Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Rozbor a ukázka automatického testování

Jakub Chládek

© 2022 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Chládek

Systémové inženýrství a informatika
Informatika

Název práce

Rozbor a ukázka automatického testování

Název anglicky

Analysis and demonstration of automatic testing

Cíle práce

Hlavním cílem diplomové práce je zpracování automatizovaných testů pro desktopovou aplikaci vytvářenou zvolenou firmou. Dílčími cíli je provést porovnání manuálního testování a testování automatizovaného a provedeno vyhodnocení.

Metodika

Na základě studia odborných a vědeckých literárních zdrojů, prakticky získaných informací a znalostí budou navrženy metody testování, vytvořeny vlastní automatizované testy. K testování budou použity zvolené technologie. Na základě provedeného manuálního a automatizovaného testování budou porovnány zvolené varianty a zpracováno jejich vyhodnocení.

Doporučený rozsah práce

40 stran

Klíčová slova

Automatické testování, C#, rozdíly testování

Doporučené zdroje informací

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6

Dustin, E., Garrett, T. and Gauf, B., 2009. Implementing Automated Software Testing. Addison-Wesley. ISBN-13: 978-0321580511

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5

STEPHENS, Matt a Doug ROSENBERG. Testování softwaru řízené návrhem. Brno: Computer Press, 2011. ISBN 978-80-251-3607-2

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

doc. Ing. Edita Šilerová, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 10. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2022

Čestné prohlášení

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

V Praze dne 14.03.2022 _____

Poděkování

Rád bych touto cestou poděkoval vedoucí práce doc. Ing. Editě Šilerové, Ph.D. za vedení a inspiraci při tvorbě bakalářské práce. Největší poděkování ale patří mé rodině a všem, kdo mě podporovali v průběhu celého studia zejména pak mé mámě magistře Monice Mlezivové za cenné rady a trpělivost.

Rozbor a ukázka automatického testování

Abstrakt

Bakalářská práce se zabývá v teoretické části problematikou testování softwaru. Je zde popsáno, proč by se měl testovat software, dále pak jsou zde také popsány typy testů, kategorie a metody testování. Praktická část je zaměřena na vytvoření vlastních systémových testů ve vývojovém programu TestComplete, které jsou použity na desktopovou účetní aplikaci POHODA a průběh testování. Dále jsou zde zhodnoceny klady a zápory manuálního a automatického testování softwaru.

Klíčová slova: testování, TestComplete, POHODA, manuální testování, automatické testy, metody testování

Analysis and demonstration of automatic testing

Abstract

The theoretical part of the bachelor thesis deals with the issue of software testing. It describes why software should be tested, and then it also describes types of tests, categories and methods of testing. The practical part focuses on the creation of custom system tests in the development program TestComplete, that are applied for the desktop accounting application POHODA and the testing progress. It also evaluates the pros and cons of manual and automated software testing.

Keywords: testing, TestComplete, POHODA, manual testing, automatic testing, testing methods

Obsah

1 Úvod

„Testování programu je velmi efektivní způsob, jak ukázat na přítomnost chyb, ale je bohužel nedostatečné k prokázání jejich nepřítomnosti.“ (E.W.Dijkstry)

Téma, kterému jsem se rozhodl věnovat ve své bakalářské práci, se týká automatického testování. Toto téma je mi blízké, protože pracuji pro společnost Certicon jako vývojář automatizovaných testů a rád bych se tak mimo jiné podělil o své zkušenosti. Cílem mé bakalářské práce s názvem „Rozbor a ukázka automatického testování“ je pokusit se nastínit všechny oblasti daného tématu.

V dnešní době je vývoj softwaru čím dál víc komplexní, a aby byl výsledný produkt dostačující kvality, tak musí být i řádně otestovaný. Testování proto začíná být důležitou součástí vývoje, ale častokrát mu bývá věnována menší pozornost, než jakou by si zasloužilo. To pak může mít neblahý vliv na výsledné kvalitě produktu a následná oprava či zlepšení se může značně prodražit.

Ve své bakalářské práci se především zabývám problematikou testování softwaru, zejména se pak zaměřuji na automatizované testování.

Teoretickou část bakalářské práce jsem rozdělil do šesti základních celků, které se týkají základních znalostí o testování. Popisuji důvody, proč je dobré testovat svůj produkt, a kdy je vhodné jeho testování ukončit. Dále představuji problematiku vyhodnocování testů a kdo by jej měl provádět. Nedílnou součástí testování je chyba, kterou se zabývám ve třetím celku. Další část je pak věnována popisu typu testů a jak jsou děleny podle toho s jakou částí systému pracují (jestli testují jen metodu, výkon nebo fungování systému jako celku). Pátá část je zaměřena na kategorie testů, které se dělí podle znalosti systému z pohledu testera nebo podle toho, jestli je test prováděný člověkem (manuálně) nebo počítačem (automaticky). Poslední část pak pojednává o dělení testů podle metody testování.

Stěžejní část mé bakalářské práce tvoří praktická část, ve které se zaměřím na tvorbu automatizovaných systémových testů a jejich spuštění v desktopové účetní aplikaci POHODA. Následně budou provedeny stejné manuální testy pro výše uvedenou účetní aplikaci. Oba typy testů (tedy automatizovaný a manuální) pak budou vzájemně porovnány a budou vyhodnoceny jejich výhody a nevýhody užití v testování

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je zpracování automatických testů pro desktopovou aplikaci vytvářenou zvolenou firmou. Dílčími cíli je provést porovnání manuálního testování a testování automatického a zhodnocení zjištěných kvalit případně i negativních faktorů u obou druhů testování.

2.2 Metodika

Na základě studia odborných a vědeckých literárních zdrojů, prakticky získaných informací a znalostí budou navrženy metody testování a vytvořeny vlastní automatizované testy. K testování budou použity zvolené technologie. Na základě provedeného manuálního a automatického testování budou porovnány kvality obou variant testování a zpracováno jejich vyhodnocení.

3 Testování softwaru

V dnešním technologické světě se každý den setkáváme s různými druhy technologií jako jsou mobilní aplikace, automatické kávovary apod., pro, než je software nezbytnou součástí. Pro tvorbu softwaru je jedním ze základních pilířů jeho testování. Mohli bychom říct, že testování je první feedback po úpravách nebo vytvoření nového softwaru.

Množství dostupných a používaných technologií se neustále zvětšuje, čímž se vývoj softwaru stává velmi komplexní záležitostí. Neustálý rozvoj technologií, a především požadavků kladených na kvalitu a stabilitu vývojového softwaru znamená, že je třeba stále více soustředit se na testování.

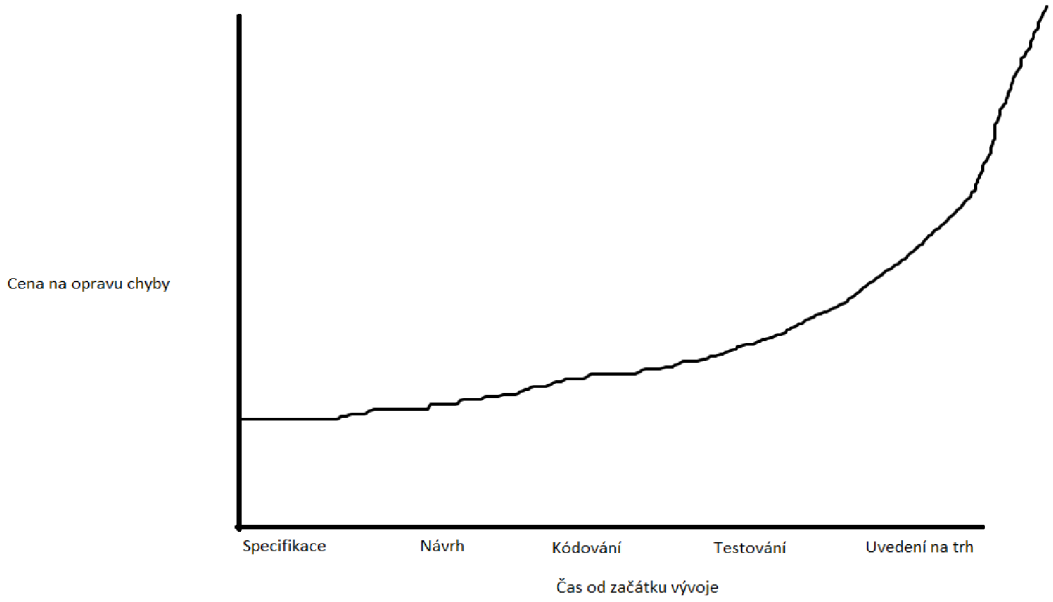
Základní cíl testování samozřejmě stále představuje minimalizace chyb, obecně však od testování požadujeme zvýšení kvality dodávaného softwaru. Důkazem tohoto trendu je fakt, že tvorba jednotkových testů se dnes považuje za běžnou součást vývoje softwaru a představuje de facto standard[1 s. 32]

Testování má mnoho různých způsobů od jednoduchých manuálních testů po velmi složité automatické testy. Software tedy můžeme otestovat manuálně, kdy člověk prochází určité fungování systému a zjišťuje, jestli se zde nenachází chyby, nebo se může zainvestovat do vývoje automatizovaných testů, tento typ testů je prováděn počítačem.

3.1 Proč testovat

Při vývoji softwaru jsou běžnou součástí chyby. Testování je proces pro odhalení těchto chyb před jejich dopadem na software, ještě před tím než můžou způsobit potíže.

Nevýhodou testovaní může být jeho finanční náročnost, může pohltit až polovinu rozpočtu na projekt, na druhou stranu neodhalená chyba postupem času vývoje může vyjít daleko dráž. Toto tvrzení se ukazuje na následujícím grafu č.1., který je inspirován Boehmovým zákonem.



Graf 1 Graf inspirován Boehmove zákon (čím dřív se na chybu přijde, tím je její oprava levnější)

Následující tabulka č.1 ukazuje poměr nákladů o něco přesněji.

Cena za opravu chyby (čísla znamenají násobky)		Fáze, kdy byla chyba detekována				
		Specifikace	Návrh	Kódování	Systémové testy	Používání
Fáze kdy chyba vznikla	Specifikace	1	3	5 - 10	10	10 - 100
	Návrh	-	1	10	15	25 - 100
	Kódování	-	-	1	10	10 - 25

Tabulka 1 Tabulka zobrazující násobek o kolik se chyba prodraží při pozdějším nalezení

Například, jestliže chyba vznikla ve fázi návrhu a přišlo se na ni až ve fázi systémových testů, pak je její oprava 15x dražší, než kdyby se na ni přišlo již ve fázi návrhu.[2 s. 18]

Avšak nalezení chyby není jediným důvodem, proč bychom měli testovat. Dalším důvodem je to, že software nebývá většinou úplně dokončen. Stále jsou v něm potřeba dělat aktualizace na zlepšení, opravy, nebo jen reagovat na aktuální trend. Díky tomu se kód (software který firma vytváří) neustále mění a lidé dost často zapomínají, co k čemu tak slouží. Nikdo pak nechce dělat úpravu, jelikož by to mohlo zanášet chyby. Je nemožné mít nějaký rozsáhlý projekt a po každé změně ho celý projít bez automatických testů. Proto velké projekty bez automatizovaných testů se v určitém bodě začnou zpomalovat, a nakonec se zastaví, protože každá další změna je riskantnější a tím pádem i dražší.

Pokud máme k dispozici vhodné automatizované testy s dobrým pokrytím, můžeme provádět změny, aniž bychom byli v ustavičném napětí, co všechno tyto změny rozbily a my to nevíme.[2 s. 20]

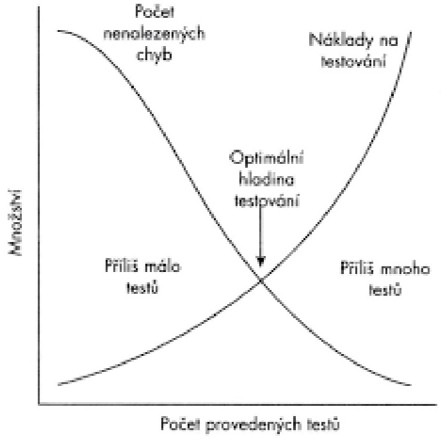
3.1.1 **Zajištění kvality softwaru**

Zajištění kvality softwaru je proces známý pod zkratkou QA (Quality Assurance) a jeho hlavní ideou je, že kvalitní proces vývoje zajišťuje kvalitní výsledky. Jedná se o komplexní aktivitu, která zajišťuje, že standardy a procedury použité během vývoje a údržby softwaru jsou vhodné a správně dodržované. Na rozdíl od testování, které je jeho podmnožinou zajištění kvality, se QA zabývá prevencí chyb, implementací a vylepšováním firemních nařízení a jako takové je to záležitost všech zainteresovaných.[2 s. 24]

3.1.2 Kdy přestat testovat

Vývoj softwaru je ekonomická záležitost / činnost. S hledáním další chyb by se mělo přestat v okamžiku, kdy náklady na nalezení a opravení chyby jsou v průměru stejné nebo větší než náklady na ponechání chyby.[2 s. 25]

Používá se pojem KKTR (Kvantita, Kvalita, Termín, Rozpočet), kdy tyto parametry musí být ve vzájemném souladu a zároveň odpovídat celkovému business plánu projektu.[2 s. 25] Tuto problematiku znázorňuje následující graf.



