



škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Student : **Victor Espinoza**
Vedoucí bakalářské práce : **Mgr. Anna Borovcová**
Oponent bakalářské práce : **Ing. Alena Buchalcegová, Ph.D**

TÉMA BAKALÁŘSKÉ PRÁCE

Nástroje pro náhodné testování (Monkey testing)

rok: 2011



Rád bych poděkoval Mgr. Anně Borovcové za vedení mé práce a za podnětné připomínky, které přispěly ke konečné kvalitě práce.

Dále bych rád poděkoval Veronice Petříkové za průběžnou pomoc při korektuře textů a podporu.



Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 29.06.2011

.....

podpis

Tato práce objasňuje význam pojmu náhodné testování, jeho využití a uplatnění v oblasti testování software.

V první kapitole představuje hlavní myšlenky testování a seznámíme se s testováním na obecné úrovni. V další části se dočteme o jednotlivých typech testování a o testování v průběhu vývoje software.

V další části práce se detailně seznamujeme s automatizovaným testováním, jeho výhodami a riziky. Navazuje na ni kapitola o náhodném testování, které můžeme považovat za podskupinu automatizovaného testování.

Po dostatečném teoretickém úvodu do problematiky náhodného testování si předvedeme možnosti automatizovaného testování přímo na konkrétní aplikaci. K tomu využijeme vybrané nástroje.

Další kapitola vysvětluje význam získaných výsledků a možnosti jejich zpracování.

Klíčová slova

Náhodné testování, testovací nástroje, automatické testování

This bachelor thesis focuses on explaining the term Monkey testing and its usage in a field of software development.

The first chapter presents principal ideas about testing and we get familiar with testing in general. In the next part we familiarize with particular types of testing and with the role of testing during software development.

Following part of this thesis introduces test automation its advantages and risks. Next chapter is about monkey testing which may be considered as a subgroup of automated testing.

After sufficient theoretical introduction into field of monkey testing we demonstrate its capabilities on concrete application. To do it we use chosen software tools.

Next chapter explains meaning of obtained results and possibilities of its processing.

Keywords

Monkey testing, testing tools, automated testing

	9
1.1	Téma práce a důvod výběru.....9
1.2	Cíl práce9
2	Testování obecně10
2.1	Úvod.....10
2.2	Co je chyba z pohledu testera11
2.3	Podstata testování software12
2.4	Testování neprokazuje, že chyby neexistují.....13
2.5	Optimální hladina testování.....13
3	Typy testů dle stavu vývoje software15
3.1	Unit testy.....15
3.2	Integrační testy.....15
3.3	Systémové testy15
3.4	Akceptační testy15
3.5	Regresní testy.....15
4	Přístupy k testování software16
4.1	Testování černé a bílé skříňky16
4.2	Testování splněním a testy selháním.....16
4.3	Vyčerpávající testování17
4.4	Statické a dynamické testování.....17
4.5	Manuální a automatizované testování18
5	Automatizované testování19
5.1	Automatizace testů19
5.2	Výhody a omezení automatizovaného testování20
5.2.1	Výhody20
5.2.2	Nevýhody20
5.3	Problémy s automatizovaným testováním21
5.4	Testovací nástroje22
5.4.1	Způsob identifikace grafických prvků.....22
5.4.2	Možnost načtení hodnot zobrazených ve vstupních a výstupních polích.....23
5.4.3	Možnost zápisu do externího souboru nebo databáze.....23
5.4.4	Pořizovací cena nástroje23
5.4.5	Přehlednost a intuitivní ovládání.....23

	23
	24
6.1	Myšlenka pokusné opice.....24
6.2	Kategorie opic24
6.2.1	Hloupá opice (Dumb monkey).....24
6.2.2	Polointeligentní opice (Semi brilliant monkey)24
6.2.3	Inteligentní opice (Brilliant monkey)24
6.3	Výhody náhodného testování25
6.4	Nevýhody náhodného testování25
7	Zkušenosti Noela Nymana26
7.1	Rozdělení opic26
7.1.1	Chytré opice26
7.1.2	Hloupé opice.....26
7.2	Příprava chytré opice26
7.3	Příprava hloupé opice26
7.4	Správný výběr26
8	Ukázka hloupé opice27
9	Experiment28
9.1	Cíl experimentu28
9.2	Testovaná aplikace28
9.3	Stanovený cíl29
9.4	Vytvořené opice29
9.4.1	Hloupá opice.....29
9.4.2	Polointeligentní opice29
9.5	Algoritmus hloupé opice.....29
9.5.1	Formulace úlohy29
9.5.2	Analýza úlohy.....29
9.5.3	Analýza29
9.5.4	Slovní popis algoritmu.....30
9.6	Popis připraveného testu.....30
9.7	Analýza algoritmu polointeligentní opice30
9.7.1	Formulace úkolu30
9.7.2	Analýza úlohy.....30
9.7.3	Slovní popis algoritmu.....31

	31
	32
10.1 WinTask.....	32
10.1.1 Doporučení	32
10.1.2 Představení.....	32
10.1.3 Vybavení.....	32
10.1.4 Vytvoření opice.....	33
10.1.5 Práce s nástrojem	36
10.1.6 Klady.....	36
10.1.7 Zápory	37
10.2 Autolt.....	39
10.2.1 Představení.....	39
10.2.2 Vybavení.....	39
10.2.3 Vytvoření opice.....	40
10.2.4 Klady.....	41
10.2.4.3 Podpůrné nástroje.....	41
10.2.5 Zápory	41
10.2.6 Osobní názor.....	42
10.3 AutoHotkey	43
10.3.1 Představení.....	43
10.3.2 Vybavení.....	43
10.3.3 Vytvoření testu	44
10.3.4 Klady.....	44
10.3.5 Zápory	44
10.3.6 Osobní názor.....	45
11 Výsledky experimentu	46
11.1 Výsledky	46
11.2 Popis provedených testů	46
11.3 Objevené chyby.....	47
11.4 Záznam průběhu testu.....	47
11.5 Reprodukce nalezených chyb a jejich analýza	48
12 Zhodnocení experimentu	49
13 Závěr.....	50
14 Terminologický slovník	51



PDF

Complete

*Your complimentary
use period has ended.
Thank you for using
PDF Complete.*

[Click Here to upgrade to
Unlimited Pages and Expanded Features](#)

.....52

1.1 Téma práce a důvod výběru

Testování software a aplikací je v dnešní době již nedílnou součástí softwarového vývoje v oblasti informačních a telekomunikačních technologií. Tento trend nebyl v minulosti samozřejmostí a osobně jsem velmi rád, že se testování v současnosti stále více odborníků. Narůstá také jeho význam a i způsob provádění je na odborné úrovni. Je příjemné sledovat, že testování již není určitou neúspěšnou činností, ale jeho provádění je již plánováno. Stále se jí své uplatnění nacházejí nástroje pro testování. Přibývá i odborníků v oblasti testování. To vše napomáhá v nahlížení na testování jako na plnohodnotnou oblast vývoje software.

Tématem mé práce je náhodné testování. Toto téma jsem si vybral na základě osobního zájmu o oblast testování software. Jedná o téma nepříliš známé a v prostředí České republiky málo rozšířené. Proto doufám, že tato práce pomůže tento typ testování objasnit budoucím studentům a osloví k výzkumu náhodného testování.

1.2 Cíl práce

Cílem této práce je představení tématu náhodného testování. Po uvedení do problematiky náhodného testování ukážu studentům i přínosy náhodného testování na praktickém příkladu. K tomu využiji ukázkovou aplikaci a vyberu si nástroje, které mi umožní zamýšlené testy provést. Po přečtení práce bych měl mít studentům jasno o výhodách a rizicích náhodného testování. Pokud bych chtěl tento typ testování využít, pak bude v daných možnostech, které má a co od jakéhoto očekávat.

