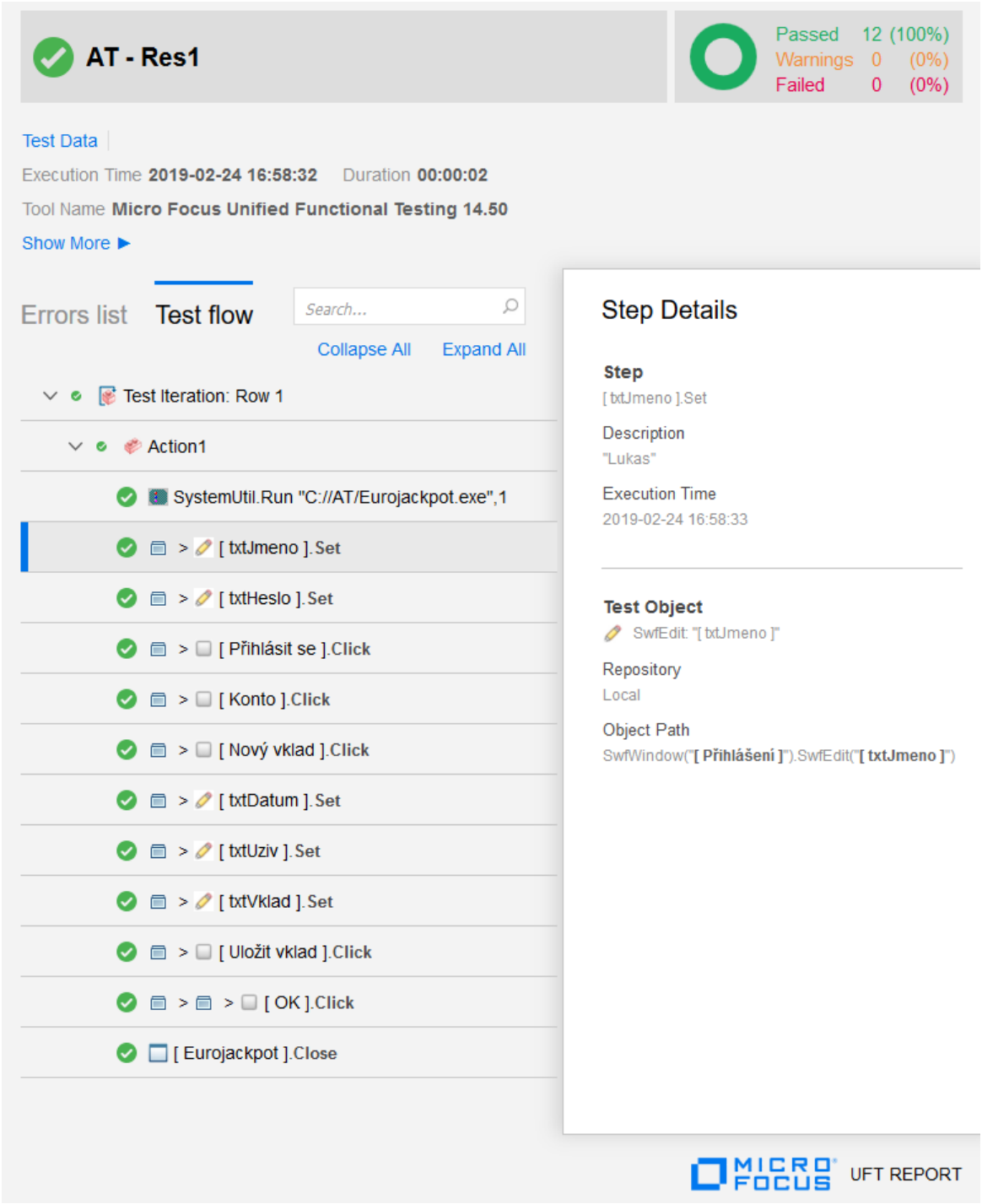
Příloha č. 4: Automatizovaný test Založení nového vkladu (vlastní zpracování)

```
'Přihlášení uživatele do aplikace
fnc_prihlaseni "Lukas", "lukas"
'Identifikace formuláře, na kterém se bude pracovat
With SwfWindow("index:=0")
     'Otevření modulu Konto
     .SwfButton("swfname:=btnKonto").Click
     'Založení nového vkladu
     .SwfButton("swfname:=btnVklad").Click
     'Identifikace formuláře, na kterém se bude pracovat
     With SwfWindow("index:=0")
          'Vyplnění data vkladu
          .SwfEdit("swfname:=txtDatum").Set "1.1.2019"
          'Vyplnění uživatele
          .SwfEdit("swfname:=txtUziv").Set "Pavel"
          'Vyplnění vkladu
          .SwfEdit("swfname:=txtVklad").Set 200
          'Uložení vkladu
          .SwfButton("swfname:=btnVklad").Click
          'Potvrzení hlášky o uložení vkladu
          .Dialog("index:=0").WinButton("index:=0").Click
     End With
     'Ukončení aplikace
     .Close
End With
```