Graf 2 Optimální hladina testování

3.2 Vyhodnocení testů

Pokud provádíme test máme nějaký výsledek, který očekáváme. Jaký má být výsledek se většinou rozhoduje před testem. Po proběhnutí testu pak analyzujeme, zda je jeho výsledek shodný s tím, co očekáváme. Pokud test odpovídá tomu, co očekáváme, pak se označuje jako prošel (Passed), pokud neodpovídá požadovanému výsledku označujeme takový test jako selhal (Failed).[2 s. 24]

Vyhodnocení může být provedeno člověkem nebo počítačem. Člověk bude například vyhodnocovat, jestli je text dostatečně čitelný, nebo jestli jsou chybové hlášky dostatečně srozumitelné atd. Takovéto věci počítač prostě nevyhodnotí, ale nechat vyhodnocování na počítači je třeba výhodné pro matematické příklady. Člověk by takto náročný úkol nezvládl nebo s velkými obtížemi, s možnou chybovostí a výraznou časovou dotací, kdyby měl projít velký počet testů a propočítat u každého výsledek, ale stroji to nebude dělat problém.

3.3 Chyba

Můžeme si definovat několik různých chyb, které mají rozdílné názvy, avšak význam bývá stejný.

Terminologie se často různí a míchají. Běžně se pro chybu v softwaru používají tyto termíny:

- Chyba (error, bug)
- Vada (flaw)
- Omyl (mistake)
- Selhání (failure)
- Porucha (fault)
- Defekt (defect)
- Závada (glitch)
- Odchylka (variace)
- Problém (problem)
- Nepříjemná událost (accident)
- Anomálie (anomaly)
- Nekonzistence (inconsistency)
- Incident (incident)

Všechny tyto termíny popisují chybu, ale často popisují trošku odlišné chyby. Pokud chceme chybu popsat nějak obecně, spadá do jedné z těchto vět.

- SW nedělá, co má dělat dle specifikace
- SW dělá, co nemá podle specifikace
- SW dělá něco, co není specifikováno
- SW nedělá něco, co není specifikováno, ale asi by to dělat měl
- SW je nesrozumitelný, špatně se s ním pracuje apod.

Stručněji pak můžeme chybu definovat takto: „Chyba je cokoliv týkající se programu, co podle některého ze stakeholderů (člověk, který má zájem na dané věci) zbytečně snižuje hodnotu programu.“[2 s. 22]

3.4 Typy testování

3.4.1 Funkční testy

Jsou to testy, které mají za účel ověřit nějakou část funkčnosti softwaru. Můžou testovat od jednotlivých metod v kódu až po formuláře na webovkách (front-end). Tento typ testů můžeme následně dělit na jednotlivé skupiny podle toho, na jaké úrovni systém testují.

Jednotkové testy (Unit tests)

Na této úrovní testů se testují jednotlivé části programu, a to bez vztahů a závislostí (resp. s minimálním vztahem) na okolním fungování systému.

Typickým příkladem jednotkového testu je test konkrétní metody, případně celé třídy. Jednotkové testy se vytváří již během vývoje softwaru. Jsou důležité pro ověření, jestli daná část softwaru (metoda) je napsaná správně bez ohledu na fungování okolí. Slouží jako dobré ověření po zásahu do dané části softwaru, jestli změny nerozbily fungování jednotlivých metod.

Integrační testy

Tyto testy jsou dosti podobné jako testy jednotkové. Mohli bychom říct, že jsou jejich nadstavbou. Jako Ilustraci integračního testu lze uvést testování DAO komponenty přímo proti databázi. V tom se liší od jednotkových testů. Ty si svá data pro testy simulují na rozdíl od integračních testů, které testují na reálné databázi. Technicky může jít o stejnou nebo velmi podobnou technologii jako v případě jednotkových testů. Zaměření a cíl těchto testů je však zcela odlišný.[1 s. 34]

Systémové testy

Představují nejvyšší formu funkčního testování, neboť smyslem těchto testů je ověřování funkčnosti již hotových aplikací v rámci celého systému. Jako příklad si můžeme uvést test na webovou aplikaci. Tento test pak prochází skrze celý systém a zaměřují se na aplikace a jejich interakce mezi nimi.[1 s. 34]

3.4.2 Nefunkční testy

Jsou testy ověřující jiné než funkční požadavky (tzn. požadavky netýkající se samotné funkčnosti aplikace). Nefunkční testy nejsou vztaženy ke konkrétní funkčnosti, ale k celému systému či softwaru. Tyto testy můžeme rozdělit na několik skupin, které si následně popíšeme.[1 s. 35]

Bezpečnostní testy

Tento druh testů bývá také označován jako penetrační testy. Jsou to testy, které se zaměřují na bezpečnost aplikace nebo celého softwaru. Cílem bezpečnostního testu je nalézt a identifikovat všechna zranitelná místa v systému či softwaru.[1 s. 35]

Výkonnostní testy

Ověřují výkonnost softwaru podle předem dohodnutých požadavků zadaných v SLA (SLA je zkratka z anglického Service Level Agreement a označuje smlouvu mezi poskytovatelem služby a uživatelem, v níž se definuje fungování dané služby (dostupnost, výkon, počet uživatelů atd.)). Výkonnostní testy můžou probíhat i mezi jednotlivými verzemi softwaru.[1 s. 35]

3.4.3 Další členění testů

Testy můžeme dělit i podle jejich užití v systému.

Průzkumné testy

Je to určitý druh testování, při kterém tester prochází aplikací, mohou být ale i zaměřeny na nějakou určitou část a dle svých zkušeností zkouší fungování systému. Tester provádějící průzkumové testy by měl mít know how o fungování systému, který testuje.

Testování není systematické, avšak pomáhá nalézt nejzávažnější (což ovšem závisí na zkušenosti testera) chyby. Pro efektivitu této metody je velmi důležité mapovat a zaznamenávat prošlou cestu v aplikaci[2 s. 41]

Podle mých zkušeností je nejlepší užití průzkumových testů na začátku testování (může být i přidání nové feature), kdy ještě nemáme sepsané test case (testovací scénáře) pro danou problematiku. Díky tomu můžeme najít spoustu chyb a nemusíme čekat s opravou na dopsání všech test case, které by na ně pak narazili.

Regresní testy

Termín regrese znamená zanášení dalších chyb do systému vlivem oprav nebo změn tohoto systému. Účelem regresních testů se ujistit se, že úprava nezhoršila původní funkcionalitu. Takto se nachází defekty, které byly zaneseny jako následek změn.[2 s. 41]

Regresní testy jsou prováděny po opravách chyb po funkčním vylepšení, kdy změněná funkčnost aplikace se otestuje dle scénářů a/nebo průzkumnými testy a nezměněný zbytek aplikace se otestuje regresními testy.[2 s. 42]

Regresní testy jsou lepší užívat jako automatizované, jelikož testovat systém manuálně po změnách by se mohlo značně prodražit a hlavně zabrat čas testerů.

Smoke testy

Jsou to prvotní testy, které by měly pokrývat již fungující části systému. Ve velkých aplikací se užívá vždy pár vybraných testů z jednotlivých funkcionalit, aby ukázaly, v jakých částích systému by mohl být problém a kde soustředit pozornost testerů.

Akceptační testování

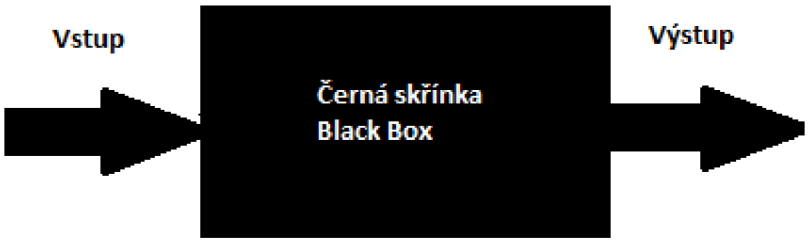
Při přebírání produktu zákazníkem probíhá akceptační testování (acceptance testing) též UAT (uživatelský akceptační test). Je to testování podle scénářů.[2 s. 43]

3.5 Kategorie testů

V testování se setkáváme s velkým množstvím druhů testů. Proto si musíme rozmyslet, kde jaký test použijeme. K dobrému otestování dojde jen tehdy, když k tomu zvolíme správný druh testu či testů. Ukážeme si tedy jaké typy testů můžeme použít a k čemu slouží.

3.5.1 Černá skříňka

Černá skřínka (Black box) je způsob testování, při kterém tester nemá znalosti o fungování systému uvnitř, a tak je test zaměřen na vstupy a výstupy programu neboli na vnější podněty od testera a následné chování systému na tyto podněty (viz Obrázek č. 1). Již podle názvu si můžeme představit černou skřínku, do které nevidíme a nevíme, co se uvnitř odehrává. Jako typický příklad lze uvést akceptační testy.



Obrázek 1 Černá skřínka

Černá skřínka testuje pouze, zda testovaný objekt poskytne po zadaném vstupu zadaný výstup.[1 s. 33]

3.5.2 **Bílá skřínka**

Bílé skřínce (white box) se v angličtině někdy také říká „glass box“, v překladu skleněná, protože oproti černé skřínce je znalost testera rozšířena o fungování systému mezi vstupem a výstupem. V tomto postupu má tester výhodu, že testovaná data může sledovat skrze systém, a tak rychle přijde na to, kde by mohl najít případné chyby co se vyskytnou. Typický výskyt užití bílé skřínky jsou jednotkové testy (Unit tests). Chceme-li totiž dobře otestovat např. nějakou metodu, měli bychom prověřit všechny cesty, kudy může program při provádění dané metody běžet a k tomu je potřeba znát definici dané metody.[1 s. 34]

3.5.3 **Šedá skřínka**

Jde o přechod mezi testováním typu černá skřínka (black-box) a bílá skřínka (white-box). Lze si to v podstatě představit jako kombinací obou dříve uvedených přístupů. Testování podle této strategie lze popsat jako testování černé skřínky (testy z pohledu uživatele), avšak s jistou znalostí vnitřních struktur a fungování softwaru.[1 s. 34]

3.5.4 **Manuální testy**

Manuální testy provádí tester, který je testuje podle daného scénáře (testcase). V dnešní době nejsou příliš preferované, jelikož jsou časově náročné i finančně, pokud by se měly opakovat. Kde je to možné, tak je již nahrazují automatizované testy.

Zvláštním druhem manuálních testů jsou testy jménem Exploratory testy. Tyto testy se liší tím, že tester, který test provádí, nejde podle scénáře, aby došel k určitému výsledku, ale cílem je prozkoumat, jak se testovaný software chová. Využívá se hlavně, když do systému přidáme nějakou novou část nebo funkci, a tak využijeme Exploratory testy, abychom odchytili zásadní chyby v chování systému.

3.5.5 Automatické testy

Automatizované testování softwaru bývá důležitou částí při vývoji softwaru, a přitom ho dost firem podceňuje. Využití automatizace znamená určitou vyšší finanční investici, ale na druhou stranu jeho užívání přináší úsporu prostředků a také zvýší kvalitu výsledného produktu.

Automat na rozdíl od člověka vykonává test „zdarma“ a relativně rychle. Automatizovaný test je tím pádem možné opakovat mnohem častěji.

Akce provedené automatizovaným testem jsou vždy stejné. Automat v rámci toho, jak je naprogramován, zopakuje dané kroky vždy stejně a neudělá chybu. Prokazatelnost výsledků testů může být v takovém případě vyšší – ale to platí jen v rámci přesně definovaného scénáře, který automatem testujeme. Automat sám od sebe jinou kombinaci dat nebo variantu procesu i mimo tento scénář již nezkusí. Manuální tester to udělat může, pokud mu intuice napovídá, že by v ní mohly být nějaké defekty. Potřebnou kreativitu testera můžeme obecně automatem těžko nahradit. Některé metody manuálního testování, závisející právě na invenci testera, například průzkumné testování (exploratory testing) nebo technika odhadu možného výskytu defektů (error guessing) v principu ani rozumně automatizovat nejdou.[3 s. 178]

3.6 Metody testování

3.6.1 Statické testování

Ke statickému testování dochází, když není zapotřebí běh daného softwaru. Při těchto testech se prochází a vyhodnocuje zdrojový kód. Při tomto způsobu testování se využívá řada metod počínaje kontrolou kódu pověřenou osobou (code review nebo peer-review) až po využívání různých nástrojů pro statickou kontrolu kódu (Checkstyle, FindBugs, …) [1 s. 33]

3.6.2 Dynamické testování

Dynamické testování může probíhat v případě, že již máme spustitelnou verzi programu. Principem je zadávání určitých vstupů a následné posuzování výstupů. Představuje testování běžícího programu nebo jeho části, při kterém se ověřuje funkčnost testovaného softwaru podle předem připravených testovacích scénářů.[1 s. 33]

Mezi dynamické testy bychom mohli zařadit jednotkové testy, integrační testy, akceptační testy atd.

3.6.3 Pozitivní vs. negativní typy testů

Tyto dva přístupy se liší zejména v tom, jestli úspěch testu je když nám test vyhodí chybovou hlášku nebo ne. Pozitivní typy testů je známější a jsou častěji využívané, negativní typy jsou méně známé a bývají i opomínané. Pozitivní testy se tedy využívají hlavně na začátku vývoje SW. Na rozdíl od negativních testů, které jsou nasazovány do procesu až potom, co se aplikace (vyvíjený software) ustálí a vykazuje stabilní fungování.