2.1 Úvod

Každá věc na světě má nějakou chybu. Doposud se nepodařilo vytvořit nic naprosto dokonalého. Toto o to více platí i ve světě vývoje software. Je zajímavé, jak moc software vstoupil do našeho života. S jednoduchými i velice složitými programy a aplikacemi se setkáváme i při běžných činnostech. Když vytáhneme krabici mléka z lednice, nastartujeme automobil nebo si koupíme kávu v automatu. Jejich bezchybnost považujeme za samozřejmou a velmi nás překvapí, když něco nefunguje tak, jak očekáváme.

nořství. Pro svou práci uvedu definici chyby dle Rona Pattona, kterou uvedl ve své knize Testování software:

„O chybě hovoříme, pokud je splněna jedna nebo více z následujících podmínek:

- 1. Software nedělá něco, co by podle specifikace produktu dělat měl.*
- 2. Software dělá něco, co by podle údajů specifikace produktu dělat neměl.*
- 3. Software dělá něco, o čem se produktová 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 – její koncový uživatel nebude považovat za správný.“ [1]*

Dle mého vlastního názoru je tato definice hodn vypovídající, je v ní obsařena v t-ina p ípad , kdy v praxi dochází k otázkám, zda se jedná í nejedná o chybu.

Ve vý-e uvedené definici chyby je uvedena i osoba testera.

„Cílem testera je vyhledávat chyby, vyhledat je co nejdříve a zajistit jejich nápravu.“ [1]

Úkolem testera je chyby vyhledat, správným zp sobem je p edat dál k dal-ímu zpracování í posouzení. Správným zp sobem je my-len hlavn dostate ný popis, jakým lze chybu znovu navodit, pop ípad jak ke konkrétní chyb dosp l samotný tester. Pokud dojde k opravení nahlá-ené chyby, pak je op t na testerovi tuto chybu znovu p ezkoumat a ujistit se tak, fle do-lo k její skute né náprav .

varu uvádí:

„Žádný program není možné otestovat kompletně, a to z následujících důvodů:

- Počet možných vstupů softwaru je příliš velký
- Počet možných výstupů softwaru je příliš velký
- Počet možných cest, které vedou skrze software, je příliš velký
- Specifikace softwaru je subjektivní. Vždycky je možné říci, že jediná chyba je pouze v očích pozorovatele“ [1]

Hlavní myšlenkou tohoto tvrzení je fakt, že vstupy do programů a aplikací lze různými způsoby kombinovat a často existuje více než pouze jeden způsob, jak dosáhnout požadovaného stavu. Také se ztotožní s myšlenkou, kdy je uvedeno, že kompletní otestování není možné z důvodu příliš velkého počtu vstupů a výstupů. Toto tvrzení ale nelze vztáhnout na všechny aplikace. Přesto je nutné si uvědomit, že z výše uvedených důvodů skutečně není možné v rámci aplikací a software otestovat beze zbytku, a proto považují tuto myšlenku za důležitou a přínosnou.

Tím je testování vystaveno určitému riziku. Toto riziko je spojené s tím, že při testování volíme určité testy, je-li jsou typologickými zástupci určité skupiny testů. Musíme tedy počítat s možností, že nebude odhalena určitá chyba a nakonec na ni narazí ať konečný uživatel.

Představme si fiktivní společnost, která má na starosti otestování software instalovaný do praček. Tento software kontroluje teplotu vody, dobu praní a je zodpovědný za správné nastavení pro jednotlivé programy praní. Bohužel je v tomto software chyba způsobující ohřátí vody o 30 °C více, než je požadováno. Testeři tuto chybu neobjevili a pračky s chybným software jsou uvedeny na trh. Avšak hned první uživatelé tuto chybu zaznamenali a nyní je na výrobci pračky vše napravit.

i testování software, byly by náklady na její opravu
t chto náklad zahrnout i finan ní výdaje spojené se

stažením vadných p ístroj z trhu, od-kodn ní zákazník apod. Náklady na opravu chyby tedy obsahují nejen náklady vynaložené na odstran ní chyby v software, ale i v-echny dal-í náklady, které p ípadn souvisí s odstran ním v-ech následk .

Náklady rostou spolu s dobou vývoje produktu. Nabízí se otázka, zda není tedy vhodn j-í více testovat a vyhnout se takovým situacím. Zde se ov-em op t vracím k my-lenkám Rona Pattona o nemofnosti otestovat software kompletn z d vodu velkého množství vstup a výstup . Pokud bychom m li software otestovat úpln , pak by náklady a as k tomuto pot ebný nevyváfil riziko spojené s nasazením software, jefl není kompletn otestovaný.

2.4 Testování neprokazuje, že chyby neexistují

Samotný fakt, fl jsou v software nalezeny chyby, nevypovídá o tom, fl jifl v software fládné dal-í chyby nejsou. Testy jsou navrflené pro otestování jedné ur íté situace, vzhledem k tomu, fl netestujeme v-echny mofné situace, není mofné v-echny chyby najít. Díky testování nalezneme dal-í a dal-í chyby, nem flme s jistotou prohlásit, fl dal-í chyby neexistují.

Pokud jsou v software nalezeny chyby, neznamená to, fl se jifl v daném programu dal-í chyby nenalézají. Provád né testy jsou navrflené k otestování ur íté situace. Vzhledem k tomu, fl netestujeme v-echny mofné varianty, není mofné v-echny chyby najít. Testování nám umofl uje prokázat, fl se v software chyby nalézají. Neumofl ují nám prokázat opak.

Navíc ím více chyb objevíme, tím více jich m flme v testovaném software o ekávat.

2.5 Optimální hladina testování

Jedná se o takové množství test , kdy je mofné íci, fl je testovaný software efektivn otestovaný a zároveň nejsou náklady na testování zbyte n vysoké. Takový bod je ov-em velice t flké stanovit.

asto záleflí pouze za osob test managera. Ten je zodpov dný za správný výb r testovací strategie a vytvo ení testovacího plánu. P í svých rozhodnutích musí brát v potaz nap íklad typ software, pofladavky zadavatele projektu, dobu dodání a mnoho dal-ího.

Zde naráfíme na problém pojmenovaný jako paradox pesticid . Tento pojem vymyslel a poprvé pouflil v roce 1990 Boris Beizer ve své knize *Software Testing Techniques*. [21] Tímto termínem se jeho autor snaflil vysv tlit fakt, fl ím více se ur ítý software testuje, tím mén



PDF
Complete

*Your complimentary
use period has ended.
Thank you for using
PDF Complete.*

[Click Here to upgrade to
Unlimited Pages and Expanded Features](#)

rá proti navrženým testům imunní stejn jako hmyz

Pokud tedy stávající testy již nevedou k objevení chyby, pak by bylo na místě další testy přidat. To ovšem může vést právě k situaci, kdy je test již mnoho a je porušena optimalita množství testů.

3 Typy testů dle stavu vývoje software

3.1 Unit testy

Tento typ test je zaměřen na otestování nejmenších částí software, což v případě objektového programování jsou to malými jednotlivými částmi a jejich metody. V případě procedurálního testování se jedná o samotné procedury a funkce. Unit testy vytváří samotní vývojáři. [6]

3.2 Integrační testy

Účelem integračních testů je ověřit, že jednotlivé části otestované již v průběhu unit testů, po vzájemném sloučení spolupracují. Testy jsou prováděny jak vývojáři, tak i testery. [6]

3.3 Systémové testy

Systémové testy mají za účel otestovat systém jako celek. Pro tyto testy jsou používány testovací scénáře simulující chování uživatele v systému. [5]

3.4 Akceptační testy

Jsou prováděny zadavatelem požadavků. Jejich účelem je ověřit, že dodané řešení obsahuje všechny požadované funkcionality. Navíc se ověřuje, že implementované funkcionality se chovají tak, jak bylo vyžadováno. [7]

3.5 Regresní testy

Z vlastní zkušenosti bych vysvětlil regresní testy jako testy vykonávané po nasazení a provedení určité změny software. Takovou změnou může být například implementace nové funkcionality, refaktoring kódu nebo modularizace. Tento typ testů slouží k ověření, stávající funkcionality zůstaly nezměněny.

4 Přístupy k testování software

Protože v software mohou být obsaženy chyby různých typů, je třeba i k testování přistupovat z více hledisek a zvýšit tak možnost objevení v těle množství chyb a chyb s vyšší závažností a dopadem. Díky tomuto získáme představu, do jaké kategorie test patří náhodné testování a jaké oproti ostatním přístupům přináší výhody a s jakými nedostatky musíme počítat.