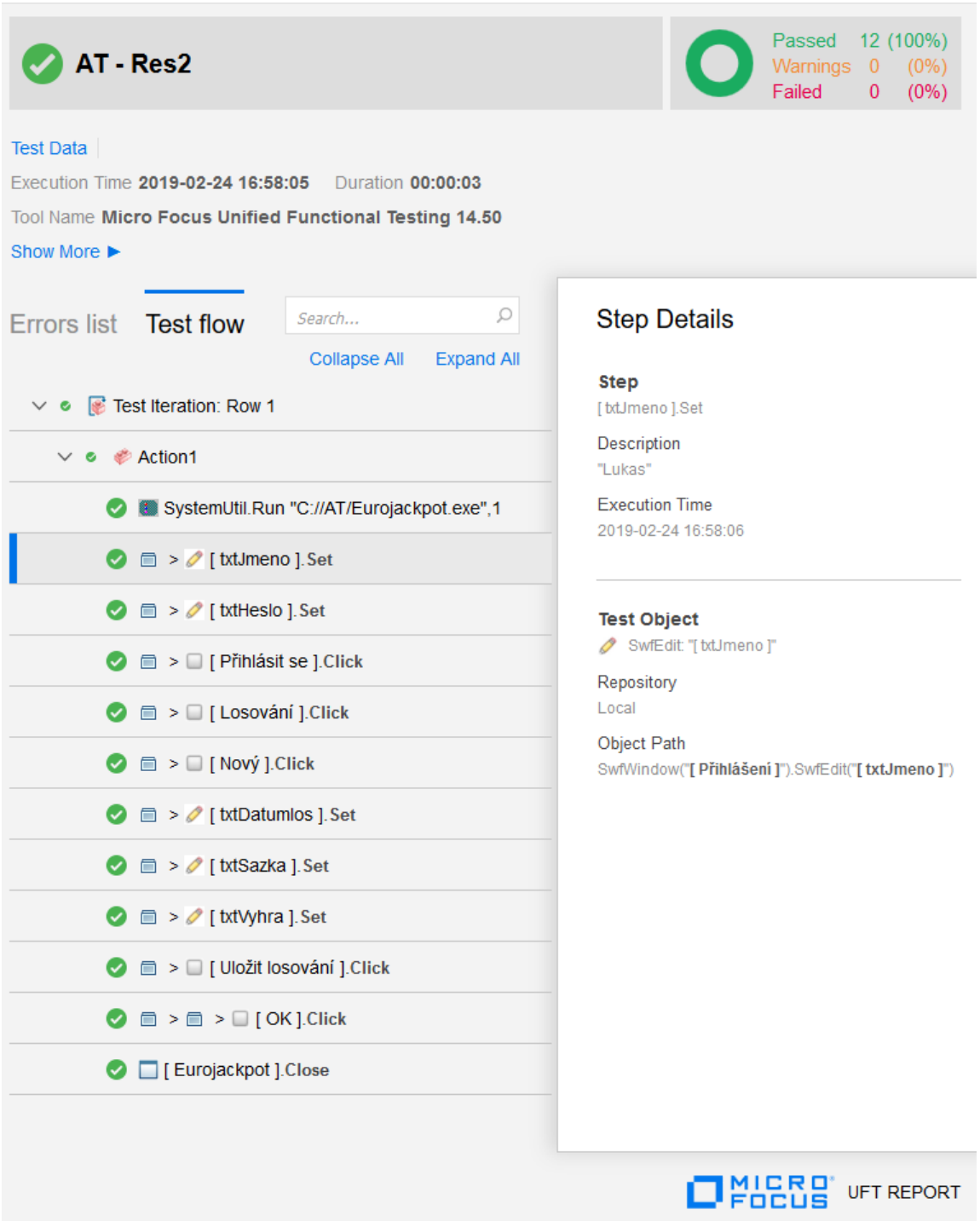
Příloha č. 5: Automatizovaný test Založení nového uživatele (vlastní zpracování)

```
'Přihlášení uživatele do aplikace
fnc_prihlaseni "Lukas", "lukas"
'Identifikace formuláře, na kterém se bude pracovat
With SwfWindow("index:=0")
     'Otevření modulu Nastavení
     .SwfButton("swfname:=btnNastaveni").Click
     'Založení nového uživatele
     .SwfButton("swfname:=btnNovy").Click
     'Identifikace formuláře, na kterém se bude pracovat
     With SwfWindow("index:=0")
          'Vyplnění jména
          .SwfEdit("swfname:=txtJmeno").Set "Petr"
          'Vyplnění příjmení
          .SwfEdit("swfname:=txtPrijmeni").Set "Novák"
          'Vyplnění uživatelského jména
          .SwfEdit("swfname:=txtUzivatel").Set "Petrn"
          'Vyplnění hesla
          .SwfEdit("swfname:=txtHeslo").Set "petr123"
          'Výběr role
          .SwfComboBox("swfname:=comboBoxRole").Select
"Administrátor"
          'Založení uživatele
          .SwfButton("swfname:=btnZaloz").Click
          'Potvrzení hlášky o uložení uživatele
          .Dialog("index:=0").WinButton("index:=0").Click
     End With
     'Ukončení aplikace
     .Close
End With
```

Příloha č. 6: Vyhodnocení testu Založení nového losování (vlastní zpracování)



Příloha č. 7: Vyhodnocení testu Založení nového vkladu (vlastní zpracování)



Příloha č. 8: Vyhodnocení testu Založení nového uživatele (vlastní zpracování)