ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KEV

BAKALÁŘSKÁ PRÁCE

Možnosti automatizovaného testování aplikací

Tomáš Jandák **2012**

Anotace

Předkládaná bakalářská práce je zaměřena na základní principy testování softwaru (dále jen SW), upozorňuje na výhody a nevýhody automatizovaného testování. Ukazuje možnosti trhu se SW nástroji na podporu automatizovaného testování vytvořeného webového programu a ukazuje automatizaci testů na praktickém příkladu. Testování SW je nedílnou součástí pro vznik SW aplikace. Automatizace je součástí přirozeného cyklu člověka pro jakoukoliv tvorbu a vývoj. Automatizace testů jsou testy provedené člověkem a pomocným nástrojem. Cílem této bakalářské práce je seznámit posluchače s možnostmi automatizace testu SW. Pro snadnější pochopení dané problematiky práce obsahuje možnosti nástrojů určených pro automatizaci funkčních testů na webové aplikaci (tj. se vstupem a výstupem pro uživatele na webový prohlížeč neboli klient - server). Bakalářská práce pojednává převážně o praktických zkušenostech a všeobecnosti teoretických metodik v oblasti testů SW dle vlastních a převzatých kolegiálních zkušeností.

Klíčová slova

Automatizované testy, chyba, dělení testů SW, funkční testy SW, nástroje pro automatizaci testů, QuickTest, principy testování SW, Selenium, testování SW, trh s nástroji pro automatizované testování

Abstract

Submitted bachelor work is focused on basic principes of software (thereinafter SW) testing, it draws attention to advantages and disadvantages of automated testing. It presents possibilities of SW tools market for automated testing support for created web program and it also shows automated testing in practical example. SW testing is entire part for beginning of SW application. Automatization is a part of human natural cycle for any creation and developement. Automated testing is a set of tests, which are done by human and helping tool. Objective of this bachelor work is familiarization with possibilities of SW automated testing. It contains possibilities of tools for automated functional testing in web application (with input and output to web browser for users by another name client – server) for easier understanding. Bachelor work mainly deal with practical experiences and universality of theoretical methodics in the area of SW testing according to personal and assumed experiences.

Key words

Automated testing, error, division of SW tests, functional SW testing, tools for automated testing, QuickTest, principes of SW testing, Selenium, SW testing, tools market for automated testing

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne Tomáš Jandák

……..

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Doc. Ing. Konstantinovi Schejbalovi, CSc. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Úvod

Testování vychází z přesnosti programátorského týmu a jeho testerů, kteří musí pochopit cíle vytvořeného SW a vcítit se, komu je SW určen [2] (koncovému uživateli [9]). Společnými silami se snaží vytvořit kvalitní produkt. Motivací lidské činnosti by měl být dobrý pocit z vlastní tvorby. Nedílnou součástí této motivace je i správná myšlenka k vytvoření produktu, která se musí řídit i potřebou dosažení kvality (plánovat a řídit). [8] Před započetím vývoje a testů se musí stanovit metriky kvality a zvolit jednotný přístup. Jelikož každý SW produkt je specifický jako práce jednotlivce, musí se zvolit jednotná metodika a pohled. Většina metodik po zvolení implementačního postupu píše, že se má v této části určit záběr testování.
Z praktického hlediska je dobré v této části začít s definicí záběru testů, který se bude průběžně upravovat až uvedením SW produktu k plnění svého účelu. Testy by měly již od začátku myšlenky vytvoření aplikace pomáhat a určovat jednoznačnost cílů. Proto se i samotné testovací týmy skládají z analytiků, architektů, vývojářů, managerů, „testerů“ a uživatelů (testerem je každý člověk, aniž by si to uvědomil). Toto pravidlo platí pro všechny profesní role použité pro tvorbu SW). Proto v metodice a metrikách kvality musí být také obsaženy jednoznačné popisy práce pro potřebné zpětné vyhodnocení, které pomáhá ke zlepšení kvality pro budoucí projekty. Testování je především o zkušenostech a o osobnostech týmu. [8] a [2]

Předkládaná práce seznamuje čtenáře s testováním SW, s možnostmi automatizace testů a hlavně s nástroji určenými pro tvorbu automatizovaných testů. Pro seznámení čtenáře popisuji hlavně funkční testy vnořené do integračních testů pro systémy založené na principu formulářových aplikací, kdy vstupem a výstupem jsou webové prohlížeče či formulářové aplikace. Na tomto druhu aplikací lze ukázat nejvíce problémů automatizace testů a zároveň zde lze snadněji pochopit testy SW. Toto téma by bylo na několik publikací, proto se budu držet základních principů. Každý výrobní cyklus prošel procesem od kusové výroby jen za pomocí lidských sil po plně automatizovanou výrobu. Výjimkou není ani vývoj SW. Bohužel obor vývoje SW je zatím na začátku, tj. zatím vzniká spousta programových jazyků a stylů programování. S tím je spojeno jeho testování tj. kontrola kvality a odhalování chyb. Každý, kdo o automatizovaném testování slyšel, počítá s tím, že se použije nástroj, který vygeneruje všechny testy s kontrolními kombinacemi, a sám je otestuje včetně nahlášení chyb, apod. Toto je doslova zatím „scifi“ pro obor testování. Pokud stejným způsobem neumíme vytvořit specifikaci, návrh a program, nemůžeme ho ani takto otestovat.

Automatizované nástroje slouží k urychlení práce testera či jeho „zefektivnění“, nikdy ne k jeho nahrazení! [8] a [2]

Text je rozdělen do čtyř částí, první z nich se zabývá základními principy testování SW, druhá uvádí zmapování trhu nejznámějších automatizovaných nástrojů na vytvořeném testovaném programu, třetí část upozorňuje na výhody a nevýhody automatizovaného testování, čtvrtá část ukazuje možnost a problémy při tvorbě automatizovaných testů v komerčním a nekomerčním nástroji na praktických příkladech.

Seznam symbolů

Apache	webový server s podporou SSL ve verzích pro mnoho operačních systémů
CSS	Cascading Style Sheet – kaskádové styly umožňující pokročilé formátování dokumentu
GB	Gigabyte - jednotka určující množství dat v informatice
HTML	HyperText Markup Language je to značkovací jazyk pro hypertext
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization - Mezinárodní organizace pro normalizaci
klient – server	síťová architektura
MB	megabyte - jednotka určující množství dat v informatice
Mysql	MySQL je databázový systém, vytvořený švédskou firmou MySQL AB
PHP	Personal Home Page - skriptovací programovací jazyk
Python	Python je interpretovaný objektově orientovaný programovací jazyk
RAM	Random-access memory - paměť s libovolným přístupem
SOAP	Simple Object Access Protocol - protokol pro výměnu zpráv založených na XML
SW	Software
WSDL	Web Services Description Language
XML	Extensible Markup Language je obecný značkovací jazyk

1 Uvedení základních principů testování SW

Jakékoliv testování slouží ke zlepšení kvality výrobku před uvedením na trh. Stejné to je i u testů SW, kde testování slouží k ladění kódu, zjišťování funkčnosti systému dle zadání, k reportu chyb a předání patřičnému tvořiteli systému (tj. vývojovému týmu či danému vývojáři). [7] Testy jsou založeny na hledání a reportování chyb. Chyby mohou být v zadání, v analýze či v naprogramování systému. Žádný program nelze otestovat komplexně = nelze otestovat všechny varianty a určit, že systém je stoprocentně funkční, jelikož počet možných vstupů softwaru je veliký, počet možných výstupů je veliký, počet možných cest skrze aplikaci je veliký a specifikace aplikace je subjektivní. Vždy lze říci, že jediná chyba je jen v očích pozorovatele. Testování je tedy založené na riziku, netestují se vždy všechny kombinace. Tester musí být schopen vytvořit optimální počet testů tak, aby jich nebylo málo nebo moc. Testy musí být zvládnutelné do termínu, včetně předpokladu oprav systému. Testování nemůže nikdy ukázat, že chyby neexistují, protože pokud testování nemůže určit existenci chyb, nemůže určit ani opak. Pokud tester nalezne více chyb, než v předešlé verzi programu, tolikrát je více chyb v systému včetně nenalezených. Ne všechny chyby se vždy opraví, proto se musí určovat priority, které určuje opět tester. Specifikace nejsou nikdy konečné. Proto nemůže být žádný SW dokonalý. [8] Mezi základní principy testování SW patří norma ISO/IEC 9126 - 1, definice chyby, různé druhy testů a další. Níže uvedu ty nejpoužívanější dle mé praxe v oboru a zdroje [21], [22], [23].

1.1 Norma ISO/IEC 9126 - 1: 2001 [5]

Tento komplexní přístup k pojetí jakosti SW musí být akceptován i v oblasti programového vybavení pro řízení, kde se často dosud přihlíží pouze k hlediskům (charakteristikám) funkceschopnost a použitelnost, potom je již SW prodejný. [2] I proto se tato norma nedodržuje v komerčním využití SW.[6] Při jakékoliv metodice řízení SW projektu (dále jen projektu) je součást i testování SW, které je na konci celého cyklu. Pro testování už je většinou jen pilotní provoz a prodej / využití aplikace. Této normy či jejích obdob se snaží držet jen největší firmy na světě a to jen při výrobě prodávanějšího SW (např. firmy Microsoft, SAP, Oracle). Bohužel největší nedodržení se dělá ve firmách, které se nezabývají primárně prodejem a vývojem SW a svůj SW si vyrábí svépomocí či s pomocí dalších firem pro vlastní účely. Zde všechny metodiky selhávají, jelikož SW není primárně určen pro finální zisk, ale „jen k lepšímu využití firemních zdrojů“. [7] Selhávají i metodiky jako např.