4.1 Testování černé a bílé skřínky

Testování černé skřínky je zaměřeno na implementovanou logiku aplikace. Tedy testuje se oproti specifikaci i dokumentaci, která je součástí objednávky. Nejde o lež, jímž předpokladem je, že takováto dokumentace je vytvořena a že je v aktuálním stavu.

V případě, že tato dokumentace ještě není hotova a je již potřeba zahájit přípravu testovacích scénářů, je možné čerpat i z jiných zdrojů. Takovými zdroji je například specifikace požadavků, tedy písemné vyjádření funkcionalit a jejich popis, které jsou pořadovány.

Hlavní rysem testování černé skřínky je validace oproti očekávanému chování, aniž bychom věděli, jakým způsobem je tohoto chování dosaženo. O vnitřních procesech a pochodech aplikace nemáme žádné informace. Přestože nevíme, jak software pracuje. Je možné v rámci tohoto typu testování praktikovat i přístup, je-li nám umožněno otestovat velké množství situací, kdy se chyby mohou projevit.

Testování bílé skřínky se na rozdíl od testování černé skřínky liší ve znalosti a možnosti přístupu ke zdrojovému kódu. Tester zná způsob, jakým byly jednotlivé pořadované funkce implementovány. Výhodou této formy testování je vyšší odhad, v jakých případech a za jakých podmínek by mohlo dojít k chybovému stavu. [20]

4.2 Testování splněním a testy selháním

Při testování splněním se tester snaží přimět najít chybu, například použitím nevhodných dat. Účelem provádění těchto testů je kontrola, že software za předpokladu obdržení správných dat a provedení očekávaných kroků reaguje, jak je požadováno. Při testování selháním je záměrem chybu vyvolat.

nou funkčnost v-echny kombinace možných zadaných vstup [8] [20].

Pokud bychom uplatňovali vyvířpávající testování, je dost pravdě podobné, že samotný proces testování by trval mnohonásobně déle než vývoj testovaného software. Oblíbeným a velice výstižným příkladem pro ukážku plných testů je obyčejná kalkulačka. Vezmeme-li v úvahu, že testovaná kalkulačka je skutečně omezená a podporuje pouze základní matematické funkce, tedy sčítání, odčítání, násobení a dělení. Navíc tato teoretická kalkulačka umí počítat pouze do jednoho milionu. Při aplikaci plných testů bychom mohli začít testovat sčítání. Začneme při sčítání jedné. Počítali bychom tedy $1 + 1$, $1 + 2$, $1 + 3$ až po $1 + 999999$. Po provedení v-ech těchto testů, bychom viděli, že sčítání jedné je skutečně funkční a my jsme pokračovali s při sčítáním k číslu 2. Takto bychom pokračovali až do 999 999. Teprve po provedení v-ech těchto testů bychom mohli přejít k testování další matematické operace. Je zřejmé, že tento způsob testování je skutečně dkladný, ale i nepředstavitelně časově a finančně náročný. To i za předpokladu, že bychom pro tyto testy použili nějaký nástroj pro automatizaci. Abychom zredukovali takovéto množství testů na realizovatelný počet testovacích případů, jsou vytvářeny tzv. třídy ekvivalence. Roztřídění ekvivalentních případů je postup, při kterém omezíme širokou a nekonečnou množinu testovacích případů do mnohem menší skupiny, která danou množinu zastupuje.

4.4 Statické a dynamické testování

Při statickém testování dochází ke kontrole software, který neběží, není spuštěn. Proto je možné s tímto typem testování začít ještě před vytvořením prvního prototypu. Předmětem testování je i dodávaná i užívaná dokumentace. Výsledkem mohou pak být i odhady náročnosti času a zdrojů pro nadcházející vývoj.

Jako příklad statického testování si můžeme představit revizi zdrojového kódu. Tato technika testování může mít formu neformálního sezení dvou nebo více programátorů, kteří po sobě revidují napsaný kód a navrhuji změny. K tomu je potřeba jistá příprava, tedy nastudování kontrolovaného kódu a příprava připomínek. Důležitou částí je zápis z takové porady, postačí jednoduchý písemný podklad, na jehož základě se budou navržené změny realizovat. Chyby objevené při revizi kódu uvedu v následující části práce.

Předpokladem pro dynamické testování je spustitelná verze testovaného software. Software

4.5 Manuální a automatizované testování

Automatizovaným testováním rozumíme využití nějaké aplikace či specializovaného software, který je určen k provádění připravených testů. Jedná se o automatizaci testů, jež jsou jinak prováděny manuálně. Přesto, že v jistých případech je využití automatizovaných testů nanejvýš vhodné, nemohou nikdy plně nahradit testy prováděné manuálně anebo osobu testera.

Manuálně prováděné testy vyžadují přítomnost osoby jako testera, jež provádí jednotlivé kroky testu. Test může provádět dle určeného zadání nebo může testovat volně dle vlastního uvážení.

hod a nevýhod automatického testování. Podle mého názoru je důležité pochopit jednotlivé aspekty tohoto přístupu k testování pro pochopení skutečné části práce o náhodného testování.

„Automatizované testování je použití nějaké aplikace, která dokáže spouštět vytvořené testy, porovnávat aktuální výstupy s očekávanými výstupy, nastavovat testovací podmínky a dokáže výsledky testu reportovat. Automatizované testování se dá všeobecně popsat jako automatizování manuálního procesu v případě, kdy se používá formalizovaný testovací proces.“ [7]

Z jistého úhlu pohledu můžeme testování považovat za dovednost. Schopnost vymyslet sledy vstupů a kroků, které nám umožní získat objektivní názor na stav testovaného software. K otestování daného software je možné připravit rozsáhlou množinu testů. Je ovšem složité a velice důležité tuto skupinu omezit na menší podskupinu, jejímž provedením odhalíme dostatečné množství chyb. Hledáme hlavně závažné chyby, které by podle stanovených kritérií neumožňovaly využívat software k zamýšlenému účelu. [2]

Jaký je takový efektivní test a jaké by měl mít správný testovací případ vlastnosti?

„Existují 4 atributy podle kterých určujeme kvalitu navrženého testu. Těmito atributy jsou:

- *Efektivita, tedy schopnost objevit či neobjevit chybu*
- *Příkladnost, test by měl otestovat více než jednu věc a zredukovat tak celkový testovací mix*
- *Úspornost, jak nákladná je analýza, vykonání a případná úprava testu*
- *Možnost test rozvíjet, zde uvažujeme náklady a úsilí, jež je třeba vynaložit na konkrétní testovací případ v případě nutné změny. Například když dojde k nějaké změně software“ (přeloženo z [2])*

Schopnost navrhnout efektivní test není pouze v jeho efektivitě, ale také v zajištění návrhu testu takovým způsobem, abychom se vyhnuli nadměrným nákladům.

5.1 Automatizace testů

Automatizace testů je také schopnost, nicméně odlišná od samotného testování. Pravdou je, že připrava automatického testu je finančně i časově náročnější než provedení identického testu manuálně. Náklady vynaložené při přípravě takového testu se navrací až po opětovném

matizovaného testu však vyplývá z návrhu testu a je provedena v rámci dle předlohy, neovlivní.

5.2 Výhody a omezení automatizovaného testování

Stejně jako jiný přístup a jiná technika testování i automatizované testování má své nesporné výhody, ale i nevýhody, které je třeba brát v potaz.

5.2.1 Výhody

5.2.1.1 Využití pro regresní testování

Jedná se o nejčastější využití, díky tomu jsou opakované spuštěné testy méně nákladné. Úsilí vyvinuté na provedení regresních testů je minimální. Pokud je třeba provést změny, jedná se zpravidla o drobné úpravy. To je dáno faktem, že testy byly aplikovány na předchozí verzi software.

5.2.1.2 Spouštění více testů častěji

Nespornou výhodou automatizace testů je možnost provést test v kratší době a provedení v kratšího počtu testů.

