České vysoké učení technické v Praze

Fakulta strojní

Ústav přístrojové a řídící techniky

Obor: Automatizace a průmyslová informatika

Diplomová práce

Automatizace testování SW v prostředí SAP

Bc. Timotej Vrátný

Vedoucí práce: doc. Ing. Josef Kokeš, CSc.

Externí vedoucí práce: Ing. Jan Vaněk

Rok: 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vrátný** Jméno: **Timotej** Osobní číslo: **456164**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídicí techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Automatizace testování SW v prostředí SAP

Název diplomové práce anglicky:

SAP testing automation

Pokyny pro vypracování:

1. Prostudujte teorii testování SW,
2. sestavte širší seznam testovacích SW, zaměřte se převážně na produkty použitelné v SAP,
3. vyberte 1-2 produkty, ve kterých vytvořte prototyp,
4. každý z prototypů vyzkoušejte a prezentujte výsledky,
5. proveďte porovnání prototypů, zformulujte doporučení a závěr.

Seznam doporučené literatury:

- Patton, Ron. Testování softwaru. Praha : Computer Press, 2002. ISBN 80-7226-636-5.
- https://www.softwaretestinghelp.com/best-sap-testing-tools/

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Josef Kokeš, CSc., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Jan Vaněk, Atos IT Solutions and Services A/S, Czech Republic, Doudlebská 1699/5, 140 00 Praha 4

Datum zadání diplomové práce: **29.10.2021** Termín odevzdání diplomové práce: **20.01.2022**

Platnost zadání diplomové práce: _____

doc. Ing. Josef Kokeš, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

2.11.2021
Datum převzetí zadání

Podpis studenta

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;

- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude k dispozici knihovně Fakulty strojní Univerzity České vysoké učení technické v Praze (dále „Univerzita“) a jeden výtisk bude k dispozici společnosti Atos IT Solutions and Services, s.r.o., IČ 44851391 dále „ATOS”;

- byl jsem seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;

- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;

- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;

- beru na vědomí, že k vypracování diplomové práce bylo využito softwaru poskytnutého společnosti Qace System s.r.o., IČ 277 91 262 a s jejím vědomím byly realizovány po zkušební dobu komerční prototypy ve společnosti ATOS IT Solutions and Services, s.r.o. Výsledky diplomové práce nelze využít ke komerčním účelům jinými subjekty kromě uvedených;

- beru na vědomí, že výstupem diplomové práce je softwarový produkt, proto se považují za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce. Očekává se, že tato část bude Univerzitou chráněna a nebude veřejně přístupná bez uzavření licenční smlouvy.

Prohlašuji,

1. že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.

2. že odevzdaná verze diplomové práce, verze elektronická nahraná do IS Univerzity a verze předaná společnosti ATOS jsou totožné.

V Praze dne

…….	…….
jméno a příjmení	podpis diplomanta

Poděkování

Rád bych zde poděkoval společnosti Atos IT Solutions and Services, s.r.o. za umožnění podílet se na reálných problémech vzniklých při vývoji programu. Jmenovitě bych chtěl poděkovat Ing. Janu Vaňkovi za cenné rady, neutuchající nadšení do problematiky, a hlavně čas věnovaný mé osobě.

Dále bych chtěl poděkovat společnosti Qace System s.r.o. za zapůjčení SW – GUI Master, ve kterém jsem mohl vytvořit funkční prototyp, konkrétně Ing. Karlovi Bařinovi za konzultace, které značně urychlily nastudování problematiky zapůjčeného SW.

V neposlední řadě bych rád poděkoval panu doc. Ing. Josefu Kokešovi, CSc., který mi umožnil zpracovat vlastní téma diplomové práce.

Nakonec, nikoli nejméně, děkuji své rodině a přátelům, kteří mi poskytli nezbytnou oporu během celého studia.

Abstrakt

Tato diplomová práce se zabývá především SW určeným k automatizaci testování v prostředí SAP a zároveň testováním samotným. V teoretické části je zaměřena na teorii testování a následnou automatizaci testování. Na teorii navazuje vlastní průzkum trhu dostupných testovacích nástrojů. V praktické části je zaměřena na testovací nástroj GUI Master a požadavek na automatizaci procesu zakládání plateb v prostředí SAP. Zhodnocení práce shrnuje výsledky vytvořeného funkčního prototypu, včetně hodnocení použitelnosti vybraného testovacího nástroje v prostředí systému SAP.

Klíčová slova:

Testování SW, Automatizace testování, GUI Master, Robotická automatizace procesů, SAP, RPA

Abstract

This diploma thesis deals mainly with software designed to automate testing in the SAP environment and testing itself. The theoretical part focuses on testing theory and subsequent automation of testing. This is followed by the market research of available testing tools. The practical part focuses on the GUI Master test tool and the requirement to automate the process of creating payments in the SAP environment. The conclusion summarizes the results and evaluation of the created functional prototype, including the evaluation of the usability of the selected test tool in the SAP system environment.

Key words:

SW Testing, Test Automation, GUI Master, Robotic Process Automation, SAP, RPA

Seznam použitých zkratek

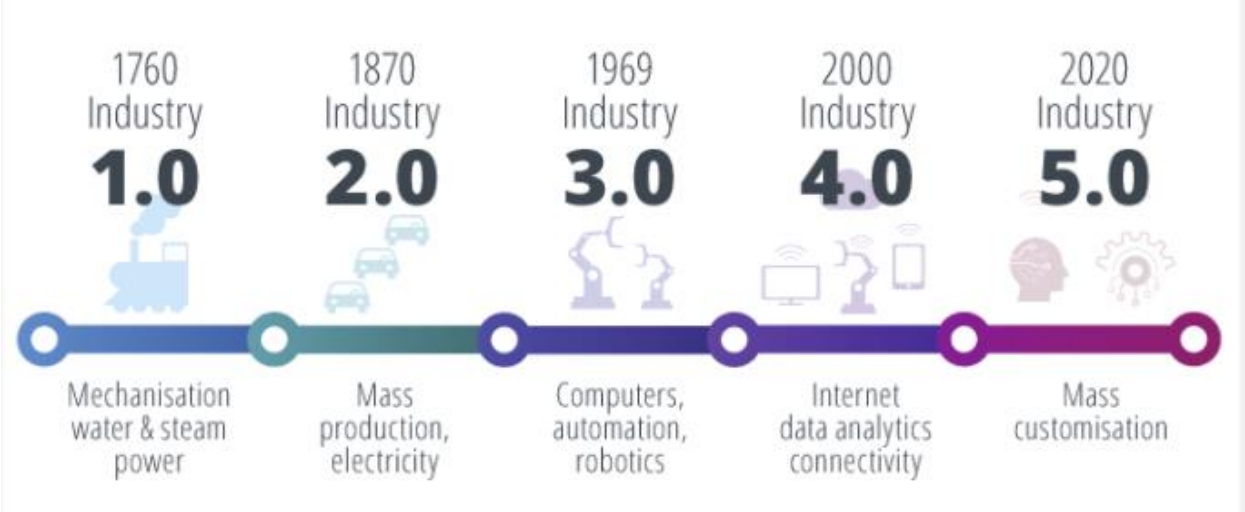
SAP ERP	- Enterprise Resource Planning Solution
SW	- Software
RPA	- Robotic Process Automation
KPIs	- Key performance indicators
SIT	- System Integration Tests
OS	- Operating system
HW	- Hardware
QA	- Quality Assurance
UML	- Unified Modelling language
UAT	- User Acceptance Testing
OLTP	- Online Transaction Processing
ABAP	- Advanced Business Application Programming
API	- Application Programming Interface
GUI	- Graphic User Interface
VBA	- Visual Basic for Applications
UI	- User Interface
BFSI	- Banking, financial services and insurance
IT	- Information technology
UWP	- Universal Windows Platform
WPF	- Windows Presentation Foundation
IDE	- Integrated Development Environment
JVM	- Jednotný výplatní modul
VPN	- Virtual private network

Obsah

1. Úvod

1.1. Vývoj automatizace nejen v průmyslu

Za poslední roky se technologický vývoj výrazně urychlil. Technologické milníky a problémy s nimi spojené společnost přeskakuje nyní bez větších obtíží. Avšak potýká se s nedostatkem pracovní síly, především kvalifikované. Vzhledem ke zvýšené poptávce na pracovním trhu, která se týká jak průmyslových, tak kancelářských pozic, nezbývá korporacím, ale i středním až malým podnikům a také státním orgánům nic jiného, než automatizovat dané procesy a agendu s nimi spojenou.



Obrázek 1 - Průmyslová revoluce [1]

Nacházíme se v dalším milníku průmyslové revoluce, tzv. Průmyslu 5.0, kdy hlavním "posunem" není náhrada člověka robotem či chytrými zařízeními, ale kooperace mezi nimi tak, aby chytrá zařízení a roboti lidem práci usnadnili a zefektivnili. Lze to také popsat jako proces, kdy společně pracují lidé a kolaborativní roboti, což označujeme složeninou „cobot". Nutno podotknout, že „coboti" se již vyskytují v digitálním světě při práci s kancelářskými automatizačními nástroji. Zároveň lze označit toto období za éru masivních "customizací" tzn., že přibývá množství požadavků na úpravu SW, webových stránek, aplikací a dalších dle konkrétního zákaznického požadavku.

Dalším pozitivním bodem koncepce Průmysl 5.0 je, že může pomoci snížit strach o pracovní místa zaměstnanců, protože staví na pracovitosti a kreativitě lidí, kterým spolupráce se stroji pomůže od těžké monotónní práce. Velká část podniků ještě nedokončila ani fázi Průmyslu 4.0 nebo jí dokonce ještě ani nezačala, tudíž se nám tyto dvě průmyslové revoluce kombinují [2].

Jednou z hlavních součástí iniciativy Průmysl 4.0 byla digitalizace, která měla především usnadnit a urychlit práci s daty. Digitalizace opět vytvořila další pracovní místa, ale většinou vysoce kvalifikovaná, protože nedílnou součástí digitalizace a také automatizace je vývoj SW, který je na danou aplikaci vhodný.

Příliv nového programového vybavení zvýšil nutnost jeho testování, protože většina SW i po nasazení do produktivního prostředí se dále rozvíjí dle nových požadavků zákazníka. Každá změna programu vyžaduje následně dostatečnou míru testování, aby se SW stabilizoval a pracoval tak, jak si představuje zákazník, ale i samotný vývojář. Nyní se vyvíjí nástroje, které pomáhají s automatizací testovacích a byznysových procesů.

1.2. SAP ERP

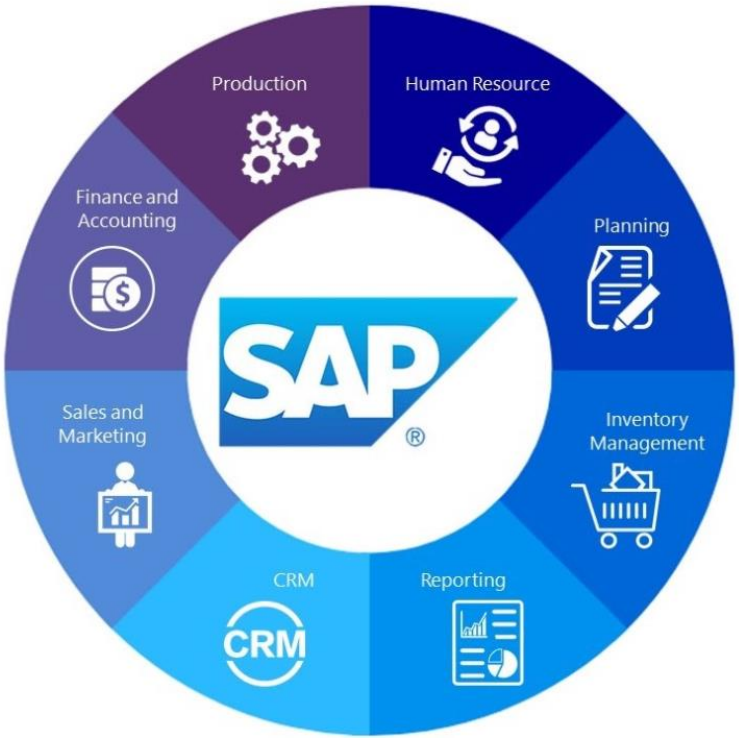
V praktické části je tato práce zaměřena na automatizační testovací nástroj v prostředí SAP ERP. V následujících kapitolách je nabídnuto stručné vysvětlení, o jaké prostředí se jedná a k čemu slouží.

1.2.1. Společnost SAP

Společnost SAP byla založena roku 1972 pěti společníky, kterým jsou Dietmar Hopp, Klaus Tschira, Hans-Werner Hector, Hasso Plattner, a Claus Wellenreuther. Původní název společnosti System Analysis Program Development, byl později nahrazen jen zkratkou SAP. Společníci přišli s nápadem spojit desítky podnikových aplikací, jako jsou například aplikace pro řízení skladu, finanční účetnictví, evidenci údržby či plánování výroby, ale také mnoho dalších, do jednoho integrovaného systému. SAP celosvětově zaměstnává více než 105 tisíc zaměstnanců. Každoročně přichází s novinkami a spolupracuje s nejznámějšími univerzitami. Devadesát devět společností ze 100 největších na světě systém SAP využívá. V České republice má společnost SAP zastoupení jako SAP ČR, s.r.o. [3].

1.2.2. ERP SW

ERP vychází z anglického Enterprise Resource Planning, v češtině se používá termín plánování podnikových zdrojů. ERP SW zahrnuje programy, které pokrývají klíčové oblasti podnikání. Vhodnou demonstrací integrace do jednoho systému těchto oblastí je - SAP ERP - využití [4].



Obrázek 2 - SAP ERP - využití [4]

SAP ERP představuje informační systém, který automatizuje a integruje enormní množství podnikových procesů. Tyto systémy včetně jeho předchůdců spadají do kategorie systémů pro online zpracování transakcí (Online Transaction Processing, zkratka OLTP), které jsou využívány zaměstnanci v rámci jednoho systému najednou. Jedná se o komplexní informační systém k efektivnímu řízení podnikových zdrojů. SAP je také vůbec první společností, která vyvinula standardizovaný SW pro byznysová (odvětvová) řešení. Na to navázala jako světový lídr v poskytování ERP řešení. Existuje mnoho verzí a rozšiřujících nástrojů systému SAP, mezi nejznámější patří SAP R/2, SAP R/3, SAP HANA, SAP S4 / HANA, SAP NetWeaver, SAP CRM, SAP PLM, SAP SCM, SAP SRM více v přílohách Tabulka 14 a Tabulka 15. SAP mimo jiné využívá vlastními silami vyvinutý objektově orientovaný programovací jazyk ABAP. Mezi konkurenční ERP SW patří např. Oracle ERP, Microsoft ERP, Unit4 ERP, Sales Force, NetSuite, Acumatica ERP [3] [5].

1.3. Cíl diplomové práce

Téma automatizace testování SW je poměrně mladé. V posledních letech přibylo na trhu mnoho společností s nabídkou nástrojů k automatizaci. Souvisí to samozřejmě s poptávkou po podobných nástrojích, kdy pro jejich užití nemusí být nezbytně nutné znalosti programátora, tudíž mají mnohem širší spektrum potencionálních uživatelů. Také jejich užití nemusí nutně sloužit k testování, ale může automatizovat či poloautomatizovat některý z vhodných procesů, které provádějí uživatelé při každodenní práci. Hlavní vliv na rozvoj automatizace měla iniciativa Průmysl 4.0.

Cílem této diplomové práce je seznámit čtenáře s teorií testování SW, aby byl schopen porozumět základním termínům, diverzifikaci testování a následně umožnit se získanými znalostmi dále pracovat. Dalším cílem je představení nástrojů určených k testování SW v prostředí SAP, protože konkrétně v tomto prostředí není tolik známých produktů k testování, a proto se jedná o zajímavé téma. Hlavním cílem práce je vytvoření prototypu v jednom nebo více nástrojích na konkrétní problematiku, prezentování dosažených zkušeností a vyvození závěru. Z hlediska praktické části, jeden z kolegů definoval cíl následovně: „***Cílem by měla být praktická realizace, která by nás posunula v oblasti rutinních testů trochu dále k profesionalitě***." Tato věta by obecně mohla vystihnout cíl testování. Jedná se o proces, který slouží ke zdokonalení dodávaného produktu, především tehdy, pokud dochází k neustálým úpravám dle požadavků zákazníka neboli "customizaci".

2. Teorie testování softwaru

Tato kapitola je věnována nejen pojmům ze světa testování SW, ale je také základním náhledem na testovací postupy a připomenutím cílů testování.

2.1. Co je testování?

Softwarové systémy jsou nedílnou součástí života, od byznysových aplikací například v bankovnictví až po spotřebitelské produkty, ve kterých se nachází, jako je automobil. Snad každý člověk se již setkal s programem, který nepracoval tak, jak si představoval. SW, který nepracuje správně, může vézt k mnoha problémům, může vyvolat ztrátu peněz, času, reputace u zákazníků v daném odvětví, nebo dokonce zranění až smrt. Testování je cesta k dosažení požadované kvality SW a snížení rizika selhání SW při provádění operací [6].

Snížení rizika selhání dosáhneme především vyhledáváním chyb. S tím souvisí jejich definování a oprava. Zajištění opravy představuje dostatečně podrobná analýza chyby a další komunikace, která povede ke korekci dané chyby. Pozdější nalezení chyby bude představovat vyšší náklady na nápravu [7].

2.1.1. SW chyba

Tím se dostáváme k otázce, co je přesně SW chyba. Jedním z uznávaných odborníků mnoha publikací na téma detekce chyb je profesor Ron Patton z britské University of Hull, [8]. Ve své knize Testování softwaru chybu definoval následovně [9]:

„O softwarovou chybu se jedná, je-li splněna jedna nebo více z následujících podmínek:

1) Software nedělá něco, co by podle specifikace dělat měl.

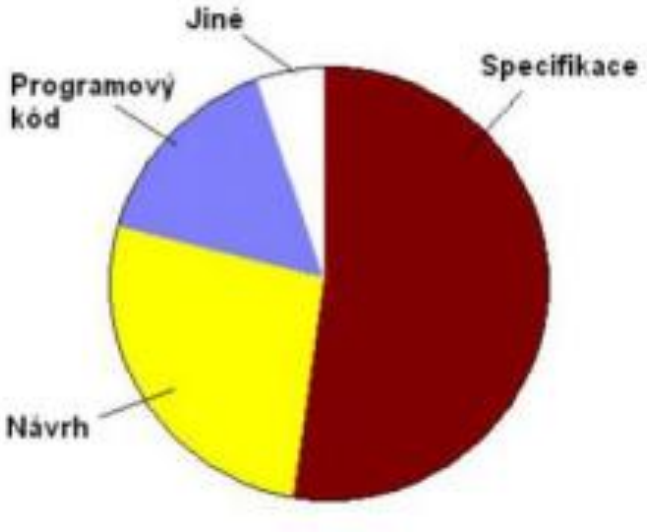
2) Software dělá něco, co by podle specifikace produktu dělat neměl.

3) Software dělá něco, o čem se specifikace nezmiňuje.

4) Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.

5) Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý nebo (podle názoru testera softwaru) jej koncový uživatel nebude považovat za správný.“

Dle této definice vychází, že SW je potřeba testovat vůči jeho specifikaci. Znázorňuje to také koláčový graf viz. Obrázek 3 - Příčiny chyb dle Pattona [9], který vychází z mnoha studií na projektech různých velikostech. Studie dospěly vždy ke stejnému výsledku. Vznik SW chyby vždy souvisí především se specifikací.



Obrázek 3 - Příčiny chyb dle Pattona [9]

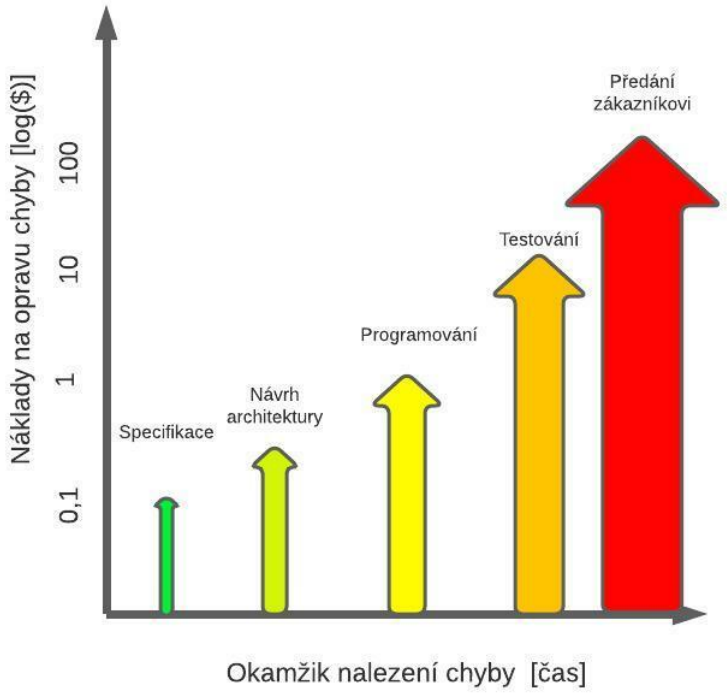
Hlavním strůjcem chyb je skutečně specifikace, a to z mnoha příčin. V řadě případů specifikaci nikdo vůbec nenapíše. Další příčinou může být specifikace, která je moc obecná nebo se neustále mění, případně se do její tvorby nezapojil relevantní vývojový tým. Pokud se plánování neprovede správně, vnese do SW chyby. Druhým největším zdrojem chyb je SW návrh, ve kterém programátor konkretizuje jednotlivé artefakty řešení a algoritmizuje požadované chování. Chyby v programovém kódu jsou až na 3. pozici. Do kategorie jiné mohou spadat například mylné předpoklady, chyby, které nakonec chybami nejsou atd., představují pouze malé procento všech vyskytujících chyb [9].

Při detekci chyb můžeme narazit na následující pojmy [10]:

- Chyba (Error): Vyjadřuje neshodu mezi hodnotou skutečnou a očekávanou například při výpočtu.
- Selhání (Failure): Vyjadřuje neschopnost systému provádět požadované operace či funkce.
- Bug: Je chyba v programu, kdy se program začne chovat neočekávaným způsobem
- Porucha (fault): Označuje nesprávný krok, proces nebo chybná data v programu, který se díky tomu začne chovat neočekávaně.
- Defekt (Defect): Obyčejně poukazuje na několik problémů s chováním, vnějším či vnitřním, softwarového produktu.

2.1.1.1. Náklady na SW chyby

Chyby můžeme nalézt hned od specifikace daného systému až po konečnou fázi, kdy již SW využívá zákazník, což názorně demonstruje Obrázek 4 - Graf závislosti nákladů na čase dle [9], sestavil autor.