RUP, SCRUM, Lean. To jen z důvodu nedostatku či špatně využitých finančních zdrojů určených k vytvoření SW. Čerpáno ze zdroje[7] a [2]

Norma ISO/IEC 9196-1 [5] nám popisuje/určuje tato kritéria:

- ***Funkceschopnost***

Schopnost obsahovat funkce, které zajišťují předpokládané nebo stanovené požadavky uživatele za stanovených podmínek používání. [5]

- ***Bezporuchovost***

Schopnost zachovat specifikovanou úroveň výkonu při používání produktu za stanovených podmínek. [5]

- ***Použitelnost***

Srozumitelnost produktu a jeho snadná obsluha určenými uživateli za stanovených podmínek. [5]

- ***Účinnost***

Schopnost poskytovat potřebný výkon vzhledem k množství použitých zdrojů za stanovených podmínek. [5]

- ***Udržovatelnost***

Schopnost produktu být modifikován s cílem nápravy nedostatků, vylepšování, adaptace a změn. [5]

- ***Přenositelnost***

Schopnost být přenesen do jiného prostředí s minimalizovanou námahou a obtížemi. [5]

Z toho vyplývají tyto druhy testů: funkceschopnost = funkční testy, použitelnost = uživatelské testy, účinnost = výkonové testy. Bezporuchovost a použitelnost jsou kritéria všech druhů testů SW (podobně jako počítání do deseti u matematiky). Z bezporuchovosti vznikl specifický druh testů = penetrační testy tj. „bezpečnostní testy“. Dále udržovatelnost a přenositelnost se zatím používají jako regresní testy, testy kompatibility, testy konfigurace a testy validity standardu. Tyto požadavky jsou důležité hlavně chceme-li testovat za pomocí automatizovaných testů. Bez nich je automatizace neefektivní. Při tvorbě systému zatím tyto dvě kritéria nejsou prioritou, jelikož životnost systému se zatím počítá na roky a ne na desítku let a více. [2], [9], [8], [7]

1.2 Chyba spojena s testováním SW

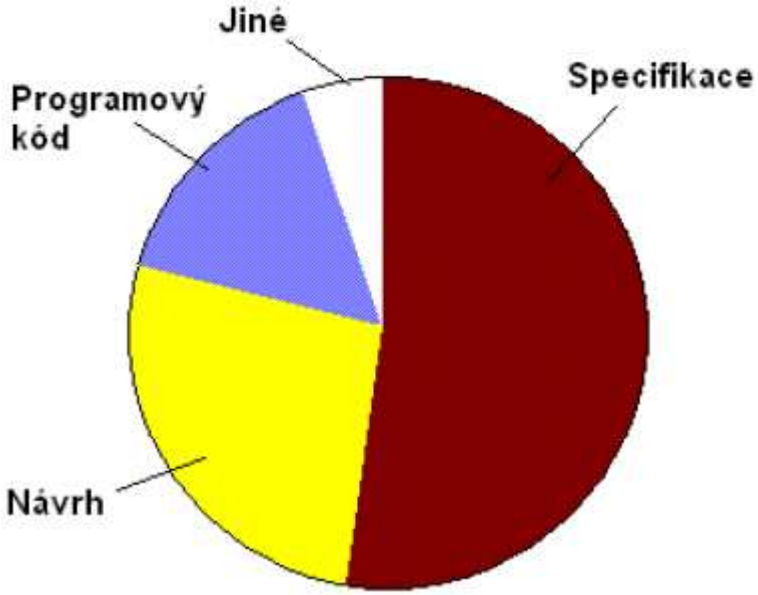
Je to pojem, na kterém je založena jakákoliv kontrola na světě, nejen testů SW. Při testu platí pravidlo, že nic na světě neexistuje stoprocentně tzn. bez chyb. Zde platí přísloví: žádný člověk není dokonalý, tudíž žádný SW nemůže být bez chyb. Každý, kdo se nezabývá SW,

zná chybu dělení nulou. Tato chyba je v dnešních systémech málo pravděpodobná (nachází se hlavně u systémů pro matematické operace). [8] U ostatních systémů ji nahrazuje chyba použití null či nedefinované hodnoty. [3] Než vysvětlím samotný pojem, uvedu, kde je největší vznik chyb v SW. Dle různých studií dle zdroje [8] i např. dle mé praxe se všichni shodneme, že největší vznik chyb je ve specifikaci. (V některých publikacích je udáno okolo 40-60% všech možných chyb – nalezených i nenalezených). Je to logické, výsledný SW produkt má být přizpůsoben i neprogramátorům („obyčejným lidem“). Tito lidé i píší prvotní specifikaci, nikdy nemohou znát všechny zákonitosti SW a zároveň trhu, který není jen pro IT společnost. [8] Například firma Microsoft toto zjistila, proto zadavateli začínají být i samotní SW inženýři, dokonce mají i program, kdy může každý z nich napsat svůj vlastní nápad / SW projekt. Nejlepší z nich jsou okomentovány samotným majitelem (B. Gatsem). Někdo může namítat, že jsou zde daleko kvalitnější produkty než od Microsoftu, ale tato firma se snaží SW přizpůsobit i pro běžné uživatele. [7] Toto například mnoho komunit odsuzuje (komunita UNIXu, linuxu apod.). Každému vyhovuje něco jiného, ale Microsoft chtě nechtě na tom vydělal nejvíce. Každý lidský pokrok vznikl pro lidské pohodlí a zisk. Při tvorbě SW tomu není jinak. [5]

Definice chyby od R. Pattona [8]:

„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."

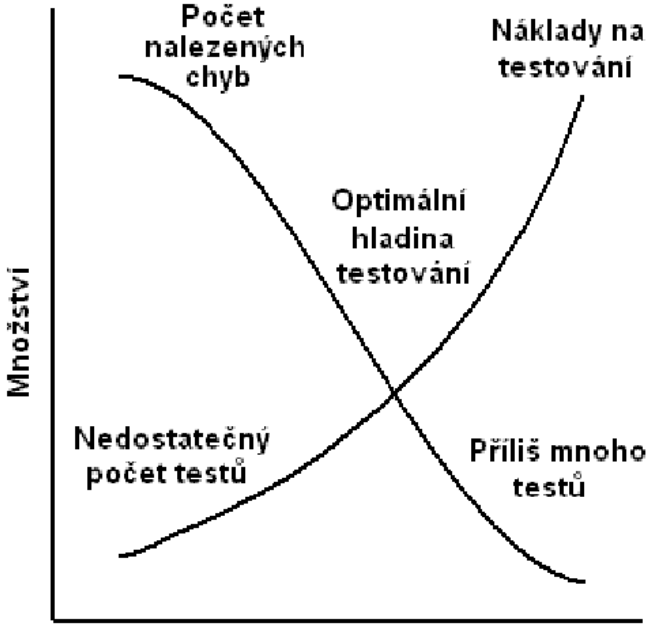
Kde chyby vznikají, je nejlépe popsáno v knize [8], kde autor také graficky zobrazuje příčiny vzniku chyb, viz obr.1. Graf je vytvořen studií mnoha různě velkých firem. [8]



Obr. 1 : Příčiny chyb od Pattona [8]

Dalším zajímavým faktem jsou náklady na chyby. Obecně [8] i prakticky [21], [22], [23] platí pravidlo, čím dříve se chyba od začátku projektu najde, tím je levnější její odstranění. [8] Toto pravidlo často používám v praxi při nalezení chyb ve specifikaci nebo analýze, pokud jim nikdo nevěnuje patřičnou pozornost. Zadavatel či pracovník pro tvorbu specifikace z 90% „nepřekousne“, že někdo mohl najít chybu v systému, který není ještě ani napsán. Např. naleznu chybu v analýze při její tvorbě či akceptaci. Sepíši tuto chybu i se spojením, kteří pracovníci různých systémů se musejí na určitý čas sejít a za další určitý čas opravit analýzu. Dohromady to dá např. x hodin – včetně mých a různých analytiků, což pro projektového managera převedu na průměrnou pracovní mzdu pracovníka a dodám dopady, které by se řešily při testech a opravou na vývoji + analýza. Stejným způsobem převedu na průměrnou mzdu všech zúčastněných. Z praxe jsem zjistil, že vždy dojdu k výsledku dvou až třikrát vyššímu než první výsledek pokud se řeší chyba při tvorbě analýzy. Pokud takto pošlu problém i s těmito dopady na pracovníka, který má na starosti řízení celého projektu, což je projektový manager, vždy mi zařídí opravu analýzy. Tento problém je pro většinu testerů neřešitelným, proto doporučuji postupovat dle výše popsaného popisu. Zadavatelé, designeři (neboli SW architekti) a projektoví manageři vždy reagují na cenu problému snáze než na oznámení problému. [2] a [4]

V dokumentacích o testech se vyskytuje zajímavý, ale v praxi nezajímavý graf jednající o optimální hladině testů. Hladina testů určitě existuje, ale ne vždy podle tohoto grafu. Shodují se zdroje [8], [2], [21], [22] a [23].



Obr. 2: Vztah nákladů testů dle [8]

Tento graf jsem uvedl z důvodu, že ne vše, co je v učebnicích, je dobře pochopeno. Bohužel spousta lidí, mezi nimi i ti, kteří tento graf zkopírovali do testování, nečekali, co tím způsobí. Pro testy je tento graf nesmyslný, jelikož každý testovací report musí mít reálný předpoklad se stejným výsledkem. [4] Pokud se zamyslíme, jak určíme před začátkem testů maximum nalezených chyb? A pokud zjistíme, jakou dobu potřebujeme k tomuto výsledku, když v této fázi už máme termín předurčen dříve a neměnný? [2] Při již započatých testech z vlastních zkušeností vím, že všichni tyto reporty upravují, aby se jim tento graf podařilo trochu napodobit. Nejlepším způsobem je říci, že tento graf o ničem nevypovídá, tudíž je zbytečné ho realizovat. Nejlepšími grafy jsou počet projetých testovacích scénářů vůči všem, které chceme provést, a dále počet nejzávažnějších chyb a jejich doba opravy s retestem. Pro testera je vždy nejhorší chyba, vůči které nelze testovat další kombinace. Díky těmto chybám se musí vždy upravit doba testů. Pokud nelze doba, tak výčet scénářů (což je v praxi častější případ. Je to další důvod proč testování je vždy s určitým rizikem a ne stoprocentní).[8], [2]

1.3 Dělení testů SW podle znalosti testovaného systému

Toto dělení vzniklo z praktických zkušeností testování SW. Je založené na znalosti testované aplikace.

- ***Testování aplikace jako černé skříňky***

Pokud se v systému zaměřujeme na vstupy a výstupy dané aplikace bez znalostí algoritmů v pozadí programu – tj. bez znalosti implementace programu. Smyslem je testovat chování SW vzhledem k očekávaným vlastnostem tak, jak ho vidí uživatel. Lze tedy testovat až po finálním vytvoření celé aplikace či daného vstupu a výstupu. [9]

- ***Testování aplikace jako bílé skříňky***

Při testování bílé skříňky má tester přístup ke zdrojovému kódu a testuje produkt na základě něj. Vidí nejen co se děje na povrchu SW, ale i vnitřní zpracování systému. Tím se ztrácí pohled uživatele, ale může lépe odhadnout, kde hledat chyby a kde ne. [9]

- ***Testování aplikace jako šedé skříňky***

Mezi těmito kategoriemi vznikla ještě třetí, testování šedé skříňky, kdy sice máme informace o vnitřních algoritmech produktu, ale ne takové, aby to bylo považováno za testování bílé skříňky. Například nemáme k dispozici celý zdrojový kód, ale jen informace o principech použitých v aplikaci. [9]

1.4 Dělení testů SW statické x dynamické

- ***Statické testování***

Statické testování jsou testy všech kontrol, pro které nepotřebujeme aplikaci spustit, tj. např. kontrola dokumentace, plánů, datových typů, číselníku v programu apod. [8]

- ***Dynamické testování***

Dynamické testování vyžaduje existenci spustitelné verze SW, probíhá porovnání vstupu s výstupy testovaného SW. [8]

1.5 Dělení testů dle počtu nehomogenních celků

Toto dělení je jedno z nejdůležitějších pro testy a automatizaci testů. Dnes většinou existují již jen systémy s více nehomogenními celky. Někoho může napadnout: tak budeme testovat jen integračně. Ale pokud lze nehomogenní celek nějakým způsobem otestovat samostatně, hodí se nám tyto testy provést vždy před nebo i při testech integrace či automatizaci testů. Dle čeho se dá toto dělení zjistit na našem systému? Nejjednodušší je zjistit, kde je v SW použit nějaký „interface“. To lze zjistit jen jste-li IT specialista a znáte systém alespoň na úrovni šedé skříňky. [9]

- ***Systémové testování***

Systémové testování jsou testy systému složené z jedné nehomogenní části subsystému.