5.2.1.3 Provádění testů, jež by bylo těžké či nemožné provést manuálně

Díky užití nástroje pro automatizaci testování je možné například simulovat souasně vyvolávání testovaného systému více uživateli najednou. Dále je možné kontrolovat věci, které není možné ověřit při manuálním testování. Například při testování grafického rozhraní dojde ke stisknutí určitého prvku. Výsledkem je spuštění události, kterou nelze okem zkontrolovat. Pomocí testovacího nástroje je však toto možné.

5.2.1.4 Efektivní využití zdrojů

Místo provádění testů, které je možné provádět automaticky, se mohou testéři zabývat testy vyžadující manuální provedení.

5.2.1.5 Konzistence testů

Opakovaná a rutinníinnost snižuje pozornost lidského testera. Toto je při automatizaci testů eliminováno. Testy jsou vždy provedeny stejným způsobem. Nemůže tedy dojít ke zkreslení výsledků prováděných testů.

5.2.2 Nevýhody

5.2.2.1 Delší čas přípravy

Příprava testu pro automatizované spuštění je časově náročnější než příprava a provedení stejného testu manuálně. Z časového hlediska se automatizace testů vyplácí až při opakovaném provedení připravených testů.

ava testu vyřaduje delší dobu. Navíc je zde potřeba
počítat s dalšími náklady spojenými s pořízením nástroje. Tím je myšlena například licence za
využívání testovacího nástroje a poplatky za jeho aktualizaci.

5.2.2.3 Omezená působnost testů a potřeba údržby

Tyto testy jsou citlivější na změny provedené v testovaném software. Každá změna
grafického rozhraní a jiného prvku může vést k nutnosti automatizovaný test upravit a
zaktualizovat. Taková změna může u testu prováděného manuálně stát bez potřeby
navržený test upravit.

Příkladem může být i prostá změna umístění tlačítka. Tester vždy zmáčkne odpovídající
tlačítko, pokud je ovšem v automatickém testu implementováno stisknutí tlačítka jako
stisknutí na určité souřadnice, pak zde dochází k problému a test je třeba upravit.

5.2.2.4 Kvalita automatizovaných testů vychází z návrhu testů

Zde je třeba si uvědomit, že automatizované testy pouze strojem vykonávají testy, které před
automatizací byly prováděny manuálně. Kvalita a efektivita těchto testů vychází z návrhu
manuálních testů. Pokud jsou tyto testy neefektivně navrženy, pak je vhodné se automatizaci
vyhnout a nejdříve tyto testy upravit. [2]

5.2.2.5 Nutná znalost práce s nástrojem

Pro přípravu automatizovaných testů musí autor takovýchto testů dobře znát prostředí
nástroje, který je pro automatizaci určen. Je zde tedy vyšší požadavek na kvalifikaci testera.

5.3 Problémy s automatizovaným testováním

Automatizované testování vyřaduje určitou úroveň provádění testů, tím je myšleno
například držení konzistentní dokumentace. Pokud samotné testy, jež se mají poté vykonávat
automatizovaně, nejsou příliš kvalitní, pak není vhodné je popisovat. Nejdříve je nutné
zlepšit samotný přístup k testování.

Od automatizovaného testování není možné očekávat nalezení vysokého počtu chyb, pokud je
takový test proveden a nedojde k nalezení chyby, pak jeho operativní spuštění nemá žádný
efekt. Přesto však testy, které neodhalují chyby, nejsou bezcenné. Minimálně nám říkají, že
nedošlo k žádným změnám v kvalitě testovaného software.

Pokud ovšem automatizované testy projdou bez chyby, není to důkazem o bezchybnosti
software. Je výrazně doporučováno kromě automatizovaných testů provádět i testy manuální.

drflbu. Pokud zm na v prost edí i funkcionalit
í manuáln , potom je vhodné se zamyslet, zda není

vhodn j-í od automatizace t chto test upustit.

Automatizované testy nemohou nahradit práci testera. ěinnost takového testu je mechanická a pokud není v testu my-ěleno na n jakou situaci i kontrolu, která by m la být provedena, pak tato je tato chyba p ehlédnuta. Automatizovaný test provede pouze to na co je napsaný, nikdy nebude um t více.

Je také t eba vzít v úvahu fakt, ěe pokud automatizovaný test objeví chybu, která nelze nasimulovat manuáln , pak ani toto není považováno za chybu, protože b ěnému uřivatelí se nem ěe poda it chybu vyvolat.

5.4 Testovací nástroje

V sou asné dob jsou nástroje pro automatizaci test na vysoké úrovni. ada z nich nabízí, krom ěmořnosti si ur ítý test pouze nahrát, vývojové prost edí s podporou vlastního jazyka, díky kterému m ěte nahrané testy upravovat nebo vytvá et testy nové od samotného po átku. Tyto nástroje uřl nabízí vlastní referen ní p íru ky a manuály, ěmořnosti takovýchto nástroj jsou skute n bohaté. Jednotlivé nástroje se ov-ěm od sebe li-í nejen ěmořnostmi a funkcemi, které podporují, ale i platformami a typy aplikací, na které je ěmořné tyto nástroje pouřít. Jeden nástroj lze vyuřít pro aplikace postavené na technologii Java, ale nepodporuje testování klient/server aplikací. Jiný je ěmořné pouřít pro klient/server aplikace a pro testování webových slufieb, ale podporuje pouze aplikace postavené na platform .Net. Mezi dal-í kritéria pro výb r nástroje bych uvedl:

5.4.1 Způsob identifikace grafických prvků

Pro svou ěinnost pot ebuují nástroje identifikovat grafické prvky aplikace i software, aby byly schopni je vyuřít a simulovat tak ěinnost testera nebo uřivatele. Identifikace m ěe probíhat podle jedine něho názvu prvku, který m ěe být íselný i slořený z et zce znak . Prvky mohou být také identifikovány pomocí sou adnic nebo kombinací t chto zp sob . Osobn se domnívám, ěe je dobré vyřadovat nástroj, který umí prvky i oblasti aplikace identifikovat pouze na základ obdrřených sou adnic. Je to z toho d vodu, ěe m ěe dojít k situaci, kdy je t eba simulovat nap . pohyby my-í v ur íté oblasti, která ov-ěm neobsahuje p ímo n jaký prvek.

zených ve vstupních a výstupních polích

dy je nap íklad simulována innost uřivatele a v pr b hu testu jsou nap íklad vypo ítávány r zné hodnoty, které je nutné dále pouřít. Bez této mořnosti bychom nemohli takový test provést.

5.4.3 Možnost zápisu do externího souboru nebo databáze

Tato mořnost je d leřitá p í logování (pr b řném zaznamenávání inností za b hu testu) nebo nap íklad p í kontrolování pr b řných výsledk apod.

5.4.4 Pořizovací cena nástroje

Cena nástroje nám m ře ovlivnit výsledné náklady na testování. Vzhledem k tomu, ře se prost edky vynaložené na automatizaci vrací ař po opakovaném spu-t ní t chto test je d leřité brát v potaz í toto hledisko.

5.4.5 Přehlednost a intuitivní ovládání

Testovací nástroj m ře být vybaven velkým množstvím uřite ných a propracovaných funkcí. Pokud ov-em je práce s nástroje obtířná a nep ehledná, pak to prodluřuje a znemořuje í p ípravu test .

5.4.6 Úroveň dokumentace

Pro správnou a efektivní práci s nástroje je dobré ádn prostudovat dodanou dokumentaci. Pokud je dokumentace nekvalitní, pak nem ře být ani následná práce s nástrojem tak efektivní, jako by tomu bylo u dokumentace kvalitní, srozumitelné a lokalizované.

io testování, jehož účelem není automatizace

regresního testování, ale určítá simulace chování uživatele. Tento typ testovacího nástroje se nazývá testovací nebo pokusná opice. Cílem náhodného testování je simulace reálného chování uživatele, ale nejen to. Hlavním přínosem náhodného testování je provést akce v aplikaci v takovém sledu, ve kterém by ho pravděpodobně běžný uživatel nepoužil. Tím nám máme odhalit neobjevené chyby.