Obrázek 4 - Graf závislosti nákladů na čase dle [9], sestavil autor

Náklady opravy chyby mají logaritmickou stupnici neboli s postupem času rostou exponenciálně. Pro chybu, kterou identifikujeme již v počátcích vývoje systému, nestojí oprava téměř nic, ve zvolené ukázce 0-0,1$. Jestliže stejnou chybu nalezneme ve fázi programování nebo testování, bude oprava stát jednotky až desítky $. Při nalezení chyb až v produkčním prostředí, se náklady na opravu vyšplhají na stovky i tisíce $. Je očividné, že testovat SW je vhodné již od úplného počátku vývoje, a to hlavně kvůli rostoucím nákladům na opravu chyby v pozdější fázi vývoje [9].

2.1.2. Správnost – důvod k testování (Turingův teorém)

Dva hlavní faktory při vývoji systému jsou specifikace a implementace. Tyto dva pojmy můžeme aplikovat na konvenční životní cyklus vývoje SW. Například, ve fázi návrhu, je specifikací dokument se specifikací systému a

implementace je detailní návrh systému. Podobně, tyto pojmy můžeme aplikovat na celý vývojový proces, kde specifikace tvoří systémové požadavky a implementace je konečná systémová realizace.

Ideálním cílem je, aby implementace vyhovovala ve všech ohledech specifikaci. Poté by implementace představovala správné řešení vzhledem k zadané specifikaci. V mnoha případech je "správnost" považována za ekvivalent takového cíle. Nicméně, dosažení takového výsledku při vývoji SW je nemožné, protože jakákoli taková metoda by musela vyřešit Turingův problém zastavení [11]:

Teorém

Neexistuje Turingův stroj E, pro jakékoliv dva Turingovy stroje p1 a p2, kde platí:

E (p1, p2) = Pravda, jestliže p1 je ekvivalent k p2

Důkaz [12]:

Jestliže takový stroj E existuje, poté může vyřešit Turingův problém zastavení. Vzhledem k jakémukoli stroji p, použijte funkci h následovně:

Stroj h(x);

y = p(x); vrací (0);

konec stroje.

h(x) = 0 jestliže p(x) končí a nekončí, když p(x) nekončí.

Jestliže z(x) = 0 pro všechna x a končí pro všechna x. Tedy E (h, z), i když a jen a pouze tehdy p ukončí pro všechna x. Tato znalost E implikuje znalost problému zastavení pro libovolné p.

Jelikož jsme si dokázali, že dosáhnout ideálního cíle, který bychom považovali za ekvivalent "správnosti" nelze dosáhnout. Tak je vždy důležité se tomu pokusit přiblížit co nejvíce. Nicméně, abychom se ke "správnému" cíli přiblížili, musíme testovat co nejefektněji.

2.1.2.1. Správnost a testování

Dosažení "správnosti" se snažíme dosáhnout všelijakými možnými pokusy testování a to tím, že detekujeme veškeré chyby v implementaci, aby mohly být odstraněny. Návrh testovacího případu, který by byl ovlivněn konkrétní závadou znamená, že chyba vedoucí k této poruše se nevyskytne při tvorbě implementace nebo vede k detekci bez nutnosti vykonání implementace a pozorování selhání SW.

Testování SW zaručí "správnost", pokud dodržíme následující body:

1. Zahrnout co nejvíce možných vstupů do testovacího scénáře (ne vždy je to nejvhodnější z důvodu náročnosti testu). Testovací scénář se musí osvědčit v odhalení co nejvíce případných závad vyskytujících se při implementaci SW. Samozřejmě musí být zohledněna omezení pro danou aplikaci
2. Výsledek v každém testovacím případě je porovnáván s očekávaným výsledkem a požadavky [11].

2.2. Testovací proces

Není dán univerzální testovací scénář, ale existují obecně známé postupy testovacích aktivit, bez kterých by testování nedosáhlo vytyčeného cíle.

Celý testovací proces ovlivňují následující body [6]:

- Model životního cyklu vývoje SW a užité projektové metodiky
- Vhodná volba testovací úrovně a druhu testování
- Produktová a projektová rizika
- Provozní omezení zahrnující:
    - Zdroje a rozpočet
    - Časové rozpětí
    - Komplexnost

    - Regulační a smluvní požadavky

- Opatření a postupy společností (zadavatele, zhotovitele)
- Vyžadované interní a externí standardy

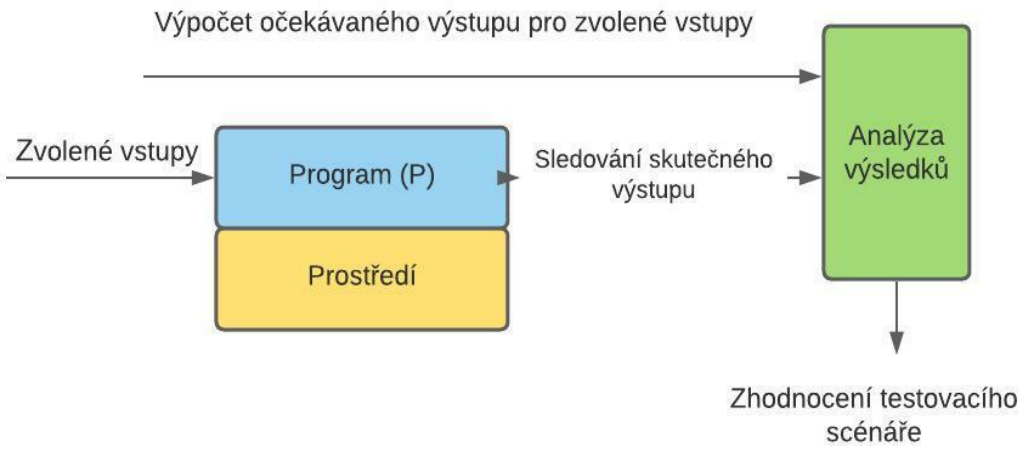
Následující body popisují obecné aspekty organizace testovacích procesů [6]:

- Testovací aktivity a úkoly, které mohou zahrnovat následující fáze:
    - Plánování testů
    - Sledování a ovládání testů
    - Návrh testů
    - Implementace testů
    - Spuštění testů
    - Dokončení testů
- Testovací pracovní produkty
- Sledování mezi základem testování a testovacím pracovním produktem

Je velice užitečné, pokud základ testování pro jakoukoliv úroveň nebo typ testování který zvažujeme, měl definována měřitelná či pozorovatelná kritéria. Tato kritéria mohou být klíčové indikátory výkonu v angličtině známé jako KPIs. Více informací o testovacím procesu dle norem se můžeme dozvědět z ISO standard ISO/IEC/IEEE 29119-2:2021 [6], resp. Standard IEEE 829-2008.

2.2.1. Testovací aktivity

V situaci funkčního a systémového testování, testovací inženýr musí provést určitou sekvenci testovacích aktivit. Následující vysvětlení daných aktivit se týká pouze jednoho testovacího scénáře a lze ho akceptovat, jako zjednodušený obecný postup. Liší se pro různé úrovně testování, ale model na obrázku níže můžeme považovat za obecnou posloupnost testovacích aktivit.



Obrázek 5 - Testovací aktivity dle [13], sestavil autor

1. Identifikace cíle testování: Prvním krokem je identifikace testovaného objektu. Čeho vlastně chceme dosáhnout daným testováním. Případně, kde očekáváme nejvíc chyb [13].

2. Zvolení vstupu: Následuje zvolení vstupů. Vstupy můžeme zvolit na základě požadavků specifikace SW, zdrojového kódu, případně našich očekávání. Testovací vstupy jsou voleny v souladu s úrovní testu. Zároveň bychom měli mít cíl testování pořád na paměti a připomínat si, proč daný test vlastně děláme.

3. Výpočet očekávaného výstupu pro zvolené vstupy: Třetí aktivita zahrnuje výpočet očekávaných výstupů pro dané vstupy. Ve většině případů to lze provést pouze na základě celkového porozumění cíle testu, což znamená, že v takovém případě nic nepočítá. Ovšem jedná se o porozumění cíli testu na vysoké

úrovni a zároveň specifikaci testovaného programu. Odvíjí se od úrovně testování [13].

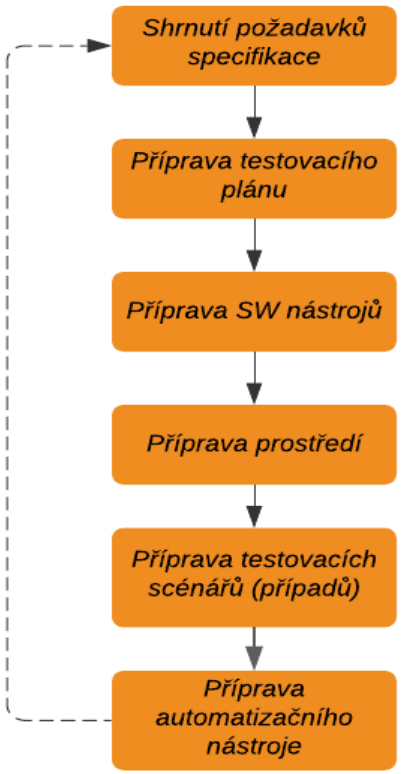
4. Nastavení spouštěcího prostředí programu: V daném kroku musí být splněny všechny externí předpoklady. Může zahrnovat přístup k síti, výběr databází, přípravu testovacích dat atd. Na závěr však nesmí chybět analýza selhání. Pochopit danou chybu, umět ji zopakovat a vhodně předat informace [13].

2.2.2. Návrh fází testování SW

Návrh SW testování lze rozdělit do tří fází: předběžná fáze testování, testovací fáze a uživatelská akceptační testovací fáze [14].

2.2.2.1. Předběžná fáze testování

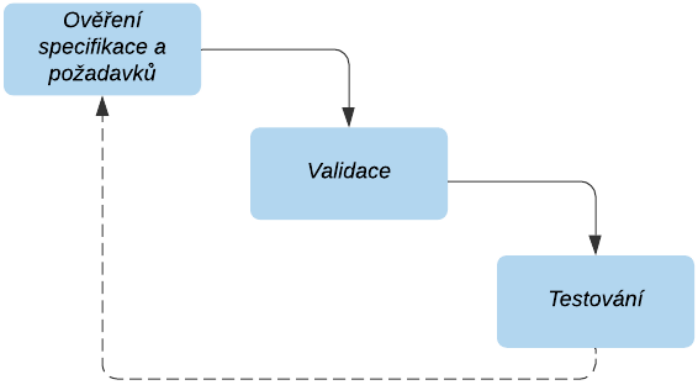
Tato fáze slouží především testerům k upřesnění specifikace a požadavků zákazníka. Dle následujícího diagramu - Obrázek 6 můžeme splnit, jak požadavky na test, tak požadavky zákazníka.



Obrázek 6 - Předběžná fáze testování - návrh

2.2.2.2. Testovací fáze

Návrh testovací fáze není třeba více rozepisovat, většinu známých postupů jsem zmínil v předchozích podkapitolách. Pro zaručení kvality testování, bychom se měli řídit dle diagramu - Testovací fáze - návrh.



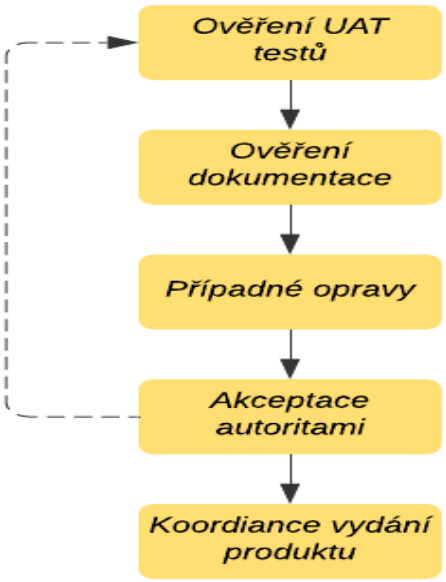
Obrázek 7 - Testovací fáze - návrh

Tento postup bychom měli provádět v souladu s Demingovým cyklem (PDCA cyklus) [15]:

- Plán (plan) – naplánování zamýšleného záměru
- Provedení (do) – Realizování naplánovaných aktivit
- Kontrola (check) – Validace výsledků oproti původnímu záměru
- Jednání (act) – Úpravy na základě ověření a následné převedení do praxe.

2.2.2.3. UAT-Uživatelská akceptační testovací fáze

UAT vychází z anglického User Acceptance Testing a návrh této fáze by se měl řídit dle diagramu - UAT - návrh, tak aby došlo k zajištění co nejvyšší kvality.



Obrázek 8 - UAT - návrh

2.3. Způsoby dělení testování

Jednotlivé způsoby testování dělíme podle:

- Fáze vývojového cyklu (úrovně): Jednotkové (Unit), funkční, systémové, akceptační,
- Znalosti kódu: Metoda černé, bílé a šedé skříňky,
- Dimenzí kvality: Výkonnosti, použitelnosti, bezpečnosti, spolehlivosti,
- Způsobu realizace: Manuální, automatické.

2.4. Úrovně testování

Na začátek je důležité zmínit, že existuje více úrovní testování, než je v této práci uvedeno. V této podkapitole je vysvětleno základní rozdělení se zaměřením na nejznámější úrovně. Testovací úrovně jsou skupiny testovacích aktivit, které jsou organizovány a řízeny společně. Liší se účelem, zdrojem, technologiemi apod. Každá úroveň testování je jedním z případů testovacího procesu, který se skládá z možných aktivit popsaných v podkapitole 2.2. Na každé úrovni se testování provádí s jiným záměrem.

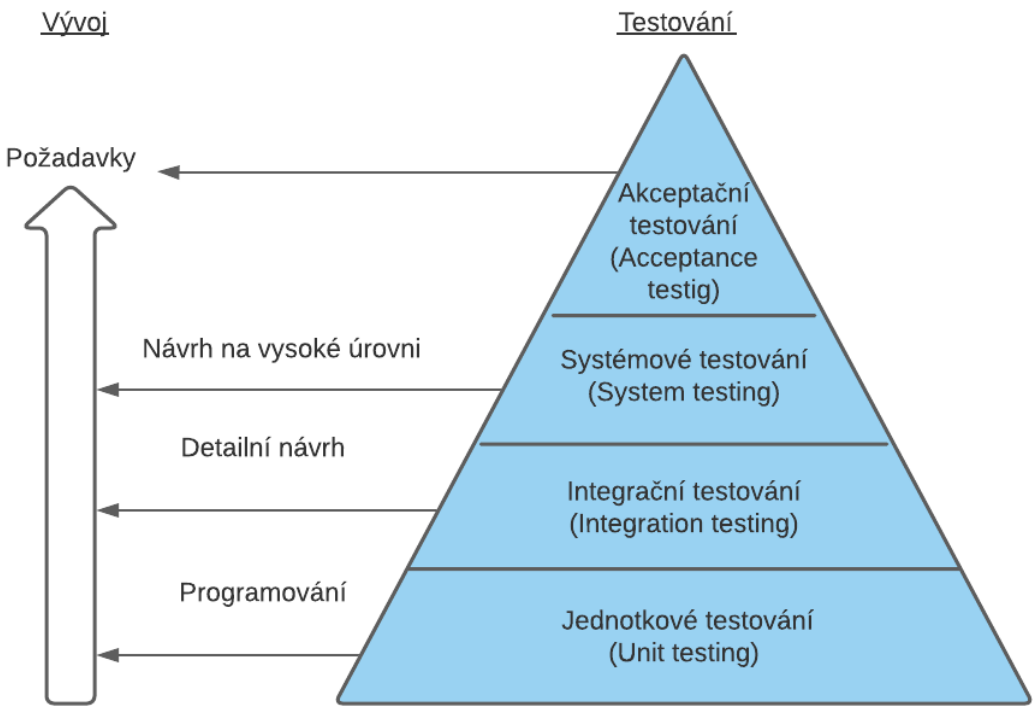
Úrovně testování popisují různé metodiky, které lze využít při provádění testů. Hlavní rozdělení úrovní je:

- Funkční testy (Functional testing): Funkční testování je založené na specifikaci SW, který má být testován. Toto testování se provádí především formou černé skříňky. Aplikace je testována za pomoci vstupních dat, a poté jsou zkoumány výsledky, které musí odpovídat specifikaci. Ověřuje operace a reakci aplikace [16].

- Nefunkční testy (Non-functional testing): Tento typ je založen na testování aplikace z jejích nefunkčních atributů. Nefunkční testování zahrnuje požadavky na výkon, bezpečnost, uživatelské rozhraní a další. Nefunkční testování je stejně důležité, jako funkční, ovlivňuje zejména spokojenost klientů [16].

2.4.1. Funkční testy

Funkční testování můžeme rozdělit do čtyř základních úrovní. První z nich je jednotkové testování, druhé integrační, třetí systémové a čtvrté akceptační. První tři úrovně testování probíhají zpravidla na straně dodavatele a čtvrtý stupeň většinou společně se zákazníkem. Model úrovní testování demonstruje Obrázek 9 - Úrovně testování [13].



Obrázek 9 - Úrovně testování

První test provádí vývojové prostředí, známý jako (Assembly test), které nedovolí uložit zdrojový kód se syntakticky nesprávným obsahem. Druhý test provádí kompilátor (kompilační test), který může odhalit další chyby programátora při kompilaci kódu do spustitelné podoby.

2.4.1.1. Jednotkové testování

Po úvodních kontrolách se dostáváme do fáze testování jednotek. V objektově orientovaném programování se jedná o testování jednotlivých tříd a metod. Testovanou jednotkou v tomto případě považujeme samostatnou část testovaného aplikačního programu, která je kompletně izolována od zbytku. Izolována znamená, že aplikace není propojena s externími závislostmi, jako jsou databáze, webové služby (protokoly) atd.

Testy jednotek se převážně provádí automaticky (preferováno). Zapisují se ve formě samostatného programového kódu, který ověřuje správnost již vytvořeného kódu testované aplikace, pouze pro danou část. Proto je logické, že automatizaci jednotkového testování obvykle vytváří a obsluhují vývojáři. Pro vytváření testů se využívá nástrojů na bázi frameworků

(v češtině známých jako rámců). Mezi nejznámější patří: Junit, NUnit, JMockit, EMMA, PHPUnit.

Testování jednotek by se mělo řídit pravidlem AAA [17]:

- Arrange (Připrav): Příprava toho, co chceme testovat, tj. proměnné, pole a vlastnosti, které potřebujeme k průběhu testu a k očekávaným výsledkům.
- Act (Jednej): Volání metody, kterou testujeme.
- Assert (Potvrď): Volání funkce k ověření, zda je výsledek (act) dle očekávání.

Příklad v jazyce C# [17]:

Testuje metodu Sum pro jednoduché sčítání. V následujícím příkladu je aplikace s názvem ApplicationToTest s třídou Calc, která má metodu Sum.

Metoda Sum ():

```
public void Sum(int b, int c)
{
    return b + c;

}
```

Jednotkový ověřovací kód (unit):

```
[Testclass]
public class UnitTest1
{
    [TestMethod]
    public void TestMethod1()
    {
        //Arrange
        ApplicationToTest.Calc ClassCalc = new ApplicationToTest.Calc();
        int expectedResult = 7;

        //Act
        int result = ClassCalc.Sum(3, 4);

        //Assert
        Assert.AreEqual(expectedResult, result);

    }
```

Je vhodné části jednotkového testování věnovat velkou pozornost, z hlediska nákladů na opravu chyby, jak jsme si znázornili na grafu - Graf závislosti nákladů na čase dle [9], sestavil autor [18].

2.4.1.2. Integrační testování

Poté, co vývojář ukončí jednotkové testování, přebírá testování testovací tým. Integrační testy zpravidla neprovádí programátor, ale jsou v kompetenci testovacího týmu. Je nutné rozlišit, zda se jedná o testy vnitřní integrace, která spočívá ve vzájemné komunikaci jednotlivých modulů a vnější, kde musí být ověřena bezchybná komunikace na vnějších rozhraních systému. U obou těchto integrací je nutné provádět testy. V této fázi se tak testuje integrace dosud jednotlivě ověřených částí. Začneme dvěma komponenty a postupně přidáváme další. U menších projektů se na tuto fázi neklade velký důraz, protože na stejné chyby bychom měli narazit v dalších fázích testování. Tudíž, kdy na chybu narazíme by nemělo mít na výsledný produkt vliv, ale opět platí, co v předchozí fázi, čím dříve na danou chybu narazíme a vyřešíme, tím budou náklady nižší [18] [7] [19].

2.4.1.3. Systémové testování

Po ověření korektní integrace je čas k systémovému testování. Tyto testy probíhají v pozdějších fázích vývoje. Ověřují aplikaci z pohledu budoucího zákazníka. Jde o ověření, zda aplikace pracuje jako celek správně. Podle připravených scénářů se simulují různé kroky, které v praxi mohou nebo by měly nastat. Obvykle probíhají v několika kolech. Nalezené chyby se opraví a následně se testy opakuji. Součástí této fáze, jsou funkční i nefunkční testy. Celé systémové testování lze považovat za jakousi výstupní kontrolu [18] [20].

Systémové a integrační testy se rozpadají na další podskupiny. Mezi nejznámější patří Smoke a Sanity testy, více o nich se můžeme dozvědět ze zdroje [21].

2.4.1.4. Akceptační testování

Akceptační testy mohou probíhat výhradně na straně zákazníka dle předané dokumentace k jeho požadavkům. Z mé zkušenosti probíhají zpravidla v kooperaci zhotovitele se zákazníkem. Zejména v prostředí SAP,

kde se provádí současně desítky customizací (záleží na zákazníkovi) je nejvhodnější sejít se zákazníkem a provést testy společně s podmínkou, že zákazník zná očekávané chování (výsledky), a je schopen je porovnat s dosaženým chováním (výsledky) upraveného systému.

Známé jsou také regresní testy, které ověřují, zda nové úpravy neovlivnily jiné části systému nežádoucím způsobem. Uplatňují se také u vývoje, který je rozdělen na etapy. Regresní testy zde ověřují, že vývoj v etapě neovlivní výstupy z etap předešlých [19].

2.4.2. Nefunkční testy

Nefunkční testy obsahují takové testy, které nesouvisí s požadavky na funkcionalitu, ale posuzují další dimenze kvality (ISO/IEC 25010:2011), jako je výkon (performance), bezpečnost (security), vstřícnost uživatelského rozhraní (user experience / usability), spolehlivost (reliability), kompatibilita (compatibility), atd. Mnohem lépe to lze identifikovat z anglického názvu Non-Functional Testing. Níže jsou uvedené alespoň některé obecně využívané.