[8888]

- ***Integrační testování***

Každý komplexní systém se skládá z množin nehomogenních subsystémů, které jsou spojeny vzájemnou komunikací (tj. spojeny „interface systémem“, nejpoužívanější jsou např. webové služby). [8]

1.6 Další druhy testů

Některé testy vznikly pro testy částí systémů, které nejsou primárně určené koncovému uživateli, či jsou součástí předešlých dělení (nebo detailem). Např. zkoumání specifikace, testování dokumentace, testy cizích jazyků, testy migrací dat apod. [2] Z těchto dělení vyplývá, že jakékoliv testování slouží ke zlepšení kvality výrobku před uvedením na trh. Stejné to je u testů SW, kde testování slouží k ladění kódu, zjišťování funkčnosti systému dle zadání, k reportu chyb a předání patřičnému řešiteli a managementu opravy chyb. [8]

1.7 Krátký přehled pracovního oboru SW testera

Od počátku testování SW prošlo výraznými změnami. Dříve se práce testera striktně dělila na více rolí. Testeři pracovali pod jednotlivými rolemi, jako jsou např. test analytik, test programátor, test architekt, tester apod. V té době firmy o testování moc nevěděly a raději si tyto pracovníky najímaly z různých specializovaných firem určených pro testování SW či si nechávaly od nich poradit, jak testovat. Dokonce i z praxe mohu usoudit a s mnoha kolegy říci, že tester může na jednom systému pracovat v kuse maximálně dva až tři roky, po delší době se dopouští více lidských chyb. Jelikož pro testování softwaru je potřeba člověka především flexibilního a specializovaného na testování, což ve většině firem nelze z důvodu pevné pracovní doby. V takovém případě byli a jsou firmami najímány potřebné lidské zdroje primárně většinou externě (tzv. outsourcingem). [2]

V průběhu vývoje SW se projevila změna: Firmy si přestávají jen najímat pracovníky na tyto pozice. A z nasbíraných zkušeností se snaží tyto pracovní pozice začlenit do svých firem za pomocí vlastních lidských zdrojů. Což byla vždy převážně špatná cesta u firem, které vytváří SW jen pro své účely (tzv. neprodávají SW). Ztrácí se zde pak různorodost pohledů, poučení od jiných společností či projektů tvorby SW. Proto dochází k reorganizaci pracovních pozic testovacího týmu, spojují se v menší celky. Člověk, který dříve zastával pouze jednu pozici, nyní musí být schopen zastat více rolí, např. tester, senior tester, které jsou vymezované dosaženými znalostmi. Požadavky na pracovní pozici testera jsou tudíž dnes

daleko vyšší než dříve. Obdobné zdroje [8], [10], [2] a [7]. Tester by měl mít tyto vlastnosti:

1) Umět se ptát (účelně a hlavně podávat otázky tak, aby se dostala jednoznačná odpověď).

2) Flexibilnost (zapomeňte na osmihodinovou pracovní dobu).

3) Umět se na daný problém podívat ze strany zadavatele, analytika, vývojáře a uživatele.

4) Umět číst dokumentaci a pochopit ji, dokázat ji odprezentovat ostatním rolím na projektu.

5) Samozřejmostí jsou základy programování v nejširším rozsahu. (tj. tester musí umět základy, syntaxi, datové modely technologií použitých v testovaném softwaru). Tento bod je nereálný, ale splňuje se tím, že tester je věčným studentem použitých technologií.

6) V důsledku předchozích bodů určovat priority a provádět testy.

[11]

2 Upozornění na výhody a nevýhody automatizovaného testování

Výhody a nevýhody automatizovaného testování vychází z manuálního a automatizovaného testu. Každého, kdo slyšel o manuálních testech, napadne: „Proč netestují stroje („kontrolní SW“)? Je to jednoduché. Člověk se snaží automatizovat cokoliv pro svoji pohodlnost. Bohužel testování je logicky až na konci časové osy tvorby SW produktu = nikdy ne na začátku. Proto je závislé na všech ostatních cyklech časové osy projektu, ale ne všechny cykly projektu jsou závislé na testování. [3] Manuální testování vzniklo z obyčejného vyzkoušení SW před uvedením na trh nebo do produkce. Tudíž manuální testy vznikly především z testů černé skříňky. [1] Jak jsem již zmínil, testy nemohou pokrýt všechny kombinace produktu. Proto se začaly používat testovací nástroje, které testerům slouží jako pomoc při testech. Bohužel nelze nahradit testovacím nástrojem testera, jen mu může pomoci při dynamickém testování a v něm jen se simulací programu, určením předběžného místa chyby a nahlášením defektu. [8] Bohužel v komerčním využití se setkávají testeři s problémy ostatních týmů použitých při vývoji systému. [2] Z praktických zkušeností mohu potvrdit, že činnost testera, simulace různých kombinací a report chyby při těchto testech zaberou 5 až 10% procent celkové práce testera. Automatizované testy mohou občas zajistit další ušetření času testera, jako je např. prvotní tvorba testovacích matic, tvorba testovacích dat a hlavně regresní testy. [7] Při manuálním testování se nejčastěji promítají dva faktory při simulaci. Zaprvé tester dělá chyby především při opakování stejné činnosti. Zadruhé, pokud testuje jednu funkčnost a systém dle bílé skříňky delší dobu, změní se mu pohled na systém v pohled programátora a ztrácí uživatelskou schopnost testera. [2]

Nejdůležitější přívlastky testovacích nástrojů pro automatizované testy jsou dle R. Pattona [8]:

- ***Rychlost***

Jsou rychlejší než manuální zkoušení.

- ***Efektivita***

Zkracují čas při simulaci.

- ***Přesnější opakovatelnost***
- ***Neúnavnost***

[8]

Bohužel testovací nástroje mají převážně více nevýhod, tudíž se moc nepoužívají. Mezi nejznámější nevýhody patří:

- ***Cena***

Komerční nástroje jsou určené pro menší počet technologií použitých při vývoji SW a vychází v rozmezí deset tisíc a jeden milion eur. [23]

U nekomerčních nástrojů je nepředstavitelně drahý specialista a delší doba vytvoření automatizovaného testu. Rentabilita nástroje se projeví až v delším časovém horizontu. [3]

- ***Chybovost***

Předpoklad dle Pattona [8]: *„Žádný SW není bez chyby.“* Tudíž ani samotný automatizovaný nástroj či operační systém. [2]

- ***Tvorba SW***

Zdrojový kód automatizovaného testu vyžaduje stejnou údržbu jako testovaný SW. [8]

- ***Technologie produktu***

Nástroje a testy jsou závislé na použité technologii produktu. [3]

- ***Komplikovaná přizpůsobivost***

Automatizovaný test je nutno neustále měnit a přizpůsobovat testovanému SW. [3]

- ***Lidskost***

Testerovu intuici a zrak ničím nenahradíte. Neexistuje nástroj, který by dokázal generovat zadání a potřebu člověka, proto nemusí vždy vše přesně zkontrolovat. [8]

- ***Verifikace***

Verifikace je obtížná. Spočívá v porovnávání, které je u webových aplikací obtížné, protože vzhled webových aplikací se často mění. Může dojít i ke kontrole toho, co se nevyžaduje. [8]

Dle Pattona [8] i [2] automatický test lze použít téměř vždy, ale nikdy nemůže zcela nahradit manuální testy. Tyto a podobné výhody a nevýhody budeme ukazovat na praktickém příkladě, kdy použijeme pro tvorbu funkčního testu komerční a nekomerční nástroj.

3 Zmapování trhu s nástroji na podporu automatizovaného testování

Při výběru testovacího nástroje bude na prvním místě vždy cena. Bohužel cena samotné automatizace testů se neodvíjí jen z ceny testovacího nástroje. (Je to prvotní mylný předpoklad, kdy si zákazník myslí, že si koupí či stáhne testovací nástroj, a ten udělá vše za něj.) Nejdražší je specialista, který bude daný automatizovaný kód psát. Specialisté, kteří dovedou psát v opensource nástroji jsou vždy dražší, než specialisté zaměřeni na jeden konkrétní nástroj. [3] a [8]

Po ceně je na řadě technologie testované aplikace. To je např. programovací jazyk, použité knihovny (i verze), jaké standardy technologií se použijí, operační systém nebo i internetový prohlížeč, ve kterém se bude webová aplikace používat. [3]

Dále je to programovací skript / jazyk, který se může použít při tvorbě automatizovaného testu. Je důležitější u opensource, kdy nemusíte být závislí jen na knihovnách daného testovacího nástroje. [3]

Pro velké firmy je především důležitá uživatelská podpora a samotná podpora testovacího nástroje. Ne vždy jsme si schopni poradit sami nebo samozřejmě může být chyba v samotném testovacím nástroji. Například u komerčních licencí si i tuto podporu lze koupit, někdy je i samozřejmostí. [3] Ale i volně šiřitelné verze mohou mít výbornou dokumentaci (příkladem může být nástroj SoapUI pro testy provolání webových služeb, kdy dokumentace je kompletně na internetu i s dokumentací komerční verze Soap Pro. Novinky v aplikacích jsou většinou placené, jelikož nástroj slouží i jako vývojový nástroj). [13a]

Dále si je potřeba při výběru testovacího nástroje uvědomit i tyto důležité faktory: kolik testů a jejich opakování se při tvorbě SW předpokládá? Počítá se s časem skriptování automatizovaných testů? [7]

Automatizované testy se používají v komerčním využití převážně k regresním testům. Test by měl být použitelný i po následujících cílených změnách. [7] Špatně se automatizují testy aplikací, které využívají komerční či nestandardní technologie např. SAP, Siebel. U těchto technologií lze použít jen rozšíření komerčně licencovaných nástrojů. [14], [15], [16], [17] Je nutné si také uvědomit, jakým nástrojem se reportují chyby. Komerční výrobci [14], [15], [16], [17] nabízí i „reportovací“ nástroje, které lze s automatizovaným nástrojem propojit.

Na trhu existuje řada komerčních i nekomerčních nástrojů pro automatizované testy funkčních testů. Představím parametry několika nejpoužívanějších na trhu v ČR, které se dají použít k uvedené webové aplikaci. Nejpoužívanější jsou odvozeny od zdrojů [21], [22], [23].