6.1 Myšlenka pokusné opice

ŠPokud bychom měli po milion let k dispozici milion opic, které by neustále „bušily“ do milionu klávesnic, statisticky vzato by mohly nakonec napsat třeba nějakou Shakespearovu hru, Jiráskovo Temno nebo nějaké jiné velké dílo. [1]

6.2 Kategorie opic

Testovací opice rozdělujeme dle Rona Pattona do 3 kategorií. [1]

6.2.1 Hloupá opice (Dumb monkey)

Jedná se o typ nejjednodušší opice. Tato opice neví o testovaném software vůbec nic a její klikání je zcela neorganizované. Pokud například posílá znaky na výstup, pak jsou zasílány i na místech, kde není žádný vstup očekáván. Přesto, může se nám zdát, že takováto opice je naprosto neúčinná, právě tento typ objevuje velmi často závažné chyby a takové chyby, je-li vedou ke zhavarování programu. Výhodou těchto opic je, že jejich tvorba je velmi snadná a rychlá s porovnáním s jejich š inteligentnějšími kolegy.

6.2.2 Polointeligentní opice (Semi brilliant monkey)

Tento druh opice se liší ve 2 základních a velice zásadních bodech. Prvním rozdílem je fakt, že opice svoje úsilí směřuje pouze na cílené testovanou aplikaci a jednotlivé kroky jsou zaznamenávány. Pokud tedy dojde k havárii software, je možné dohledat, jaký sled kroků toto způsobilo. Druhým rozdílem je, že opice rozpozná, že došlo k havárii a ukončení činnosti software. Díky tomuto upozornění, je možné ušetřit spoustu času, kdy opice nekliká dále již naprosto bez smyslu, ale je možné ji znovu spustit a objevit další chyby.

6.2.3 Inteligentní opice (Brilliant monkey)

Tato opice je již velice sofistikovaným nástrojem. Jejími hlavními výhodami je fakt, že tato opice ví, kde se nachází, co na tomto konkrétním místě může udělat, kam může přejít, kde již byla a jestli to, co vidí je správné. Nemusí se proto zaměřit pouze na hledání závažných chyb způsobujících havárii software, ale může provádět i klasické testy, pakliže jsou v jejím programu naimplementovány.

ní

test nebo osoba testera. Stačí v daný okamžik spustit test a zkontrolovat, zda odhalil nějakou chybu. Stejně tak není třeba přizpůsobit testovací prostředí. To by v podstatě bylo na škodu, nebo hlavní myšlenka náhodného testování by v takovém prostředí byla potlačena. Vývoj hloupých opic je levný a lze jej zvládnout za krátkou dobu. Noel Nyman uvádí, že v Microsoftu vývoji hloupých opic pro určitou aplikaci nevěnují více než 30 minut. [5]

Další výhodou náhodného testování spatřuji v přeskočení přípravy testovacích dat (data potěbná jako vstupy zadávaná během testu). To nám velice urychluje přípravu samotného testování. Vezmeme-li v úvahu, že často získat potřebná data pro zamýšlený test je často náročné, není-li potřeba a provedení samotného testu, pak zde objevujeme velkou úsporu času i nákladů.

6.4 Nevýhody náhodného testování

Náhodné testování je určitá doplňková metoda. Je sice pravdou, že díky této technice je možné objevit řadu chyb a to i velice závažných, ale stále je zde potřeba využít i jiné způsoby testování. Slabým bodem je i reprodukce chyby. Z toho důvodu si dovoluji tvrdit, že pešlivé zaznamenávání průběhu testu a jednotlivých provedených úkonů je nejméně tak důležitá jako test samotný. Za nevýhodu to považuji z toho důvodu, že pokud se nám povede objevit určitá chyba a neměli bychom dostatečné množství informací k jejímu nasimulování, pak jsme zbytečně vynaložili prostředky. Zde je tato situace navíc ještě složitější oproti klasickému automatizovanému testování a to kvůli náhodnému generování jednotlivých úkonů.

Dále si musíme uvědomit, že opice k objevení potřebuje dlouhou dobu. Výjimkou nejsou testy běžící několik hodin nebo den. Ani potom ovšem nemáme zajištěno, že k objevení nějaké chyby nebo chyb dojde.

Nutností je pečlivá analýza činností prováděných v průběhu testu. Oproti klasickému testu prováděnému manuálně dle určitého testovacího scénáře, kdy přesně víme jaký krok nebo sled kroků chybový stav vyvolal, je zde potřeba pečlivě analyzovat log pro simulaci chyb.

výláník [5], kde se podílil o své zkušenosti

s užíváním náhodného testování ve společnosti Microsoft. V době, kdy tento výláník vyšel, zastával Noel pozici Test inženýra. Jeho výklad není pouze teoretický, jak je tomu v případě Rona Pattona, ale uvádí i tipy pro využívaní náhodného testování, jeho možné problémy a chyby, kterým je dobré se vyhnout.

7.1 Rozdělení opic

V článku [5] jsou uvažovány pouze 2 typy opic:

- Chytré opice (Smart monkeys)
- Hloupé opice (Dumb monkeys)

7.1.1 Chytré opice

Chytré opice mají určitou znalost o tom, jakým způsobem využívat uživatelské rozhraní a mají povědomí o stavech a reakcích, které mají nastat po jejich úkonu.

7.1.2 Hloupé opice

Hloupé opice se chovají jinak než jejich chytré varianty. Nemají ponětí o možných stavech aplikace nebo o platných a neplatných vstupech do software.

7.2 Příprava chytré opice

Vytvořit chytrou opici je hodně nákladné, může se stát, že náklady vydané za vytvoření chytré opice jsou vyšší než užitek z nalezených chyb. Nevýhodou k tomu může být i fakt, že takto vytvořenou opici je velmi těžké ať nemohlo využít pro jiný projekt i testování jiného software. Proto pokud jsme se rozhodli pro testování chytrou opici vytvořit, je vhodné ji využít maximálně, aby se tak vynaložené prostředky vrátily zpět ve formě objevených chyb a tím i ušetřených nákladů na opravy chyb v pozdějších fázích vývoje.

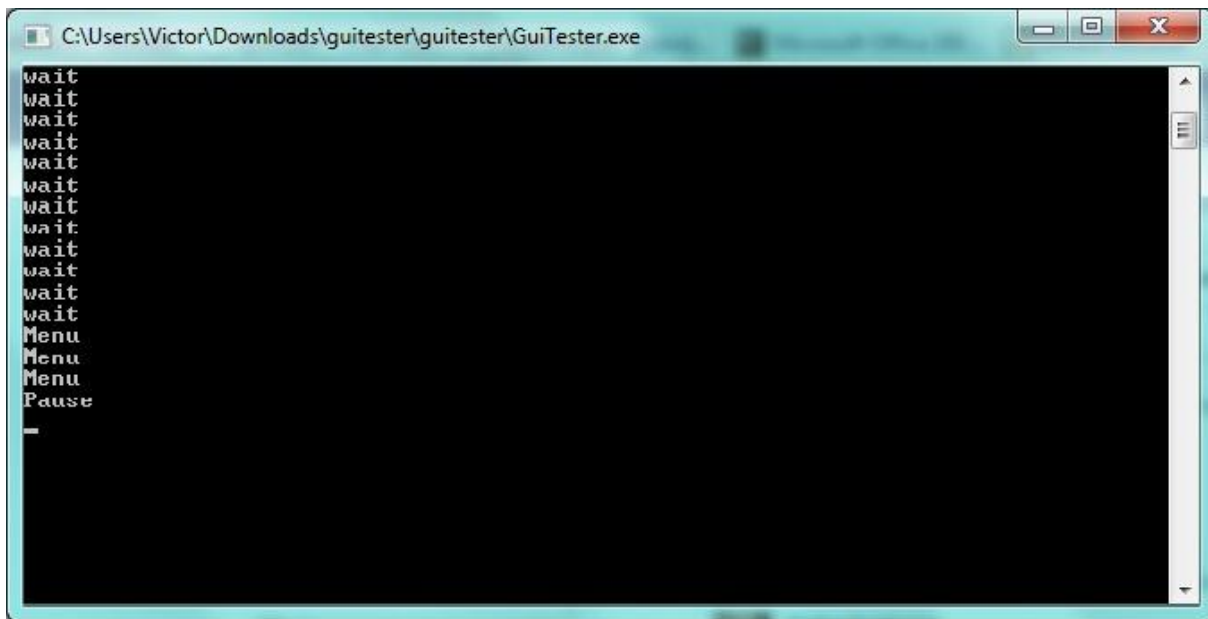
7.3 Příprava hloupé opice

Hloupé opice jsou nesrovnatelně levnější než opice chytré. Hloupá opice může testovat téměř všechny aplikace, které běží na daném operačním systému. Nejlepšími výsledky je dosaženo u hloupých opic, které mimo pouhého náhodného klikání, mají i určité minimální znalosti o zajímavých oblastech k otestování nacházejících se v okně aplikace.