2.4.2.1. Testování výkonnosti

Je velmi důležité u testování webových aplikací, kdy typickým příkladem může být e-shop, u kterého vysoká návštěvnost způsobí selhání serveru, nebo jsou příliš pomalé reakční časy, či zobrazuje chybové zprávy. Existují různé příčiny, které se podílejí na snížení výkonu SW [16]:

- Zpoždění sítě
- Zpracování na straně klienta
- Zpracování databázových transakcí
- Zátěž mezi servery
- Vykreslování dat

Testování výkonnosti se provádí zejména z hlediska následujících aspektů:

- Rychlost (časy odezvy, vykreslení a přístup k datům)

- Kapacita
- Stabilita
- Škálovatelnost

Mezi typicky měřené ukazatele výkonnosti patří počet zpracovaných požadavků za sekundu (RPS – z angl. requests per second), počet souběžně aktivních uživatelů (simultaneously active users) a průměrná doba odezvy (average response time). Testování výkonosti může mít kvalitativní nebo kvantitativní charakter. Testování výkonnosti může být dále rozděleno na další dva podtypy [7].

2.4.2.2. Zátěžové testování (Load testing)

Load testing je proces, při kterém se pozoruje chování SW při maximální zátěži z hlediska počtu přístupů a manipulaci s velkým množstvím vstupních dat. Tento údaj nám sdělí maximální kapacitu SW při vrcholových podmínkách zátěže. Zde lze velmi dobře uplatnit nástroje k automatizaci testování, jako jsou Load Runner, AppLoader, IBM Rational Performance tester, Apache Jmeter, Silk performer, Visual Studio Load Test a další [16].

2.4.2.3. Zátěžové testování (Stress testing)

Stress testing prověřuje systém za abnormálních podmínek. Může nastat například při odebrání některých zdrojů nebo zatížením nad fyzický limit zařízení. Cílem je nalézt bod zlomu, při kterém SW selže. Selhání lze dosáhnout různými scénáři, jako jsou [16]:

- Náhodné/řízené vypnutí nebo restart síťových portů.
- Zapnutí nebo vypnutí databáze.
- Spouštění různých procesů, které zatěžují procesor, paměť, server, zdroj a další.

Obě testování, pokud aplikace obstojí, jsou známkou spolehlivosti

2.4.2.4. Testování bezpečnosti

Testování bezpečnosti je proces, který ověří, zda systém chrání svá data a funguje z pohledu zabezpečení tak, jak bylo předpokládáno. Jedná se o jedno z nejnáročnějších testování vůbec, protože u každého SW existují jiné postupy k jeho zneužití. Pracovní pozice, která se konkrétně zaměřuje na testování bezpečnosti je etický hacker a je jednou z nejlépe placených pozic v IT. Mezi hlavní předměty kontroly patří [7] [22]:

- XSS (cross-site scripting) – útočník dokáže do webových stránek podstrčit svůj vlastní kód.
- SQL injection – napadení databázové vrstvy vlastním SQL skriptem přes nezabezpečený vstup na stránce.
- Nahrání škodlivého souboru na server a jeho následné spuštění.
- Získání informací o konkrétních objektech aplikace bez příslušné autorizace.
- CSRF (cross site request forgery) – Napodobení formuláře na stránce.
- Nedostatečně ošetřené chybové stavy aplikace-mohou útočníkovi sdělit informace použitelné pro další útok.
- Útoky na přihlašovací část aplikace.
- Nezabezpečené ukládání šifrovaných dat.
- Nezabezpečená komunikace – útočník může číst komunikaci, která mu není určena.
- Zobrazování citlivých informací na stránkách, které nevyžadují autorizaci.

Uvedené příklady jsou pro testování vnějších útoků.

Mnohem častějším předmětem bezpečnostního testování jsou oprávnění pro stávající a budoucí uživatele v souvislosti s realizovanými změnami. Zda mají uživatelé přístup pouze k takovým transakcím, které jsme

jim předem povolili, zda jsou stále relevantní, v souladu s organizační strukturou, interními předpisy apod. To je typické hlavně pro prostředí SAP.

2.4.2.5. Testování použitelnosti a uživatelského rozhraní

Pro testování použitelnosti je vhodné využít někoho, kdo od počátku nepracoval na vývoji SW, aby do testování vnesl nový pohled a upozornil na případné nedostatky. Tým testerů plní roli uživatelů téměř od počátku vývoje, proto mu většina postupů může přijít zcela jasná, což nemusí platit u nových uživatelů. Pro tuto oblast se na trhu práce profiluje profese UX specialista.

Existuje několik definicí použitelnosti dle standardů, jako jsou ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, atd. [16].

2.4.2.6. Testování dokumentace

Nedílnou součástí je také testování dokumentace. Testy se provádí z hlediska funkcionality, tzn. zda na jejím základě lze SW využívat. Kontroluje se gramatika, typografie. Důležité je dokumentaci pravidelně aktualizovat současně s vývojem SW. K této činnosti může patřit kontrola hlášení programu, aby z nich uživatel pochopil vzniklý problém a mohl najít jeho řešení vlastními silami.

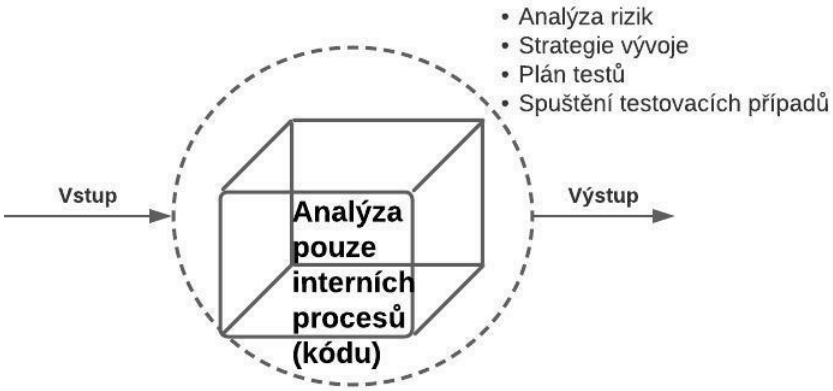
2.5. Známé metody testování

Existují tři nejznámější metody testování:

1. Metoda testování bílé skříňky.
2. Metoda testování černé skříňky.
3. Metoda testování šedé skříňky.

2.5.1. Metoda testování bílé skříňky

Je založena na detailním průzkumu interní logiky a struktury kódu. Při této metodě je pro testera nutná znalost zdrojového kódu. Metoda je aplikovatelná na jednotkovou úroveň testovacího procesu. Tester je zaměřen zejména na zdrojový kód.



Obrázek 10 - Metoda bílé skříňky

Výhody:

- Odhalí chybu ve skrytém kódu, odstraněním dalších řádků kódu.
- Vedlejší efekty jsou přínosem.

Nevýhody:

- Je velice nákladné a vyžaduje zkušeného testera.
- Některé části kódu mohou být přehlédnuty [23] [24].

2.5.2. Metoda testování černé skříňky

Jedná se o metodu testování bez jakékoliv znalosti vnitřní struktury aplikace. Zkoumá pouze chování a reakce systému. Tester musí znát architekturu a nemá přístup ke zdrojovému kódu. Může pokrýt všechny čtyři úrovně testování, typicky se užívá v rámci funkčních a akceptačních testů.

Obrázek 11 - Metoda černé skříňky

Výhody:

- Efektivní pro velké segmenty kódu.
- Pro pochopení testera je to jednoduché.
- Náhled testera (jako uživatele) a vývojáře je oddělený, vzájemně se neovlivňují.
- Rychlejší vývoj testovacích případů.

Nevýhody:

- Pouze omezené pokrytí.
- Bez jasné specifikace je náročné sestavit testovací případy.
- Nemusí se jednat o zcela efektivní testování [23] [24].

Konkrétním příkladem pro tuto metodu je přihlašovací obrazovka, kde chceme otestovat přihlášení do určitého systému ke konkrétnímu účtu. Nachází se zde dvě pole, uživatelské jméno a heslo, jako vstup a očekávaným výstupem je přihlášení do systému nebo chybová hláška (chybné přihlašovací údaje). Tato metoda nezahrnuje specifikaci zdrojového kódu.

2.5.3. Metoda testování šedé skříňky

Bílá + černá skříňka = šedá skříňka využívá z obou metod jejich základní principy. Tester by měl mít znalosti využitých interních datových

struktur a algoritmů k sestavení testovacího případu. Techniky využité při tvorbě testovacích případů jsou model architektury, UML, stavový diagram [23] [24].

Obrázek 12 - Metoda šedé skříňky

Výhody:

- Zvýší pokrytí testování tím, že se zaměří na více vrstev jakéhokoliv systému kombinací veškerých existujících testování černé a bílé skříňky.
- Tester převážně pracuje na uživatelském rozhraní a funkční specifikaci více než na zdrojovém kódu.
- Vytvoření kvalitních testovacích scénářů.
- Test je prováděn spíše z uživatelského hlediska než vývojářského.
- Objektivní testování.

Nevýhody:

- Pokrytí testování je limitováno přístupem ke zdrojovému kódu, mohou být dostupné pouze jeho části, což může mít za důsledek další bod.
- Hodně částí programu může zůstat neotestováno.
- Jestliže vývojář již provedl některé testovací případy, tak tato metoda může být nadbytečná [23] [24].

2.6. Manuální x automatické testování

Testy lze také rozdělit podle toho, zda jsou vykonávány ručně nebo automatizovaně. Pokud test vyžaduje lidské hodnocení a úsudek je velmi těžké ho automatizovat, tudíž je vhodnější použít manuální testování. Pro opakované spuštění velkého množství testů nebo testu generujícím velké množství dat je vhodné automatizovat testy. Postupně se začínají využívat nástroje s umělou inteligencí nebo "coboti", kteří kooperují s uživatelem, v našem případě s testerem [7].

Mezi jednu z technik manuálního testování patří průzkumné testování, které je vhodné používat jako doplněk ostatních formálních technik pro zvýšení efektivity. Jedná se o techniku založenou na zkušenostech a intuici testera. Hlavním podkladem jsou informace o testovaném produktu. Správné průzkumné testování je interaktivní a kreativní. Problémem průzkumného testování je reprodukovatelnost nalezených defektů. V momentě, kdy k selhání vedl určitý souběh kroků, si tester musí uvědomit, jakými kroky k selhání dospěl a musí být schopen je opakovaně nasimulovat [25].

Osobně mám více zkušeností s manuálním testováním, kdy první zkušenosti s automatizovaným testováním získávám právě při tvorbě diplomové práce.

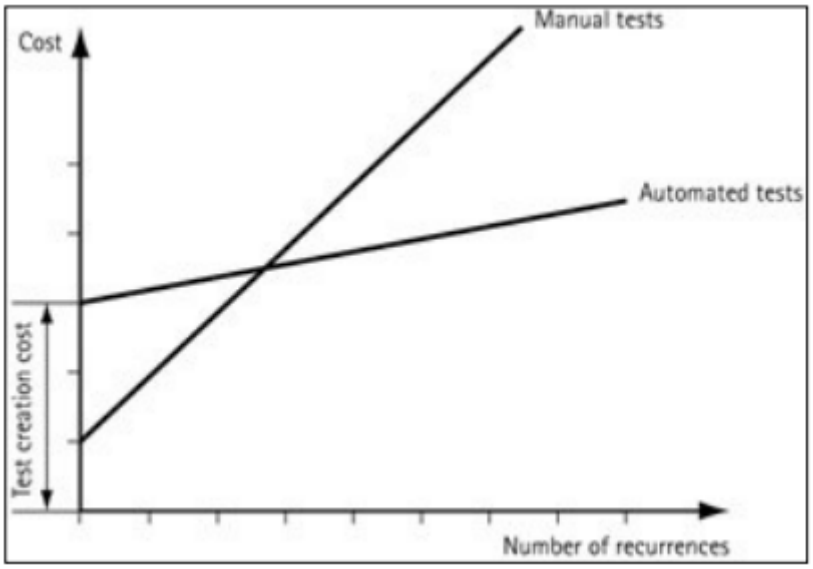
Z hlediska manuálního testování se vždy jedná o práci intuitivní, ale je důležité si připravit "správná" testovací data. Tato činnost je občas zdlouhavá, ale nutná. Pokud se data nepřipraví včas, test se může významně prodloužit, protože tester zpravidla narazí na potíže, které nesouvisí s testovanou funkcionalitou.

Tester také musí pochopit, jak se SW má zachovat dle nové specifikace, ne vždy je to zřejmé. Výhodou testera je znalost a základní orientace v programovacím jazyce, ve kterém je SW vyvíjen, aby mohl upozornit nejen na funkcionalitu, ale zároveň na místo jejího vzniku a tím usnadnil a urychlil nápravu. Další podstatnou dovedností testera je odhalit nestandartní postupy a kombinace, které by vývojáře, ale i samotného uživatele nemusely napadnout. Tím předchází chybě v budoucnosti, což je základem průzkumného testování.

Mnohem náročnější je danou chybu analyzovat za delší čas, až když na ni upozorní zákazník. Opět si musíme osvojit a pochopit danou problematiku. Je velmi důležité komentovat jak kód, tak specifikaci a průběžné změny stále zaznamenávat tak, aby byly zpětně dohledatelné. Jedná se o jednu z nejúnavnějších aktivit spojenou s vývojem SW (testováním) a ne vždy je vykonávána správně.

2.6.1. Porovnání nákladů

Předtím, než začneme proces automatizovat, měli bychom finančně vyjádřit náklady časové náročnosti předpokládané automatizace. Graf níže demonstruje náklady spojené s manuálním a automatizovaným testováním. Manuální testování má nižší náklady pro malý počet opakování. Naopak automatické testování má vyšší náklady na začátku při jeho vývoji, ale počtem opakování se zdokonaluje a tím klesají i náklady na jeho provedení. Lze říci, že od určitého počtu iterací jsou náklady na manuální test vyšší než náklady na automatický test [26].



Obrázek 13 - Náklady na manuální x automatizované testování [26]

Ačkoliv průběh vztahu grafu výše deklaruje bod zlomu, automatické testy nemusí být vždy tak často využívány, jak jsme předpokládali. Jejich vytvoření může být nákladnější, než bylo původně očekáváno. Tudíž je určitě na místě udělat i zpětnou analýzu, zda se nám investice vyplatila a případně přijmout opatření ke zvýšení efektivity.

2.7. Složení testovacího týmu

Každá organizace má svoji vlastní týmovou strukturu, ale je zde několik pozic, které by měly být pokryty, pokud chceme mít úspěšný testovací tým. Profesionální testovací tým byl měl být sestaven z pracovních pozic: QA Engineer, Test Manager, Test Engineer, Test Analyst a Test Automation Engineer. Vždy nemusí být dostatek zdrojů na všechny pozice. Často zastupuje jeden nebo dva testeři více rolí. Popis jednotlivých zaměstnání dle [27]:

1. QA Engineer: Inženýr kvality kontroluje všechny fáze vývojového procesu a ručí za kvalitu dodávaného SW. Zajišťuje, že SW pracuje hladce a bez chyb, než je uveden do produkce.

2. Test Manager: Organizuje a kontroluje celý testovací proces. Buduje a vede celý testovací tým k úspěchu na projektu. Definuje cíle testování v souladu s doručením SW. Nasazuje a spravuje zdroje určené k testování. Zaručuje viditelnost, dohledatelnost a kontrolu testovacího procesu.

3. Test Analyst: Test analytik se zaměřuje na byznys problémy. Obecně navrhuje, připravuje, spouští a realizuje testy, aby zachytil defekty nebo chyby v před produkčních prostředích.

4. Test Engineer: Tester obecně zastřešuje mnoho specializací. Může odkazovat na inženýry specializované na různé testovací přístupy, jako je manuální testování, průzkumné testování, testování výkonu atd. Často je takto označována pozice, ve které se minimálně spoléhá na automatizaci.

5. Test Automation Engineer: Většinou odpovídá pozici programátora, který se výhradně soustředí na automatizaci testovacích procesů. Využívá testovacích nástrojů vhodných pro danou aplikaci a typ testování jako jsou Selenium, Jubula, UiPath, Worksoft aj. Musí se vyznat v návrhu grafického uživatelského rozhraní a SW testování.

2.7.1. Charakteristiky testera

Existuje mnoho pohledů na to, jaké vlastnosti by měl tester mít, ale obvykle nenajdeme jednoznačnou a jasnou definici. Důležitou dovedností testera je umět klást správné otázky a mezi požadované vlastnosti patří přirozené vnímání chyb, tudíž být trošku perfekcionista, ale pouze do vhodné míry z hlediska užitečnosti. Jedním z hlavních dovedností testera by mělo být kritické myšlení. Seznam níže popisuje vhodné vlastnosti [28]:

- Mít kreativní myšlení. Díky tomu vymýšlí testy, které by ostatní nenapadly, používá SW nestandartním způsobem. Sada testů, kterými má produkt projít, rychle roste.
- Být zvídavý, snažit se objevit skryté. Testeři se nesnaží rozbít nebo pokazit produkt (ten už pokažený může být). Testeři se snaží odhalit do jaké míry je pokažený, a ukázat ostatním, co je jim skryto a vhodně vysvětlit, co je nutné opravit.
- Být pečlivý a trpělivý. Pocit z dobře odvedené práce bývá největší odměnou, protože málokdo umí ocenit, zda je testování dobře odvedené nebo ne.
- Umět dobře komunikovat s okolím. Vysvětlení nalezených chyb a zajištění jejich opravy je podstatná část práce, ke které má tester ve skutečnosti jen jediný nástroj, a to svoji schopnost komunikovat.
- Být průbojný. Stává se, že testeři přijdou na projekt a dozví se pouhý zlomek toho, co potřebují. Neví pořádně, kde najít dokumentaci, co se ohledně projektu rozhodlo, na jaké otázky se ptát a koho. Vyžádat si půl dne času někoho zkušenějšího, kdo má hodně práce, si žádá nutnou porci odvahy a odhodlání.

3. Automatizace testování

Ve světě IT se zpravidla můžeme setkat s anglickým pojmem Test Automation nebo Automation Testing. Než přejdeme k samotnému vytváření automatizovaných testovacích scénářů, je důležité řídit se několika základními body [29]:

- Určit, kde chceme automatizovat
- Určit, co chceme automatizovat (co je cílem, souvisí spolu)
- V jakém případě bychom měli / musíme automatizovat
- Jakým způsobem můžeme automatizovat
- Zvolit vhodné testovací nástroje
- Konfigurovat stabilní testovací prostředí
- Vytvořit první automatizované testy

3.1. Jak začít automatizovat testování

Prvním krokem je stanovit místo, kde budeme automatizované testy provádět (v jaké úrovni). Tím místem mohou být testy funkční: jednotkové, integrační, systémové, akceptační nebo nefunkční: testování výkonnosti, bezpečnosti atd. Nejvhodnější je začít jednotkovými (unit) testy. Ovšem lze začít na více úrovních testování, pokud máme dostatečně agilní testovací tým. Praktická část diplomové práce je realizována v úrovni systémového testování.

Jsou případy, kdy je vyloženě vhodné využít automatizaci pro testování. Například transakce registračního formuláře, přihlašování, jakákoliv oblast, kde se uživatelé přihlašují do systému a zároveň generují (využívají) velké množství dat. Kromě toho lze využít veškeré položky v grafickém uživatelském rozhraní: spojení s databázemi, validace vstupních políček, které lze efektivně testovat automatizací manuálního procesu za pomoci vhodných testovacích nástrojů. Tím se budu zabývat v praktické části

diplomové práce. Nicméně, v některých případech to není zcela možné, nebo by vytvoření automatizace bylo příliš náročné. Zejména v případech, kde se uživatel musí rozhodovat na základě nějaké znalosti, která je nadřazena SW.

K automatizaci mohou přispět i podpůrné programovací jazyky, jako je například VBA (Visual Basic for Applications), ale většinou se realizuje specializovanými nástroji pro automatizaci. Existuje mnoho frameworků, které jsou vhodné zejména pro jednotkové (unit) testy. Začínalo se s populárními "record and play" (nahraj a spusť) nástroji. Tuto funkcionalitu stále nabízí většina nástrojů, i když pro automatizaci na vyšší úrovni je nedostačující.

Když se rozhodneme, že využijeme některý z dostupných testovacích nástrojů, je důležité zvolit ten "správný". U výběru nástroje rozhoduje mnoho proměnných jako jsou cena, podpora (českého jazyka, školení, OS atd.), možnosti úprav, vhodnost na danou aplikaci. Více se tématem nástrojů budu zabývat v kapitole 4. Nástroje pro automatizaci testování.

Pro nasazení automatizovaného testování musíme nakonfigurovat stabilní testovací prostředí. Tím se myslí takové prostředí, kde se nachází veškerý nutný SW a HW, který je připraven pro testovací týmy k provádění samotných testů.

Pokud je vše vyřešeno, můžeme začít s automatizovanými testy. Pro složité produkty je vhodné začít s poloautomatickými testy. Začneme s opakujícími činnostmi, které často využíváme, přihlášení, filtrování logů a zároveň se vyznačují jedinou variantou realizace (průchodu aplikací). Mně se vždy osvědčilo začít s přihlášením do testované aplikace. Vhodné je také vytvoření kontinuálního testu, který by se mohl vykonávat například každý den, resp. každou hodinu.