Pozitivní testy

Cílem je ověřit normální fungování systému. Pozitivní testy provádějí verifikaci konkrétního (požadovaného)chování za očekávaných podmínek fungování testovaného programu.[2 s. 27]

Negativní testy

Mohou být také nazvané jako testy vynucených chyb. Tento druh testů se zaměřuje na ověřování chování systému za neakceptovatelných, abnormálních nebo neočekávaných podmínek.[2 s. 27] Bývají často vynechávány ať už z lenosti, kvůli nákladům, časovému presu nebo z přesvědčení, že se prostě tohle nemůže stát. Negativní testy ověřují, zda program na takové situace vhodně reaguje. Vhodnou reakcí je např. přehledné chybové hlášení nebo jiná reakce (např. automatická oprava hodnot, vyhození výjimky), která je popsaná ve specifikaci a nevede k pádu programu. Naopak nevhodnou reakcí je pád programu, ztráta dat atp. [2 s. 27]

4 Praktická část

Praktická část je zaměřena na zpracování automatizovaných testů pro zvolenou desktopovou aplikaci, kterou je účetní aplikace POHODA. Na základě odborných literárních zdrojů jsou zde navrženy a aplikovány různé metody testování a podle nich vytvořeny odpovídající automatizované testy. K tvorbě testů a testování byl zvolen program TestComplete. Testování dané aplikace je provedeno automatizovaným i manuálním druhem testu a následně je provedeno porovnání výsledků z obou typů testování.

4.1 Cíle praktické části

1. Vytvoření automatických testů
2. Provedení manuálního testování a spuštění automatizovaných testů
3. Porovnání manuálních a automatických testů a určení jejich výhod a nevýhod.

4.2 Použité programy

4.2.1 TestComplete

Pro vytvoření automatizovaných testů byl vybrán komerční program TestComplete vyvíjený firmou SmartBear Software. Testy se zde dají vytvářet více způsoby, například je zde integrované nahrávání záznamu uživatelských akcí, lze užít i klíčových operací, ale testy je možné psát i manuálně pomocí podporovaných skriptovacích jazyků. Pro účely vytvoření automatizovaného testu užitého v bakalářské práci bylo použito integrované nahrávání záznamu s následnou úpravou ve skriptovacím jazyce.

Podporované skriptovací jazyky:

- VBScript
- JScript
- DelphiScript
- C++ Script
- C#Script
- JavaScript

TestComplete umožnuje testovat desktopové aplikace, webové aplikace, ale i mobilní zařízení.

Tvorba testů je v tomto vývojovém programu poměrně jednoduchá a intuitivní. Díky integrovanému rekordéru umožňuje tento program tvorbu záznamu testů téměř komukoliv. Ověření jednotlivých kroků jde udělat několika následujícími způsoby:

- Textem nebo vlastností (Text or property)
- Obrázkem / Snímkem Obrazovky (Image)
- Souborem (File)
- Tabulka (Table or Grid)
- Database
- Webové metriky (Web metrics)

4.2.2 **POHODA**

POHODA je komplexní ekonomický systém pro vedení účetních, majetkových, skladových a mzdových agend, řízení firmy a mnoha dalšího. Má plno funkcí ve třech základních řadách – POHODA, POHODA SQL a POHODA E1 a spoustu dalších doplňků v bonusových balíčcích, které si můžete dokupovat podle svých potřeb a zájmů. POHODA je program vhodný pro živnostníky, malé firmy i střední a větší firmy s více zaměstnanci a požadavky. Vyzkoušet si jej můžete zdarma díky demoverzi a následně si vybrat podle funkcí, které ve svém podnikání využijete. Ceny se rozlišují podle toho, jaký typ programu si vyberete a zda to bude varianta Mini, Standard, Profi nebo jiná.[4]

Aplikace Pohoda byla vybrána jako aplikace pro účely testování softwaru na základě internetového srovnání účetních programů. Výběr testovacího programu byl proveden podle následující tabulky číslo 2, kdy rozhodujícím faktorem bylo zejména všestranné použití programu, cenová dostupnost, přehlednost a osobní hodnocení uživatelů.

	POHODA	**ABRA Flexibee**	**PROFIT**	**HELIOS Red**	**MoneyS3**
Finální hodnocení	9.9	9.3	9	8.5	8.4
Vhodné pro	Živnostníci, menší firmy, střední firmy, větší firmy	Živnostníci, menší firmy	Živnostníci, menší firmy	Živnostníci, menší firmy	Živnostníci, menší firmy, střední firmy

Recenzovaný tarif	Pohoda Standard	Business	Profit	Trial verze	Money S3 Business
Finance					
Délka testovací doby	3 měsíce	1 měsíce	Neomezená - Profit je zcela zdarma	1 měsíce	1 měsíce
Měsíční paušál	od 1980 Kč-15 980 Kč	od 3950 Kč-11950 Kč	990	od 4900 Kč-18 300 Kč	od 2490 Kč-14 990 Kč
Verze zdarma	Ano	Ne	Ano	Ano	Ano
Funkce					
Adresář, kontakty	Ano	Ano	Ano	Ano	Ano
Banka a pokladna	Ano	Ano	Ano	Ano	Ano
Elektronická evidence tržeb (EET)	Ano	Ano	Za příplatek	Ano	Za příplatek
Fakturace	Ano	Ano	Ano	Ano	Ano
Daňová evidence a jednoduché účetnictví	Ano	Ano	Ano	Ano	Ano
Podvojné účetnictví	Ano	Ano	Ano	Ano	Ano
Evidence majetku	Ano	Ano	Ano	Ano	Ano
Kniha jízd	Ano	Ano	Ano	Ano	Ano
Mzdy a personalistika	Ano	Ano	Za příplatek	Ano	Ano
Skladové hospodářství	Ano	Ano	Ano	Ano	Ano
Další funkce					
Napojení na e-shop	Ano	Ano	Ano - neomezený počet	Ano - neomezený	Ano
Hotovostní prodej (kasa)	Ano	Ano	Ano	Ano	Za příplatek
Daně (elektronická podání daňových přiznání)	Ano	Ano	Ano	Ne	Za příplatek

Podpora					
Telefon	Ano	Ano	Ano	Ano	Ano
Kontaktní formulář/e-mail	Ano	Ano	Ano	Ano	Ano
On-line konzultace	Ano	Ano	Ne	Ne	Ano
Pobočky	Ano	Plzeň	Ano - partnerské firmy	Ano (Praha, Brno, Ostrava, Hradec Králové)	Brno
Odpověď do 24 hodin	Ano	Ano	Ano	Ne	Ano
Odpovědi	10/10	10/10	10/10	10/10	9/10
Nápověda					
FAQ	Ano	Ano	Ne	Poradna s kompletními návody	Ne
Videonávody	Ano	Ano	Ano	Ano	Ano
Semináře a školení	Ano	Ano	Ano	Ano	Ano
Pomohou s instalací	Ano	Ano	Ano	Ano	Ano
Pomohou při práci v systému	Ano	Ano	Ano	Ano	Ano
Pomohou vybrat variantu	Ano	Ano	Ano	Ano	Ano
Příručky ke stažení	Ano	Ano	Ano	Ne, ale na stránkách je poradna s kompletními návody	Ano

Tabulka 2 Srovnání účetních programů[4]

4.3 Metody testování

4.3.1 Statická metoda

Testy byly již při jejich tvorbě automaticky staticky testovány vývojovým programem TestComplete proti syntaktickým chybám v kódu scriptů. Vytvořené testy byly pak následně prohlídnuty a upraveny v rámci Self-Review.

4.3.2 Dynamická metoda

Pomocí vývojového softwaru pro automatizované testy TestComplete byly vytvořena série testů pro otestování některých základních funkcionalit POHODA Lite, podle kterých byly vytvořeny automatizované testy ve skriptovacím jazyce Javascript a následně byly stejné testy provedeny manuálně. Podle kapitoly 3.5.1 černá skřínka víme, že testy byly prováděny bez znalosti fungování uvnitř programu, takže testem typu černá skříňka.

Dále byly vytvořeny pozitivní testy, které již známe z kapitoly 3.6.3. Tyto testy nám ověřují očekávané fungování systému. Vytvořené testy pokrývají i testy negativní nebo taky někdy známé jako testy vynucených chyb, které se zaměřují na neobvyklé chování programu a jeho odpověď. Testy ověřují, zda testovaný program POHODA vhodně reaguje na neočekávané akce z pohledu uživatele chybovým hlášením.

4.4 Tvorba a průběh testování

4.4.1 Automatizované testování

Automatizované testy byly tvořeny pomocí integrovaného nahrávání ve vývojové aplikaci TestComplete, kde byly namapovány jednotlivé prvky testované účetní aplikace POHODA a nahrány kroky automatizovaného testu. Následně nahrávka byla převedena do podoby JavaScriptu a upravena do finální podoby automatizovaného testu. Tímto způsobem bylo uděláno následujících 14 automatizovaných testů, které prochází vybrané funkcionality testovaného účetního programu. Veškeré zdrojové kódy testů lze nalézt v příloze 1.

1. Aktivování EET v účetním programu (Aktivace_EET_Test_Rezim)
2. Odeslaný doklad do EET bez přiřazeného EET profilu pro aktuálního uživatele (Neodeslane_EET_Chybny_EET_Profil)

3. Vytvoření Profilu pro EET a následné nastavení pro aktuálního uživatele (Vytvoreni_A_Nastaveni_EET_Profilu)
4. Přidání nového řidiče do účetní jednotky (Pridani_Ridice)
5. Přidání nového vozidla do účetní jednotky (Pridani_Vozidla)
6. Přidání nového řidiče, vozidla a následně vytvořená nová cesta pro vytvořeného řidiče s vozidlem (Nova_Jizda)
7. Přidání nového řidiče, vozidla a následně pokus o vytvoření nové cesty pro vytvořeného řidiče s vozidlem kde datum odjezdu je až po datu příjezdu (Nova_Jizda_Error)
8. Založení nové testovací účetní jednotky (Zalozeni_Ucetni_Jednotky)
9. Smazání testované účetní jednotky (Smazani_Ucetni_Jednotky)
10. Změna informací v účetní jednotce (Ucetni_jednotka_zmena_informaci)
11. Kontrola zobrazených informací na hlavní stránce a jejich záložkách (Hlavni_Stranka)
12. Založené nové vydané faktury bez uvedení položek (Vydane_Faktury_Bezpolozkova)
13. Založení nové vydané faktury s uvedeným jednotlivých položek na dané faktuře (Vydane_Faktury_Polozkova)
14. Pokus o založení vydané faktury ve formě kladného dobropisu/vrubopisu a následně uložení jako opravný daňový doklad (Vydane_Faktury_Error)

4.4.2 **Manuální testování**

Manuální testy byly prováděny na základě vytvořených automatizovaných testů, tak aby jednotlivé kroky odpovídaly krokům, které jsou vykonávány automatizovanými testy.

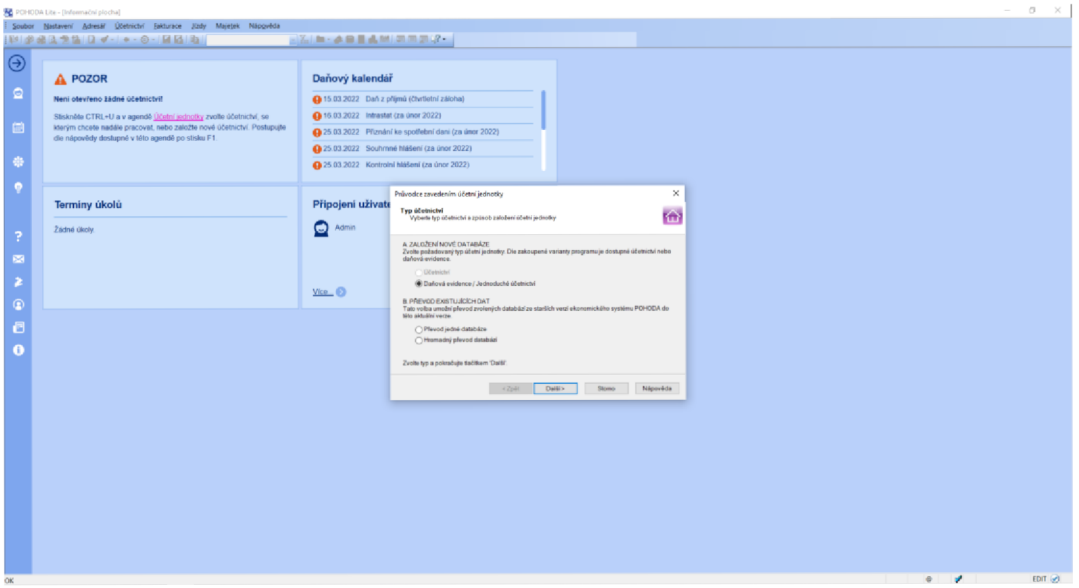
4.4.3 **Průběh testování**

Na následujících dvou příkladech je uveden průběh testování, byl proveden jeden pozitivní a jeden negativní scénář testu, u ostatních testů byl použit stejný postup. Průběh testování znázorňují jednotlivé kroky, které se vykonaly při provádění jak manuálního, tak automatizovaného testu v testované účetní aplikaci POHODA. Zdrojový kód automatizovaných testů byl rozdělen na dané kroky popisované v průběhu testu a byl přidán komentář popisující o jaký krok se jedná.

Příklad pozitivního testu Založení účetní jednotky

Testování začíná prvním krokem, kdy si otevřeme testovací aplikaci POHODA.