V parametrech nejsou uvedeny ceny, jelikož výrobci dávají různé ceny licencí různým uživatelům i s různou podporou (support). Ceny se pohybují v rozmezí 10 000 – 300 000 eur. [23] Od popisu výrobců, jsem zjistil jednu zajímavost týkající se podporovaných technologií. Firmy se předhánějí v podporovaných technologiích, tj. v jakých technologiích je testovaná aplikace napsána. Proto je vždy nejlepší zeptat se firmy, jestli danou technologii podporuje a pokud ano, tak každý solidní výrobce nabízí plné zkušební verze na vyzkoušení a potvrzení jejich údajů. [14], [15], [16], [17]

3.1 Testovaná aplikace a její technologie

Před tím, než popíši testovací nástroj, si musím uvědomit, co chci testovat. [8] Pro moji praktickou ukázku automatizace testů a výběru testovacího nástroje na trhu, jsem vytvořil tuto webovou aplikaci. S pomocí technologie PHP, HTML, CSS, webové služby (s využití technologie SOAP) a databáze Mysql. Aplikaci jsem napsal pomocí všech zdrojů mé práce. Aplikace běží na triadě Xampp [xamp.com]. Xampp je druh aplikace, ve které běží webový server Apache a databáze Mysql. Díky této technologii stačí Xampp s vytvořeným programem nahrát na disk počítače. [12]

Aplikace má uživatelské webové rozhraní pro internetový prohlížeč. Účel aplikace je tzv. kniha návštěv, kdy s pomocí formulářových polí plním vstupní údaje. Tyto údaje se uloží do databáze pomocí webové služby. Dále je umožněno zobrazení již podaných zpráv pod formulář, které lze i smazat z databáze, opět za pomocí webových služeb. Program je vytvořen pomocí QuickTest Professional 10.00 System.[18]

3.2 QuickTest Professional 10.00 System

Program QuickTest Professional 10.00 System (dále jen QuickTest) je vyroben od výrobce Hewlett-Packard Company. Nástroj slouží k automatizaci funkčních testů formulářových a webových aplikací. Požadavky a parametry samotného nástroje podané od výrobce [16]:

- ***Požadavky na CPU:***

Pentium III a vyšší.

- ***Operační systém, ve kterém lze spustit nástroj:***

Windows 2000 Professional
Service Pack 4
Update Rollup 1 for Windows 2000 Service Pack 4
Windows XP Professional 32-Bit Edition, Service Pack 2 or Service Pack 3
Windows XP Professional 64-Bit Edition, Service Pack 2
Windows Server 2003 32-Bit Edition, Service Pack 2
Windows Server 2003 R2 (32-Bit x86)
Windows Vista 32-Bit Edition or Windows Vista 32-bit Edition, Service Pack 1
Windows Vista 64-Bit Edition or Windows Vista 64-bit Edition, Service Pack 1
Windows Server 2008 32-Bit Edition
Windows Server 2008 64-Bit Edition
Quality Center klient nepodporuje 64-bitové operační systémy, tudíž pro tyto operační systémy nefunguje spojení s produktem od HP Quality Center. Quality Center je systém sloužící pro řízení projektu od psaní dokumentace specifikace a testovacích scénářů. Hlavně se používá k reportům již provedených scénářů a hlášení chyb dle libovolné metodiky.

Tabulka 1: Podporované Operační systémy pro QuickTest

- ***Požadavky na RAM paměť':***

Minimální kapacitu 512 MB. Minimálně 1 GB při využití možnosti nahrávat filmy v průběhu běhu programu.

- ***Grafická karta:***

Grafická karta s 4 MB paměti video (8 MB a výše doporučeno). Minimální High Color (16 bit).

- ***Volné místo na disku, na kterém poběží testovací nástroj:***

650 MB volného místa na disku pro aplikaci, když jsou instalovány pouze výchozí add-ins. 800 MB (doporučeno 1 GB), pokud jsou nainstalovány všechny add-ins je potřeba dalších 120 MB volného místa. Pro správný chod doporučuje mít výrobce dalších 150 MB volného místa pro chod Operačního systému a QuickTestu.

- ***Podporované prohlížeče pro testované webové aplikace:***

Výrobce nabízí Microsoft Internet Explorer6.0 Service Pack 1, Microsoft Internet Explorer7.0 a Microsoft Internet Explorer8.0 Beta 2. V diskuzi od HP je možné použít i Firefox 3.0.X Pro dalších pár prohlížečů lze využít starší verze programu QuickTest - např.

verze 9.2 se může použít pro webový prohlížeč Microsoft Internet Explorer 6.0 Service Pack 1 or Microsoft Internet Explorer 7.0. (required), Netscape Browser 8.1.2 , Mozilla Firefox 1.5 and 2.0 a AOL 8.0 and 9.0 (optional).

[16]

3.3 TestComplete 7

Parametry programu TestComplete 7 (dále jen TestComplete) pro automatizaci funkčních testů od výrobce AutomatedQA Corporation.

- ***Požadavky na CPU:***

Intel Pentium II 400 MHz nebo vyšší (uvedeno pro Operační systém Microsoft Windows 2000, jelikož dle výrobce se zvýší verzí OS požadavky liší).

- ***Operační systém, ve kterém lze spustit nástroj:***

Microsoft Windows 7 (oba 32-bit a 64-bitové verze).
Microsoft Windows Vista (32-bit a 64-bitové verze).
Microsoft Windows Server 2008 (oba 32-bit a 64-bitové verze).
Microsoft Windows XP (32-bit a 64-bitové verze).
Microsoft Windows Server 2003 (oba 32-bit a 64-bitové verze).
Microsoft Windows 2000.

Tabulka 2: Podporované Operační systémy pro TestComplete

- ***Požadavky na RAM paměť:***

256 MB paměti RAM (uvedeno pro Operační systém Microsoft Windows 2000, jelikož dle výrobce se s vyšší verzí OS požadavky liší).

- ***Grafickou kartu výrobce neuvádí***

Dle výrobce má stačit grafická karta podporující režim SVGA (800 × 600) nebo vyšší rozlišení (uvedeno pro Operační systém Microsoft Windows 2000, jelikož dle výrobce se s vyšší verzí OS požadavky liší).

- ***Volné místo na disku, na kterém poběží testovací nástroj:***

460 MB místa na pevném disku (uvedeno pro Operační systém Microsoft Windows 2000, jelikož dle výrobce se s vyšší verzí OS požadavky liší).

[16]

- ***Podporované prohlížeče pro testované webové aplikace:***

Internet Explorer podle verze testovacího nástroje. Firefox do verze 3 [15]

3.4 SilkTest

[17]

Parametry programu SilkTest pro automatizaci testů pro funkční testy od výrobce BORLAND.

- ***Požadavky na CPU:***

Intel Pentium III a vyšší.

- ***Operační systém, ve kterém lze spustit nástroj:***

Windows NT4 Service Pack 6a.
Windows 2000.
Windows XP.
Windows 2003.
Vista.
Windows 7.
Windows Server 2008.

Tabulka 3: Podporované Operační systémy pro SilkTest

- ***Požadavky na RAM paměť:***

Minimálně 256 MB RAM.

- ***Grafickou kartu výrobce neuvádí.***
- ***Potřebné místo na disku výrobce neudává.***
- ***Podporované prohlížeče pro testované webové aplikace:***

Internet Explorer 5.0 a novější verze.

[17]

3.5 Rational Robot

[14]

- ***Požadavky na CPU:***

Minimum 1.5 GHz Intel(R).

- ***Operační systém, ve kterém lze spustit nástroj:***

Windows 2000.
Windows 2003.
Windows 98.
Windows Me.
Windows NT.
Windows XP.

Tabulka 4: Podporované Operační systemy pro Rational Robot

- ***Požadavky na RAM paměť':***

1 GB RAM.

- ***Grafická rozlišení nebo grafická karta: výrobce neudává.***
- ***Volné místo na disku pro instalaci, na kterém poběží testovací nástroj:***

200 MB.

- ***Podporované prohlížeče pro testované webové aplikace:***

Internet Explorer 4.x, 5.x a 6.0 nebo Netscape.

[14]

4 Ukázka možnosti automatizovaného testování aplikací na praktickém příkladu

Pro tuto ukázku jsem si vybral nástroj open source Selenium [20] s kombinací knihoven skriptovacího jazyka Python pro funkční testy přes GUI aplikace, systémové testy interface webových služeb SoapUI [13]. Dále z komerčních nástrojů QuickTest + addon pro testy webových služeb. [16] Tyto nástroje jsem si vybral především z důvodu, že patří mezi nejznámější v oboru. Dle popisu výrobců mají být pro podobné aplikace doporučovány.

4.1 Vytvoření testované webové aplikace

4.1.1 Popis GUI webové aplikace

Webová aplikace se skládá z vstupního formuláře: Jméno, Příjmení, E-mail, Předmět, Zpráva a Tlačítko „Přidat zprávu“.

Obr. 3:Náhled na GUI testované aplikace

Výstup: po vložení údajů do formuláře a potvrzením pomocí tlačítka „Přidat zprávu“ se promítnou údaje pod formulář, kde je i možnost smazat jednotlivý záznam.

4.1.2 Popis databáze

Aplikace ukládá data do databáze Mysql, kterou lze vytvořit přes soubor install.php. Pokud se spustí cesta k souboru přes webový prohlížeč, vytvoří se tabulky v databázi. Je zde i

ošetření opětovného spuštění, tabulka se nejprve smaže a následně se vytvoří. Popis databáze vychází ze souboru install.php, který lze spustit přes webový prohlížeč (viz. seznam souborů na přiloženém CD).

```
<?

require "./lib/sql.php";

$db = "guestbook";
$remove_db = true;
$create_db = false;

try {
    $sql = new SQL();

    if ($create_db) {
        $dbs = $sql->listDBS();
        while ($row = mysql_fetch_object($dbs)) {
            if ($row->Database == $db) {
                if ($remove_db) {
                    $sql->query("DROP DATABASE " . $db);
                }
                else {
                    throw new Exception("database $db exists");
                }
            }
        }
        $sql->query("CREATE DATABASE " . $db);
    }

    $sql->selectDB($db);

    $sql->query("

        CREATE TABLE `guestbook` (
            `id` VARCHAR(256) NOT NULL,
            `date` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
            `firstName` VARCHAR(128) NOT NULL,
            `lastName` VARCHAR(128) NOT NULL,
            `email` VARCHAR(128) NOT NULL,
            `subject` VARCHAR(256) NOT NULL,
            `message` TEXT NOT NULL,
            PRIMARY KEY (`id`)
        ) COLLATE utf8_czech_ci

    ");
}
catch (Exception $e) {
    echo "Caught exception: ",  $e->getMessage(), "\n";
}

?>
```