3.1.1. Příklad automatizace testování

Jedním ze základních případů, kde lze vyzkoušet poměrně jednoduše automatizaci testování, je aplikace Kalkulačka v prostředí MS Windows. K napsání takového testu jsem využil inspiraci na stránkách společnosti Microsoft viz. [30]. V prostředí MS Visual Studio Enterprise 2019 využívám funkcionality knihovny MS Coded UI (UI-uživatelské rozhraní). K napsání kódu

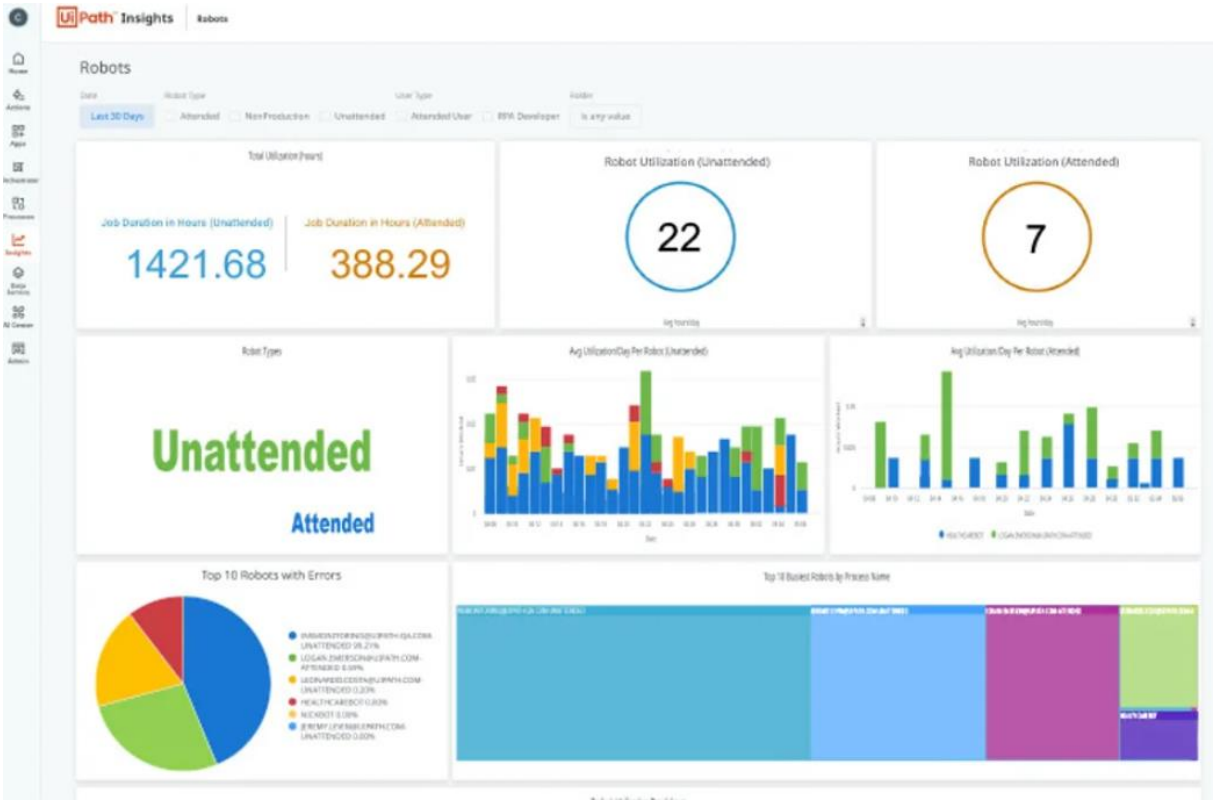
jsem využil znalostí ze Stack Overflow. Bohužel tato knihovna již není dostupná v novějších verzích Visual Studia.

```
[TestMethod]
public void TestCalculator()
{

    //Spuštění aplikace
    var app = ApplicationUnderTest.Launch("C:\\Windows\\System32\\calc.exe");

    //Vykonání pokynů
    Mouse.Click(button6);
    Mouse.Click(buttonAdd);
    Mouse.Click(button2);
    Mouse.Click(buttonAdd);
    Mouse.Click(button4);
    Mouse.Click(buttonEqual);

    // Zhodnocení výsledků – Zde je očekávaný výsledek v případě, že se
nezobrazí, vyskočí hláška s upozorněním
    Assert.AreEqual("12", txtResult.DisplayText,"Kalkulačka nezobrazuje
očekávaný výsledek 12);

    //uzavreni aplikace kalkulačka

          app.Close();
```

Výsledek v mém případě byl vždy správný. Kalkulačka je dlouholetý nástroj, který se využívá na Windows platformě již od jejího vzniku. Tento příklad slouží k jednoznačné demonstraci automatizace testování. K očekávánému výsledku se velmi často používá funkce Assert nebo Checkpoint, záleží na využitém nástroji. Zde je také využito pravidlo AAA.

3.1.2. Report z testů

Velmi důležitou součástí automatických testů je samozřejmě zobrazení výsledků z testů. Aktuální doba přináší nutnost práce s velkým obsahem dat a jejich vizualizaci. Tento požadavek se přenáší také na testovací nástroje. Jedním z příkladných výkaznictví automatizovaného testování poskytuje nástroj od společnosti UiPath - Obrázek 14. Bohužel se může jednat o placenou nadstavbu základní verze programu. Proto se v menších společnostech výsledky testování zasílají vedoucím pracovníkům zpravidla e-mailem doplněným výkazem s vhodnými grafy. S tím však souvisí zhodnocení nákladů na přípravu takového výkazu narůstajících s rozsahem a četností testů.



Obrázek 14 - Reporting UiPath [31]

Jak je z příkladu vizualizace výsledku testů vidět, jedná se o komplexní řešení vhodné pro manažerské týmy k rychlému přehledu na projektech automatizace.

3.2. Druhy Automatizace testování

Dělení automatizace testování je různé, každá organizace může mít odlišné, můžeme vycházet z informací podkapitoly 2.4. Ve které jsou druhy testování rozdělené do jednotlivých úrovní. U automatizovaného testování je dělení podobné. Lze ho rozdělit dle funkcionalit testovacích nástrojů na:

- Automatizace testování (dále Test Automation, ve zkratce TA)
- Robotická automatizace procesů (dále Robotic Process Automation, ve zkratce RPA)

Na trhu existuje mnoho dalších typů nástrojů k automatizaci podle zaměření, patří mezi ně:

- Jednotkové testování (Unit testing)
- Testování vlastností, funkcí (Feature, Functional testing)
- Testování grafického uživatelského rozhraní (GUI testing)
- Testování webových aplikací (Web testing)
- Testování výkonnosti (Performance Testing)
- Testování zabezpečení (Security Testing)

3.2.1. Automatizace testování - Test Automation

Obecně je TA proces automatizace testovacích postupů pomocí programovacího kódu nebo SW nástroje.

3.2.1.1. Test automation - přínosy a zápory

Tabulka 1 - Test Automation - přínosy

TA přínosy
1. Neomezený počet opakování testů.
2. Nalezení chyb, které nepokrylo manuální testování.
3. Rychlost – doba provedení automatických testů je kratší než manuálních.
4. Rychlá zpětná vazba pro vývojáře.
5. Možnost snížení nákladů s rostoucím počtem opakování.

Tabulka 2 - Test Automation - zápory

TA zápory
1. Vazba TA na aktuální verzi produktu, tj. Čím častější aktualizace programu, webu atd., tím častěji musí být testovací scénář upravován.
2. I malé změny mohou narušit testovací případy.
3. Závislost na konfiguraci testovacího prostředí (záplatování, nastavení antiviru, síťových prostupech apod.)

3.2.2. Automatizace testování - RPA

Robotická automatizace procesů je SW technologie, která se užívá především v automatizaci obchodních procesů. Jedná se o autonomní práci SW robotů nebo "cobotů" k emulaci lidské činnosti vůči elektronickým systémům a programům. Na nejvyšší úrovni automatizace je již využívána také umělá inteligence ve fázi strojového učení. Roboti jsou sestaveni jako stavové automaty, které využívají podmínek (jestliže..., pak...) známější z anglického (if and then) nebo technologií strojového učení. Automatizace je pracovní postup, který vytvářejí vývojáři SW či testeři. Roboti následně tyto pracovní postupy zpracovávají jednou nebo vícekrát s využitím API nebo skriptovacího jazyka. "Coboti" čekají na interakci uživatele [32] [33].

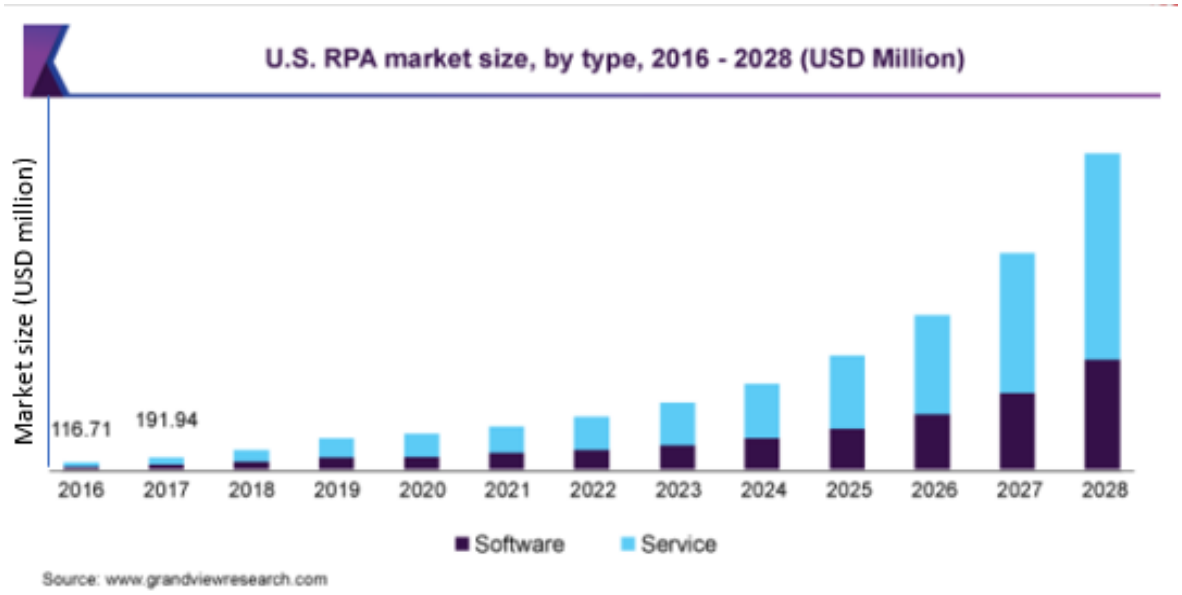
RPA nevyžaduje člověka k psaní kódu. Místo toho RPA systém vytváří seznam instrukcí (akcí) sledováním toho, jak uživatel provádí úkony v grafickém rozhraní aplikace. Poté je schopen tyto kroky opakovat. Stejně jako lidé, využívají roboti události generující klávesnice a myš u PC, ale nepotřebují obrazovku, pracují ve virtuálním prostředí [33]. Není pravidlem pro všechny RPA nástroje.

3.2.2.1. RPA tržní hodnota

Globální tržní hodnota RPA byla ohodnocena pro rok 2020 na 1,57 miliardy dolarů. Očekává se její roční růst o 32,8 % od roku 2021 do roku 2028. Což by mělo přibližně odpovídat globální tržní hodnotě v roce 2028 13,74 miliardy dolarů. Robotická automatizace je na vzestupu a očekává se její rozšíření do více sektorů a společností. Předpokládá se, že RPA dosáhne téměř univerzální akceptace do roku 2030. Náklady na RPA oproti zaměstnanci, který přijel za prací ze zahraničí tzv. (offshore employee), jsou třetinové. Oproti zaměstnanci, který je lokálně zaměstnaný tzv. (onshore employee), jsou náklady až pětinové vychází z dat [34] [35].

Předpokládaný růst tržní kapitalizace, který je znázorněn na grafu Obrázek 15 - Očekávaná tržní hodnota RPA [35], byl zaměřen zejména na oblasti Severní Ameriky, Evropy, Asie a Tichomoří, Latinské Ameriky, Střední a Východní Afriky. Dominantními zeměmi analýzy byly USA, Kanada, Francie, Německo, Velká Británie, Čína, Indie, Japonsko, Austrálie a Nový Zéland, Singapur, Sdružení národů jihovýchodní Asie, Brazílie, Mexiko. Tento graf zároveň podpoří argument, že se jedná o aktuální téma a je na místě se o

podobných nástrojích bavit z hlediska rostoucí tržní kapitalizace a možnosti zisku v budoucnosti [35].



Obrázek 15 - Očekávaná tržní hodnota RPA [35]

Klíčovými hráči na tomto rychle se rozvíjejícím trhu jsou společnosti seřazeni podle velikosti tržní kapitalizace na trhu v tabulce níže:

Tabulka 3 - RPA společnosti - seřazení

Pořadí	Společnost	Pořadí	Společnost
1.	UiPath	7.	NTT Advanced Technology Corp.
2.	Automation Anywhere	8.	EdgeVerve Systems Ltd.
3.	Blue Prism	9.	FPT Software
4.	NICE	10.	OnviSource, Inc.
5.	Pegasytems	11.	HelpSystems
6.	KOFAX, Inc.		

Na trh se také snaží probojovat malé lokální společnosti se snahou prosadit se v tomto velmi konkurenčním prostředí. Snaží se toho docílit osobnějším přístupem a mnohdy řešením na míru.

Dalším bodem, který posílí aktuálnost tématu je globální pandemie, která zásadně změnila chod většiny korporátů, ale i malých podniků skrz veškerá odvětví. Globální pandemie narušila interní i externí obchodní procesy. Odvětví, jako jsou maloobchod, výroba, IT, telekomunikace, automotive aj. čelila poklesu rozvoje podnikání. Zdravotnické organizace byly přeplněny rostoucím počtem případů Covid-19. Tato událost ovlivnila a nadále ovlivňuje kancelářské operace, což vedlo k prodloužení reakční doby administrace, selhání zaměstnanců (z nejrůznějších důvodů, např. nemoc, psychika, práce z domova), hromadění dokumentace, narušení odběratelsko-dodavatelského řetězce atd.

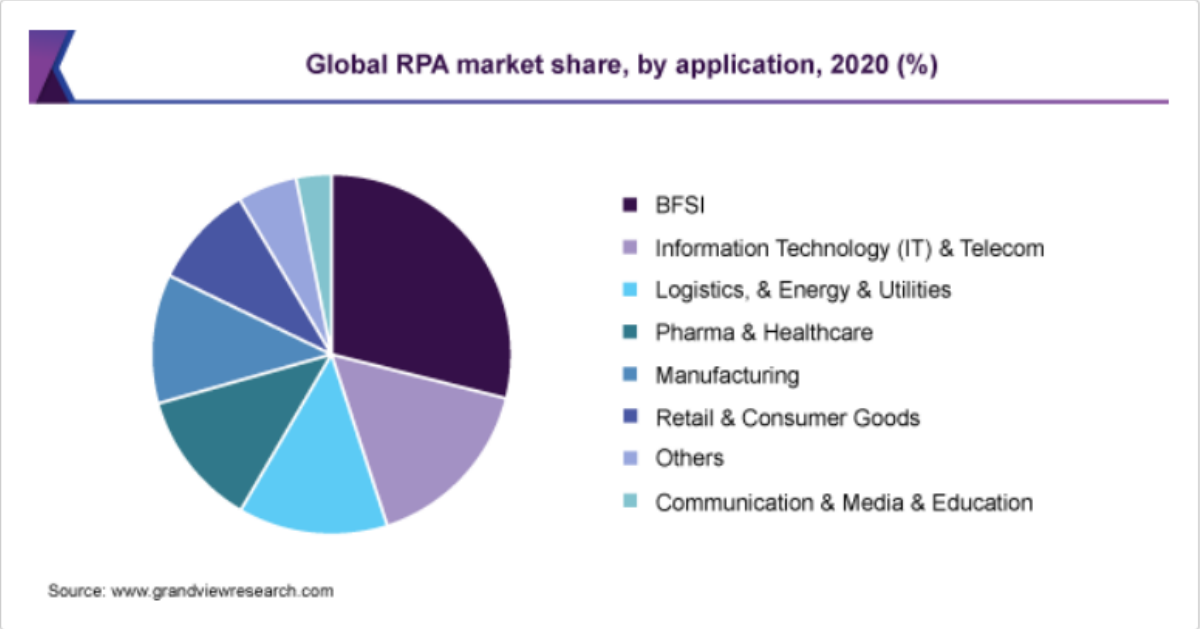
Implementace RPA řešení bylo proto jedním z vítaných faktorů pro optimalizaci nákladů, poskytování rychlejších služeb a zefektivnění testování a výkaznictví. Kromě toho pandemie urychlila poptávku digitální pracovní síly napříč podniky a otevřela nové příležitosti pro poskytovatele automatizačních nástrojů. Očekává se, že automatizace bude definovat práci po pandemii, protože společnosti jsou svědky strukturálních změn se zaměstnanci pracujícími z domova. Pozitivní dopad implementace během pandemie posílí poptávku a podpoří růst trhu s automatizačními nástroji RPA a jim podobnými. Zmíněné aspekty podporují pozitivní fundament v tomto odvětví [35].

Pokrytí jednotlivých oblastí RPA nám demonstruje graf Obrázek 16 - RPA - podíl na trhu (dle zaměření) [35]. Oblast bankovnictví, finančních služeb a pojištění (BFSI) je dominantním odvětvím, pokrývá 29% podíl na výnosech v globálním trhu. Většina procedur spojeným s financemi se přesouvá do online světa. Ruší se pobočky, snižuje počet zaměstnanců k osobním konzultacím s bankéři. Tím vzniká větší objem dat a strukturovaných transakcí spojených s poskytováním služeb, což vyžaduje řešení, které opakované úkoly a objem dat zpracuje nezávisle na zaměstnancích finančního sektoru.

Následuje IT a telekomunikace, kde dochází k zejména vysoké poptávce po přizpůsobení SW a samozřejmě po testování, které je s úpravou SW nezvratně spojené.

V logistice jsou podobné nástroje již nějaký čas využívány, ale stále dochází k jejich navyšování, projevuje se zvýšená globalizace a poptávka po

transportních a doručovacích službách, které nejen díky pandemii Covidu-19 zažívají nevídaný růst.



Obrázek 16 - RPA - podíl na trhu (dle zaměření) [35]

Pandemie Covidu-19 zejména ovlivnila farmaceutický a zdravotnický segment. Testování na tuto nemoc přineslo nezměrný počet dat, se kterými bylo nutné pracovat a na základě nich vyhodnocovat závažnost situace. RPA zde pomáhá zpracovat data, zároveň organizovat termíny vakcinace pacientů, záznamy o testování, průběhy nemoci a další agendu s nimi spojenou. V každém z odvětví v grafu výše se očekává další nárůst využití RPA [35].

3.2.2.2. RPA přínosy a zápory

Přínosy a zápory se mohou lišit dle odvětví v jakém chceme RPA využívat. Některé z bodů, které se vyskytovaly častěji jsou shrnuty do tabulek níže.

Tabulka 4 - Přínosy RPA

RPA přínosy
1. Může pracovat 24/7/365.
2. Vysoce komplexní řešení ke splnění vysokých nároků poptávky.
3. Vykonává úkoly rychleji.
4. Nižší chybovost a vyšší konzistentní kvalita.
5. Nahradí rutinní úkoly zaměstnanců a tím jim umožní pracovat na dalších projektech, kreativní a duševní práci.
6. Nasazuje nové funkcionality rychleji než jiná IT řešení.
7. Levnější integrace se systémy skrz uživatelské rozhraní.
8. Řízený návrat investice.
9. Dohledatelná chyba a možnost řízené opravy.
10. Měřitelná a škálovatelná výkonnost.

Tabulka 5 - Zápory RPA

RPA zápory
1. Vhodné pouze pro procesy, které zahrnují úkoly založené na pravidlech.
2. Může se jednat pouze o dočasné řešení.
3. Zvýšená složitost procesu v případě, který vyžaduje trvalou obsluhu pracovníkem.
4. Vytváření nových úkolů pro roboty musí být kontrolováno.

3.2.3. Rozdíly mezi RPA a Test Automation

Na první pohled se zdají být velmi podobné. Firmy je využívají k automatizaci úkolů, šetří náklady a uvolňují čas zaměstnanců na další úkoly. Když se podíváme na RPA a TA z technického hlediska můžeme narazit na další podobné znaky. Zejména UI test automation je velmi podobné RPA, protože oba nástroje pracují s automatizací uživatelského rozhraní. Velká část testovacích nástrojů již nyní nabízí obě funkcionality, aby pokryla větší část automatizace nejen testování a umožňuje jejich kombinaci, kdy na část můžeme použít RPA a poté si v kódu opravit či doplnit některé funkcionality testovaného produktu [33]. Hlavní rozdíly:

- RPA není nutně využíván k testování, ale k automatizaci veškerých byznysových procesů. Je možné ho využívat z mnohem širšího aspektu než Test automation, které je navržené pouze k testování daného produktu (SW, webu).

- Test automation obvykle vyžaduje, aby tester napsal script, zatímco RPA nevyžaduje psaní kódu k automatizaci úkolu.
- Což vede k dalšímu hlavnímu rozdílu, testovací skript vytvořený pro test automation závisí na testovaném prostředí. Nástroje RPA jsou nezávislé na prostředí, na kterém procesy probíhají [33].
- Test automation je obvykle limitováno uživateli, kteří mají dostatečné znalosti pro daný nástroj. Pro RPA není zpravidla nutná tak vysoká kvalifikace.

3.2.3.1. Rozdíly mezi RPA a Selenium

Jedním z nejznámějších představitelů Test automation je Selenium, které slouží především k testování webových aplikací. Narazil jsem na vhodné srovnání, které může demonstrovat obecně další rozdíly mezi RPA a Test automation. Rozdíly jsem srovnal do tabulky níže.

Tabulka 6 - Srovnání RPA a Selenium inspirováno [36], sestavil autor

Porovnání na základě	RPA	Selenium
Druh automatizace	Flexibilní	Na bázi programování
Náklady	Nutné nástroje (většinou nejsou zdarma)	Open-source tzn., že užívání SW je zdarma
Závislost na prostředí	Nezávislé, může pracovat na mobilu, ploše počítače, webové aplikaci atd.	Pouze webové aplikace
Využité komponenty	Využívá roboty	Využívá webové ovladače
Úroveň testování	Všechny administrativní procesy jako je zadávání dat, bankovní výpočty	Pouze funkční, regresní a výkonnostní testy
Bude automatizovat	Byznysové procesy	Webové aplikace
Znalost programování	Minimální znalost	Vyžaduje základní až pokročilou znalost
Případ použití	Veškerá administrace, účetnictví, lidské zdroje, dokumentace, komunikace a zpracování dat atd. (“backend” = kancelářské procesy)	Využívá aktuální webovou stránku

4. Nástroje pro automatizaci testování

Kapitola je zaměřena na trh s automatizačními nástroji. Nástroje jsou rozděleny do tabulek. Jedná se o vyhledaná data z dostupných internetových zdrojů. Jsou využity i některé studie, které se podobným zaměřením zabývaly, ale jelikož se jedná o hodně konkurenční prostředí, studie nejsou zdarma nebo jsou dostupné jen jejich části.

4.1. Testovací nástroje zdarma

Existují nástroje, které jsou zcela zdarma a podporují testování uživatelského rozhraní. Uvedené testovací nástroje by měly sloužit k testování desktopových aplikací.

Tabulka 7 - Testovací nástroje zdarma

Testovací nástroj	Klady	Zápory	Podpora OS	Podpora aplikací
WinAppDriver	Vyvíjen Microsoftem Využívá výhod průmyslového standardu Web Driver protocol Kompilace s JSON wire protocol a Appium	Zastavení vývoje	Windows	UWP, WPF, WinForms, Win32
Winium	Podpora mnoha programovacích jazyků (Java, PHP, Python, Node.js, C# atd.) Lze využít s jakýmkoliv testovacím frameworkem	Vypadá, že už dlouho nebyl aktualizován	Windows	WinForms, WPF
Jubula	Tvorba testovacího scénáře bez kódování	Nepodporuje headless mode (tzn. načítá UI prohlížeče při exekuci testu) Podpora Java RCP/ SWT/ JAVAFX, Swin app	Windows,Linux, Unix and MAC	Java app

Testovací nástroj	Klady	Zápory	Podpora OS	Podpora aplikací
Pywinauto	Poslední aktualizace podporuje MS UI Automation Python knihovna	Nepodporuje ostatní programovací jazyky	Windows, MAC, Linux	Win forms, WPF, Qt browsers, Win32
OATS – Oracle Application Testing Suite	Vhodné k regresním, výkonnostním a databázovým testům	Nevhodné k jiným než Oracle aplikacím	Windows, Linux	Založených na Oracle
SikuliX	Vhodné pro aplikace, kde se těžko hledá jiné řešení Automatizace všeho, co máme na obrazovce Metoda rozpoznání obrazu	Metoda rozpoznání obrazu nemusí být vždy úplně spolehlivá	Windows, MAC, Linux/Unix	Pracuje na základě rozpoznání obrazu (podobné GUI masteru)
FlaUI	Vhodné pro automatizaci rozhraní oken .NET knihovna	Podpora pouze jazyk C# Chybí opakující se mechanismy a další vlastnosti	Windows	Win32, WinForms, WPF, Store apps
Autoit	Snadná syntaxe kódování	Nepodporuje Javu	Windows	COM
Appium	Testeři ho znají Je vhodný pro mobilní aplikace	Pozastaveno Microsoftem (Funkcionality spojené s **WinAppDriver)**	MAC, Windows	iOS, Android, Windows SDKs
Robot Framework	Snadná instalace a ovládání Podpora různých technologií a knihoven (SSH, REST, SOAP) Velmi aktivní uživatelská komunita	Nepodporuje populární IDE	Windows, Linux	Podpora například javaFX app, Java Web start app a další

4.2. RPA nástroje

Následující část uvádí RPA nástroje s významným tržním podílem, jejichž výrobce lze zařadit k lídrům v daném odvětví a zároveň mají návaznost na SAP.