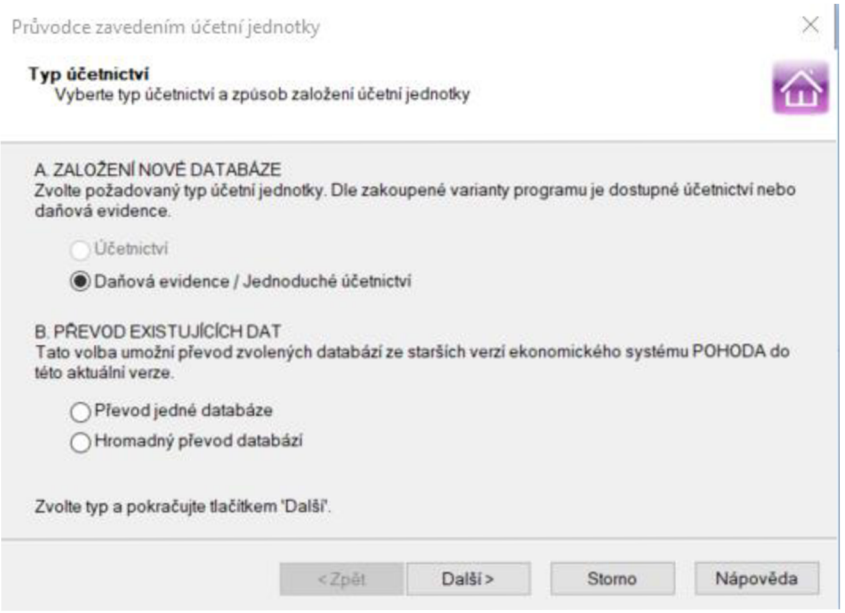
Pokud program otevíráme a ještě nemáme založenou žádnou účetní jednotku, program nás automaticky vyzve k založení nové účetní jednotky viz Obrázek č. 2.



Obrázek 2 Hlavní stránka bez účetní jednotky s dialogovým oknem

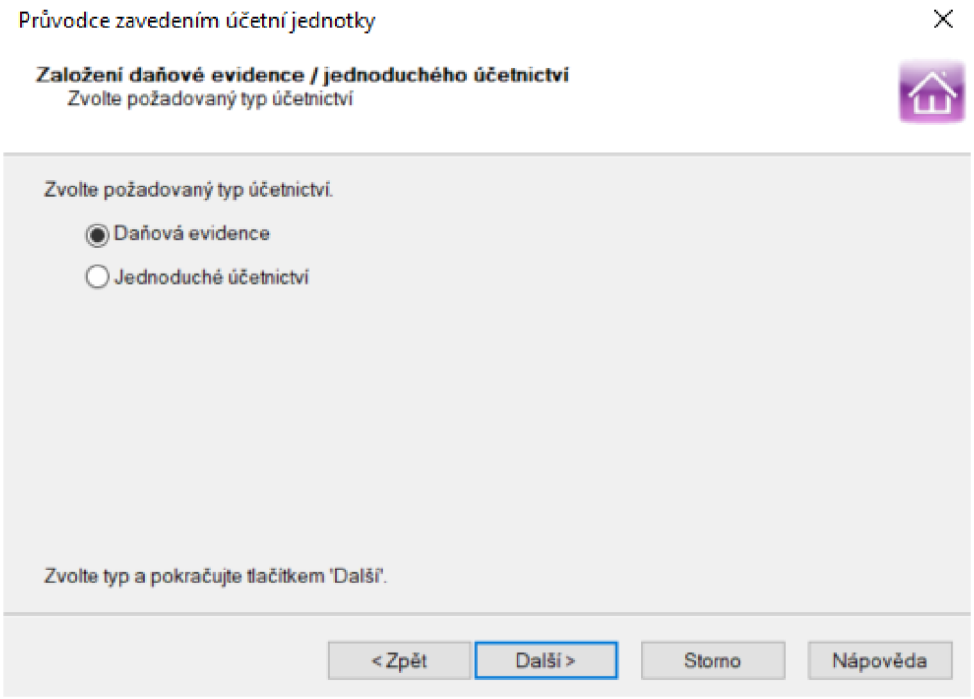
V druhém kroku přesuneme pozornost k dialogovému oknu uprostřed stránky viz Obrázek č. 3, v tomto okně se nám označí automaticky výběr Daňová evidence / Jednoduché účetnictví a následně stiskneme tlačítko další.

Zde se postup může lišit podle toho, jakou verzi programu POHODA používáme.



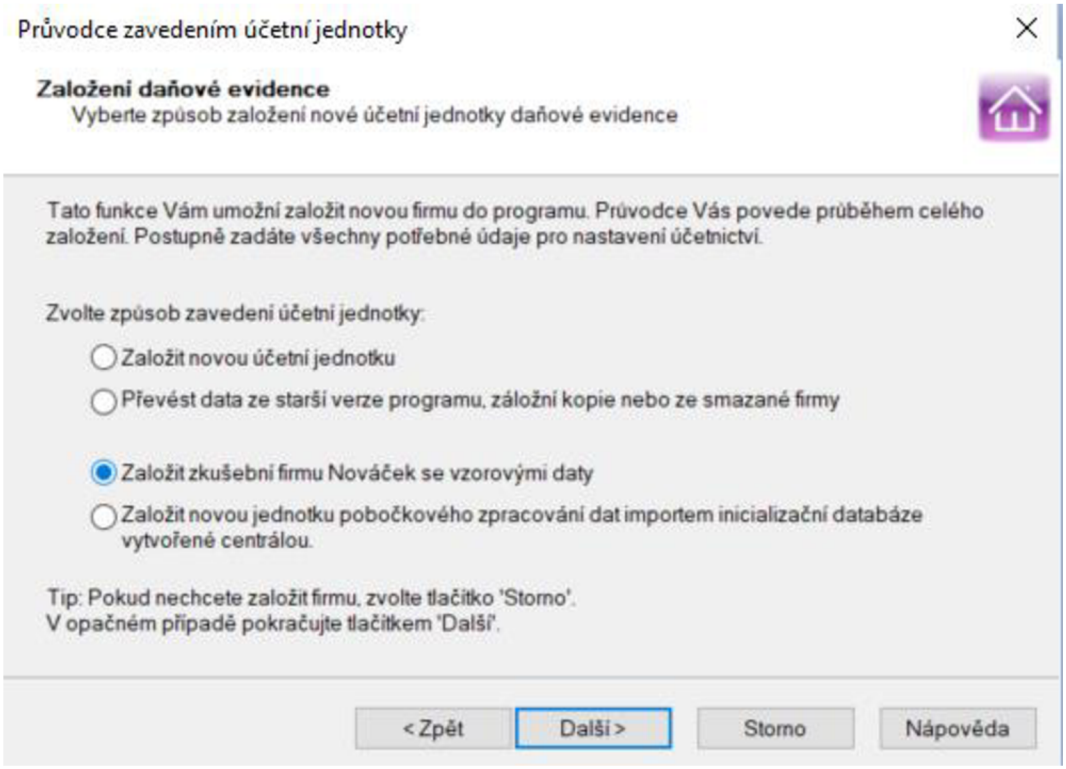
Obrázek 3 Dialogové okno Typ účetnictví

Ve třetím kroku necháme označenou volbu Daňová evidence a klikneme na tlačítko další viz Obrázek č. 4.



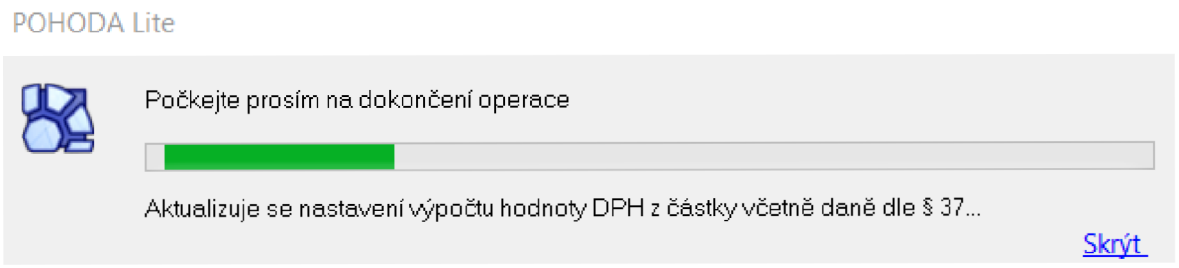
Obrázek 4 Dialogové okno Založení daňové evidence / jednoduchého účetnictví

Ve čtvrtém kroku označíme výběr Založit zkušební firmu Nováček se vzorovými daty a stiskneme tlačítko další viz Obrázek č. 5.



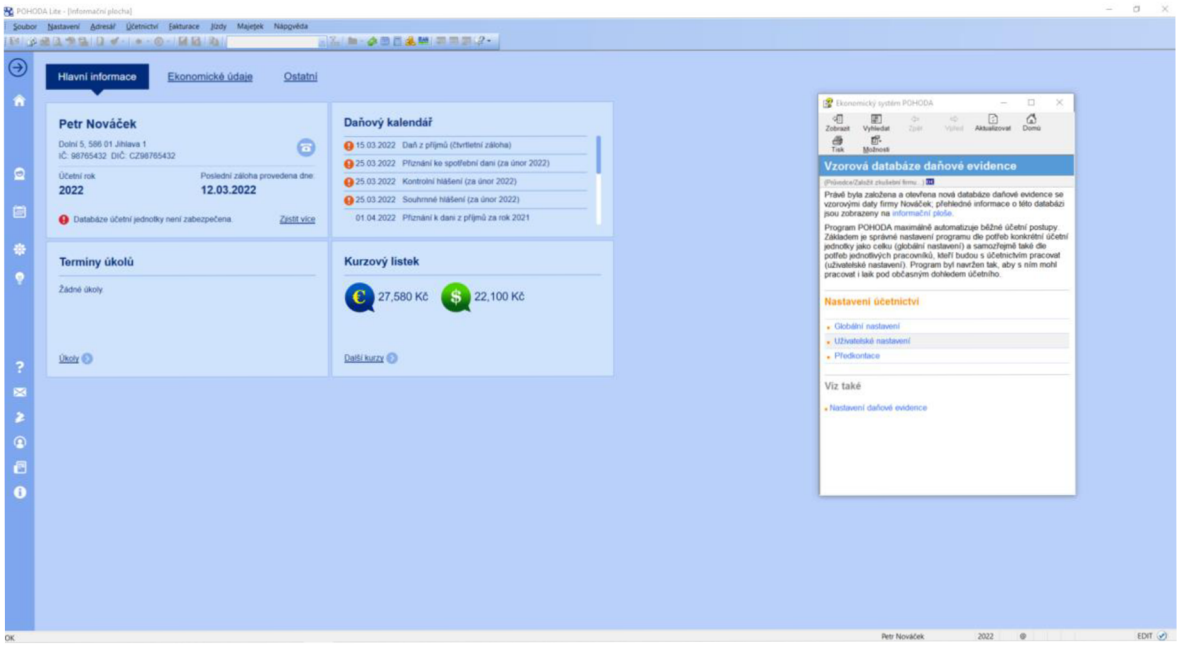
Obrázek 5 Dialogové okno Založení daňové evidence

Po dokončení kroku čtyři se nám zobrazí načítací okno na spodku obrazovky, které ukazuje načítání vzorových dat viz Obrázek č. 6.



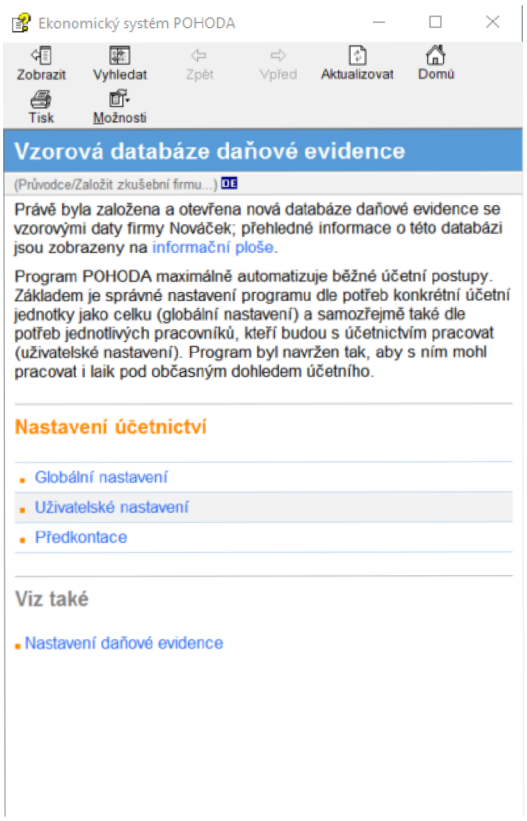
Obrázek 6 Načítací okno

Po načtení se nám zobrazí hlavní stránka pro založenou / naimportovanou účetní jednotku viz Obrázek č. 7.