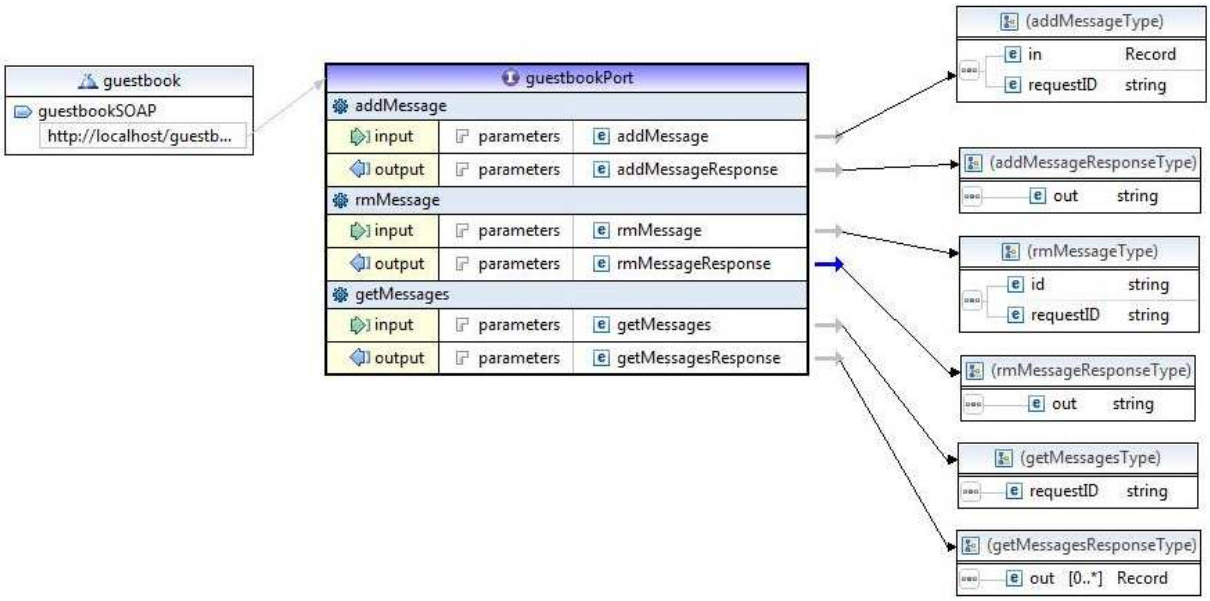
Obr. 4: Náhled do souboru install.php 1

id	firstName	lastName	email	subject	message

Tabulka 5: Vzhled databázové tabulky

4.1.3 Popis webových služeb

Vytvořená aplikace komunikuje s databázovou tabulkou přes webové služby s jedním WSDL. Webové služby jsou vytvořeny pomocí programu Eclips. [19] Nástroj Eclips umožňuje i technický náhled na WSDL webových služeb.



Obr. 5: Model webové služby 1

4.1.4 Test v Selenium

Nejprve jsem musel prostudovat možnosti knihoven [20]. Dále jsem si vybral verzi pro podporu skriptovacího jazyka Python 2.6.5 (dále jen Python). Pro načtení knihoven jsem použil skript, viz obr. 6.

```
import unittest
#import time
#import re
import urllib
import codecs
import ConfigParser
from lxml import etree
from StringIO import StringIO
from selenium import selenium
```

Obr. 6: Vstupní knihovny testu

Jelikož mnoho částí skriptu testu bude mít dost společných vlastností kódu, vytvořil jsem proto třídu, která bude obsahovat kód společný pro další potomky této třídy. Na obrázku 7. je ukázka třídy class Template(unittest.TestCase). Třída obsahuje vytvoření objektu RawConfigParser() a načtení konfigurace z *.ini souboru do proměnných. Určité údaje je potřeba konfigurovat na určité prostředí a cesty testovaného programu. Proto je načítám

z fyzického souboru. Ve třídě je potřeba si uvědomit, že se používají jiné datové typy pro načítání řetězců, čísel a boolean. Pokud by načítaná hodnota neodpovídala předpokládanému datovému typu, objeví se výjimka. Toto jsem odladil při opakovaném spuštění samotného testu. [20]

```
class Template(unittest.TestCase):
    screenshot_count = 0
    screenshot_preffix = ""
    def setUp(self):
        rcp = ConfigParser.RawConfigParser()
        rcp.read("selenium-tests.ini")
        self.service_log = rcp.get("Service", "log")
        self.host = rcp.get("Selenium", "host")
        self.port = rcp.getint("Selenium", "port")
        self.browserStartCmd = rcp.get("Selenium", "browserStartCmd")
        self.browserURL =  rcp.get("Selenium", "browserURL")
        self.speed = rcp.getint("Selenium", "speed")
        self.wait_for_page_to_load = rcp.getint("Selenium", "wait_for_page_to_load")
        self.window_maximize = rcp.getboolean("Selenium", "window_maximize")
        self.screenshot_enable = rcp.getboolean("Screenshot", "enable")
        self.screenshot_output = rcp.get("Screenshot", "output")
        self.screenshot_suffix = rcp.get("Screenshot", "suffix")
        self.add_message_csv = rcp.get("Input", "add_message")

        self.verificationErrors = []

        self.selenium = selenium(self.host, self.port, self.browserStartCmd, self.browserURL)
        self.selenium.start()
        self.selenium.set_speed(self.speed)

    def go_home(self):
        if (self.window_maximize):
            self.selenium.window_maximize()

        self.selenium.open(self.browserURL)
        self.selenium.click("link=guestbook-client.php")
        self.selenium.wait_for_page_to_load(self.wait_for_page_to_load)
```

obr. 7: Ukázka skriptu třídy template (unittest.TestCase)

Pozor jsem si musel dávat na řetězcové hodnoty v konfiguračním souboru, které se nesmí uzavírat do uvozovek, protože by se staly součástí řetězce.

```
[Service]
log = http://localhost/guestbook/guestbook-service.log

[Selenium]
host = localhost
port = 4444
browserStartCmd = *firefox
browserURL = http://localhost/guestbook
speed = 2000
wait_for_page_to_load = 30000
window_maximize = true

[Screenshot]
enable = true
output = D:/test_selenium/screenshots/screenshot
suffix = .png

[Input]
add_message = D:/test_selenium/add_message.csv
```

Obr. 8: Náhled do souboru selenium-tests.ini

Skript obsahuje pořizování tisk obrazovek (dále jen screenshot). Pokud je v selenium-test.ini souboru aktivováno pořizování screenshotu, tak se zvedne čítač o jedničku, vygeneruje název výstupního souboru a provede screenshot (bohužel tyto metody podporuje jen internetový prohlížeč Firefox). Proto je tuto funkčnost potřeba zapnout v souboru selenium-tests.ini. Tyto tisky obrazovek se používají ke zpětnému dohledání vzhledu obrazovek po provedení testu. Nezáleží na tom, jestli test proběhne bez chyby nebo s chybou, screenshot se vždy vytvoří. [20]

Dále jsem musel vytvořit objekt HTMLParser() pro detekci objektů v testované aplikaci. Skript vyhledává předdefinované objekty, které se nachází na zobrazené stránce, za pomoci zdrojového kódu stránky v prohlížeči. Tato část kódu je použita k dohledání objektů v logu a samotné aplikaci. Souběžně porovnává objekty v aplikaci s logem. Pokud se tato činnost nepodaří, test skončí s chybou.

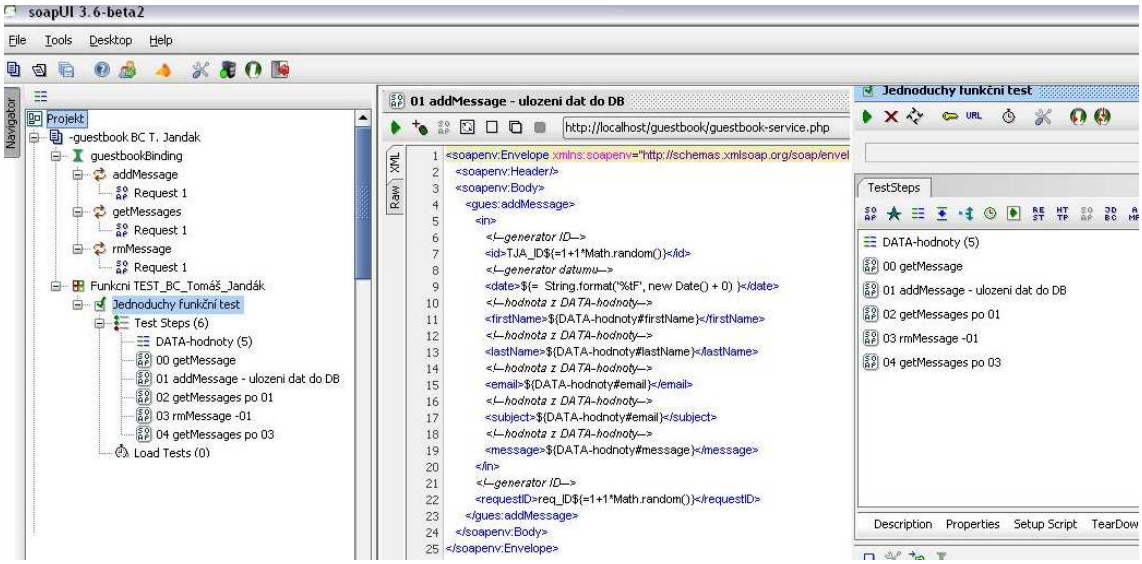
Dále je skript vybaven vstupní plnící maticí dat, která se plní do formuláře aplikace. Tímto lze vytvořit různé kombinace testu na základě různých vstupních dat. Výsledkem testu je doba provedeného testu, správnost či chyba a tisk obrazovek. Dále testovací nástroj umožňuje instalaci doplňku Firefox, pomocí kterého se spustí i grafický průběh testu. (Celý použitý skript s verzí Selenia je v Příloze.)