Tabulka 8 - RPA nástroje

Společnost	Počet zákazníků	FREE trial - dny	Příjmy 2021 – mil.USD	Návrh automatizace	Vhodné pro SAP	Cena plné verze ($/rok[1])
UiPath	>7900	ANO - 60	608	Grafický (hybrid)	ANO	7000
Automation Anywhere	>2800	Ano - 30	2,700	Textový kód	ANO	5000
Blue Prism	>2000	Ano - 30	108	Grafický	ANO	3000
NICE	>700	ANO - 2 roky	2	Textový kód	ANO	5800 (až 3. rokem)
Pegasystems	>677	ANO - 30	1,190	Grafický (hybrid)	ANO	Na vyžádání
KOFAX	>25000	Ano - 90	760	Grafický (hybrid)	ANO	Na vyžádání
NTT Advanced Technology Corp.	<100	Na vyžádání	18	Grafický (hybrid)	ANO	Na vyžádání
EdgeVerve Systems Ltd.	>370	ANO - neuvedeno	365	Grafický (hybrid)	ANO	Na vyžádání
FPT Software	>700	Ano - neuvedeno	513	Grafický (hybrid)	ANO	Na vyžádání
OnviSource	<100	ANO - 30	13	Textový kód	Spíše NE	Na vyžádání
HelpSystems	>8700	Ano - 30	137	Grafický (hybrid)	ANO	2000
SAP (IRPA)	>22500	ANO - 30 (rozšíření na 90)	9,230	Grafický (hybrid)	ANO (SAP produkt)	565/měsíc

Velmi často se u RPA nástrojů ještě dělí licence na roboty, které vyžadují kooperaci s člověkem a na ty samostatné bez kooperace s ním (attended and unattended robots - bots). Cenu ovlivní také počet robotů (botů). U menších společností se cena za licenci mění dle konkrétního řešení, a tudíž není fixně zadaná. Při vyhledávání a náhledu do daných prostředí, ve kterých se vyvíjí automatizační skript, lze konstatovat, že většina prostředí si je velmi podobná, a to zejména graficky na první pohled. Za významný hodnotící faktor jednotlivých produktů lze stanovit počet zákazníků. Rozsáhlá komunita uživatelů významně zvyšuje podporu nástroje a usnadní dostupnost informací ke školení. Počet zákazníků neodpovídá počtu

[1] pokud není uvedeno jinak v řádku

uživatelů, který je mnohonásobně vyšší. K momentální špičce na trhu se řadí první tři zmíněné společnosti.

4.3. Test automation nástroje

Tabulka 9 - Test automation nástroje

Nástroj TA	Počet zákazníků	Podpora programovacích jazyků	OS a podpora aplikaci	Vhodné pro SAP	Licence	Znalost programování
Selenium	>54000	Python, JavaScript, Java, Ruby, C#	Křížová platforma Pouze WEB	ANO	Open Source	ANO
SoapUI	<100	Groovy a JAvaScript	Křížová platforma Pouze WEB	NE	Open Source PRO placena 599$/rok	ANO V PRO NE
Appium	Nenalezeno Git - komunita)	Java, Pythong, Ruby, C#, JavaScript, PHP	Křížová platforma Desktop i web	NE (vhodné pro mobilní aplikace	Open Source	ANO
Cucumber	>3700	Původně jen Ruby, nyní Java, C++, C#, node.js, další jsou neoficiální (PHP, Python)	Křížová platforma Desktop i web	NE	Open Source PRO - 11EUR/měsíc	ANO
UFT (QTP)	>3700	Visual Basic scripting (VB)	Windows	ANO	3200$/rok	NE
Tricentis Tosca	>1800	Visual Basic scripting (VB)	Windows	ANO	Placená – na vyžádání	NE
Ranorex	>2400	Visual Basic scripting, VB.NET a C#	Křížová platforma Desktop, WEB i mobil	ANO	Placená 3 typy 2890 EUR 4790 EUR 690 EUR	NE
Katalon studio	>10000	Visual Basic scripting, Groovy, Java	Křížová platforma	ANO	Verze zdarma i PRO 839 $/rok	NE
LoadRunner	>9000	C, C#, Java and Javascript	Windows	ANO	Placená	ANO

Nejznámějším a nejvyužívanějším nástrojem je bez pochyb Selenium. Svědčí o tom rozsáhlá komunita uživatelů a také požadavek na znalost tohoto nástroje v pracovních nabídkách. Je očividné, že většina nástrojů, kde je nutná znalost programování jsou většinou zdarma nebo za nízký poplatek a naopak, kde není nutná znalost nebo základní, jsou tyto nástroje poměrně drahé. Pro přesnější posouzení hospodárnosti a efektivity výběru TA, resp. RPA by bylo nutné provést analýzu nákladů na zkušeného testera s pokročilými znalostmi programování s porovnáním nástroje, a také rozdílu, kolik lidí je schopno využívat takový nástroj.

4.4. Testovací nástroje vhodné pro SAP

Pro úplnost je v následující tabulce uvedena kategorie nejvhodnějších testovacích nástrojů pro SAP jednak dle oblíbenosti uživatelů a zároveň dle mého průzkumu, doplněna o charakteristické hodnotící znaky.

Tabulka 10 - Testovací nástroje vhodné pro SAP inspirováno [37], sestavil autor

Nástroj	Počet zákazníků	Free trial – dny	Licence - plná verze	Hlavní výhoda
Workoft	> 600	Ano - na vyžádání	6000$/rok	Zlatý standard pro SAP (Pokrytí všech SAP aplikací). Snadná integrace s dalšími testovacími nástroji. Ověřený nástroj.
RightData	<100	Ano - na vyžádání	Na vyžádání	Schopnost nalézt data, která je třeba testovat na základě dotazování, analýzy a profilování.
Testimony	>100	Demo - na vyžádání	Na vyžádání	Využívá tiketový systém na analýzu chyb a přiřazuje stupeň závady.
Qualibrate	>50	ANO	Na vyžádání	Jedná se o cloudové řešení pro SAP projekty.
Leapwork	>50	Ano - 14	10000EUR/rok	Velmi snadný nástroj k naučení. Během 14 dní lze nabýt dostatečné znalosti na práci s programem.

Z předchozích tabulek bych mezi nejvhodnější pro SAP dále přiřadil z RPA kategorie UiPath, Automation Anywhere, Blue Prism a SAP IRPA. Ze skupiny Test automation nelze vynechat Selenium, Ranorex a Tricentis Tosca.

Uvedené informace jsou vyhledané z oficiálních webových stránek společností, uživatelských recenzí a mailových komunikací se zástupci společností. Je těžké doporučit konkrétní produkt pouze na základě obecných znalostí daného nástroje. Rozhodujícím faktorem je vyzkoušení daného produktu na konkrétní úloze, kterou chceme automatizovat, čemuž se budu věnovat v praktické části.

5. Automatický testovací nástroj GUI Master

V této kapitole shrnuji poznatky o nástroji, který jsem využíval k automatizaci zadaného procesu a jeho využíváním jsem postupně získával potřebné zkušenosti. Hlavním kritériem výběru tohoto nástroje byla možnost bezplatného zapůjčení jeho výrobcem, unikátní technologie nástroje uvedená v následujícím odstavci a skutečnost, že zaměstnavatel hledá nové nástroje, které pomohou optimalizovat a usnadnit spravované procesy. Mým úkolem tedy bylo automatizovat zadaný proces, ověřit vybraný nástroj na praktickém příkladě a poté rozhodnout, zda by mohl být vhodný pro širší využití v týmu.

5.1. Obecné informace GUI Master

„GUI master je aplikace určená pro automatizované testování grafických uživatelských rozhraní. Pracuje na principu vyhledávání zadaných bitmapových vzorů na obrazovce. Bitmapové vzory následně řídí realizaci testovacího scénáře přes" [38]:

- Zadání údajů do vstupních polí
- Výběr položek z výčtových seznamů a nabídek (menu)
- Rolování obrazovek
- Synchronizace obrazovek
- Vyhodnocování chybových stavů
- Sběr dat z obrazovek pro účely verifikace výsledků
- Vizuální grafickou kontrolu výstupních obrazovek nebo jiných dokumentů

GUI Master pracuje na technologii identifikace bitmap, kdy považuje rozhraní každé aplikace za soubor bitmapových vstupů, které jsou vyhodnoceny speciální sadou algoritmů. Tato koncepce umožňuje nezávislost na testovaných aplikacích, včetně mobilních nebo terminálových. Pracuje v prostředí OS Windows a funkcionality nástroje se stále rozvíjí.

Nástroj využívá vlastní skriptovací jazyk, pomocí kterého lze nástroj modifikovat podle projektových požadavků [38].

Nástroj se skládá ze dvou licencovaných programů:

- GUI Master Editor – program, který slouží k tvorbě automatických scénářů pomocí vývojových diagramů (stromového kódu) a makropříkazů z vlastní knihovny, které nabízí specifické vstupy, výstupy a vlastnosti.
- GUI Master Launcher – slouží ke správě již hotových testovacích scénářů a jejich řízené exekuci. Součástí výstupů běhů scénářů jsou výkazy, které zachycují úspěšné či neúspěšné provedení.

Tabulka 11 - Cena GUI Master

	Cena trvalé licence (tis. Kč)
GUI Master Editor	100
GUI Master Launcher	5

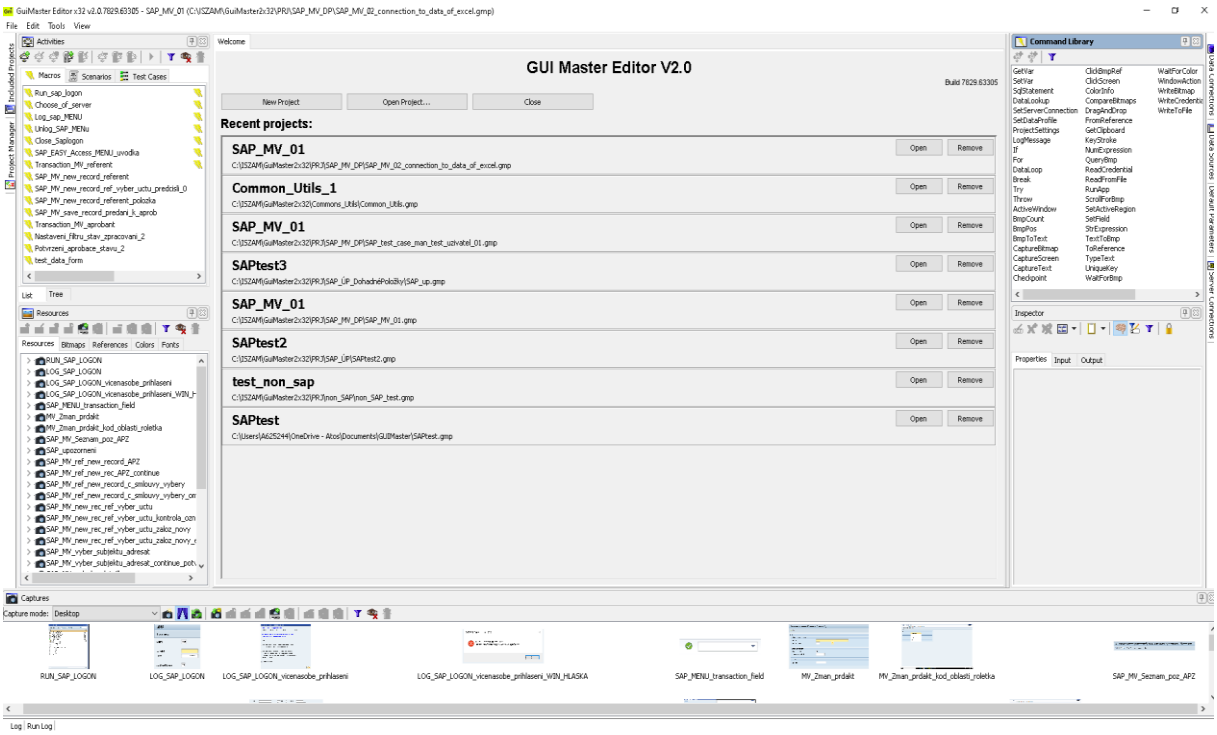
(Cena je orientační)

Lze tedy říct, že GUI master je určitou kombinací nástrojů RPA a Test Automation.

5.2. GUI Master Editor

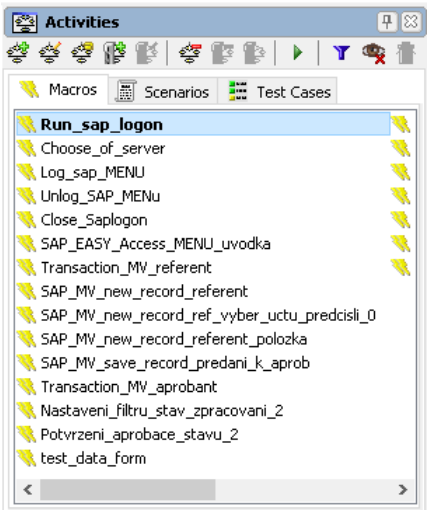
Není cílem se zde věnovat různým nastavením nástroje a prvním spuštěním. Informace, které jsou nutné k práci s GUI Masterem jsou uvedeny na produktových stránkách [39]. Pro návaznost na hodnocení nástroje v následující kapitole, je uvedeno alespoň několik náhledů z programu.

5.2.1. Náhled do nástroje



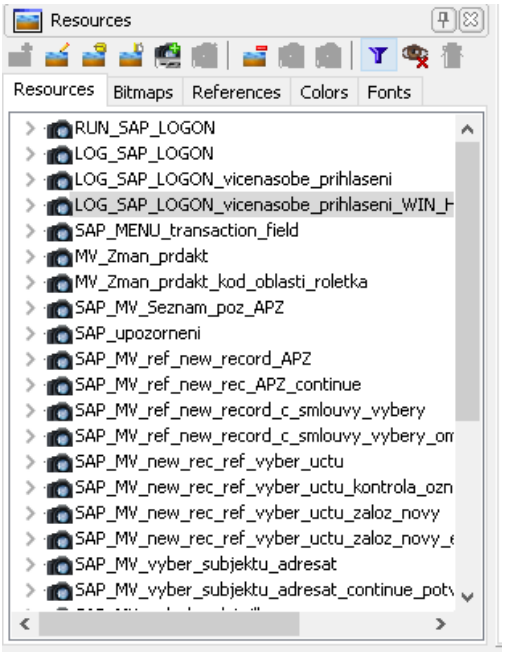
Obrázek 17 - GUI master Editor - základní náhled

Na obrázku výše můžeme vidět základní náhled do nástroje GUI Master Editor. Skládá se z karet Aktivity, Zdroje, Zachycení (captures), Knihovna příkazů a Inspektor. Nachází se zde samozřejmě mnoho dalších funkcionalit, ale pro základní znázornění principu postačuje těchto pět. Prvním krokem před prací je však založení projektu, kde se ukládají veškerá data spojená s tvorbou skriptu.



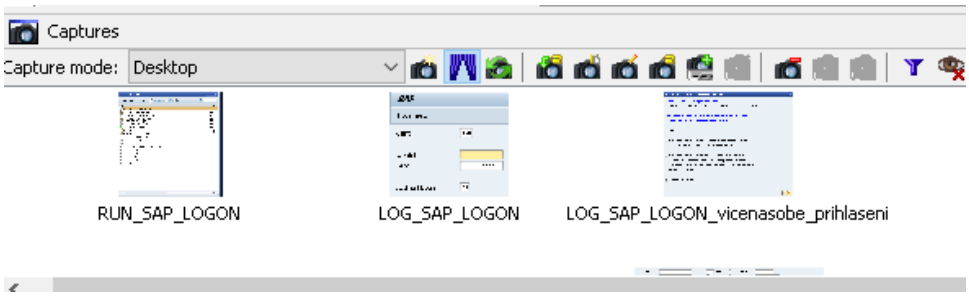
Obrázek 18 - Editor - Aktivity

Karta činnosti (Activities) se dále dělí na makra, scénáře a celé testovací případy. Každý z nich má stejné vlastnosti a slouží zejména pro větší přehled v jednotlivých scénářích, které mezi sebou můžeme kombinovat.



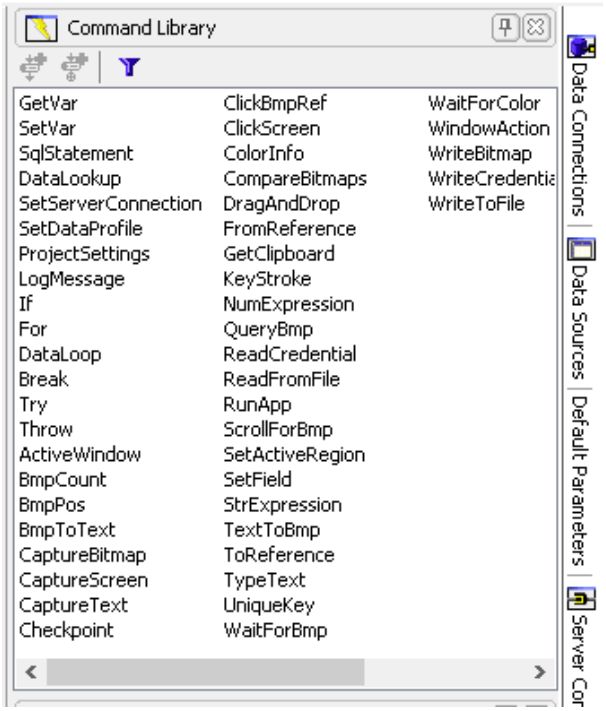
Obrázek 19 - Editor - Zdroje

Do zdrojů (Resources) si ukládáme veškeré objekty, které dělíme na obrázky, bitmapy, referenční body, barvy a fonty. Musíme vždy vybrat vhodný druh zdroje pro danou aplikaci tak, aby na dané obrazovce byl tento objekt unikátní. Zdroje slouží zejména k identifikaci objektů, které budeme na obrazovce vyhledávat a pracovat s nimi.



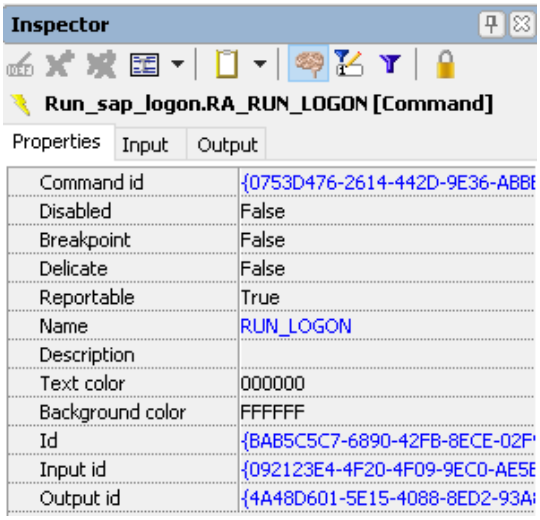
Obrázek 20 - Editor - Captures

Snímky obrazovky (Captures) slouží k zaznamenání jednotlivých částí obrazovky a vytváří databázi bitmapových vzorů, se kterými pracujeme v rámci automatizace.



Obrázek 21 - Editor - Knihovna příkazů

V knihovně příkazů (Command Library) se nachází veškeré příkazy, ze kterých lze sestavit jednotlivé skripty.



Obrázek 22 - Editor - Inspektor

V inspektoru (Inspector) se nachází vlastnosti jednotlivých příkazů. Některé jsou předdefinované a je nutné je v kontextu testovacího případu upravit. Inspektor obsahuje vstupy a výstupy, se kterými pracujeme.

5.2.2. Hodnocení Editoru

Za největší slabinu Editoru považuji jeho samotný vzhled. Když se podíváme do základních náhledů podobných SW, například od UiPath, na první pohled vypadají konkurenční produkty mnohem moderněji.

Za nejpřínosnější považuji přehlednost a propracovanost jednotlivých karet, které využíváme. Nástroj má jednoznačně velmi promyšlený obsah s ohledem na to, že téměř veškeré nutné karty pro tvorbu skriptu máme hned na úvodní obrazovce a pro základní tvorbu skriptu nemusíme zdlouhavě hledat další místa. Téměř vše je na jednom místě.

V průběhu práce s programem jsem si vedl bodový deník, ze kterého bych rád zmínil několik dalších podstatných nálezů:

- Pro získání popisu funkcionality jednotlivých příkazů je vytvořena stránka: https://wiki.getguimaster.com/index.php?title=Hlavní_strana, na které se nachází detailní online dokumentace. Při práci s jednotlivými příkazy nemusí být optimální se při potřebě nápovědy koukat někam jinam. Bylo by vhodné mít možnost se

podívat na stručný popis příkazu přímo v programu, například přes funkční klávesu F1 (Nápověda), jak je zvykem.

- V případě umístění myši na jednotlivé příkazy v knihovně nebo v inspektoru vlastností, se objeví pouze strohá nápověda (tooltip), ale jen po velmi krátkou chvíli (rychle zmizí). Uživatel nabídnutý text obvykle nestihne ani přečíst. Možným řešením je vytěžit předchozí bod, případně prodloužit dobu zobrazení tooltipu, nebo vázat zobrazení na současný stisk pomocné klávesy (Alt, Shift nebo Ctrl + MouseOver).

- Podobný problém vidím při využití Debuggeru. Na kartě se zobrazí číslo chyby nebo krátký popis, například "Bitmap is too small for mask" bez dalšího vysvětlení. Uživatel by jistě ocenil sborník chyb s popisem jejich pravděpodobných příčin a vhodného řešení.