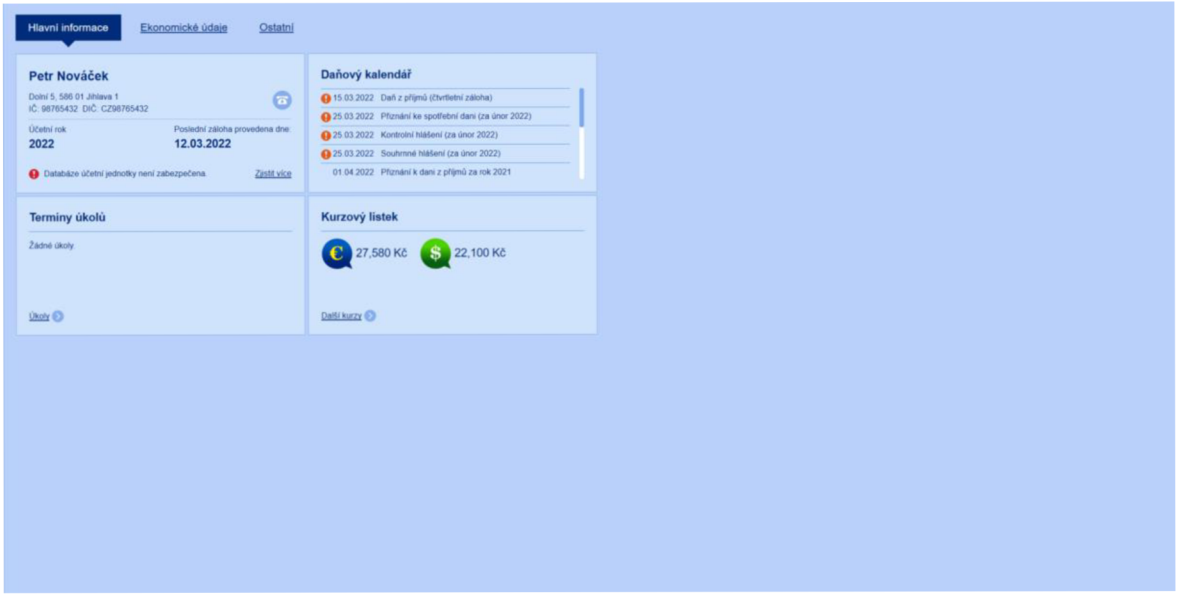
Obrázek 7 Hlavní stránka po založení účetní jednotky

V pátém kroku vypneme vyskakovací informační nápovědu viz Obrázek č. 8 v pravé části na hlavní stránce.



Obrázek 8 Nápověda Vzorová databáze daňové evidence

V kroku šest zkontrolujeme informace na hlavní stránce na záložce Hlavní informace podle Obrázku č. 9.



Obrázek 9 Kontrolní obrázek pro založenou účetní jednotku

Zdrojový kód pro automatizovaný test

```
function Zalozeni_Ucetni_Jednotky()
{
  //krok jedna
  //Runs the "Pohoda" tested application.
  TestedApps.Pohoda.Run();

  //krok dva
  //Clicks the 'btnDal_' button.
  Aliases.StwPh.dlgPr_vodceZaveden_m_etn_Jednotky.btnDal_.ClickButton();

  //krok tři
  //Clicks the 'btnDal_' button.
  Aliases.StwPh.dlgPr_vodceZaveden_m_etn_Jednotky.btnDal_.ClickButton();

  //krok čtyři
  //Clicks the 'radio' radio button.
  Aliases.StwPh.dlgPr_vodceZaveden_m_etn_Jednotky.page32770.radio.ClickButton();
  //Clicks the 'btnDal_' button.
  Aliases.StwPh.dlgPr_vodceZaveden_m_etn_Jednotky.btnDal_.ClickButton();

  //krok pět
  //Closes the 'wndHHParent' window.
  Aliases.StwPh.wndHHParent.Close();
  //Delays the test execution for the specified time period.
  Delay(500);
  //Clicks the 'AfxFrameOrView140' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView140.AfxFrameOrView140.Clic
k(320, 28);
  //Delays the test execution for the specified time period.
  Delay(500);
```

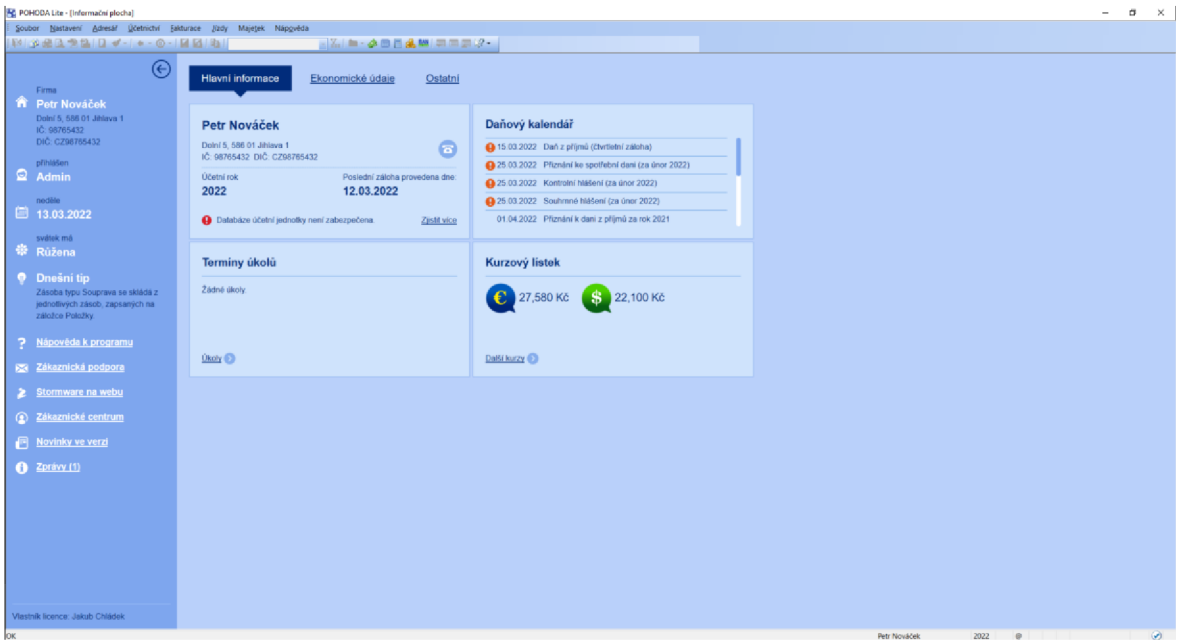
```
//krok šest
//Compares the HlavniS Stores item with the image of the Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView140.AfxFrameOrView140 object.

Regions.HlavniS.Check(Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView140.AfxFrameOrView140, false, false, 0, 0, "HlavniS_Mask");

//Zavření aplikace po skončení testu
//Closes the 'wndAfx' window.
Aliases.StwPh.wndAfx4.Close();
//Clicks the 'btnKonec' button.
Aliases.StwPh.dlg.btnKonec.ClickButton();
```

Příklad negativního testu Chybně odeslaný EET účet

V prvních pěti krocích si provedeme setup prostředí testované aplikace POHODA. Jako první krok si otevřeme testovací aplikaci POHODA. Následně se zobrazí naposledy otevřená účetní jednotka viz Obrázek č. 10.

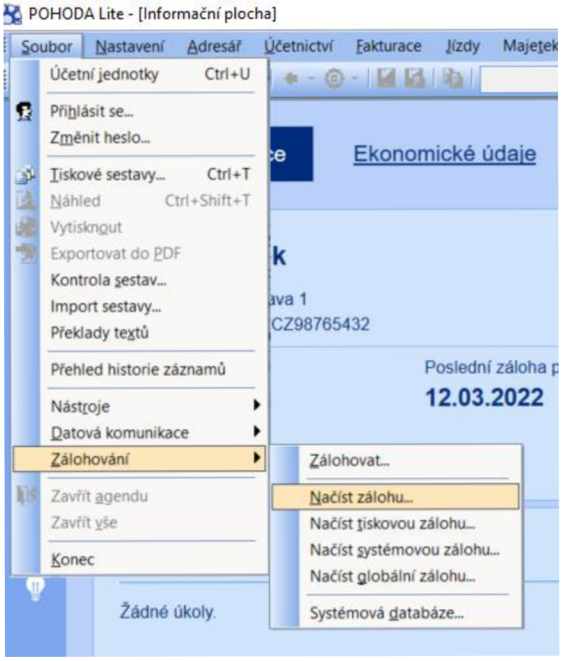


Obrázek 10 Hlavní stránka účetní jednotky

Krok dva zavřeme boční okno šipkou v horní části viz Obrázek č. 11.

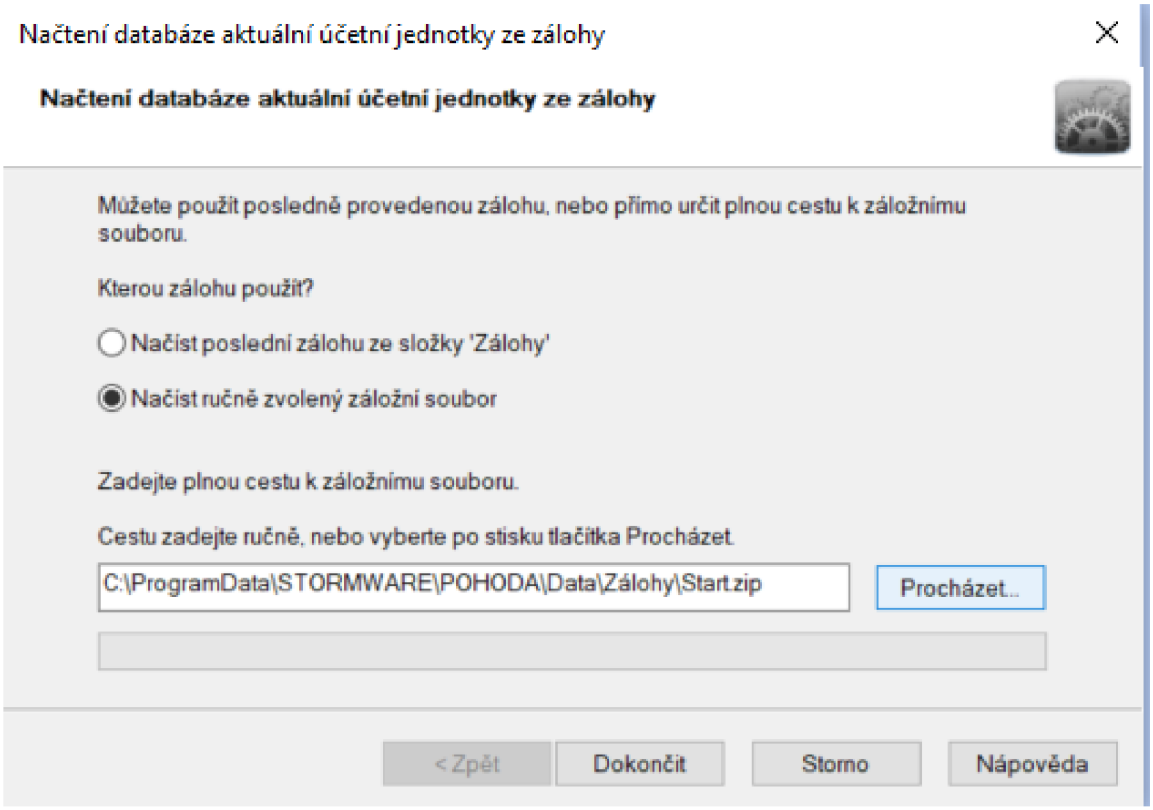
Obrázek 11 Boční panel na hlavní stránce

V kroku tři klikneme na horní navigační liště na Soubor>Zálohování>Načíst Zálohu… viz Obrázek č. 12. Stejného cíle můžeme docítil kliknutím na Soubor a zmáčknutí SHIFT + zn.



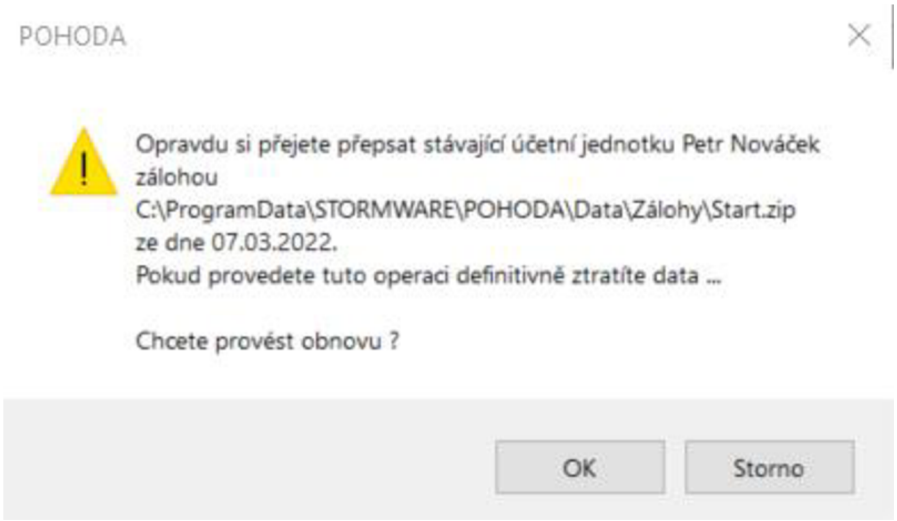
Obrázek 12 Navigace na Načíst zálohu…

V kroku čtyři klikneme na možnost Načíst ručně zvolený záložní soubor, následně zadáme cestu k naší záloze, tj. *C:\\ProgramData\\STORMWARE\\POHODA\\Data\\Zálohy\\Start.zip* viz Obrázek č. 13.