[20]

4.1.5 Unit test webové služby v SoapUI

Při testu v Seleniu jsem si všiml, že by bylo dobré otestovat před spuštěním velké vstupní matice a ověřit funkčnost webové služby. Pro tuto činnost jsem použil další nekomerční

nástroj SoapUI, které je open source pro testy webových služeb. Pro tento test do nástroje SoapUI postačí zadat cestu k danému WSDL webové služby a spustit generování provolání se základním testem. Potom stačí jen doplnit alespoň jednu vstupní kombinaci pro webové služby a spustit test. Test nám ověří validitu xml a zda se vrátila nějaká odpověď. [13]



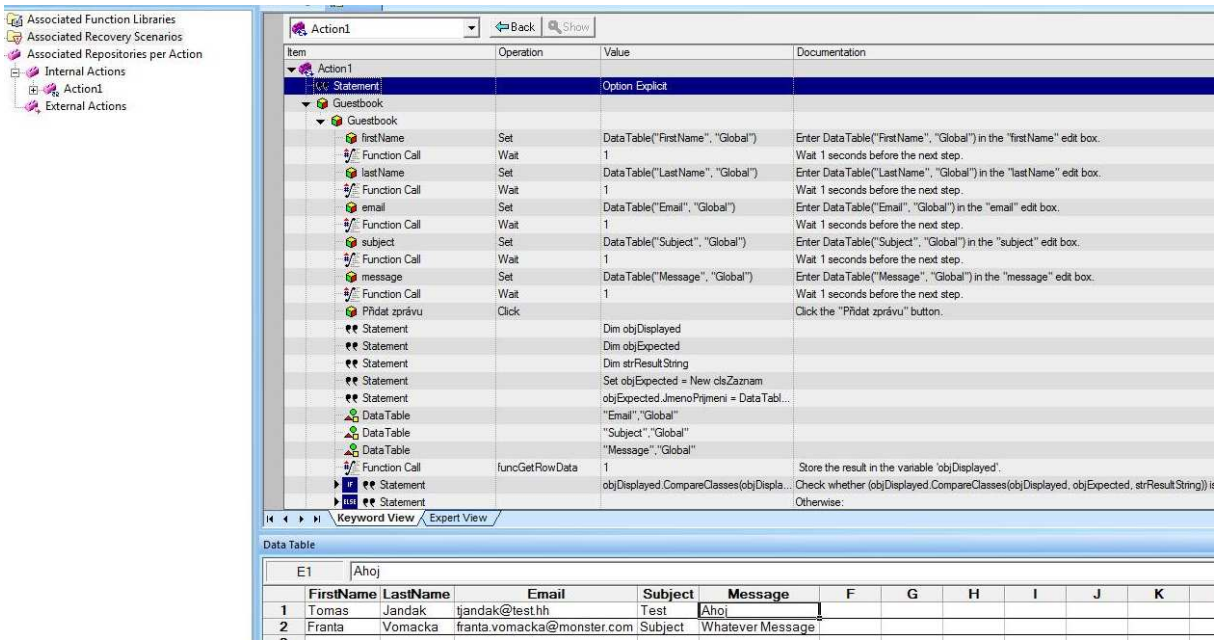
Obr.9: Ukázka testu webové služby v SoapUI 1

4.1.6 Automatizovaný test webové služby v QuickTestu

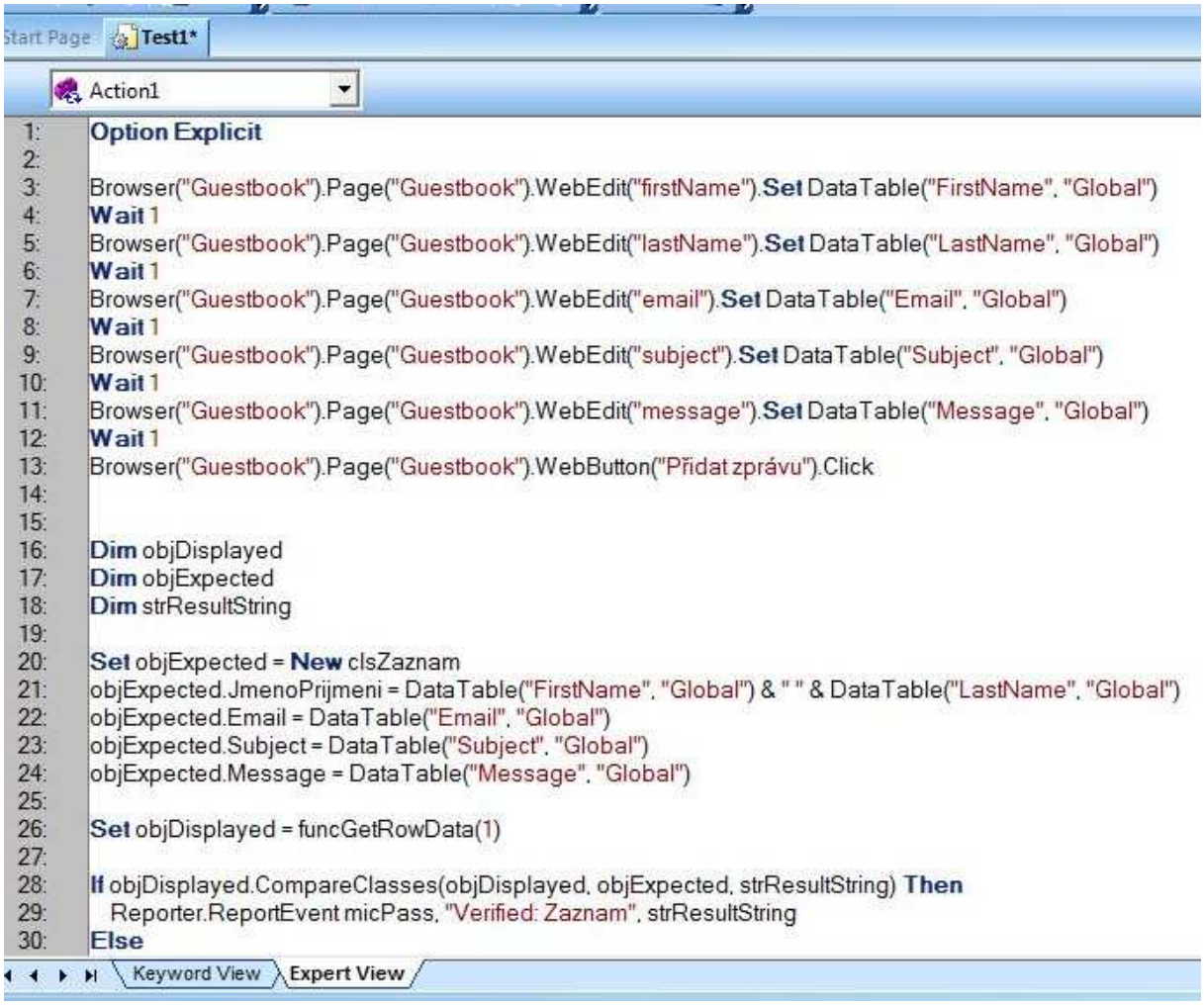
Po přečtení dokumentace obsažené v helpu programu a od samotného výrobce jsem vytvořil automatizovaný test. [16] Pro praktickou ukázku jsem použil Trial verzi programu.

Nejprve jsem využil přednost tohoto nástroje a to automatické generování skriptu pro naplnění webového formuláře testované aplikace, dále jsem využil vstupní matici určenou pro naplnění těchto hodnot (matice musí být uložena v testu, ale lze ji naimportovat např. z Excelu). Chtěl jsem pokračovat vytvořením modulární prázdné třídy, bohužel samotný testovací nástroj měl na tuto funkčnost chybu a dle výrobce se čeká na opravu. Tento problém jsem vyřešil vytvořením automatické třídy libovolného objektu na testované aplikaci a potom tuto třídu přepsal.

Testovací nástroj umožňuje skriptovací jazyk Vbskript, ve kterém mohu samostatně vytvořit objekty nebo již vygenerované objekty upravovat. Následující obrázky ukazují náhled na testovací nástroj a testovací skript.



Obr. 10: Náhled na kroky QuickTestu



Obr. 11 : Náhled na skript v QuickTestu 1

Výsledkem funkčního testu je test naplnění různých hodnot do vstupních “políček”, identifikace objektů v aplikaci, jako například zobrazení hodnot na formuláři po vložení do formuláře a porovnávání se vstupními daty.

Dále jsem zjistil, že samotný nástroj umožňuje stažení dalších vlastních modulů jako například testy webových služeb. Proto jsem vytvořil i test webové služby, který provolává webovou službu a kontroluje výstupní hodnoty podle předdefinovaných hodnot. Webové služby jsem v testu seřadil za sebe jako v aplikaci. Tudíž test nejprve provolá webovou službu pro naplnění dat do databáze a zkontroluje odpověď podle předdefinované hodnoty.

Další v pořadí jsem vložil provolání webové služby pro čtení údajů, které se porovnávaly s údaji ze vstupu předešlé webové služby.

Poslední část testu provolává webovou službu pro mazání a ověřuje odpověď dle předdefinované hodnoty. Pokud test spustíme cyklicky, to jest s více druhy naplnění, potom jsme schopni pomocí tohoto testu zkontrolovat, zda se data z databáze doopravdy smazala.

Další obrázky testovacího nástroje, i například výsledek testu, je v Příloze D.

[16]

5 Závěr

Z praktického příkladu mohu potvrdit, že automatizovaný test lze provést v komerčním i nekomerčním nástroji. Samotná tvorba testovacího skriptu je však v každém z těchto dvou typů nástrojů odlišná. Rozdíly jsou shrnuty v následujících bodech.

Čas strávený na vytvoření testu v nekomerčním nástroji je daleko větší, než čas strávený při tvorbě v komerčním nástroji. Pokud tento čas převedeme na finance, tak při větším počtu vytváření automatizovaných testů porostou. Tudíž pro větší SW nebo delší vývoj je i ekonomicky drahé použít nekomerční nástroje pro automatizované testy. Dále i programátorské zkušenosti testera v nekomerčním nástroji musí být vyšší než v komerčním. V praxi zkušenější specialista je dražší než méně zkušený. Toto nám cenu automatizace také ovlivňuje. Dále je čas strávený při tvorbě skriptu ovlivněn i druhy testů. Dle mého praktického příkladu pro základní test webové služby a funkční test postačí jen samotný komerční nástroj se svými doplňky. Při použití nekomerčního nástroje je nutno použít dva testovací nástroje.

Nekomerční nástroje jsou zdarma, ale mají nedostatečnou nebo až moc složitou dokumentaci. Zato komerční nástroje mají uživatelsky přívětivou dokumentaci, která je většinou obsažena přímo v aplikaci a také ve formě podpory od výrobce (odvíjí se od ceny licence). Toto je nejspíš hlavní hledisko, proč se ve větších společnostech a vývojových týmech používají komerční nástroje. Vytvoření automatizovaných testů je časově náročné a společnost si nemůže dovolit, aby nástroj přestal být podporován ze strany výrobců.

Při tvorbě skriptů jsem našel neocenitelnou výhodu komerčního nástroje a to umožnění ladících utilit samotného testovacího skriptu. U nekomerčního nástroje jsou chyby skriptu zjištěny až po finálním vytvoření samotného testovacího skriptu.

Trendem automatizovaných nástrojů dle trhu je jednotné nebo podobné prostředí pro všechny typy testů, ale s jiným modulem samotné testované aplikace. Výhoda toho je použití společných tříd kódu.

Při tvorbě automatizovaných testů se projevilo pravidlo potřeby úpravy/dodržení určitých pravidel při tvorbě kódu testované aplikace. Formuláře musí mít unikátní identifikaci, jinak by se v něm objekt nenašel jednoznačně. Např. při změně umístění vstupních políček formuláře v testované aplikaci by se data plnila do jiného objektu nebo by objekt byl kontrolován vůči jinému. Další příklad je logování, které musí obsahovat unikátní id volání a

unikátní id jednoho procesu. Důsledkem byly úpravy webové služby s těmito parametry, za účelem testu porovnání logu se vstupem.

Na trhu téměř neexistuje nekomerční nástroj, který by byl primárně určen k testům nestandardních technologií (například framework SAP a Siebel).

Dále se ukázalo, že i samotný testovací nástroj může mít chybu. Například v QuickTest se objevila chyba při vkládání třídy přes Properties a aplikaci se nedařilo ji vytvořit. Tento krok jsem musel obejít vytvořením jiné třídy automaticky vygenerované a přepsat ji. Dále se také ukázala chyba v podobě nerozpoznaného objektu při tvorbě skriptu pomocí ladícího programu. U nekomerčního nástroje Selenium se potvrdila sice veliká možnost použití i jiných knihoven ze samotného skriptovacího jazyka, ale při použití určitých funkcí, jako například tisk obrázku do souboru, fungují jen pod jedním druhem prohlížeče Firefox (tím je znemožněn test na ostatních prohlížečích).