- Velmi nepříjemná situace nastává v případě chyby, která se objeví při běhu ve smyčce, protože Debugger za pomoci Analyzátoru zvládne analyzovat jen poslední chybu v dané sekvenci. Tzn. k analýze není dostupná první chyba (která způsobila zastavení sekvence), ale až následující chyba, která je pouze důsledkem nezobrazení očekávané sekvence (bitmapy atd.). Nemohu analyzovat příčinu ale pouze důsledek. Standartně by mělo stačit využít debuggování skriptu po jednotlivých krocích a chybu nalézt, ale to nestačí na náhodné chyby objevující se pouze v zacyklení. Přesně určit danou chybu není možné a je nutné hledat příčinu náhodně. Na tento druh náhodných chyb mi bylo doporučeno nahrání obrazovky pro celý scénář a následná detekce chyb. To nepovažuji za dostačující řešení.

- Zvýšení použitelnosti SW by určitě prospěla knihovna s příklady hotových skriptů, do kterých by uživatel doplnil pouze vstupy. Nesměl by chybět popis příkladů a jejich funkcionality, také informace, kde se konkrétně využily. Jsem si jistý, že by to urychlilo pochopení programu a zároveň umožnilo další šíření nástroje.

- K prevenci neočekávaných chyb patří nutnost tzv. ověřujících kliknutí do daného pole. Například funkce SetField by to měla zaručit, ale při mnohonásobném opakování se stane, že se pole nezaktivuje. Podobných řešení se objevuje více, ale nemusí se jednat nutně o chybu GUI Masteru, ale nýbrž o nestandartní chování aplikace či prostředí, kde je test prováděn. Na vrub tohoto problému je však nutné poznamenat, že zákazník se s podobným vysvětlením nespokojí a "vinu" nefunkčnosti přisoudí celému řešení.

- Za jeden z největších kladů Editoru považuji implementaci stromového kódu, ale jedná se o můj subjektivní pohled, kdy mi tvorba skriptu tímto způsobem přišla přehledná. Naopak zkušenější programátoři by mohli postrádat možnost tvorby skriptu ve svém preferovaném jazyce. V některých situacích by si tím určitě mohli usnadnit práci.

- Velmi oceňuji možnost plnění dat testovacích scénářů z externích zdrojů pomocí funkce DataLoop.

- Náhodný pád Editoru mne rozhodně překvapil. Stejně jako triviální doporučení, tj. ukládat častěji svůj projekt.

- Zatím není dostupná nějaká komunita uživatelů, která by se mezi sebou mohla bavit o problémech s programem a jejich řešení, což je v dnešní době online skupin, blogů a sociálních sítí velmi překvapivé. Není čas řešit každý detail s konzultanty, nehledě na to, že jsou hodně zaneprázdnění. Podobná komunita by určitě pomohla rozšířit SW mezi další zájemce a mohla by si získat své příznivce.

- Specificky přínosné shledávám práci GUI masteru na vzdálené ploše, kdy lze okno vzdálené plochy zmenšit do malé velikosti a SW nadále vykonává scénář. Tento směr k celkové virtualizaci běhu automatizovaného testu (tj. Vzdálené spuštění na virtuálním PC bez monitoru) však překračuje rozsah a cíl této práce.

- Při použití techniky “táhni a pusť” (drag&drop) nelze uchopený objekt “upustit” na nedefinovaných místech (například vložení příkazu do tvorby skriptu (workflow), resp. aktivní oblast není vždy intuitivní, např. přesun identifikace bitmapy do vlastností bitmap v inspektoru.

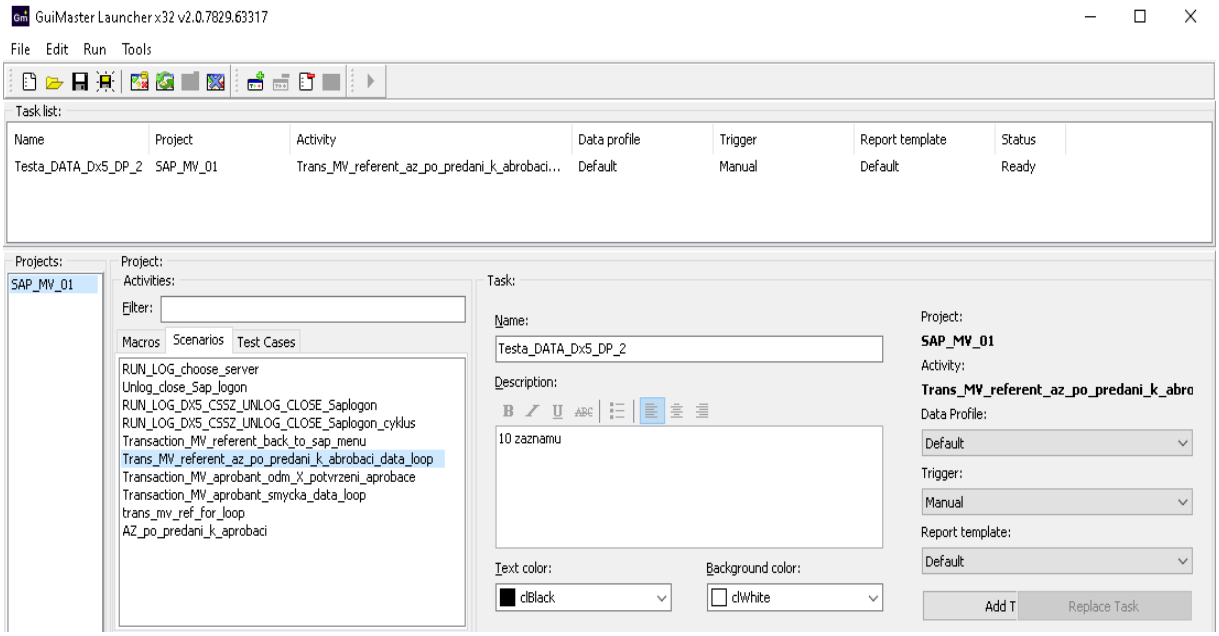
- V případě, kdy chci eliminovat chyby již při vývoji skriptu v Editoru je vhodné zacyklit každou vytvořenou sekvenci zvlášť. Tím předejdu počtu vzniklých chyb při běhu celého scénáře (na které bych narazil až při spuštění scénáře z Launcheru) a zvýším tím spolehlivost automatického prototypu. Zároveň tím zjistím, které příkazy nemají 100% funkčnost v daném prostředí.

Mezi hodnotící kritéria můžeme zařadit také časovou náročnost na pochopení základní práce s Editorem. Na základě vlastní zkušenosti trvalo přibližně 78 hodin, než jsem se naučil základní ovládání, funkce a konstrukce a byl jsem schopen se posunout se znalostmi programu dále. Celkový počet hodin strávených prací s tímto programem je v rozsahu cca (200-300), protože vytvořený skript lze již považovat dle konzultanta výrobce za pokročilejší.

5.3. GUI Master Launcher

Launcher slouží především k exekuci vytvořených scénářů a následné analýze.

5.3.1. Náhled do nástroje



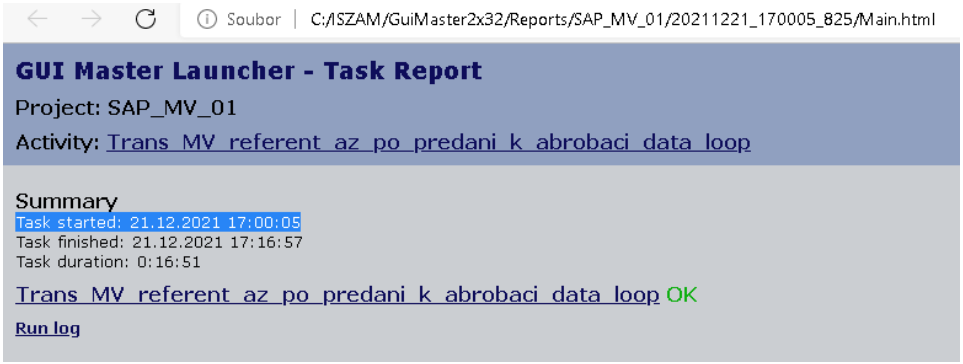
Obrázek 23 - Launcher - Základní náhled

Po zvolení projektu, který chceme spustit, můžeme vidět, že náhled obsahuje podobné položky jako Editor. V levé části máme Projekt, hned vedle aktivity s vybraným scénářem, poté již pojmenování testovacího scénáře s popisem a úplně napravo nastavení datového profilu a vzhledu reportu. Přiřadíme-li scénář do seznamu úkolů, můžeme jej následně již jen spustit. Je to velmi snadné a intuitivní.

Pracovat s programem pro exekuci skriptu může opravdu každý, což je velmi užitečné pro administraci a realizaci automatizovaných scénářů.

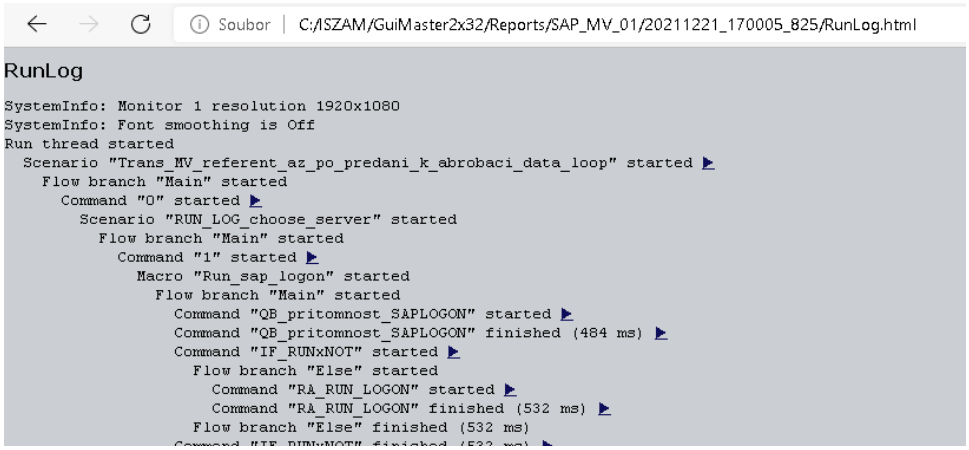
5.3.2. Výkaznictví (Reporting)

Po vykonání testu se vygeneruje výkaz, který obsahuje výsledek a jednotlivé časy běhu. Výkaz je připraven ve formě HTML souboru s názvem Main.html umístěného ve složce projektu s názvem Reports.



Obrázek 24 - Launcher - Report

Report obsahuje vždy název úvodní činnosti. Když ji rozklikneme, zobrazují se další detaily do úrovně makra, ze kterých se daný scénář skládá, vždy s časem začátku, konce a dobou trvání stejně, jak je tomu na obrázku výše.



Obrázek 25 - Launcher - Run Log

Kliknutím na odkaz Run log (viz Obr. 25), dostaneme se do protokolu běhu skriptu, kde nalezneme informace o veškerých příkazech, které v daném scénáři proběhly (viz. Obr. 26).

5.3.3. Hodnocení Launcheru

V následujících bodech shrnuji vlastní postřehy z užití a testování programu Launcher nástroje GUI Master:

- Za největší nedostatek považuji nemožnost přiřadit externí zdroj dat (v mém případě tabulku z MS Excel) v Launcheru. Konzultant

výrobce potvrdil, že tato možnost tady v předchozích verzích byla a lze předpokládat, že na základě většího počtu žádostí může být opět doplněna. Podobně by mohla být umožněna také změna vstupních parametrů.

- Nepříjemné je chování aplikace při užití "citlivých proměnných (delicate variables - proměnné využívané k přihlášení uživatele do systému), kdy je nutné si je v přes záložku edit navolit. Program hodnoty citlivých proměnných neukládá, tudíž je pochopitelné, že se nepropisují z Editoru. Téměř vždy na to zapomenu a jsem si jistý, že další uživatelé taky. Na místě by tedy byla upomínka před spuštěním testu na nutné vyplnění.

- Za velmi nepříznivé považuji chybějící možnost zastavení běhu aplikace, tj. tlačítko stop nebo přerušit).

- Slabší částí nástroje je reportování běhu scénářů. V případě většího počtu dat mi přijde v porovnání s konkurencí nedostačující a hlavně nepřehledná. Určitě by pomohla lepší vizualizace dat. Pro základní kontrolu výsledku je však dostatečná.

5.4. GUI Master dokumentace

Osobně se mi návod k programu GUI Master moc nelíbil. Pro někoho, kdo se má učit s novým nástrojem samostudiem považuji dokumentaci za nedostatečnou. Vyvstává otázka, komu přesně má dokumentace sloužit. Opět jsem si poznamenal pár bodů:

- Dle návodů není vše úplně intuitivní, tj. ani po přečtení návodů není zřejmé, jakým postupem lze dosáhnout požadovaného cíle.

- Některé body již nejsou aktuální (což je sice pochopitelné, protože se produkt neustále vyvíjí, nicméně vzhledem k existenci online wiki dokumentaci je překvapivé, že návody nejsou dostupné online nebo nejsou aktualizovány).

- Nastavit a pochopit program nelze pouze na základě dokumentace a některé části jsou v různých dokumentech, vše

by mělo být sjednoceno na jednom místě. Podporuje to například formulace věty “Strukturu složek určuje (praxe) uživatele“. Pro nováčka by měl být návod průvodcem.

- Dokumentace se až zbytečně opírá o školení, které je nadstavbou. Zdokonalení návodů a podpora větší komunity uživatelů může zásadně ušetřit finance zákazníka, a hlavně čas na školení.

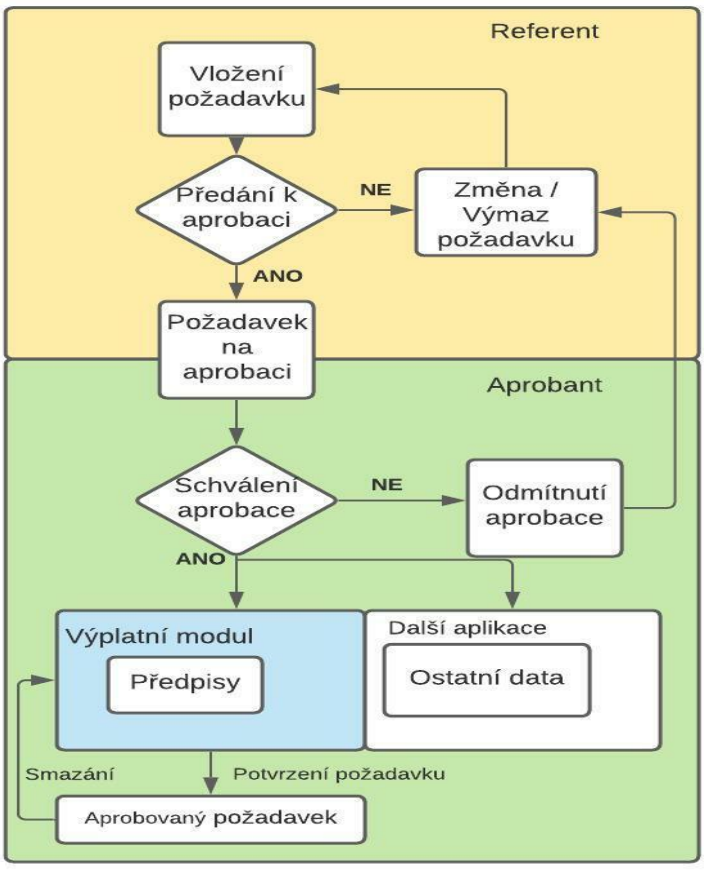
- Obecně bych vypustil body týkající se běžně známých vlastností ovládání Windows (např. uzavření okna tlačítkem X v pravém horním rohu pracovního okna apod.). Naopak doporučuji rozšířit témata funkčních postupů.

6. Vlastní návrh automatizovaného testu

Požadavkem na automatizaci bylo založení platby za pomoci manuálního vstupu v oblasti APZ (Aktivní politika zaměstnanosti) pro následné testování zpracování plateb. Jedná se o jeden z nejčastějších požadavků na test při změnách v systému JVM (Jednotný Výplatní Modul), jak ve vývojovém, tak v testovacím prostředí. Jedná se zároveň o optimální ověření, zda je v systému JVM vše nastaveno a naprogramováno tak, aby platby byly úspěšně zpracovány.

Mým úkolem tedy bylo zautomatizovat krok po kroku vše, co uživatel musí udělat, aby úspěšně založil platbu v manuálním vstupu. Tato část se skládá ze dvou transakcí:

- ZMAN_PRDAKT_SEZNAM_R (Referent)
- ZMAN_PRDAKT_SEZNAM_A (Aprobant)



Obrázek 26 - Proces manuálního vstupu - diagram

Pro obě transakce jsem vytvořil dva automatické scénáře za pomoci nástroje GUI Master, ve kterých transakce Referenta předá platbu do transakce Aprobant a následně je schválena nebo odmítnuta dle varianty scénáře. Po schválení Aprobantem je předána do transakce Uvolnění a zpracování požadavků výplat. Zde můj automatický testovací scénář končí. Následně se musí provést ještě několik manuálních kroků v transakci ZVYP_PRDAKT_ZPR, aby byla platba odeslána a zpracována. Požadavek lze tedy označit za automatickou přípravu plateb manuálním vstupem pro následné zpracování modulem JVM.

6.1. Kroky před vytvořením scénáře (skriptu)

6.1.1. Cíl automatizace

Cílem by měl být automat, který na základě vstupních dat vytvoří platbu v manuálním vstupu a předá ji k dalšímu zpracování.

6.1.2. Místo automatizace

S cílem automatizace je jednoznačně spojeno také, kde budeme automatizovat, jelikož se jedná o zejména automatizaci uživatelské práce, je nejvhodnější testovat přímo na uživatelském rozhraní.

6.1.3. Vhodný testovací nástroj

Zásadním předpokladem, vzhledem k požadavku na automatizaci je, zda je nástroj určen k testování na uživatelském rozhraní, a to GUI Master splňuje. Dalším důležitým předpokladem je, zda je možné využití v prostředí SAP, a to také splňuje, protože má univerzální využití.

Na základě dosaženého výsledku, tj. vytvořeného skriptu v nástroji GUI Master, mohu konstatovat, že GUI Master je na podobné aplikace jednoznačně vhodný. Tento nástroj byl sice vybrán k ověření použitelnosti na platformě SAP vedením společnosti, nicméně před mojí realizací byl nástroj doporučován pouze výrobcem. Mým úkolem bylo faktické ověření na praktickém příkladě, tedy test připravit, provést a vyhodnotit. Je nutné poznamenat, že k úspěšnému výběru nástroje výrazně pomohou zejména praktické zkušenosti s podobnými nástroji.

6.1.4. Stabilní testovací prostředí

Před samotnou tvorbou skriptu jsem si musel s kolegy dohodnout a vytvořit vhodné místo k testování, kde bude dostupný veškerý SW a HW. Kolegy zmiňuji, protože hlavní myšlenka ve využití testovacího nástroje je v neustálém, opakovaném využívání, aby se co nejvíce vyplatila jeho koupě. V rámci přípravy prostředí jsme vyhradili specifické PC, na které byly instalovány veškeré programy a licence nutné k tvorbě testu v prostředí SAP. Předpokládalo se, že se na této stanici vytvoří testovací scénář a následně ho využijeme například u zákazníka nebo kdekoliv jinde. V tomto bodě později nastal problém, protože ke spuštění scénáře je stále nutné zakoupit licenci GUI Master Launcher. To shledávám jako jednu z největších slabin v možném rozšíření tohoto programu.

Při přípravě prostředí jsem se setkal s první možnou automatizací a tou je přihlášení do zákaznické VPN. Jedná se o první věc, kterou uživatel musí provést, aby se mohl přihlásit prostřednictvím aplikace SAP Logon k cílovému serveru. Musí být připojen ke správné síti. Jelikož se jedná o velmi častou činnost, její automatizace je určitě na místě. K tomuto jsem však nakonec nevyužil nástroj GUI Master, ale jednoduchý dávkový skript.

Ten zaručí spuštění VPN klienta (zde Cisco AnyConnect Secure Mobility) a autentizaci prostřednictvím přihlašovacích údajů. Obsah dávkového spustitelného souboru:

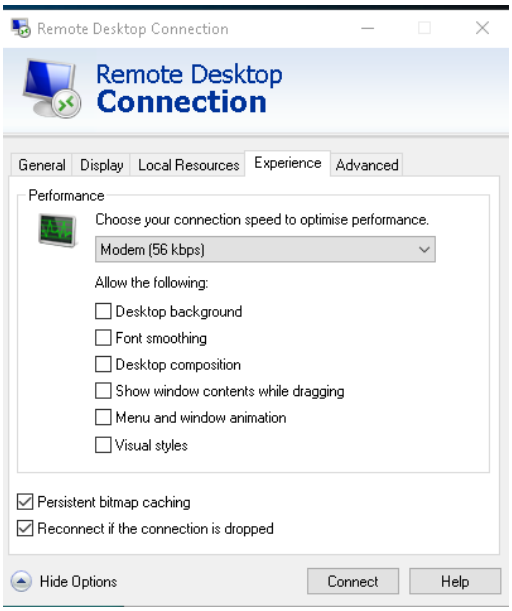
```
@echo off
taskkill/F /IM vpnui.exe
"%ProgramFiles(x86)%\Cisco\Cisco AnyConnect Secure Mobility
Client\vpncli.exe" -s < "C:\Users\a785109\AppData\Roaming\.login_info.txt"
"%ProgramFiles(x86)%\Cisco\Cisco AnyConnect Secure Mobility
Client\vpnui.exe"
```

Obsah textového souboru login_info.txt s přihlašovacími údaji (údaje co vyplňujeme jsou *):

```
Connect **.***.**.**
Y
DEV-any
Prihlasovaci jmeno
Heslo
```

Jednoduchý skript nám velmi urychlí přihlášení do VPN.

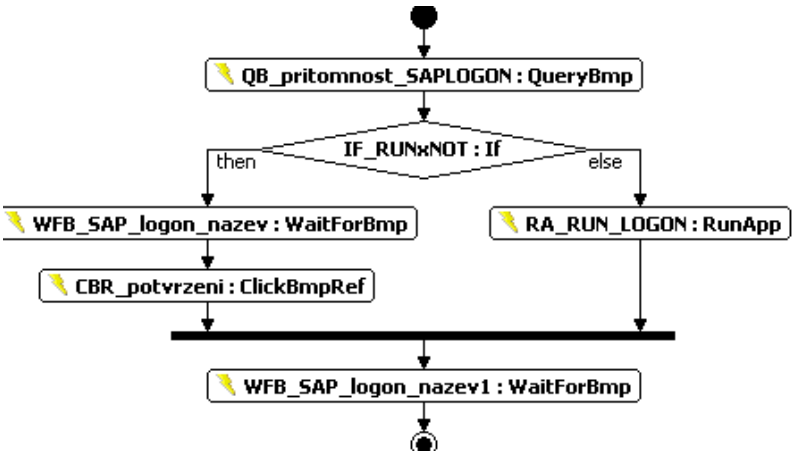
Nechci uvádět veškeré nastavení systému, které je nutné pro správnou práci GUI Masteru, většina je uvedena v uživatelské příručce, která je volně dostupná od výrobce, a zároveň ji uvádím v přílohách této práce. Nicméně, nastavení vzdálené plochy se v něm nenachází a je nutné, protože jinak by GUI Master nemusel pracovat správně.