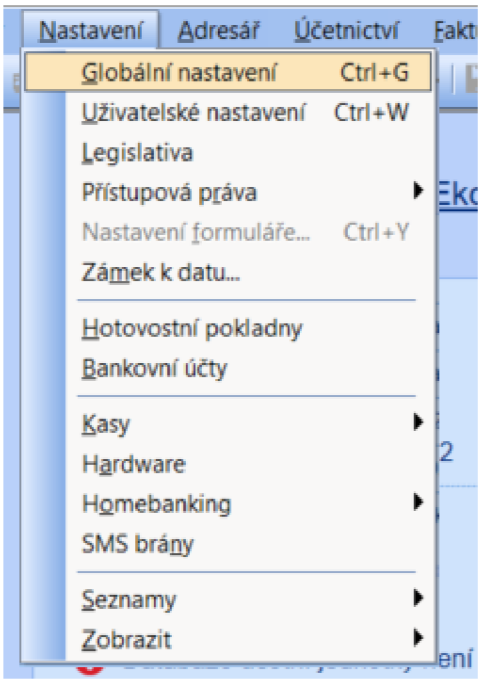
Obrázek 13 Dialogové okno Načtení databáze aktuální účetní jednotky ze zálohy

V kroku pět na nás vyskočí dialogové okno s upozorněním, že se chystáme přepsat aktuální data zálohou. Klikneme na tlačítko OK viz Obrázek č. 14.



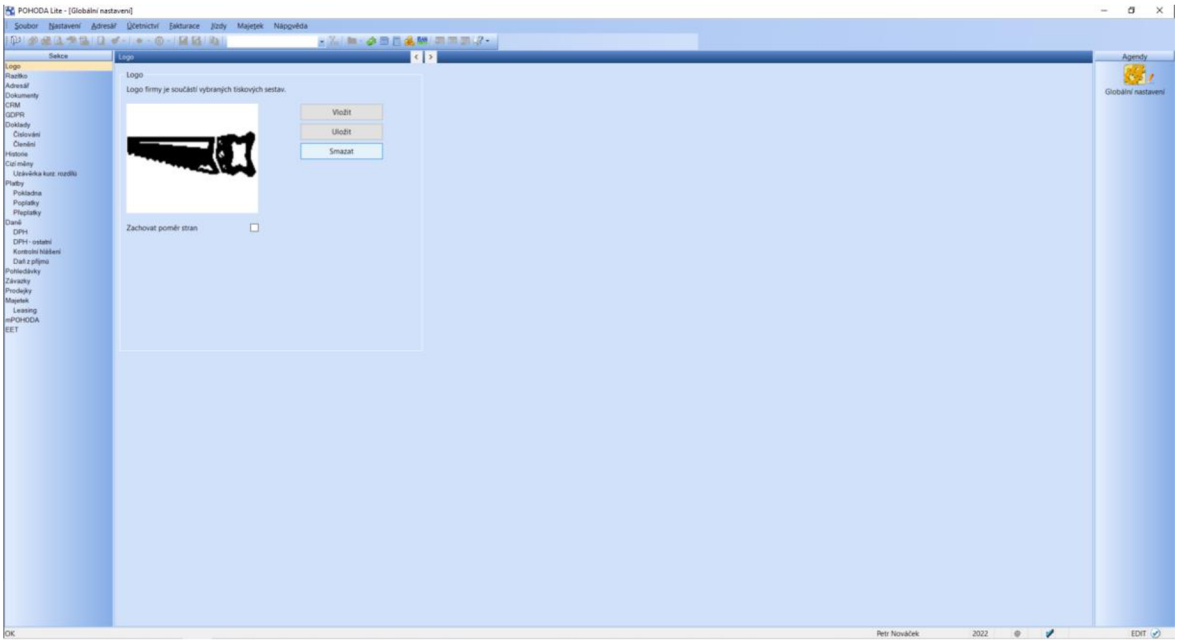
Obrázek 14 Dialogové okno upozorňující na přepsání aktuálních dat zálohou

Krok šest je kliknutí v horní liště na Nastavení>Globální Nastavení viz Obrázek č. 15. Stejného výsledku můžeme docílit stisknutím tlačítek CTRL + G.



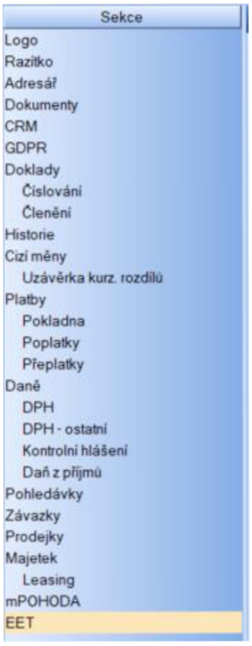
Obrázek 15 Navigace do Globálního nastavení

Po výběru se zobrazí obrazovka pro Globální Nastavení viz Obrázek č. 16.



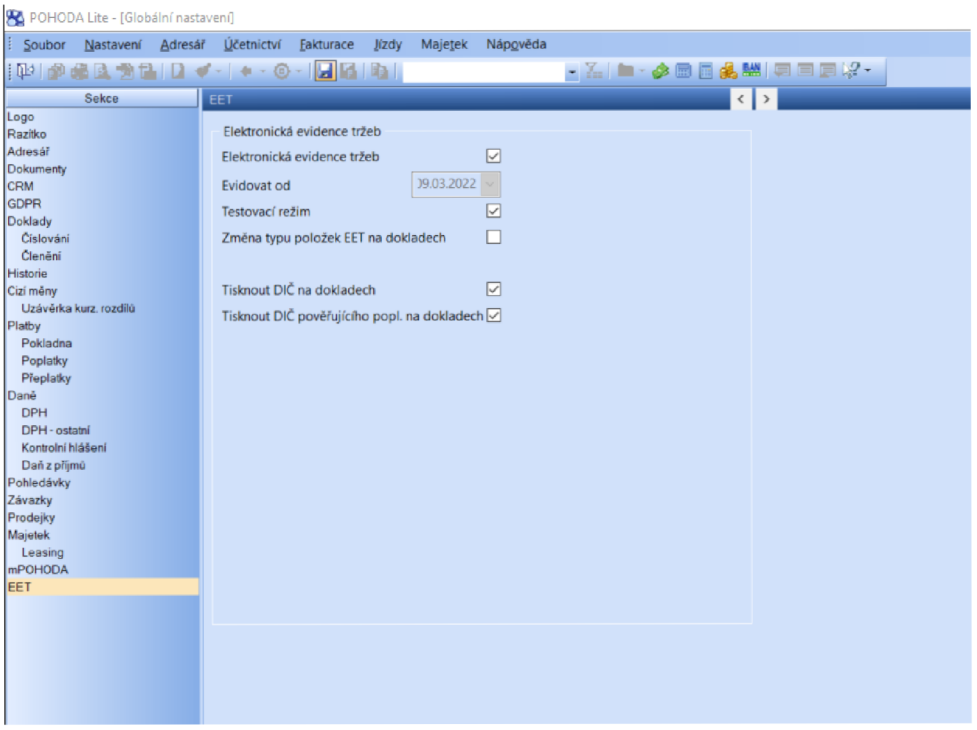
Obrázek 16 Globální nastavení

V kroku sedm klikneme na poslední záložku EET v seznamu záložek Globálního nastavení v levé části obrazovky viz Obrázek č. 17.



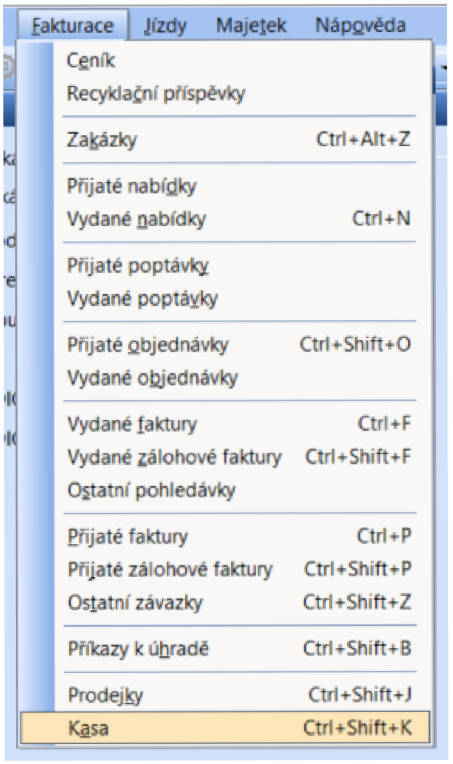
Obrázek 17 Seznam záložek v Globálním nastavení

Krok osm zobrazuje nastavení pro EET. Zaškrtneme checkbox pro Elektronická evidence tržeb, dále pak ještě vyplníme datum 09.03.2022 v kolonce Evidovat Od, pak ještě zaškrtneme Testovací režim a klikneme na uložit v horní části okna viz Obrázek č. 18.



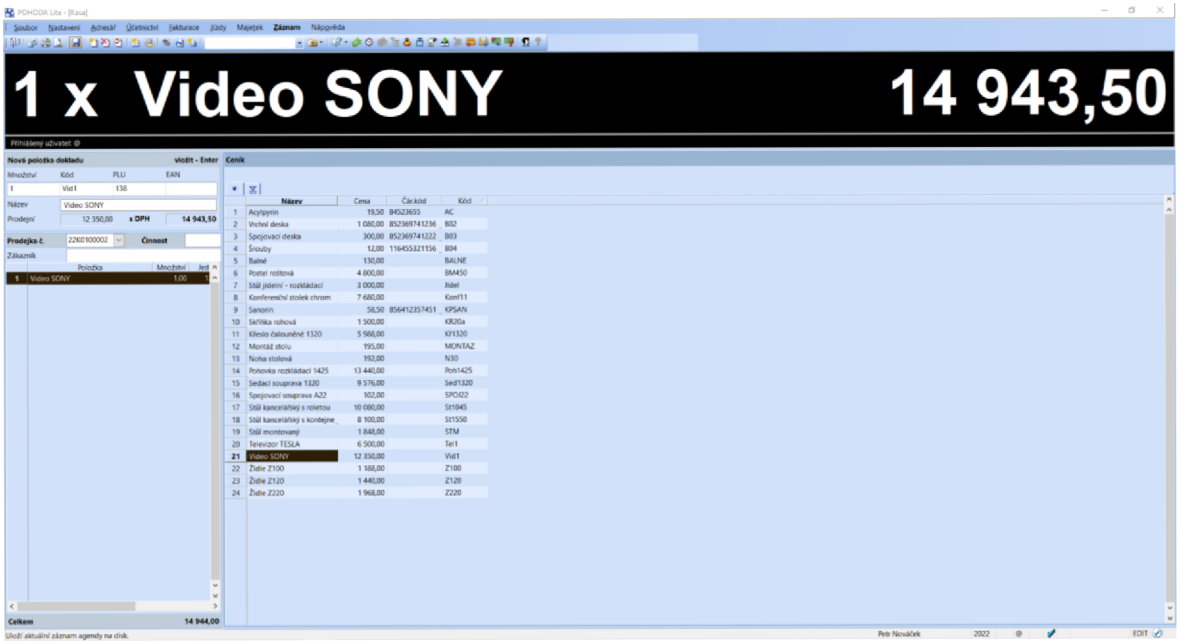
Obrázek 18 Globální nastavení pro EET

V kroku devět klikneme na horní liště na Fakturace>Kasa viz Obrázek č. 19. Stejného výsledku jde docílit stiskem CTRL + SHIFT + K.



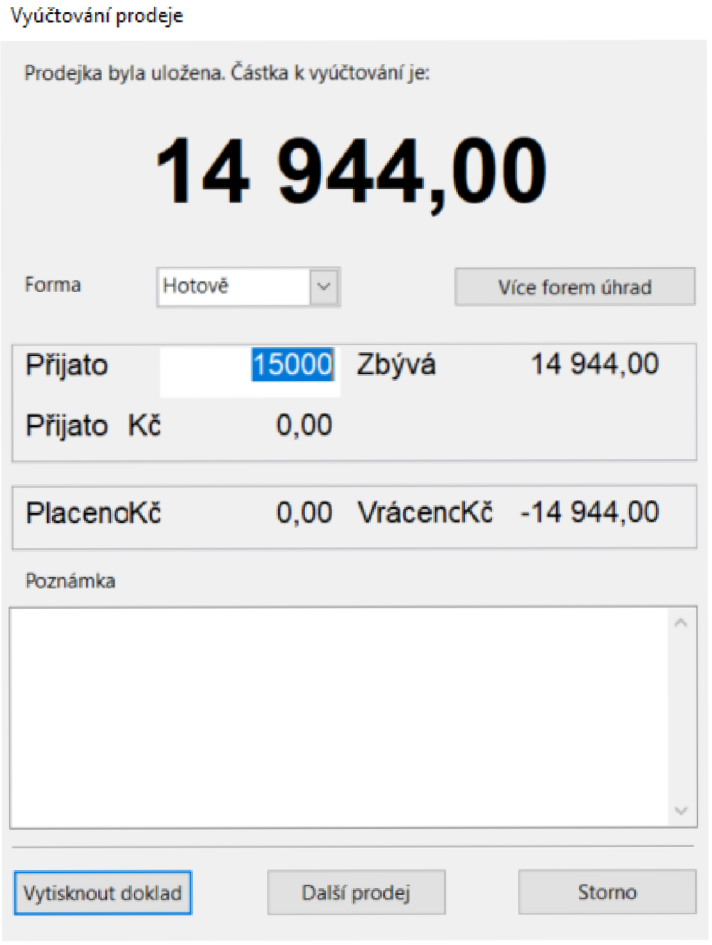
Obrázek 19 Navigace na Kasa

V kroku deset se nám zobrazí Kasa. Dvojklikem klikneme na položku 21 s názvem Video SONY. Ta se nám zobrazí v levé části obrazovky, poté klikneme na tlačítko uložit v horní části obrazovky viz Obrázek č. 20.