Z výše uvedeného popisu rozdílů mezi komerčními a nekomerčními nástroji vyplývá, že před použitím konkrétních nástrojů při automatizaci testů je dobré si nejprve testovací nástroj vyzkoušet, zda splňuje potřebné technologie. Osoba, která nástroj bude používat, musí být také schopna v daném skriptovacím jazyce programovat (skriptovací jazyk je určen testovacím nástrojem). Důležité si je také uvědomit, jakým jsou automatizované testy přínosem pro samotný průběh testů, a podle toho se rozhodovat pro samotnou automatizaci a nástroj k jejímu provedení. Automatizované testy jako takové probíhají až na konci testovacího procesu, tj. v době, kdy je aplikace již téměř odladěna manuálními testy. Vzhledem k problémům s nástroji pro automatizaci a její časové, cenové a technologické náročnosti je nejlépe ji využít v regresním testování, tj. při testování, kdy se zjišťuje, že starší programy umí pracovat společně s novými změnami.

6 Použitá literatura

[1] Cockburn Alistair, 2005, Use Cases – Jak efektivně modelovat aplikace, ISBN 80-251-0721-3
[2] Crispin Lisa, Janet Gregory, 2009 - Agile Testing: A Practical Guide for Testers and Agile Teams, Adobe Reader , Addison-Wesley Professional, ISBN-10: 0-321-53446-8
[3] Dustin Elfriede, Thom Garrett, Bernie Gauf, 2009, - Implementing Automated SW Testing, Addison-Wesley Professional, ISBN-10: 0-321-58051-6
[4] Gunderloy Mike, 2007, Z kodéra vývojářem – Nástroje a techniky pro opravdové programátory, ISBN 978-80-251-1517-6
[5] ISO/IEC 9126 - 1: 2001 Softwarové inženýrství - Jakost produktu - Část 1: Model jakosti automatizace
[6] McConnell Steve, 2006, Odhadování softwarových projektů, Computer Press, ISBN 80-251-1240-3
[7] Page Alan, Ken Johnston, Bj ollison, Jak testuje SW Microsoft, Computer Press, ISBN 978-80-251-2869-5
[8] Patton Ron,2002 ,Testování Softwaru,Computer Press, ISBN 80-7226-636-5
[9] Rick D.Craig, Stefan P. Jaskiel, 2002 - Systematic SW Testing,ISBN: 1-58053-508-9

Internetové zdroje:
[10] [online] 1.4.2010, URL: http://www.kaner.com/
[11] [online] 10.4.2010, URL: http://www.satisfice.com/kaner/?p=11
[12] [online] 15.4.2010, URL: http://www.apachefriends.org/en/xampp.html
[13] [online] 2.4.2012, URL: http://www.soapui.org/
[14] [online] 3.4.2010, URL: http://www-01.ibm.com/software/awdtools/tester/robot/features
[15] [online] 10.4.2010, URL: http://www.automatedqa.com/products/testcomplete/
[16] [online] 29.5.2010, URL: http://welcome.hp.com/country/cz/cs/prodserv/software.html/
[17] Borland, [online] 15.5.2010 URL: http://techpubs.borland.com/silk_gauntlet/SilkTest/
[18] [online] 20.4.2010 URL: http://www.root.cz/
[19] [online] 20.4.2010 URL: http://www.eclipse.org/
[20] [online] 1.5.2010 URL: http://seleniumhq.org/

Konzultace se speciality z různých firem zabývající se tvorbou a testy SW:
[21] T-mobile: ing. Petr Kunstát, 10.5.2011, pozice: vedoucí manager testovacího týmu.
[22] Unicorn: Bc.Tomáš Novák, 11.5.2012, pozice: test architect / test manager
[23] Cleverlance: Marek Šťastný, 10.5.2012, pozice: senior tester/ testautomatic specialist

Seznam Přílohy:
Příloha A – test za použití Selenium
Příloha B – test v testovacím nástroji QuickTest a další obrázky prostředí
Příloha C – Popis uložení programu na přiloženém CD

7 Přílohy

7.1 Příloha A – test za použití Selenium

Firefox 3.5
Firefox - selenium IDE
Firefox - screengrab.org

Python 2.6.5
lxml 2.2.2 py2.6
pyreadline-1.5-win32-setup
ipython-0.10.win32-setup
selenium-remote-control-1.0.3.
server: selenium-server-1.0.3
selenium-python-client-driver-1.0.1 (do Python 2.6.5 adr LIB)
Příkaz v v příkazové řádce: C:\selenium-server-1.0.3> java -jar selenium-server.jar -port 5000
Microsoft Windows [Verze 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. Všechna práva vyhrazena.
c:\Users\Tomas\Downloads>test3.py

Script v Selenium:

```
#! /usr/bin/env python2.5
# -*- coding: utf-8 -*-

import unittest
import time
import re
import urllib
import codecs
import ConfigParser
from lxml import etree
from StringIO import StringIO

#import sys
#sys.path.insert(0, "/home/public_html/tomas/selenium/selenium-python-client-driver-1.0.1/")

from selenium import selenium

class Template(unittest.TestCase):
  screenshot_count = 0
  screenshot_preffix = ""

  def setUp(self):
    rcp = ConfigParser.RawConfigParser()
    rcp.read("selenium-tests.ini")
    self.service_log = rcp.get("Service", "log")
    self.host = rcp.get("Selenium", "host")
    self.port = rcp.getint("Selenium", "port")
    self.browserStartCmd = rcp.get("Selenium", "browserStartCmd")
```

```
  self.browserURL = rcp.get("Selenium", "browserURL")
  self.speed = rcp.getint("Selenium", "speed")
  self.wait_for_page_to_load = rcp.getint("Selenium", "wait_for_page_to_load")
  self.window_maximize = rcp.getboolean("Selenium", "window_maximize")
  self.screenshot_enable = rcp.getboolean("Screenshot", "enable")
  self.screenshot_output = rcp.get("Screenshot", "output")
  self.screenshot_suffix = rcp.get("Screenshot", "suffix")
  self.add_message_csv = rcp.get("Input", "add_message")

  self.verificationErrors = []

  self.selenium = selenium(self.host, self.port, self.browserStartCmd, self.browserURL)
  self.selenium.start()
  self.selenium.set_speed(self.speed)

def go_home(self):
  if (self.window_maximize):
    self.selenium.window_maximize()

  self.selenium.open(self.browserURL)
  self.selenium.click("link=guestbook-client.php")
  self.selenium.wait_for_page_to_load(self.wait_for_page_to_load)

def screenshot(self):
  if (self.screenshot_enable):
    self.screenshot_count += 1
    image = "%s%s%04i%s" % (self.screenshot_output, self.screenshot_preffix,
        self.screenshot_count, self.screenshot_suffix)
    self.selenium.capture_entire_page_screenshot(image, None)

def etree_parse(self):
  parser = etree.HTMLParser()
  return etree.parse(StringIO("<html>%s</html>" % self.selenium.get_html_source()),
        parser)

   def assert_not_success(self, action, id):
  if (id):
    message = "%s success, id = %s" % (action, id)
    fo = urllib.urlopen(self.service_log)
    for line in fo.readlines():
      if (line.find(message) != -1):
        return True

  self.fail("addMessage failure: message '%s' not found in %s" % (message,
        self.service_log))
  return False

def get_form_ids(self, tree, name):
  expr = "//*[local-name() = $name]"
  #form_elts = tree.xpath('/form [@name="getMessages"]]')
```

```
    id = []
    form_elts = tree.xpath(expr, name="form")
    for form_elt in form_elts:
      if (form_elt.get("name") == name):
        for element in form_elt.getchildren():
          if (element.get("name") == "id"):
            id.append(element.get("value"))

    return id

  def tearDown(self):
    self.selenium.stop()
    self.assertEqual([], self.verificationErrors)

class TestAddMessage(Template):
  screenshot_preffix = "AddMessage"

  def addMessage(self, record):
    self.selenium.type("firstName", record["firstName"])
    self.selenium.type("lastName", record["lastName"])
    self.selenium.type("email", record["email"])
    self.selenium.type("subject", record["subject"])
    self.selenium.type("message", record["message"])

    tree = self.etree_parse()
    id = self.get_form_ids(tree, "addMessage")
    self.selenium.click(u"//input[@value='Přidat zprávu']")
    self.selenium.wait_for_page_to_load(self.wait_for_page_to_load)

    if (id):
      self.assert_not_success("addMessage", id[0])
    else:
      self.fail("form['addMessage'].id not found")

  def test(self):
    self.go_home()

    fo = codecs.open(self.add_message_csv, "r", "utf-8")

    self.screenshot()
    for line in fo.readlines():
      line = line.strip()
      csv = line.split(";")
      items = ["firstName", "lastName", "email", "subject", "message"]
      if (len(csv) == len(items)):
        i = 0
        record = {}
        for item in items:
          record.update({item: csv[i].strip()})
```

```
        i += 1

       self.addMessage(record)
       self.screenshot()
      else:
       self.fail("bad line '%s' in %s" % (line, self.add_message_csv))

    fo.close()

class TestRmMessage(Template):
  screenshot_preffix = "RmMessage"

  def rmMessage(self, id):
    self.selenium.click("rmBtn_%s" % id)
    self.selenium.wait_for_page_to_load(self.wait_for_page_to_load)
    self.assert_not_success("rmMessage", id)

  def test(self):
    self.go_home()
    self.screenshot()

    tree = self.etree_parse()
    ids = self.get_form_ids(tree, "rmMessage")
    for id in ids:
      self.rmMessage(id)
      self.screenshot()

if __name__ == "__main__":
  unittest.main()
```