Obrázek 27 - Nastavení vzdálené plochy

Je nutné si vzdálenou plochu nastavit dle obrázku výše, protože by například mohlo docházet k nerozpoznání některých bitmap. Jedná se o doporučení od vývojáře GUI Masteru.

6.1.5. První testovací skript



Obrázek 28 - První testovací skript

Na obrázku - První testovací skript, je můj první vytvořený skript, který nejprve detekuje, zda již není aplikace SAP Logon spuštěna. Pokud je nalezena, pouze se aktivuje okno SAP Logon, pokud není, spustí se aplikace a nástroj čeká na unikátní bitmapu na úvodním okně.

6.2. Automatizační skripty

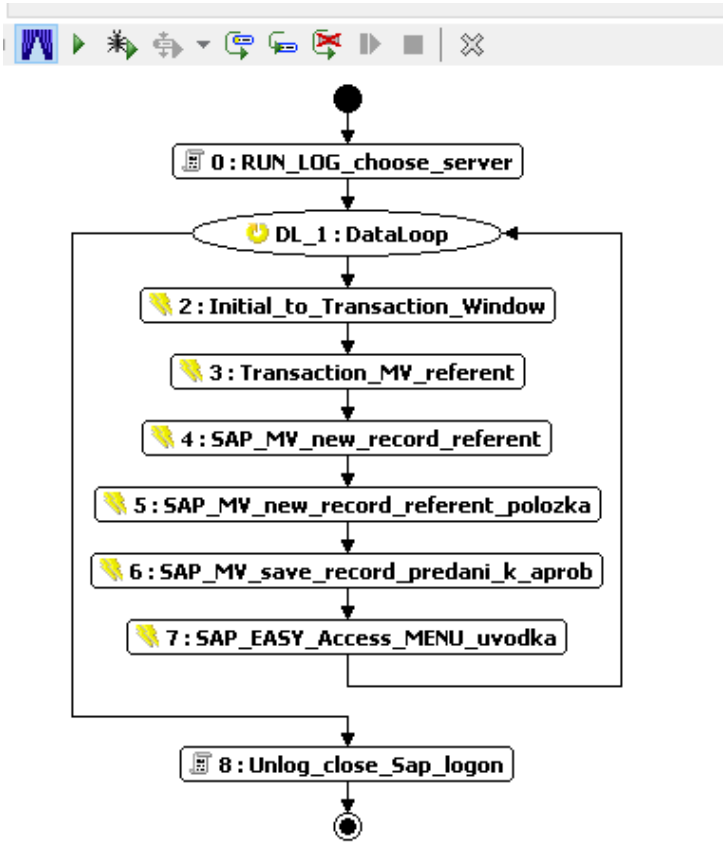
Zde popíšu dva moje hlavní testovací scénáře, které byly požadavkem na automatizaci. Jedná se pouze o základní popis k vytvoření představy způsobu nahrazení uživatelské práce pro dvě transakce:

- ZMAN_PRDAKT_SEZNAM_R (Referent)
- ZMAN_PRDAKT_SEZNAM_A (Aprobant)

6.2.1. Skript pro transakci referenta

Skript by měl provést stejné kroky, které musí udělat uživatel, aby založil platbu v manuálním vstupu. Kroky jsou:

1. Spuštění programu SAP logon (verze 760)
2. Výběr serveru
3. Přihlášení k serveru (vyplnění klienta a přihlašovacích údajů)
4. Aktivace transakčního okna, vložení a potvrzení transakce
5. Vyplnění kódu oblasti a organizační jednotky v transakci
6. Založení nového záznamu
7. Vyplnění všech nutných polí platby
8. Uložení
9. Předání k aprobaci
10. Odhlášení a uzavření aplikace

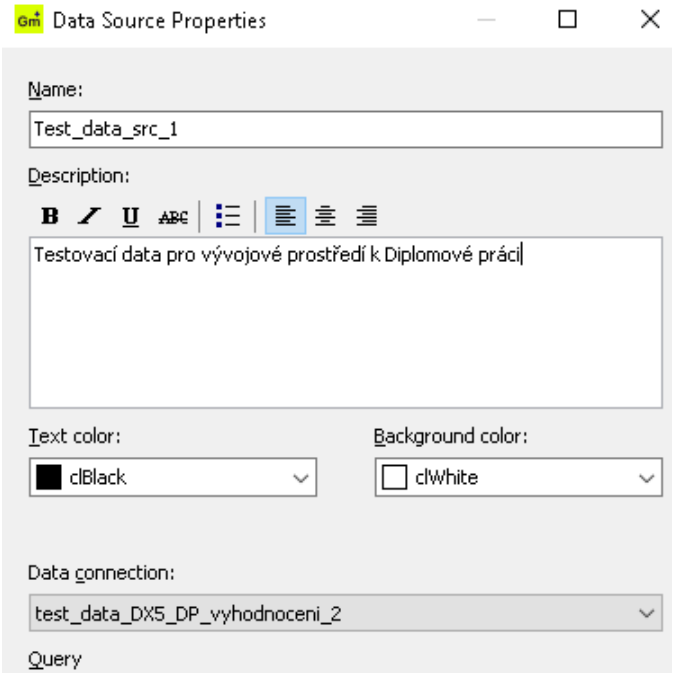


Obrázek 29 - Automatizační skript - Referent

Skript se skládá z osmi hlavních maker, které se případně dále rozpadají do dalších atd. V zásadě skript kopíruje kroky uživatele s úpravou vedoucí k opakování a minimalizaci chybovosti. Kroky skriptu jsou:

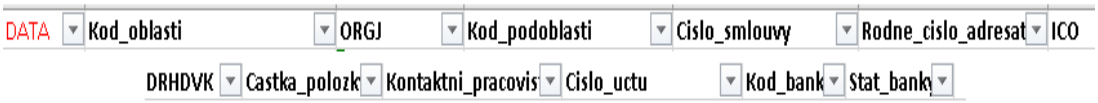
0: Obsahuje spuštění, výběr serveru a přihlášení do SAP logon

1: Datová smyčka, která obsahuje vstupní soubor s daty, které se mají plnit do jednotlivých polí. Zde je nejprve nutné vytvořit datovou konektivitu, v mém případě přes ODBC ovladač, který podporuje excelové soubory (xls, xlsx atd.). Součástí definice spojení je určení excelového souboru. Toto datové propojení se dále přiřadí do datového zdroje, kde výběr dat určuje SQL příkaz. Nakonec je nutné tento datový zdroj přiřadit do vstupních parametrů datové smyčky.



Obrázek 30 - Napojení datového zdroje

Jednotlivá makra jsou napojena na vstupní parametry (názvy níže). Ve vstupních datech můžeme pracovat s větším počtem záznamů. V mé realizaci jsem běžně používal pro přípravu testovacích dat 10 a více záznamů.



Obrázek 31 - Vstupní data

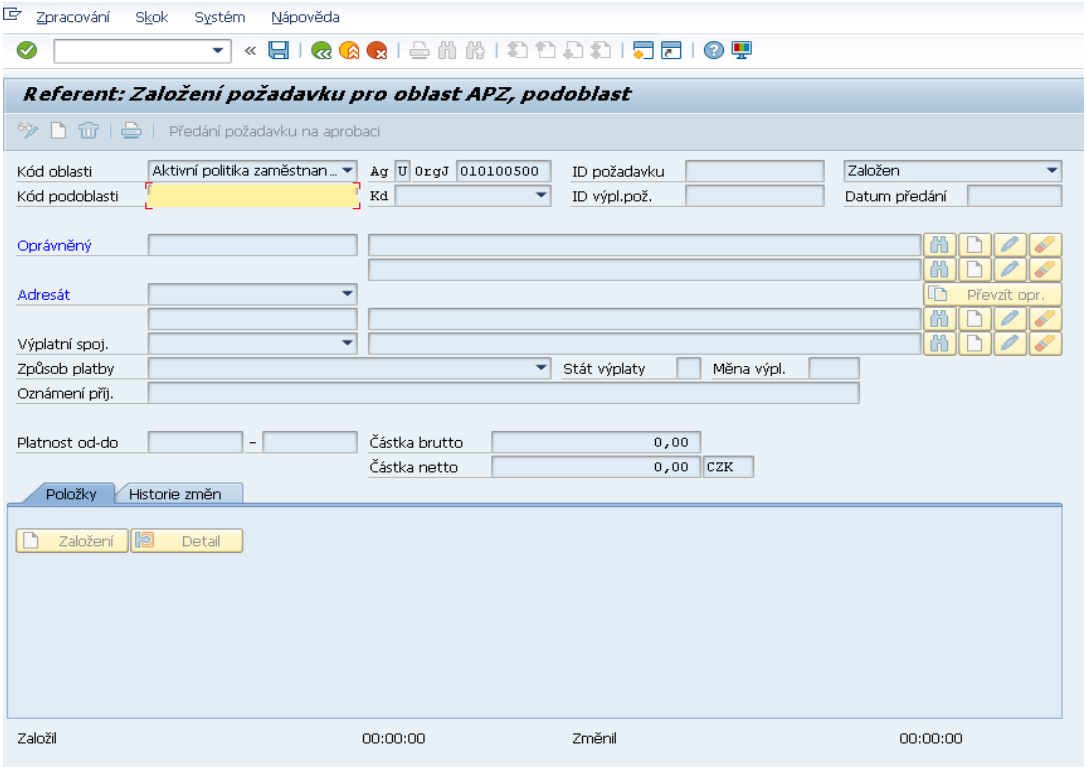
Při přípravě vstupních dat je nutná znalost souvislostí vstupních dat, aby byly kombinace přípustné pro danou oblast v manuálním vstupu a kontroly programu v SAPu platbu přijaly.

2: Tento krok slouží k inicializaci transakčního okna (uzavře vše navíc), abychom vždy mohli začít/pokračovat s čistým prostředím pro nový záznam. Jedná se o běh v cyklu datové smyčky. Může se tedy stát, že některý ze záznamů bude obsahovat chybu, může být špatně nastaven nebo upraven program v SAPu, kdy chybu

identifikujeme až při analýze v Launcheru. Nechceme, aby nám chyba jednoho záznamu zastavila celý scénář.

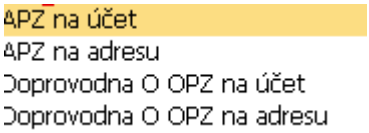
3: Vložení transakce, výběr oblasti a organizační jednotky, potvrzení.

4: Zde již plnění platby v manuálním vstupu daty

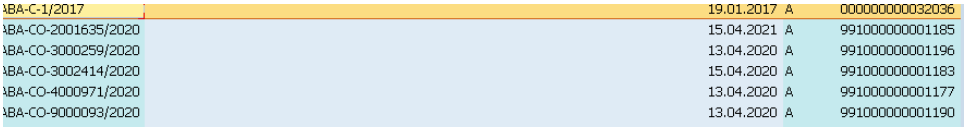


Obrázek 32 - Náhled do transakce Referenta - SAP

Data se vpisují přímo do textových polí, vybírají se z rolovacích lišt nebo se vyhledávají přes výběr možností (funkční klávesa F4 - matchcode). Tento výběr GM provádí tak, že se text převádí na bitmapu, pro kterou se vyhledává shoda.



Obrázek 33 - rolovací lišta - SAP



Obrázek 34 - Výběr dat přes matchcode - SAP

V tomto případě je nutné vyplnit jen a pouze přesný počet znaků, které názvy obsahují, protože se text převádí na bitmapy a vyhledává je.

5: Založí položku, stejný postup jako u kroku 4.

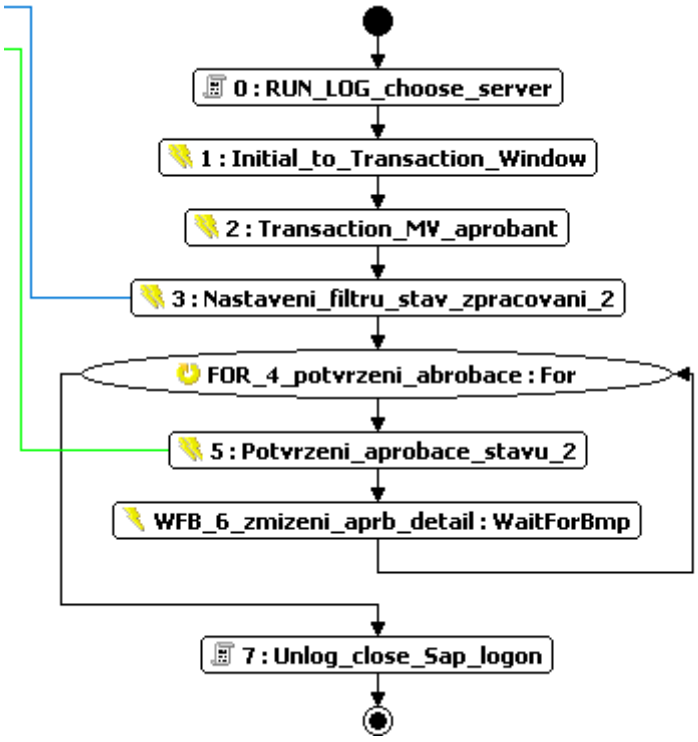
6: Uloží a předá se k aprobaci

7: Krok nás vrací na úvodní obrazovku (mohl by se vynechat, zde je spíše pro využití u jiných transakcí, ale v cyklu se osvědčil. Krok inicializace je dostatečný). Zde končí také datová smyčka, po dokončení běhu všech vstupních dat.

8: Poslední krok je odhlášení a uzavření aplikace SAP logon.

6.2.2. Skript pro transakci aprobanta

Dle požadavku se výběr omezil pouze na oblast a organizační jednotku a poté potvrzení aprobace u všech záznamů.



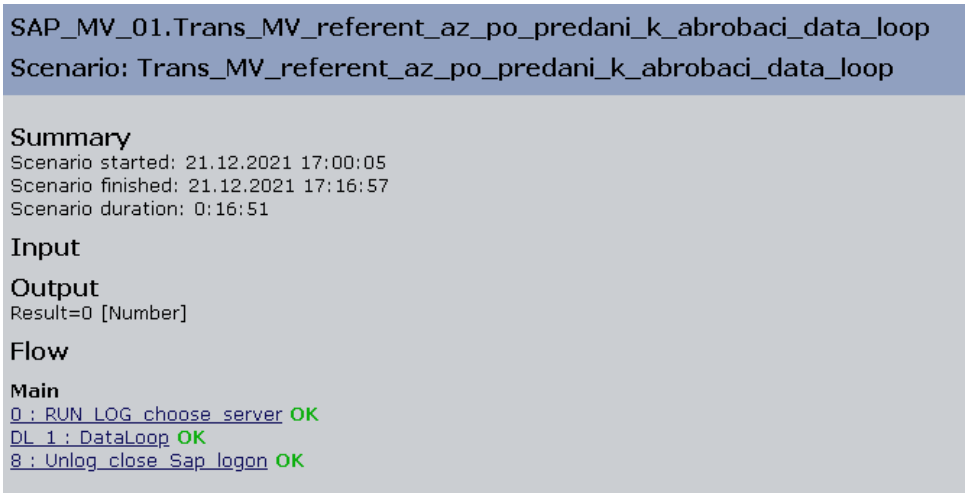
Obrázek 35 - Automatizační skript - Aprobant

Pracovní logika je stejná jako u předchozího skriptu s tím, že po vstupu do transakce, skript nezakládá žádné platby, ale vyfiltruje takové, které mají zvolený stav. Vstupem je pouze organizační

jednotka a stav plateb pro filtr. Následně každou platbu GM otevře a potvrdí aprobaci. Tento scénář slouží opravdu jen k praktickému urychlení, protože tato transakce slouží především jako další kontrola plateb tzv. kontrola čtyř očí. Skript významně zrychluje celý proces testování zpracování platby.

6.2.3. Reporty z testů

Pro oba skripty jsem využil deset sad vstupních dat, které by měly být dostačenou ukázkou spolehlivosti automatizace.

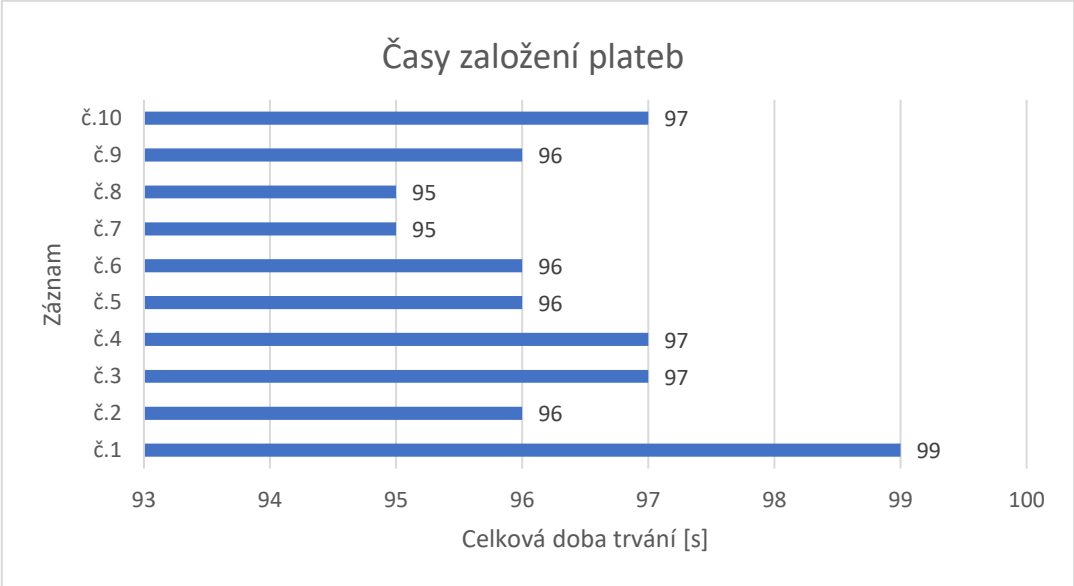


Obrázek 36 - Výsledek testu - skript Referent

Celková doba na založení 10 plateb mým automatem také s přihlášením a odhlášením do aplikace SAPu je 16 minut a 51 sekund. Náhled v Launcheru zobrazuje celkové časy, a zda se jednotlivé testy provedly a nic víc. Detailnější analýzu jsem připravil do tabulky - Tabulka 12 z analýzy jednotlivých maker k dalšímu porovnání.

Tabulka 12 - Časy založení záznamů automatem

Záznam (platba)	Inicializace [s]	Transakce [s]	Založení záznamu [s]	Založení položky [s]	Uložení a předání [s]	Úvodní obrazovka [s]	Doba trvání celková [s]
č.1	0	13	42	34	7	3	99
č.2	1	11	42	33	7	2	96
č.3	1	11	42	33	7	3	97
č.4	1	11	41	34	7	3	97
č.5	1	11	42	33	6	3	96
č.6	1	11	41	33	7	3	96
č.7	0	11	41	33	7	3	95
č.8	0	11	41	33	7	3	95
č.9	0	11	42	33	7	3	96
č.10	1	11	41	34	7	3	97
Všechny	6	112	415	333	69	29	964



Obrázek 37 - Graf - Časy založení plateb

Založení záznamu je zaokrouhlené na celé sekundy. Z grafu lze vyčíst, že se časy založení jednotlivých záznamů výrazně neliší, což považuji za výraz stability. Při tvorbě skriptů jsem vytvořil takovýchto scénářů mnohonásobně víc, než jsem se dostal k uspokojivému výsledku, tj. téměř nulová chybovost v běhu. Nyní se objevují chyby při založení automatem jen výjimečně.

Obrázek 38 - Výsledek testu skript - Aprobant

Celková doba pro potvrzení 10 plateb byla 2 minuty a 13 sekund, jednalo se o potvrzení všech plateb na základě výběru organizační jednotky.

Zkoušel jsem testovat také testy selhání, kdy jsem úmyslně vyplnil nevhodná vstupní data, která by se neměla v SAPu objevit. Výsledek byl opět očekávaný, tj. nezaložila se žádná z plateb, která neměla. Samozřejmě jsem zkoušel různé kombinace nestandartních vstupů a skript obstál.

Vstupní data musím zadat přesně, tak jak jsou definována v SAPu s nulovou tolerancí, což je také požadavkem na takový automat.

Flow

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Loop
2 : Initial to Transaction Window OK
3 : Transaction MV referent OK
4 : SAP MV new record referent **FAIL**

Obrázek 39 - Test selhání

Na obrázku - Test selhání, se nachází výsledek testu, ve kterém jsem nastavil cíleně chybná vstupní data pro kód podoblasti. Jak můžeme vidět, skript vždy selhal na správném místě pro všech 10 záznamů, a to při výběru kódu podoblasti z rolovací lišty. Důvodem bylo nenalezení odpovídající bitmapy k textu, což bylo očekávaným chováním v tomto testu.

Output
Result=96 [Number]
Error=Command SAP_MV_new_record_referent failed: Command SAP_MV_new_record_referent.CBR_potvrzeni_vyberu_kodu_podoblasti failed: Bitmap not found [String]

Flow

Main
CBR 3 : ClickBmpRef OK
KS 3 : KeyStroke OK
CBR new record APZ : ClickBmpRef OK
CBR Vyber kodu podoblasti : ClickBmpRef OK
TTB Kod podoblasti : TextToBmp OK
CBR potvrzeni vyberu kodu podoblasti : ClickBmpRef **FAIL**

Obrázek 40 - Test selhání detail

Na obrázku výše můžeme vidět detailnější popis dané chyby. V analýze lze takto proklikat až ke konkrétní chybě v daném makru a podle názvu si poté snadněji dohledat chybu ve skriptu.

Podobných testů jsem provedl desítky, ať už se jednalo o krokování skriptu v Editoru nebo při exekuci v Launcheru. Díky tomu jsem narazil i na některá chybějící nastavení v SAPu a na základě těchto testů jsem je doplnil.

6.3. Dokončení testu v SAPu

Pro kompletní test nových nastavení v SAPu je nutné pokračovat několika dalšími kroky, které již nebyly automatizované, ale domnívám se, že pro dokončení logiky testu je vhodné je alespoň zmínit a ukázat očekávaný výsledek.

1. Uvolnění, zpracování návrhu platby a export souboru.
2. Vygenerování testovacího bankovního výpisu.
3. Nahrání bankovního výpisu.
4. Provedení jobu automatem v SAPu.
5. Zhodnocení, zda se všechny platby úspěšně nahrály a zaúčtovaly do sytému.