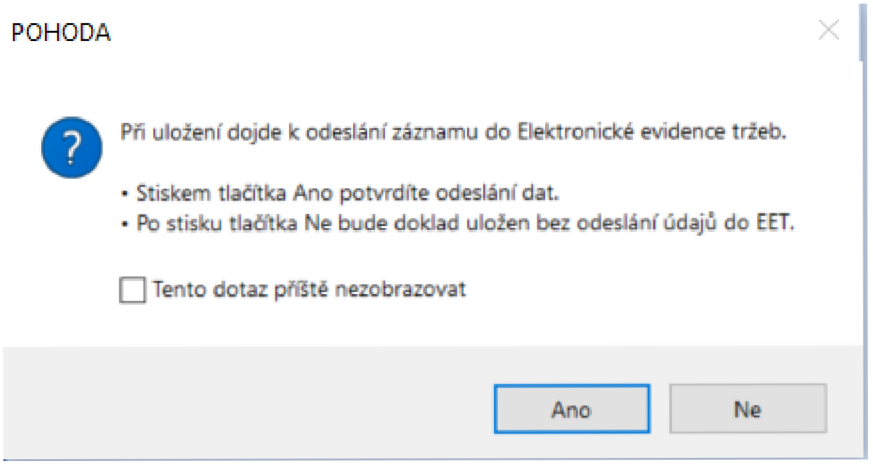
Obrázek 20 Kasa

V kroku jedenáct se nám zobrazí dialogové okno Vyúčtování prodeje, kde necháme formu Hotově, do řádku Přijato zadáme hodnotu 15000 a klikneme na tlačítko Další prodej viz Obrázek č. 21.



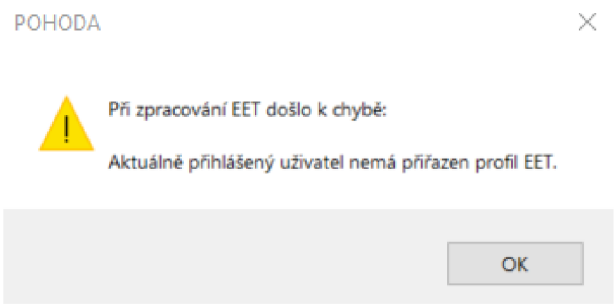
Obrázek 21 Vyúčtování prodeje

V kroku dvanáct se nám zobrazí informační okno, že při uložení dojde k odeslání do EET. Zde klikneme na tlačítko Ano viz Obrázek č. 22.



Obrázek 22 Informační okno o odeslání záznamu do EET

V kroku třináct se zobrazí chybová hláška o chybě při odesílání EET. Zkontrolujeme, že jako důvod se nám zobrazí hláška: „*Při zpracování EET došlo k chybě: Aktuálně přihlášený uživatel nemá přiřazen profil EET.*“, následně klikneme na tlačítko OK viz Obrázek č. 23.

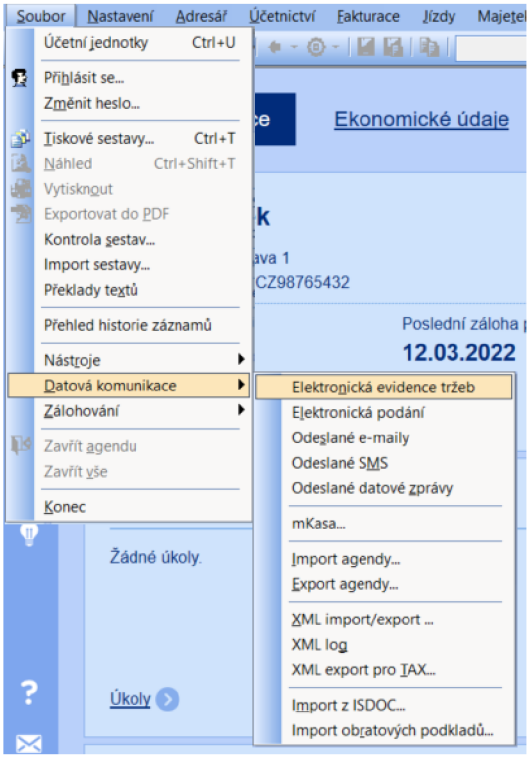


Obrázek 23 dialogové okno o chybě při odesílání EET

V kroku čtrnáct klikneme na tlačítko Zavřít agendu které se nachází v panelu nástrojů úplně vlevo v horní části obrazovky viz Obrázek č. 24.

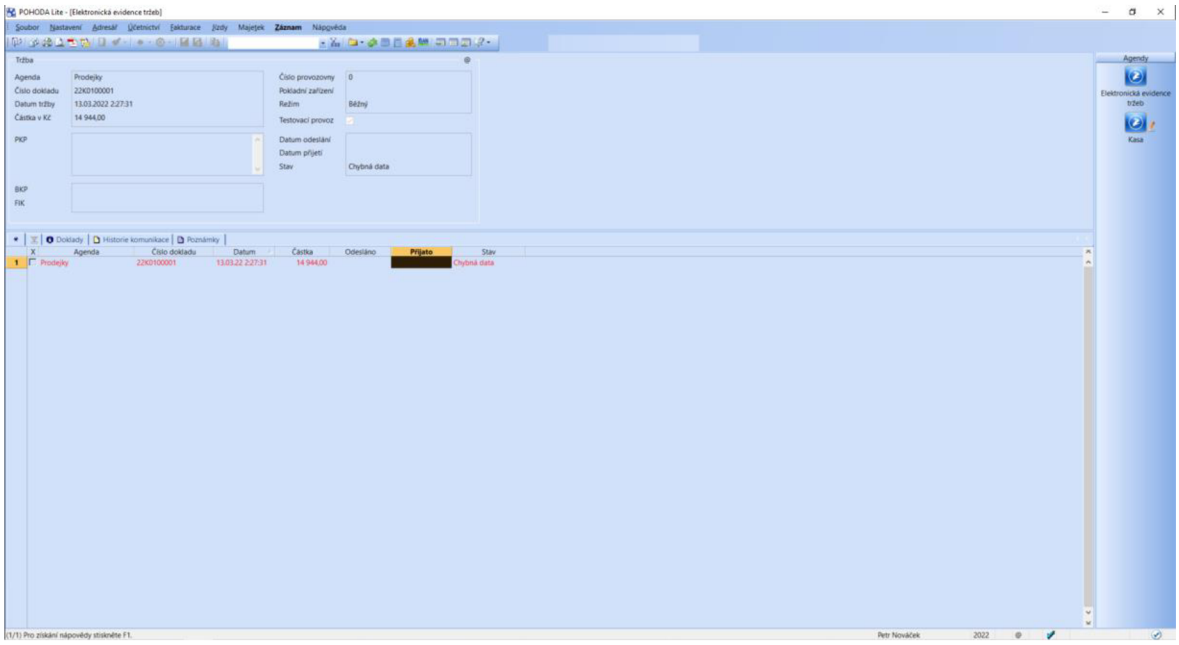
Obrázek 24 Panel nástrojů – Kasa

V kroku patnáct klikneme na Soubor>Datová komunikace>Elektronická evidence tržeb. Stejného výsledku jde docílit klikneme-li na Soubor a stiskneme klávesy SHIFT + D a pak n viz Obrázek č. 25.



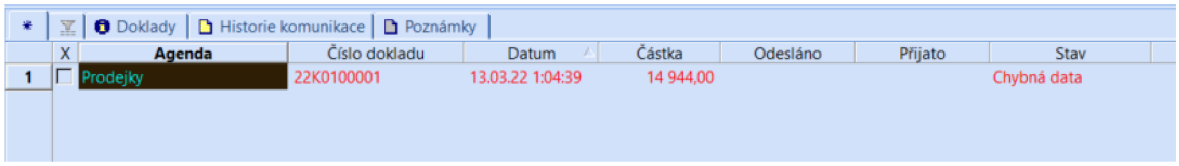
Obrázek 25 Navigace na Elektronická evidence tržeb

Po výběru se zobrazí obrazovka s komunikací EET viz Obrázek č. 26.



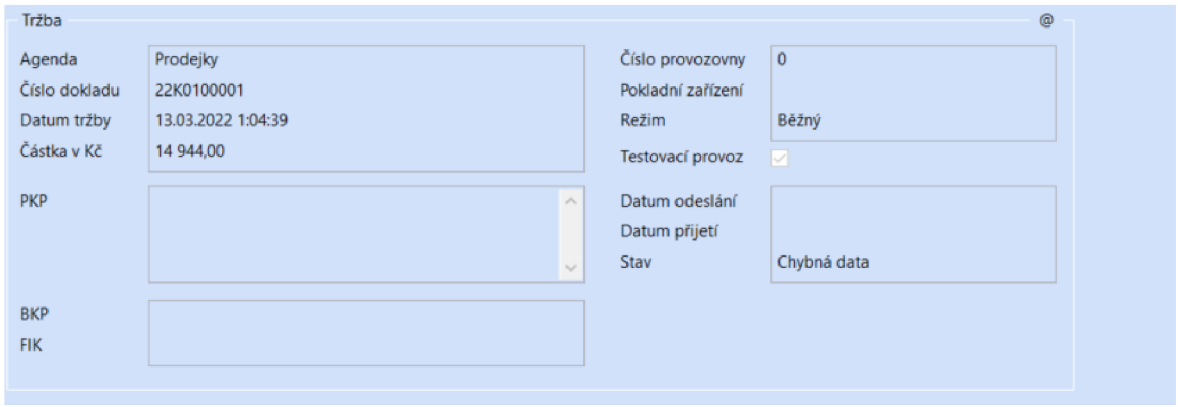
Obrázek 26 Komunikace s EET

V kroku šestnáct provedeme kontrolu údajů z dolní části obrazovky (při kontrole ignorujeme datum) podle Obrázku č. 27.



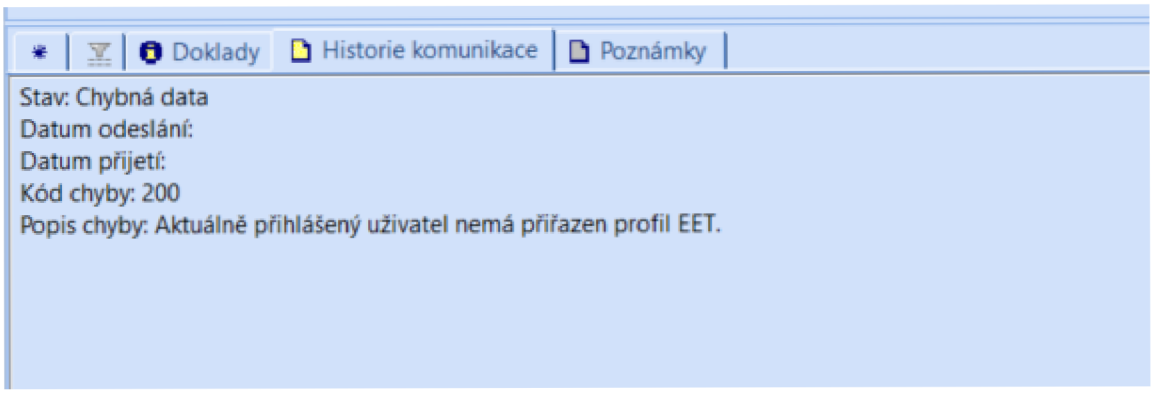
Obrázek 27 Tabulka s komunikací EET

V kroku sedmnáct provedeme kontrolu údajů z horní části obrazovky (při kontrole ignorujeme datum) podle Obrázku č. 28.



Obrázek 28 Detail položky z tabulky komunikace EET

V kroku osmnáct klineme na záložku Historie komunikace v dolní části obrazovky. Zde zkontrolujeme detail chyby podle Obrázku č. 29.