7.4 Správný výběr

Náhodné testování by určitě nemělo být jediný přístup k testování software. Opice neznají dostatečně testované aplikace a neobjeví mnoho chyb. Avšak chyby, které objeví, jsou často závažné a vedoucí ke spadnutí aplikace a tedy nejzávažnější chyby.

program, jehož autorem je Luigi Poderico a svůj
výtvar, GUI tester, nechává k volnému stažení na stránkách www.poderico.it/guiterster. [9]
Tato opice provádí náhodné kliky v rámci daného okna a výsledky zapisuje na konzoli.
Nicméně se nemůže jednat o polointeligentní opici, nebo tato opice bez problémů testovanou
aplikaci vypne a pokračuje se svými náhodnými kliky.



```
C:\Users\Victor\Downloads\guiterster\guiterster\GuiTester.exe
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
Menu
Menu
Menu
Pause
-
```

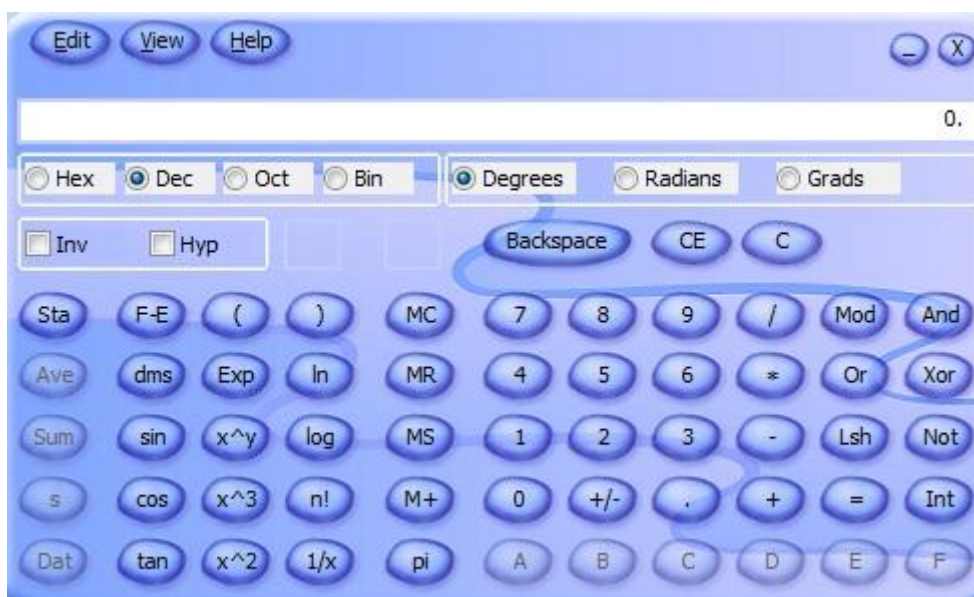
Obr. 8 – Výpis opice vytvořené Luigim Podericem (zdroj: Victor Espinoza)

9.1 Cíle experimentu

Cílem tohoto experimentu je aplikovat na níže popsanou aplikaci techniku náhodného testování. O očekávaným výstupem plánovaných testů je získání představ o efektivitě vytvořených opic a o přínosech této metody testování v praxi.

9.2 Testovaná aplikace

Za předmět svého experimentu jsem si vybral aplikaci Microsoft Calculator Plus [10].



Obr.9.2 - Microsoft Calculator Plus (zdroj:Victor Espinoza)

Tento kalkulátor kromě matematických operací umožňuje požitání v různých číselných soustavách, převod jednotek nebo různé způsoby zobrazení uživatelského rozhraní.

Grafické rozhraní se skládá z tlačítek (button), přepínačů (radiobutton) a ovládacích rámečků (checkbox).

V první fázi experimentu jsem se nejdříve nechal inspirovat již zmíněným nástrojem *Guiterster* od Lugiho Poderica. Bohužel výsledky, lépe řečeno průběh testu pro mě nebyl uspokojivý. Hned po první minutě testování došlo k opuštění testované aplikace. Situace se při opětovném spuštění testu opakovala a v několika případech došlo i k ukončení aplikace. Použil jsem se tedy z uvedených nedostatků a rozhodl se vlastní opice vzdát takovým způsobem, aby k uvedenému chování nedocházelo nebo aby k němu docházelo minimálně.

o zjistit, zda testovaný software obsahuje chyby.

K objevení chyb použijí technik náhodného testování, konkrétně hloupou a polointeligentní opici. Pokud se v testované aplikaci objeví chyby, pak je součástí výstupu testu i informace o průběhu testu. Díky tomu je pak možné chybu dostatečně analyzovat a popípadě zreprodukovat.

9.4 Vytvořené opice

Pro nastíněný experiment jsem se rozhodl vytvořit 2 druhy opic, hloupou a polointeligentní opici.

9.4.1 Hloupá opice

Pro účely testování má opice k dispozici název okna, jeho umístění a rozměry. Nemá žádné znalosti o prvcích v daném okně, které má k dispozici ani o stavech aplikace. Dokonce nekontroluje, zda je okno testované aplikace stále otevřené. Souadnice každého kliknutí jsou uloženy do externího souboru nebo do databáze.

9.4.2 Polointeligentní opice

Tato opice je vzdělanější než její hloupá verze v tom smyslu, že má k dispozici seznam prvků, které může v daném okně použít. Dále opice nemůže zavolat okno aplikace. K tomu by došlo pouze v případě neočekávaného chování, tedy chyby. Typ akce a použitý prvek je spolu s časem provedení akce zapsán do externího souboru.

9.5 Algoritmus hloupé opice

9.5.1 Formulace úlohy

Provést stanovený počet kliknutí v okně aplikace Microsoft Calculator Plus.

9.5.2 Analýza úlohy

9.5.2.1 Vstupní údaje

- Celkový počet kliknutí
- Horizontální a vertikální souadnice pro kliknutí

9.5.2.2 Výstupní údaje

- Použité souadnice uložené v externím souboru i databázi

9.5.3 Analýza

Simulovat kliknutí myší na obdržené souadnice. Tuto činnost opakovat dokud není proveden stanovený počet kliknutí.

2. Získej informace o souřadnicích a rozměrech okna spuštěné aplikace
3. Proveď kliknutí na obdržení souřadnicích
4. Zapiš použité souřadnice do určeného média
5. Byl proveden požadovaný počet kroků?
 - a. Ano – Ukonči činnost
 - b. Ne – Vrať se na bod 3

9.6 Popis připraveného testu

Tato opice nebude mít žádné znalosti týkající se grafických prvků aplikace, které by mohla využívat. Jediné dostupné informace bude aktuální umístění okna a jeho rozměry, tedy šířka a výška. Opice bude provádět kliky na jakékoli místo. To znamená, že může využít jakýkoli grafický prvek, ale také se může stát, že dojde ke kliknutí na oblast, kde je pouze prázdný prostor okna aplikace. U každého provedení kliknutí jsou uloženy použité souřadnice.

K vytvoření takovéto opice bychom měli mít nástroj možnosti:

- Simulovat kliknutí na stanovené souřadnice
- Získat informace o určeném okně
- Zapsat použité souřadnice na externí médium

9.7 Analýza algoritmu polointeligenční opice

9.7.1 Formulace úkolu

Použití prvku grafického rozhraní aplikace Microsoft Calculator Plus, tuto činnost provádět v předem stanoveném rozsahu.