7.2 Příloha B - Script test1 – funkčního testu v QuickTestu:

```
Option Explicit

Browser("Guestbook").Page("Guestbook").WebEdit("firstName").Set
        DataTable("FirstName", "Global")
Wait 1
Browser("Guestbook").Page("Guestbook").WebEdit("lastName").Set DataTable("LastName",
        "Global")
Wait 1
Browser("Guestbook").Page("Guestbook").WebEdit("email").Set DataTable("Email",
        "Global")
Wait 1
Browser("Guestbook").Page("Guestbook").WebEdit("subject").Set DataTable("Subject",
        "Global")
Wait 1
Browser("Guestbook").Page("Guestbook").WebEdit("message").Set DataTable("Message",
        "Global")
Wait 1
Browser("Guestbook").Page("Guestbook").WebButton("Přidat zprávu").Click


Dim objDisplayed
Dim objExpected
Dim strResultString

Set objExpected = New clsZaznam
objExpected.JmenoPrijmeni = DataTable("FirstName", "Global") & " " &
        DataTable("LastName", "Global")
objExpected.Email = DataTable("Email", "Global")
objExpected.Subject = DataTable("Subject", "Global")
objExpected.Message = DataTable("Message", "Global")

Set objDisplayed = funcGetRowData(1)

If objDisplayed.CompareClasses(objDisplayed, objExpected, strResultString) Then
        Reporter.ReportEvent micPass, "Verified: Zaznam", strResultString
Else
        Reporter.ReportEvent micFail, "Review: Zaznam", strResultString
End If

Set objDisplayed = Nothing
Set objExpected = Nothing

Class clsZaznam

        Private m_strJmenoPrijmeni
        Private m_strEmail
        Private m_strSubject
```

```
Private m_strMessage

Private Sub Class_Initialize
        m_strJmenoPrijmeni = "#N/A#"
        m_strEmail = "#N/A#"
        m_strSubject = "#N/A#"
        m_strMessage = "#N/A#"
End Sub

Private Sub Class_Terminate
        ' Statements go here.
End Sub

'       JmenoPrijmeni
Public Property Get JmenoPrijmeni
        JmenoPrijmeni = m_strJmenoPrijmeni
End Property

Public Property Let JmenoPrijmeni(InputValue)
        m_strJmenoPrijmeni = InputValue
End Property

'       Email
Public Property Get Email
        Email = m_strEmail
End Property

Public Property Let Email(InputValue)
        m_strEmail = InputValue
End Property

'       Subject
Public Property Get Subject
        Subject = m_strSubject
End Property

Public Property Let Subject(InputValue)
        m_strSubject = InputValue
End Property

'       Message
Public Property Get Message
        Message = m_strMessage
End Property

Public Property Let Message(InputValue)
        m_strMessage = InputValue
End Property
'       ToString
Public Function ToString()
```

```
        ToString = "JmenoPrijmeni = " & JmenoPrijmeni & " | " & "Email = " & Email & " | " & "Subject = " & Subject & " | " & "Message = " & Message

End Function
'       Compare
Public Function CompareClasses(ByRef objDisplayed, ByRef objExpected, ByRef strResultString)

        Dim strExpected
        Dim strDisplayed
        Dim arrList
        Dim blnResult
        Dim i

        arrList = Array("JmenoPrijmeni", "Email", "Subject", "Message")
        blnResult = True

        For i = 0 To UBound(arrList)

                Select Case arrList(i)

                        Case "JmenoPrijmeni"
                                strExpected = objExpected.JmenoPrijmeni
                                strDisplayed = objDisplayed.JmenoPrijmeni

                        Case "Email"
                                strExpected = objExpected.Email
                                strDisplayed = objDisplayed.Email

                        Case "Subject"
                                strExpected = objExpected.Subject
                                strDisplayed = objDisplayed.Subject

                        Case "Message"
                                strExpected = objExpected.Message
                                strDisplayed = objDisplayed.Message

                End Select

                If strExpected <> "#N/A#" Then

                        If strExpected <> strDisplayed Then
                                blnResult = False
                                strResultString = strResultString & "***** Property " & arrList(i) & " neni stejne" & vbCRLF & _
                                                "{Expected Value = " & strExpected & "}" & vbCRLF & _
                                                "{Displayed Value = " & strDisplayed & "}" & vbCRLF & vbCRLF
```

```
                    Else
                        strResultString = strResultString & "Property " & arrList(i) & " je stejne" & vbCRLF & _
                            "{Expected Value = " & strExpected & "}" & vbCRLF & _
                            "{Displayed Value = " & strDisplayed & "}" & vbCRLF & vbCRLF
                    End If

                End If

            Next

            CompareClasses = blnResult

        End Function

End Class

Function funcGetRowData(ByVal intRowNum)

        Dim objDisplayed
        Dim objZaznam
        Dim strPom
        Dim strSubject
        Dim strEmail
        Dim intStart
        Dim intEnd

        With Browser("Guestbook").Page("Guestbook")

            Set objZaznam = .WebElement("html tag:=TD", "class:=menu", "index:=" & intRowNum - 1)
            If Not objZaznam.Exist Then
                Reporter.ReportEvent micFail, "Review: funcGetRowData", _
                    "Function did not execute because of kontejner neexistuje."

                Exit Function
            End If

            Set objDisplayed = New clsZaznam
            objDisplayed.JmenoPrijmeni = objZaznam.Link("href:=mailto:.*", "index:=0").GetROProperty("innertext")

            strEmail = objZaznam.Link("href:=mailto:.*", "index:=0").GetROProperty("href")
            objDisplayed.Email = Replace(strEmail, "mailto:", "")

            strPom = objZaznam.WebElement("html tag:=DIV", "class:=menu", "index:=0").GetROProperty("innerhtml")
```

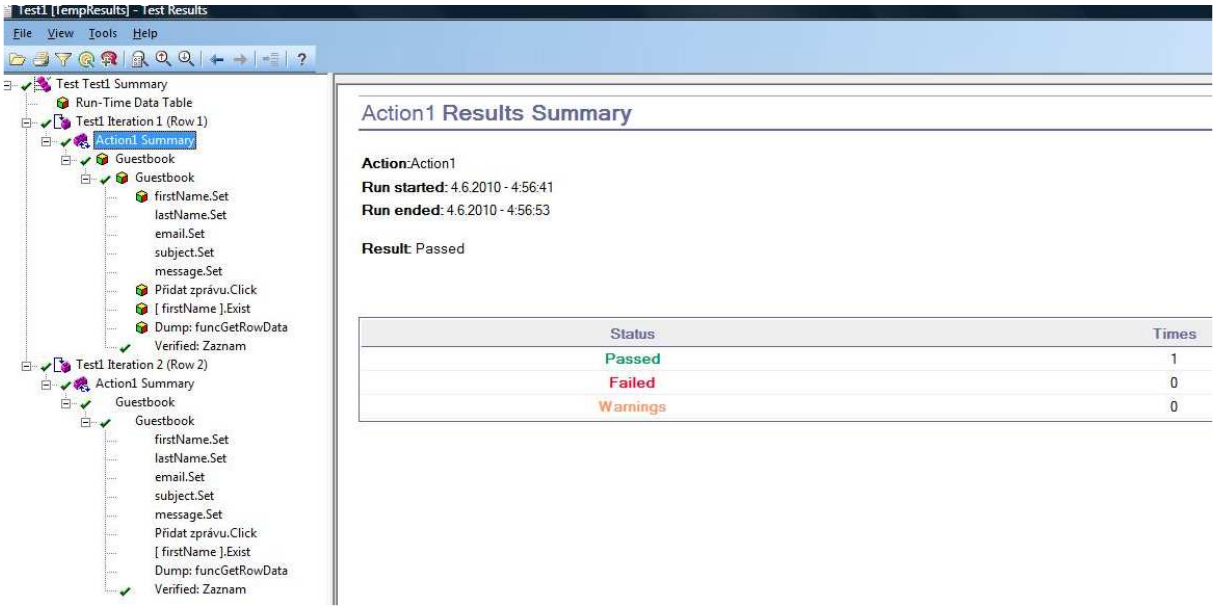
```
            intStart = InStr(1, LCase(strPom), "</a><br>", 1)
            intStart = intStart + 8

            strPom = Mid(strPom, intStart)
            intEnd = InStr(1, LCase(strPom), "<form", 1)

            strSubject = Left(strPom, intEnd - 1)
            objDisplayed.Subject = strSubject

            objDisplayed.Message = objZaznam.WebElement("html tag:=DIV",
        "class:=main", "index:=0").GetROProperty("innertext")


        End With

        Reporter.ReportEvent micDone, "Dump: funcGetRowData", objDisplayed.ToString()
        Set funcGetRowData = objDisplayed
        Set objZaznam = Nothing

End Function

'Browser("Guestbook").Page("Guestbook").WebButton("Smazat").Click
'Wait 10
```

Test2 – script testu webové služby v QuickTestu:

```
Dim intID

Randomize
intID = Int((999 * Rnd) + 100)
DataTable("addMessage_In_XML_id", dtGlobalSheet) = intID

Set con = createobject("ADODB.connection")
Set rs = createobject("ADODB.Recordset")
Set com = createobject("ADODB.command")
con.open("Driver={MySQL ODBC 5.1 Driver};Server=localhost;Database=guestbook;
        User=root;Password=;Option=3;")

strSQLQuery = "delete from guestbook"

'com.ActiveConnection = con

'com.Text=strSQLQuery
'com.Execute

rs.Open strSQLQuery, con

con.Close

' guestbook Web service object steps
```

```
addMessage = WebService("guestbookService").addMessage(XMLWarehouse("Record"),
        DataTable("addMessage_In_requestID", dtGlobalSheet))
WebService("guestbookService").Check CheckPoint("addMessage")

Set getMessages =
        WebService("guestbookService").getMessages(DataTable("getMessages_In_requestI
        D", dtGlobalSheet))
WebService("guestbookService").Check CheckPoint("getMessages")

rmMessage = WebService("guestbookService").rmMessage(DataTable("mmMessage_In_id",
        dtGlobalSheet), DataTable("mmMessage_In_requestID", dtGlobalSheet))
WebService("guestbookService").Check CheckPoint("rmMessage")
```

Obrázek výsledku testů v QuickTestu:



Obr. 12: Výstup testu v QuickTestu 1

7.3 Příloha C – Přiložené CD

Testovaná aplikace s aplikací xampp, která nám umožňuje spuštění webového serveru APACHE a databáze MYSQL.

Cesta k Testované aplikaci
\xampp\htdocs\guestbook\install.php soubor pro vytvoření \xampp\htdocs\guestbook\databaze
\xampp\htdocs\guestbook\guestbook-service.php
\xampp\htdocs\guestbook\guestbook-service.log log webové služby
\xampp\htdocs\guestbook\guestbook-client.php vstupní soubor webové aplikace
\xampp\htdocs\guestbook\guestbook-client.log log formuláře
\xampp\htdocs\guestbook\guesbook.wsdl webová služba

\xampp\htdocs\guestbook\css\default.css knihovna kaskádových stylů webové aplikace
\xampp\htdocs\guestbook\sql.php připojení k databázi
\xampp\htdocs\guestbook\logging.php připojení k databázi

\xampp\xampp-controll.exe slouží ke spuštění webového serveru Apache a databáze Mysql (je již nakonfigurován na aplikaci, stačí celou složku xampp\ nakopírovat na disk spustit xampp-controll.exe , kde se jen zadá start Apache a start Mysql. Nyní pokud se zadá v internetovém prohlížeči URL: http://localhost/guestbook/guestbook-client.php

Dále použité knihovny k testů a samotné testy:

\ QuickTest Professional testy v QuicjkTestu

Selenium knihovny a Python knihovny:

\ selenium-python-client-driver-1.0.1
\ selenium-server-1.0.3
\ test_selenium test v selenium - spouští se z příkazového řádku selenium-tests.py
\ SoapUI\guestbook-BC-T--Jandak-soapui-project.xml – testy v SoapUI ve verzi: soapUI 3.6-beta2, copyright (C) 2004-2010 eviware software ab