Obrázek 29 Detail chyby komunikace s EET na záložce Historie komunikace

Zdrojový kód pro automatizovaný test

```
function Start()
{
  //krok jedna
  //Runs the "Pohoda" tested application.
  TestedApps.Pohoda.Run();
  Delay(500);

  //krok dva
  //Clicks the 'AfxFrameOrView140' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView140.AfxFrameOrView140.Click(326, 27);

  //krok tři
  //Clicks the 0 item of the 'Menu' toolbar.
  Aliases.StwPh.wndAfx4.Menu.Menu.ClickItem(0, true);
  //Enters 'ZN' in the 'Menu' object.
  Aliases.StwPh.wndAfx4.Menu.Menu.Keys("ZN");
```

```
 //krok čtyři
 //Clicks the 'radio' radio button.
 Aliases.StwPh.dlg2.page32770.radio.ClickButton();
 //Clicks the 'Edit' object.
 Aliases.StwPh.dlg2.page32770.Edit.Click(166, 18);
 //Enters the text 'C:\ProgramData\STORMWARE\POHODA\Data\Zálohy\Start.zip' in the
'Edit' text editor.

Aliases.StwPh.dlg2.page32770.Edit.SetText("C:\\ProgramData\\STORMWARE\\POHOD
A\\Data\\Zálohy\\Start.zip");
 //Clicks the 'btnDokon_it' button.
 Aliases.StwPh.dlg2.btnDokon_it.ClickButton();

 //krok pět
 //Clicks the 'btnOK' button.
 Aliases.StwPh.dlgPOHODA.btnOK.ClickButton();
}
function Neodeslane_EET_Chybny_EET_Profil()
{
 //krok 1-5
 Setup.Start();
 //Delays the test execution for the specified time period.
 Delay(500);

 //krok šest
 //Enters '^g' in the 'AfxFrameOrView140' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView140.AfxFrameOrView140.Key
s("^g");
```

```
 //krok sedm
 //Clicks the 27 item of the 'ListBox' list box.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1403.page32770.ListBox.Click
ItemXY(27, 4, 7);

 //krok osm
 //Sets the state of the 'checkElektronick_EvidenceTr_eb' check box to cbChecked.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1403.page327702.page32770.c
heckElektronick_EvidenceTr_eb.ClickButton(cbChecked);
 //Clicks the 'editEvidovatOd' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1403.page327702.page32770.e
ditEvidovatOd.Click(8, 11);
 //Enters the text '09.03.2022' in the 'editEvidovatOd' text editor.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1403.page327702.page32770.e
ditEvidovatOd.SetText("09.03.2022");
 //Sets the state of the 'checkTestovac_Re_im' check box to cbChecked.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1403.page327702.page32770.c
heckTestovac_Re_im.ClickButton(cbChecked);
 //Clicks the 33370 item of the 'Standardn_' toolbar.
 Aliases.StwPh.wndAfx4.Menu.Standardn_.ClickItem(33370, false);

 //krok devět
 //Clicks the 4 item of the 'Menu' toolbar.
 Aliases.StwPh.wndAfx4.Menu.Menu.ClickItem(4, true);
 //Enters 'a' in the 'Menu' object.
 Aliases.StwPh.wndAfx4.Menu.Menu.Keys("a");
```

```
//krok deset
//Double-clicks the 'HGrid' object.

Aliases.StwPh.wndAfx4.MDIClient.wndKasa.AfxMDIFrame140.AfxMDIFrame140.Afx
MDIFrame140.page32770.HGrid.DblClick(211, 525);
//Clicks the 33370 item of the 'Kasa' toolbar.
Aliases.StwPh.wndAfx4.Menu.Kasa.ClickItem(33370, false);

//krok jedenáct
//Drags the 'editP_ijato' object.
Aliases.StwPh.dlgVy_tov_n_Prodeje.editP_ijato.Drag(69, 5, 108, -2);
//Enters the text '15000' in the 'editP_ijato' text editor.
Aliases.StwPh.dlgVy_tov_n_Prodeje.editP_ijato.SetText("15000");
//Clicks the 'btnDal_Prodej' button.
Aliases.StwPh.dlgVy_tov_n_Prodeje.btnDal_Prodej.ClickButton();

//krok dvanáct
//Clicks the 'btnAno' button.
Aliases.StwPh.dlgPOHODA.btnAno.ClickButton();

//krok třináct
//Checks whether the 'WndCaption' property of the Aliases.StwPh.dlgPOHODA.Static3
object equals 'Při zpracování EET došlo k chybě:
//
//Aktuálně přihlášený uživatel nemá přiřazen profil EET.'.
aqObject.CheckProperty(Aliases.StwPh.dlgPOHODA.Static3, "WndCaption", cmpEqual,
"Při zpracování EET došlo k chybě:\r\n\r\nAktuálně přihlášený uživatel nemá přiřazen
profil EET.");
//Clicks the 'btnOK' button.
Aliases.StwPh.dlgPOHODA.btnOK.ClickButton();
```

```
//krok čtrnáct
//Clicks the 57602 item of the 'Kasa' toolbar.
Aliases.StwPh.wndAfx4.Menu.Kasa.ClickItem(57602, false);

//krok patnáct
//Clicks the 0 item of the 'Menu' toolbar.
Aliases.StwPh.wndAfx4.Menu.Menu.ClickItem(0, true);
//Enters 'Dn' in the 'Menu' object.
Aliases.StwPh.wndAfx4.Menu.Menu.Keys("Dn");

//krok šestnáct
//Compares the EETError Stores item with the image of the Aliases.StwPh.wndAfx.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.HGrid object.

Regions.EETError.Check(Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.HGrid, false, false, 0, 0, "EETError_Mask");

//krok sedmnáct
//Double-clicks the 'HGrid' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.HGrid.DblClick(245, 35);
//Compares the EETDetailError Stores item with the image of the Aliases.StwPh.wndAfx.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page327702 object.

Regions.EETDetailError.Check(Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page327702, false, false, 0, 0, "EETDetailError_Mask");
```

```
  //krok osmnáct
  //Clicks the 'AfxWnd140' object.

Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.AfxWnd140.Click(279, 3);
  //Checks whether the 'wText' property of the Aliases.StwPh.wndAfx.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.Edit object equals 'Stav: Chybná data
  //Datum odeslání:
  //Datum přijetí:
  //Kód chyby: 200
  //Popis chyby: Aktuálně přihlášený uživatel nemá přiřazen profil EET.
  //
  //'.

aqObject.CheckProperty(Aliases.StwPh.wndAfx4.MDIClient.wndAfxFrameOrView1404.AfxMDIFrame140.page32770.Edit, "wText", cmpEqual, "Stav: Chybná data\r\nDatum odeslání: \r\nDatum přijetí: \r\nKód chyby: 200\r\nPopis chyby: Aktuálně přihlášený uživatel nemá přiřazen profil EET.\r\n\r\n");

  //Zavření aplikace po skončení testu
  Setup.End();
}
```

4.4.4 **Porovnání testů**

V následující tabulce číslo 3 si můžeme porovnat výsledky trvání času provádění jednotlivých testů.

Název Testu	Čas na provedený test	
	Manuální test	Automatizovaný test
Aktivace_EET_Test_Rezim	00:25	00:14
Neodeslane_EET_Chybny_EET_Profil	01:05	00:16
Vytvoreni_A_Nastaveni_EET_Profilu	01:18	00:19
Pridani_Ridice	03:55	00:13
Pridani_Vozidla	01:55	00:14
Nova_Jizda	04:55	00:40
Nova_Jizda_Error	04:50	00:28
Zalozeni_Ucetni_Jednotky	00:23	00:15
Smazani_Ucetni_Jednotky	00:21	00:07
Ucetni_jednotka_zmena_informaci	03:20	00:16
Hlavni_Stranka	01:13	00:13
Vydane_Faktury_Bezpolozkova	01:59	00:15
Vydane_Faktury_Polozkova	01:33	00:24
Vydane_Faktury_Error	01:37	00:14

Tabulka 3 Výsledky časů provádění jednotlivých testů

Výsledky a pozorování z testování automatickými testy a manuálními testy byly porovnány a zobrazeny v následující tabulce číslo 4.

	Manuální testy	Automatizované testy
Průměrný čas na tvorbu jednoho testu	15 minut	1 hodina
Celkový čas na provedení vytvořených testů	28minuty 49sekund	4minuty 8 sekund
Čas, kdy můžeme test vykonávat	Manuální testy můžeme vykonávat v běžné pracovní době pracovníka, který vykonává manuální testy	Automatické testy můžeme provádět kdykoliv.

Dokumentace provedených testů	Dokumentaci k testům provádí pověřený tester, který daný test provádí. Má možnost provést dokumentaci krok po kroku (to záleží kde daný manuální test provádí, jelikož některé programy podporují manuální testy a jde zde odškrtávat jednotlivé kroky během provádění testu a psát si k tomu poznámky), nebo po provedeném testu, ať už úspěšném (kdy se nenajde chyba během programu) tak i neúspěšném zapsat výsledky manuálního testu	Dokumentaci k provedeným testům provádí test přímo za běhu testu, proto není nutnost další práce na dokumentovaní provedených kroků a popřípadě nalezených chyb
Kdo může testy vytvářet / provádět	Tvorbu manuálního testu by měl provádět pracovník, který má znalost, jak má daná aplikace fungovat a vytvořit podle toho daný test. Provádět manuální testy můžou pověřené osoby, které nemusejí mít žádné odborné znalosti, ale je dobré, když mají z fungováním dané aplikace zkušenosti.	Automatizované testy může vytvářet vývojář, ale pustit je pak už může prakticky kdokoliv.

Tabulka 4 Porovnání automatizovaných a manuálních testů

Z tabulky 4 můžeme vidět, že tvorba manuálních testů je daleko rychlejší a levnější, jelikož na jejich tvorbu nepotřebujeme vývojáře. Bohužel jejich vykonávání je závislé na pracovní době testera a také jeho zkušenosti a pozornosti. Při provádění manuálních testů je největší riziko v lidském faktoru. Významným faktorem automatizovaných testů je zkrácení doby průběhu testování.

5 Zhodnocení výsledků

Při mém vykonávání manuálních testů jsem narazil na několik chyb při kontrole údajů z nepozornosti (přepsání se ve slovech, číslech apod.). Zde se pak čas navyšuje, protože je nutné je pak opravit, je to patrné zejména při složitějším testu.

I když automatické testy potřebují větší čas a náklady na přípravu testů, zůstává jejich nepopiratelnou výhodou to, že jdou pouštět stále dokola se stejným výsledkem, podobným časem trvání, můžeme nastavit jejich spuštění kdykoliv a jejich následné testování již nic nestojí. Testování automatickými testy jednoznačně šetří čas, odhalují chyby v dřívějších fázích a tedy snižují náklady a dělají vlastně testování zábavnější, neboť nikdo nemá rád opakující se manuální rutinu. Díky automatizaci dochází i k zpřesnění, protože pravděpodobnost, že se tester při manuálním testování někde splete nebo něco přehlédne je vysoká.

V dnešní době rychlého vývoje již není možné tolik testovat manuálně. Neznamená to však, že manuální testování definitivně skončilo, ale objem testovaného softwaru se jednoznačně přehoupl na stranu automatizovaných testů.

6 Závěr

Ve své praktické části jsem se zaměřil na problematiku automatizovaného testování. Snažil jsem se, aby byla má práce pochopitelná lidem, kteří nemají moc zkušeností s automatizovanými testy a mohli se tak o nich dozvědět víc. Využil jsem také vlastních zkušeností z praxe, které jsem nabral jako vývojář automatizovaných testů.

Na základě studia odborných a literárních zdrojů jsem stanovil, jaké metody použiji pro své testování. Jako testovací desktopovou aplikaci jsem si zvolil na základě internetového srovnání účetní program POHODA, na kterém jsem testovací metody uvedl do praxe tvorbou automatizovaných testů na základní funkcionality tohoto účetního programu. Po provedení automatizovaných testů a zaznamenání jejich výsledků do tabulky jsem pak následně zopakoval stejný postup manuálně podle kroků vytvořených pro automatizované testy. Výsledky jsem si zaznamenal do tabulky a provedl porovnání s výsledky automatizovaných testů.

Při porovnávání výsledků testování různými druhy testů se potvrdilo, že automatické testování má oproti manuálnímu typu testu znatelné výhody, což v podstatě potvrdilo má očekávání před zahájením testování. Manuální testování sice vychází jako výhodnější z podhledu nákladů a času na výrobu, ale jsou ovlivněna možnou lidskou chybou ve výsledcích, delším časem na provedení testů a jejich provádění závisí na pracovní době manuálních testerů. Naopak u automatizovaných testů je doba výroby testů značně delší a bude i nákladnější, jelikož výrobu testů provádí vývojář automatizovaných testů, ale následně to pak přináší výhody, které rozhodně převažují zvýšené náklady a čas na výrobu. Mezi tyto výhody patří zkrácení doby, kdy se testy provádí (v mé praktické části vyšlo zkrácení doby až na 1/6 oproti manuálním testům), odstranění chyby lidského faktoru z testů (testy běží při každém průběhu naprosto stejně), testy můžeme pouštět kdykoliv bude potřeba a nakonec testy nás za jejich provedení nic nestojí.

7 Seznam použitých zdrojů

[1] ARNOŠT HAVELKA a RUDOLF PECINOVSKÝ. *JUnit 5*. B.m.: Grada Publishing, 2018. ISBN 978-80-271-0733-9.

[2] PAVEL HEROUT. *Testování pro programátory*. První. B.m.: KOPP, 2016. ISBN 978-80-7232-481-1.

[3] MIROSLAV BUREŠ, MIROSLAV RENDA, MICHAL DOLEŽEL, PETER SVOBODA, ZDENĚK GRÖSSL, MARTIN KOMÁREK, ONDŘEJ MACEK, a RADOSLAV MLYNÁŘ. *Efektivní testování softwaru*. nedatováno. ISBN 978-80-247-5594-6.

[4] 5NEJ.CZ. *Srovnání účetních programů* [online]. 12. březen 2022. Dostupné z: https://www.5nej.cz/srovnani-ucetnich-programu/

[5] STORMWARE. *Účetní program POHODA* [online]. 12. březen 2022. Dostupné z: https://www.stormware.cz/pohoda/

[6] SMARTBEAR SOFTWARE. *TestComplete* [online]. 12. březen 2022. Dostupné z: https://smartbear.com/product/testcomplete/features/

[7] TESTOVANISOFTWARU. *Testování Softwaru - TestComplete* [online]. 12. březen 2022. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/testcomplete/

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů

Přílohy

Příloha 1 – TestComplete.zip (Zdrojové kódy testů a záznamy)