9.7.2 Analýza úlohy

9.7.2.1 Vstupní údaje

- Celkový počet operací k provedení
- Jednoznačný identifikátor prvku a typ akce, který je třeba provést

9.7.2.2 Výstupní údaje

- Typ provedené operace a prvek, se kterým byla operace vykonána
- Informace o nedostupnosti aplikace

9.7.2.3 Analýza

Simulovat dostupné operace s danou aplikací. Tuto činnost opakovat ve stanoveném počtu.

1. Spust' testovanou aplikaci
2. Je aplikace dostupná a požadované okno aplikace aktivní?
 - a. Ne – zapiš tuto skutečnost do určeného média
 - b. Ano – proved' určenou činnost a zapiš informace o provedení do určeného média
3. Byl proveden požadovaný počet operací?
 - a. Ne – vrať se na krok 2
 - b. Ano – ukonči aplikaci

9.8 Popis připravovaného testu

K vytvoření polointeligentní opice je třeba programově otestovat její vlastnosti, aby mohla být za polointeligentní považována.

Tato opice bude znát prvky, které může využít. Dále bude v testu, zda je okno testované aplikace zobrazené a jestli je aktivní. Každý provedený krok je zaznamenán do určeného média, v tomto případě se jedná o externí textový soubor, kam je zapisováno ve formátu CSV. Jednotlivé hodnoty jsou oddělovány znakem středník (š ; õ). Pokud by v průběhu testu došlo k zavření okna aplikace, pak je i tato situace do externího souboru zapsána a test je ukončen. Tato opice tedy nebude klikat na místa aplikace, kde se nenachází žádný funkční prvek, ale pokaždé při svém úkonu využije dostupný ovladač.

Pokud potěbuji vytvořit výše popsanou opici, pak bych měl použít nástroj možnosti:

- Uchovávat seznamy tlačítek, například, zaškrtávacích rámečků a kláves ke stisknutí
- Simulovat kliknutí na tlačítka, zaškrtnutí například nebo zaškrtávacího rámečku a stisk klávesnice
- Zapisovat do externího souboru
- Kontrolovat zda je určené okno otevřené a zda je aktivní

1. leflité si vyjasnit, co a jakým zp s obem budeme testovat. To je d leflitý pro ur ení jaké techniky, metody a postupy testování budeme aplikovat. V p ípad , fle je nutné pouflít i n jaký specializovaný software, pak je d leflité práv z t chto údaj vycházet a ur it tak, který nástroj je pro dané pot eby nejvhodn j-í.

Na základ pot eb, které vyplývají z popisu algoritmu testovacích opic, jsou jasné pot eby, které je t eba zváflit p í výb ru nástroj pouflitých pro testování.

10.1 WinTask

10.1.1 Doporučení

WinTask jsem se rozhodl pro svou bakalá skou práci pouflít na základ doporu ení Joe Strazzere, který je významným p isp vatelem diskusního fóra *sqaforum.com* a zastává pozici manaflera pro zaji- ování kvality (QA Manager) ve spole nosti Newriver. V minulosti zastával stejnou pozici ve spole nostech Dun & Bradstreet nebo FairMarket Inc. Vystudoval Bentley Collegue Graduate school a University of Lowell. Krom toho, fle p ispívá do r zných diskusních fór, vede i soukromý blog www.strazzere.blog.com, jeho fl prost ednictvím sdílí r znorodé informace z oblasti testování software. Doporu ený nástroj dle jeho vlastních slov pat í v posledních letech k jeho nejoblíben j-ím.

10.1.2 Představení

Tento nástroj pochází z dílny francouzské spole nosti Taskware, která existuje jifl od roku 1997. Samotný nástroj byl poprvé p edstaven sv tu v roce 1998 a byl vytvo en pro Windows 95. Postupem ásu byl nástroj upravován a modifikován a v sou ásné dob je v jeho poslední verzi podporována i verze Windows 7.

Prvotní ú elem nástroje byla automatizace úkon v prost edí Windows. Pozd ji se rozvojem internetu nástroj obohatil i o automatizaci úkon v prost edí webových stránek a aplikací.

Nástroj je vyufllíván zejména k:

- Automatizaci úkon na osobních po íta ích
- Úkon m na serveru
- Automatizaci regresní test webových a desktopových aplikací

10.1.3 Vybavení

Nástroj je uzp soben k ufllívání lidmi, kte í nemají fládnou zku-enost s programováním.

K tomuto slouflí nahrávací mód, díky kterému jsou pot ebné skripty automaticky generovány v jazyce vyufllívaném prost edím.

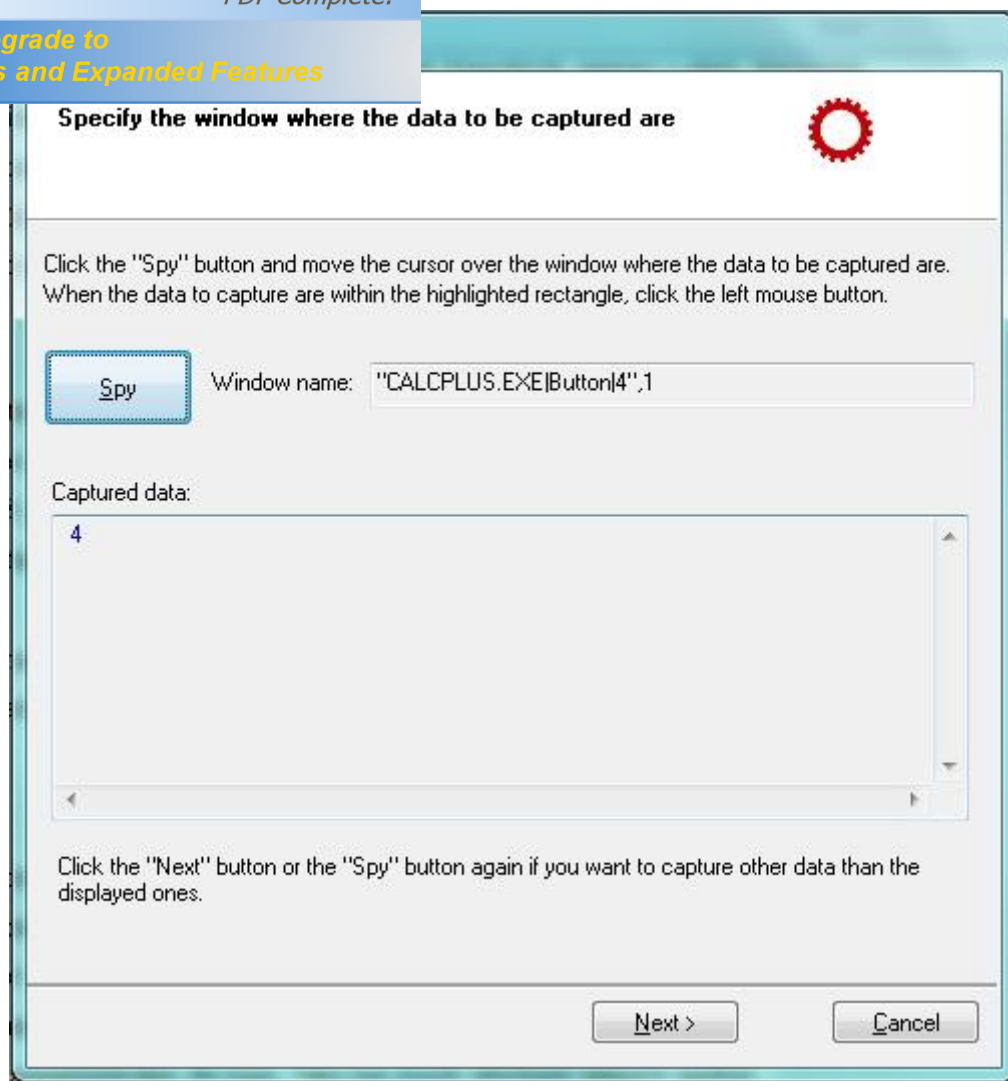
Wiz WinTask i možnost použít vlastního
jazyka je velice podobná jazyku Visual Basic.