08.12.2021	12:49:57	MAN.CSKE.PRE.52.20211208	Kontace	✗	-	65756	automat
08.12.2021	12:49:56	MAN.CSKE.PRE.52.20211208	Agregace	✓	-	65755	automat
08.12.2021	12:49:56	MAN.B___.20211208.__.___.__	Kontace	✗	-	65754	automat
08.12.2021	12:49:55	MAN.BEUR.20211208.VS.EUR.01	Agregace	✓	-	65753	automat
08.12.2021	12:49:55	EEC.U02.20211208	Kontace	✗	-	65752	automat
08.12.2021	12:49:54	EEC.U2320211208.1221	Kontace	✗	-	65751	automat
08.12.2021	12:49:54	EEC.U02.20211208	Agregace	✓	-	65750	automat
08.12.2021	12:49:53	EEC.U2320211208.1221	Agregace	✓	-	65749	automat
08.12.2021	12:49:50	K132_03122021.VYP	Párování	✓	-	65748	automat
08.12.2021	12:49:49	K132_03122021.VYP	Import dat	✓	-	65747	automat
08.12.2021	12:08:19	K132_08122021_52.XML	Export dat	✓	-	65746	Timotej Vrátný
08.12.2021	12:05:30	MAN.CSKE.PRE.52.20211208	Dávka ZD	✓	-	65745	Timotej Vrátný
08.12.2021	12:03:36	K068_08122021_22.PLA	Export dat	✓	-	65744	Timotej Vrátný
08.12.2021	12:02:56	EEJ.U22.20211210	Dávka ZD	✓	-	65743	Timotej Vrátný
08.12.2021	11:58:07	K132_08122021_01.PLA	Export dat	✓	-	65742	Timotej Vrátný
08.12.2021	11:57:24	MAN.BEUR.20211208.VS.EUR.01	Dávka ZD	✓	-	65741	Timotej Vrátný
08.12.2021	11:55:13	K068_08122021_02.PLA	Export dat	✓	-	65740	Timotej Vrátný
08.12.2021	11:54:13	EEC.U02.20211208	Dávka ZD	✓	-	65739	Timotej Vrátný
08.12.2021	11:52:18	K068_08122021_37.PLA	Export dat	✓	-	65738	Timotej Vrátný
08.12.2021	11:51:22	MAN.OPUC.PRT.37.20211208	Dávka ZD	✓	-	65737	Timotej Vrátný
08.12.2021	11:49:47	K068_08122021_35.PLA	Export dat	✓	-	65736	Timotej Vrátný

Obrázek 41 - SAP - náhled do zpracování souboru

Zde je náhled do vyhodnocení nahraných plateb. Pro mou část testování je podstatné, zda se provedly operace párování a agregace. Další

kroky již souvisí s účtováním. Tester po dokončení testu pouze zaznamená, že je nějaké nastavení nebo úprava v programu SAPu chybně, v případě potřeby analyzuje tuto chybu a výsledek předává dalšímu členu týmu ke zpracování.

7. Zhodnocení a doporučení

Testování dle zadání této práce bylo realizováno ve třech zásadních směrech. Zaprvé jsem testoval samotný nástroj GUI Master, resp. jeho praktickou použitelnost (vzhledem ke specifické technologii využívající bitmapy) pro prostředí SAP. Zadruhé, proběhlo testování automatizačního skriptu realizovaného dle zadání této práce. Zatřetí, za pomoci tohoto skriptu proběhlo testování programových úprav realizovaných na dotčených modulech v prostředí SAP. Tyto tři roviny se mezi sebou vzájemně prolínaly a analýza chyb nebyla vždy jednoduchá, protože jsem vždy musel nejprve zjistit, o kterou chybu z těchto tří rovin se vlastně jedná.

7.1. GUI Master

V zásadě musím hodnotit GUI Master jako povedený produkt, který se neustále vyvíjí a je schopný se přizpůsobit novým situacím a požadavkům zákazníka. Jak jsem zmiňoval v kapitole *5*, není dořešeno několik podstatných bodů, uživatelská komunita nemá společné prostředí, kde by mohla probíhat spontánní diskuse a sdílení možných řešení problémů. Dokumentace jako hlavní zdroj informací není dostatečná, a analýza provedených testů je povrchní. Postrádám možnost cloudového řešení produktu, které je na aktuálním trhu obvyklé i žádané. Tyto nedostatky u mě částečně kompenzuje přehlednost a tvorba skriptu za pomoci stromového kódu, který mi vyhovuje.

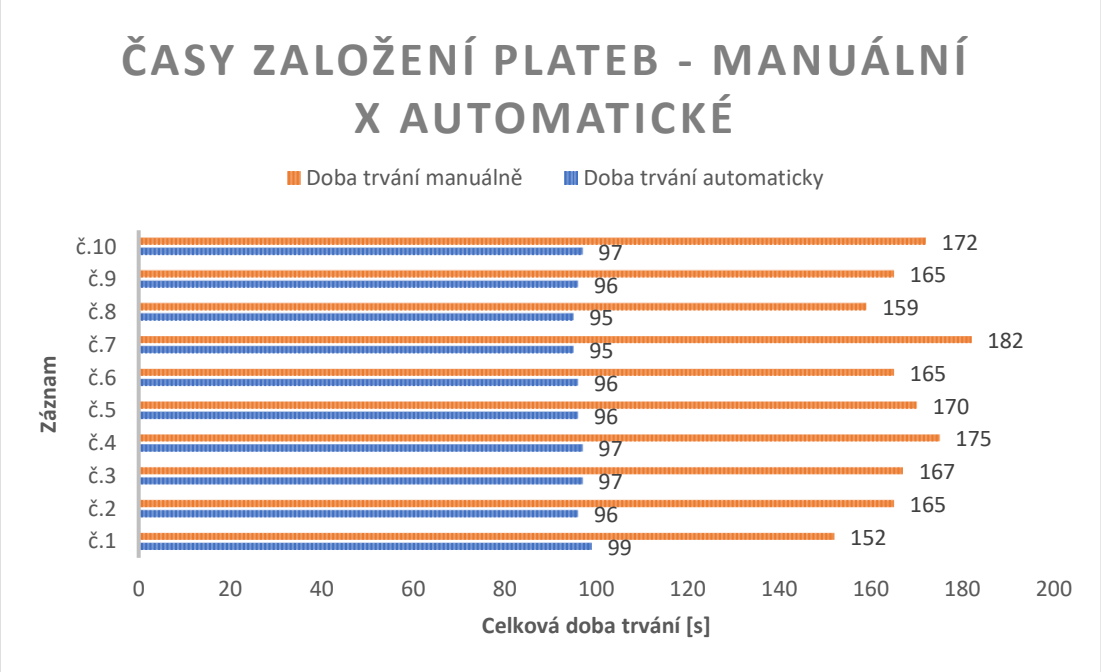
7.2. Vhodnost testování SAP

Nástroj, který má univerzální využití, a zároveň nemá větší problém s testováním uživatelského rozhraní platformy SAP, má obrovský potenciál. SAP sám o sobě má mnoho funkcí a některá řešení, nemusí být vždy optimální. Není jednoduché nalézt konkrétní řešení (automatizace) testování. Zde GUI Master obstál svými možnostmi nejen výběru přes bitmapy, ale také detekcí barev, a hlavně typů písma. Ve finále je vždy rozhodující, o jaké testování nebo automatizaci procesů se jedná. Je nutné mít možnost tento produkt zdarma vyzkoušet. Stejně jako u většiny nabízených nástrojů na trhu je nutné kontaktovat výrobce GM s informacemi, za jakým účelem chceme nástroj využívat a na jak dlouho nám jej zapůjčí.

Pro snadnější vyzkoušení bych preferoval volně dostupnou trial verzi s nějakým omezením a na určitou dobu.

7.3. Automatizace plateb manuálního vstupu

Na základě průzkumu trhu se domnívám, že pro vybranou úlohu automatizaci plateb by řada nástrojů neobstála. Nicméně, jako hlavní kritérium zde považuji časy založení jednotlivých plateb.



Obrázek 42 - Časy založení plateb - manuální x automatické

Z grafického porovnání časů manuálního a automatického založení plateb, můžeme konstatovat, že doba založení jedné platby ručně je téměř dvojnásobná než doba založení automatem. Dále lze konstatovat, že časy založení pro jednotlivé platby automatem jsou téměř totožné, průměrně 96,4 sekund. Ruční založení je limitováno uživatelem (znalost prostředí, motivace, únava, rozdílná reakční doba, přesnost atd.). Tudíž lze říct, že čím více záznamů daný člověk založí, tím déle mu založení každého dalšího záznamu bude trvat. Automat se s podobnými limity potýkat nemusí.

Tabulka 13 - Časy zapnutí a vypnutí aplikace SAP

Činnost	Manuálně [s]	Automaticky [s]
Zapnutí a přihlášení	51	19
Odhlášení a vypnutí	19	5

V tabulce - Tabulka 13 je porovnání manuálního a automatického spuštění, ukončení aplikace SAP logon a přihlášení, odhlášení k příslušnému serveru. Z tabulky vyplývá, že je automat přibližně 2,5krát rychlejší při zapnutí a přihlášení a téměř 4krát rychlejší při odhlášení a vypnutí.

Pro větší komerční využití prototypu a zároveň rozšíření nástroje by musely být splněny další předpoklady. Za zásadní považuji, aby byl Launcher zdarma, případně za symbolický poplatek v řádech jednotek Euro. Dále předpokládám možnost definice externího zdroje se vstupy v Launcheru. V našem konkrétním případě by momentálně uživatelé museli vždy zaslat vyplněný excelový soubor oprávněnému uživateli licence Editoru a ten by poslal zpět upravený projekt k exekuci v Launcheru. Takový postup by již nemusel být tak časově výhodný a jednoduchý. Zároveň by mohlo být vhodné tímto omezením rozlišit trial verzi Launcheru od placené. Při splnění těchto předpokladů by skript mohl sloužit k založení všech plateb manuálním vstupem, přičemž by si uživatelé měnili vstupní parametry pro různé oblasti. Lze říct, že by tak mohl téměř nahradit práci jednoho nebo více uživatelů, kteří tento proces obstarávají. Dále by bylo možné vyhledat obdobné procesy a také je stejným způsobem automatizovat. Po splnění předpokladů by se prototyp a nástroj mohli více rozšířit a lépe by sloužili ke svému účelu.

Požadavky na zadaný test byly splněny a konstatuji, že nástroj vyhověl a lze jej doporučit na podobné procesy v prostředí SAP. Výsledkem testu je významné zrychlení procesu testování plateb. Nástroj prokázal, že existují další možnosti, jak usnadnit procesy a nahradit manuální práci v SAPu. V případě pořízení takového nástroje se musí hledat aktivně procesy, které lze efektivně automatizovat, aby se nástroj maximálně využil. To nemusí být vždy snadné nebo na první pohled viditelné.

Vizi testovacích nástrojů vidím v cloudovém řešení, kdy se vždy na daný test nebo automatizaci procesů přihlásíme na daný čas, například jednou týdně nebo jednou měsíčně, záleželo by na četnosti testů (procesů) a tak by se testovací nástroj plně využil v definovaném čase.

Závěr

Teoretická část diplomové práce může sloužit k edukativním účelům v oblasti testování SW. V případě, kdy čtenář bude uvažovat o automatizaci testování, informace ho nasměrují správným směrem. Volba testovacího nástroje je jedním z klíčových okamžiků procesu přípravy automatizace. Základní přehled trhu demonstruje seznam testovacích nástrojů se zaměřením na systémy SAP. Průzkum trhu nadále ukazuje, že oblast RPA má v současnosti i do budoucna značný potenciál pro rozvoj. Na základě této informace lze uvažovat o vývoji podobného testovacího nástroje z hlediska komerčního a edukativního.

Praktická část posloužila zejména společnosti Atos IT Solutions and Services, s.r.o. k ukázce, zda má smysl se podobnými nástroji zabývat ať už za účelem zvýšení kvality dodávaného SW klientům, ale také jako možnost dalšího rozšíření byznysu v oblasti automatizace ERP procesů. Zde považuji za velmi přínosný vytvořený prototyp pro zakládání plateb v prostředí SAP. Demonstruje, jak by se mohla snížit časová náročnost testování plateb v oblasti JVM. Po splnění specifických podmínek lze sestavený prototyp potenciálně nabízet dalším zákazníkům. Dále ukazuje společnosti Qace System s.r.o. některé z nedostatků nástroje GUI Master, které mohou být podnětem pro další optimalizaci.

Seznam obrázků

Seznam tabulek

Seznam příloh

Tabulka 14 - Verze SAP část 1

SAP R/3 verze	Rok	SAP ECC verze	Rok	SAP S/4 Hana	Rok
SAP R/1 System RF	1972	AP ERP Central Component (ECC) 5.0	2004	SAP S/4HANA Finance 1503	2015
SAP R/2 Mainframe System	1979	SAP ERP Central Component (ECC) 6.0	2005	SAP S/4HANA 1511	2015
SAP R/3 Enterprise Edition 1.0A	1992	SAP Enhancement Package 1 for SAP ERP 6.0	2006	SAP S/4HANA Finance 1605	2016
SAP R/3 EnterpriseEdition 2.0	1993	Business All-in-One		SAP S/4HANA 1610	2016
SAP R/3 EnterpriseEdition 3.0	1995	SAP Enhancement Package 2 for SAP ERP 6.0	2007	SAP S/4HANA 1709	2017
SAP R/3 EnterpriseEdition 4.0B	1998	SAP Enhancement Package 5 for SAP ERP 6.0	2010	SAP S/4HANA 1809	2018
SAP R/3 EnterpriseEdition 4.3	1998	Business Suite 7		SAP S/4HANA 1909	2019
SAP R/3 Enterprise Edition 4.5B	1999	SAP Enhancement Package 6 for SAP ERP 6.0	2012	SAP S/4HANA 2002	2020
SAP R/3 Enterprise Edition 4.6C	2001	SAP Enhancement Package 7 for SAP ERP 6.0	2013	SAP S/4HANA 2005	2020
SAP R/3 Enterprise Edition 4.6F	2001	SAP S/4 Simple Suite for HANA	2015		
SAP R/3 Enterprise Edition 4.7	2003				

Tabulka 15 - Verze SAP část 2

SAP S/4 Hana Cloud	Rok	SAP Netweaver	Rok	SAP Fiori	Rok
AP S/4HANA Cloud 1603	2016	**AP NetWeaver 7.0**	2005	**SAP Fiori 1.0**	2013
SAP S/4HANA Cloud 1605	2016	**SAP NetWeaver 7.0 EHP 1**	2008	**SAP Fiori 2.0**	2016
SAP S/4HANA Cloud 1608	2016	**SAP NetWeaver 7.0 EHP 2**	2010	**SAP Fiori 3.0**	2019
SAP S/4HANA Cloud 1611	2016	**SAP NetWeaver 7.0 EHP 3**	2011		
SAP S/4HANA Cloud 1702	2017	**SAP NetWeaver 7.1**	2007		
SAP S/4HANA Cloud 1705	2017	**SAP NetWeaver 7.1 EHP 1**	2009		
SAP S/4HANA Cloud 1708	2017	**SAP NetWeaver 7.2**	2009		
SAP S/4HANA Cloud 1711	2017	**SAP NetWeaver 7.3**	2010		
SAP S/4HANA Cloud 1802	2018	**SAP NetWeaver 7.3 EHP 1**	2011		
SAP S/4HANA Cloud 1805	2018	**SAP NetWeaver 7.4**	2013		
SAP S/4HANA Cloud 1808	2018	**SAP NetWeaver 7.5**	2015		
SAP S/4HANA Cloud 1811	2018	**AS ABAP 7.51**	2016		
SAP S/4HANA Cloud 1902	2019	**AS ABAP 7.52**	2017		
SAP S/4HANA Cloud 1908	2019	**ABAP Platform 1809**	2018		
		ABAP Platform 1909	2019		

- Automatizační skripty, projekt (na CD)
- Využité dokumentace (na CD)

Reference:

[1] *Industry 5.0: How the mass customization era can be the solution for overproduction* [online]. [vid. 2021-11-23]. Dostupné z: https://www.platforme.com//blog/industry-5-0-how-the-mass-customization-era-can-be-the-solution-for-overproduction

[2] Průmysl 5.0 – když lidé pracují se stroji | Rockwell Automation Česká republika. *Rockwell Automation* [online]. [vid. 2021-11-23]. Dostupné z: https://www.rockwellautomation.com/cs-cz/company/news/blogs/prumysl-5-0-kdyz-lide-pracuji-se-stroji.html

[3] What is SAP? | Definition and Meaning. *SAP* [online]. [vid. 2021-12-03]. Dostupné z: https://www.sap.com/about/company/what-is-sap.html

[4] Top 7 Best ERP Software on the market: Updated 2022. *Magenest - Run your eCommerce store with ease!* [online]. 19. listopad 2021 [vid. 2021-11-23]. Dostupné z: https://magenest.com/en/erp-software/

[5] KALINA, Miroslav. Informační systém SAP. 2019.

[6] *Foundation Level Syllabus - ISTQB® International Software Testing Qualifications Board* [online]. [vid. 2021-11-24]. Dostupné z: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html

[7] PIETRIK, Michal. *Automatizované testování webovỳch aplikací*. B.m., 2012. PhD Thesis. Masarykova univerzita, Fakulta informatiky.

[8] *Professor Ron Patton* [online]. [vid. 2021-12-29]. Dostupné z: https://www.hull.ac.uk/staff-directory/ron-patton.aspx

[9] PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 978-80-7226-636-4.

[10] Difference between defect, error, bug, failure and fault. *tfortesting* [online]. 3. září 2012 [vid. 2021-11-26]. Dostupné z: https://tfortesting.wordpress.com/2012/09/03/difference-between-defect-error-bug-failure-and-fault/

[11] LAYCOCK, Gilbert Thomas. *The theory and practice of specification based software testing* [online]. B.m., 1993. PhD Thesis. Citeseer. Dostupné z: https://www.researchgate.net/publication/236031163_The_Theory_of_Software_Testing

[12] HOWDEN, William E. *Functional program testing and analysis*. B.m.: McGraw-Hill, Inc., 1986.

[13] NAIK, Kshirasagar a Priyadarshi TRIPATHY. *Software Testing and Quality Assurance: Theory and Practice*. B.m.: John Wiley & Sons, 2011. ISBN 978-1-118-21163-2.

[14] LAWANNA, Adtha. The Theory of Software Testing [online]. 2012, 6. Dostupné z: https://www.researchgate.net/publication/236031163_The_Theory_of_Software_Testing

[15] *Demingův cyklus PDCA* [online]. [vid. 2021-12-30]. Dostupné z: https://www.systemonline.cz/sprava-it/deminguv-cyklus-pdca.htm

[16] *Software Testing - Levels* [online]. [vid. 2021-11-26]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm

[17] *unit-testing Tutorial => The general rules for unit testing for all...* [online]. [vid. 2021-12-30]. Dostupné z: https://riptutorial.com/unit-testing/topic/9947/the-general-rules-for-unit-testing-for-all-languages

[18] *Úrovně provádění testů | Testování softwaru* [online]. [vid. 2021-11-26]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/

[19] *Druhy testování v procesu vývoje SW* [online]. [vid. 2021-12-30]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11

[20] HAMILTON, Thomas. *What is System Testing? Types & Definition with Example* [online]. 12. leden 2020 [vid. 2021-12-30]. Dostupné z: https://www.guru99.com/system-testing.html

[21] HAMILTON, Thomas. *Sanity Testing Vs Smoke Testing: Introduction and Differences* [online]. 13. leden 2020 [vid. 2021-12-30]. Dostupné z: https://www.guru99.com/smoke-sanity-testing.html

[22] MCWHERTER, Jeff a Ben HALL. *Testing ASP.NET web applications*. Indianapolis, Ind: Wiley, 2010. ISBN 978-0-470-49664-0.

[23] KHAN, Mohd Ehmer a Farmeena KHAN. A comparative study of white box, black box and grey box testing techniques. *Int. J. Adv. Comput. Sci. Appl*. 2012, **3**(6).

[24] KAEMARUNGSI, Kamol a Prashant KRISHNAMURTHY. On the use of adaptive OFDM to preserve energy in ad hoc wireless networks. In: *Proceedings of 13th MPRG/Virginia Tech Symposium on Wireless Personal Communications*. B.m.: Citeseer, 2003.

[25] *Guideline: Průzkumné testování* [online]. [vid. 2021-12-30]. Dostupné z: http://metodikamezitest.asp2.cz/Metodika_testovani/guidances/guidelines/pruzkumne_testovani_AD696237.html

[26] LINK, Johannes a Peter FRÖHLICH. *Unit testing in Java: how tests drive the code*. San Francisco, Calif: Morgan Kaufmann, 2003. ISBN 978-1-55860-868-9.

[27] EGILMEZ, Ismail. *4 Software Testing Roles* [online]. [vid. 2021-11-28]. Dostupné z: https://blog.thundra.io/4-software-testing-roles

[28] BOROVCOVÁ, Anna. Testování webovỳch aplikací. 2008.

[29] *Software Testing - Quick Guide* [online]. [vid. 2021-12-09]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_quick_guide.htm

[30] MIKEJO5000. *Coded UI tests - Visual Studio (Windows)* [online]. [vid. 2021-12-09]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/test/use-ui-automation-to-test-your-code

[31] INC, UiPath. *RPA Analytics Tool - Measure RPA Operations | UiPath* [online]. [vid. 2021-12-10]. Dostupné z: https://www.uipath.com/product/rpa-insights

[32] PÁNEK, Dominik. Robotická automatizace procesů. 2020.

[33] Robotic Process Automation: Is it the Same As Test Automation? *SmartBear.com* [online]. [vid. 2021-12-09]. Dostupné z: https://smartbear.com/blog/robotic-process-automation-is-it-same-as-test-auto/

[34] Robotic Process Automation vs Traditional Test Automation. *qentelli* [online]. [vid. 2021-12-10]. Dostupné z: https://www.qentelli.com/thought-leadership/insights/robotic-process-automation-vs-traditional-test-automation

[35] *Robotic Process Automation Market Size Report, 2021-2028* [online]. [vid. 2021-12-10]. Dostupné z: https://www.grandviewresearch.com/industry-analysis/robotic-process-automation-rpa-market

[36] Difference between RPA and Selenium. *GeeksforGeeks* [online]. 7. červenec 2020 [vid. 2021-12-10]. Dostupné z: https:/www.geeksforgeeks.org/difference-between-rpa-and-selenium/

[37] *Top 10+ Best SAP Testing Tools (SAP Automation Tools)* [online]. [vid. 2021-12-15]. Dostupné z: https://www.softwaretestinghelp.com/best-sap-testing-tools/

[38] *Automatizované testování « GLOBTECH spol. s r.o.* [online]. [vid. 2021-12-16]. Dostupné z: http://www.globtech.cz/automatizovane-testovani/

[39] *GuiMaster - Wiki* [online]. [vid. 2021-12-21]. Dostupné z: https://wiki.getguimaster.com/index.php?title=GuiMaster