WinTask má v sobě sadu již zabudovaných funkcí, nutno dodat že množství jej nelze srovnávat s jiným, plnohodnotným programovacím jazykem, ale i přesto je díky těmto funkcím možné provádět velice užitečné úkony. Kromě klasických operací s okny, kurzorem myši a simulace stisku kláves, je schopný získávat hodnoty určitých HTML elementů, text ve specifikovaném okně i umoci změnit toku programu a podporuje cykly. Užitečných a použitelných funkcí je více, nicméně mě zaujala jednoduchá práce s externími soubory a velice snadné připojení k databázi.

10.1.4 Vytvoření opice

K vytvoření polointeligentní opice jsem využil vývojové prostředí, které je součástí nástroje. Tato opice má znalosti o všech prvcích, které může použít. Dále kontroluje, zda okno testované aplikace není zavěšeno. V takovém případě již dále nepokračuje v testu, ale do logu zapisuje informaci o ukončení aplikace. Do logu také zapisuje všechny provedené operace.

V první části programu jsem definoval všechny prvky, které lze využít. K tomu jsem využil datovou strukturu seznamu. Jednotlivé prvky aplikace jsem identifikoval pomocí nástroje *Spy Wizard*.



Obr. 10.1.4a – WinTask Spy Wizard (zdroj: Victor Espinoza)

Tento nástroj je ve WinTask součástí základní výbavy a jeho účelem je právě pomoc při rozeznávání grafických prvků. Identifikované prvky jsem rozdělil dle typu do datových prvků obecně známých jako kolekce i seznam. Protože aplikace přijímá jako vstupy i stisknutí určitých kláves, zahrnul jsem do možných akcí i stisky kláves. Seznam těchto kláves obsahuje všechny klávesy, které vyvolávají nějakou reakci aplikace. Ostatní klávesy nejsou do seznamu zahrnuty.

Protože obsažené grafické prvky se skládají z checkbox, tlačítek a radiobutton. Bylo třeba pro každou tuto operaci nadefinovat vlastní metodu. Dále opice obsahuje metodu pro simulaci stisku kláves. Jednotlivé klávesy mají své vlastní označení k jejich správnému pojmenování a užití jsem použil uživatelskou příručku nástroje.

na do externího souboru. Zde je tato informace uložena ve formátu CSV. Tento formát jsem zvolil kvůli možnosti tento log dále zpracovat. O tom se rozepíší v další části práce.

```

Step 1: 5/5/2011 5:00:26 AM - ;keys;; <Num 6>
Step 2: 5/5/2011 5:00:29 AM - wasn't found radiobutton: Byte
Step 3: 5/5/2011 5:00:29 AM - ;radiobutton;; Bin
Step 4: 5/5/2011 5:00:30 AM - ;keys;; n
Step 5: 5/5/2011 5:00:30 AM - ;checkbox;; Hyp
Step 6: 5/5/2011 5:00:31 AM - ;checkbox;; Inv

```

Obr. 10.1.4b – Výpis testu (zdroj: Victor Espinoza)

Z obrázku je zřejmé, že záznam v logu se skládá z:

- Číslo kroku v daném testu
- Data a času, kdy byl určený krok testu vykonán
- Typu provedené akce
- Názvu prvku, který byl pro danou akci užit

a p ehledn . Nástroj jako celek má intuitivní ovládání.

Knihovna obsažených funkcí, oproti jiným nástroj m není nijak zvlá-t bohatá, ale i p esto nástroj obsahuje pro svoje ú ely v-echny pot ebné funkce.

10.1.6 Klady

10.1.6.1 Jednoduchost

Tento nástroj je velice jednoduchý a intuitivní. Práce s nástrojem se nau í velice rychle a snadno. P esto v-ak obsahuje ádu funkcí, které z n j d lají uflite ný a -íroce vyuflitelný nástroj. Vyufliváný skriptovací jazyk je velice jednoduchý a poradí si s ním i uflivatelé bez p edchozích programovacích zku-eností.

10.1.6.2 Vybavenost

V plné verzi nástroje je zabudováno velké množství uflite ných funkcí, jefl napomáhají k automatizaci zamý-lených inností.

10.1.6.3 Modifikovatelný vzhled

Krom okna editoru je mofné si prost edí upravit dle vlastních pot eb, okno s knihovnou funkcí, panel kompilátoru i jednotlivé li-ty s menu.

10.1.6.4 Bohatá dokumentace a podpora

K software je dodávána bohatá uflivatelská p íru ka, referen ní p íru ka a dále je mofné tuto p íru ku nalézt p ímo v prost edí nástroje. Také je zde online podpora, kdy je mofné p ímo výrobce kontaktovat kafdý v-ední den od 3 do 11.30 dopoledne, a to formou chatu. K nástroji je také založeno fórum, kam p íspívají fanou-ci a uflivatelé nástroje. Moderáto i t chto fór jsou v t-inou zam stnanci Taskware. Toto fórum ov-em není p íli-vyuflíváno, více rad je mofné naleznout na testovacích fórech. Mimo to je pro uflivatele p ípravena áda manuál a tutoriál , jefl umofní se s nástrojem rychleji sfít.

Na webových stránkách výrobce je uvedeno, že nástroj –iroce podporuje řadu jazyků a neorientuje se pouze na anglický jazyk. Hned v prvních okamžicích, co jsem používal tento nástroj, jsem narazil na problém s podporou e-tiny.

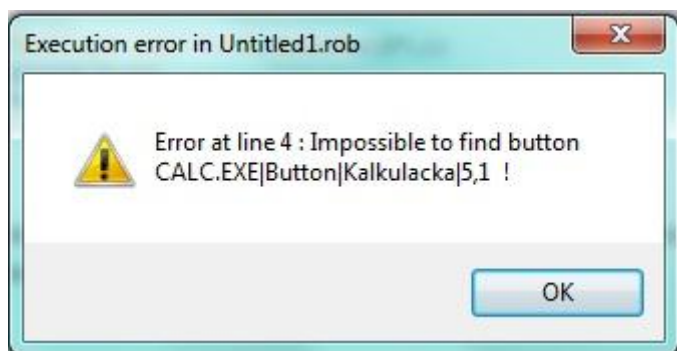
10.1.7.1.1 Nalezený problém

V prvních chvílích jsem si pro vyzkoušení rekord módu nástroje zkusil vytvořit jednoduchý skript, který na klasické Windows kalkulačce stiskne klávesy 1, 2 a 3. Výstupem byl tento zdrojový kód:

```
0001 Shell("C:\Windows\System32\calc.exe",1)
0002
0003 UseWindow("CALC.EXE|#32770|Kalkulačka|2",1)
0004     Click(Button,"|5")
0005     Click(Button,"|11")
0006     Click(Button,"|16")
0007
```

Obr. 10.1.7.1.1a – problémové volání okna aplikace (zdroj: Victor Espinoza)

Skript obsahuje spuštění kalkulačky a stisknutí kláves 1, 2 a 3. Po spuštění tohoto kódu pro vykonání došlo k zobrazení následující hlásky:



Obr.01.1.7.1.1b – chybové hlášení (zdroj: Victor Espinoza)

Z této hlásky je viditelné, že nástroj nemohl identifikovat příslušné tlačítko. Tento důvod je prostý, rekordér identifikoval okno, jako \$CALC.EXE|#32770|Kalkulačka|2\$, nicméně při vykonávání skriptu je \$Kalkulačka\$ zaměněna za \$Kalkulacka\$. Proto nelze tento skript vykonat.

10.1.7.3 Nevýhody při vývoji

Nejedná p ímo o nedostatky, ale spí-e o m j osobní dojem a nám t na zlep-ení. Celý jazyk je case insensitive, tedy je mofné psát libovoln velká a malá písmena a stále se bude jednat o stejná klí ová slova i prom nné. Toto m fle být pohodlné, ale nenapomáhá to k vytvá ení p ehledného kódu. Dále není zabudován klasický na-eptáva funkcí (ctrl + mezerník). Výrazným nedostatkem pro m byla absence zvýrazn ní vybrané prom nné skrz skript. Poté je dal-í lad ní programu t fl-í a zdlouhav j-í.

10.1.7.4 Skriptovací jazyk

Tento bod se týká nutnosti deklarovat využívané funkce a procedury na začátku skriptu. Zvlá-t programáto i, jeřl se pln ztotořl ují s my-lenkami OOP, zde zařlijí -ok. P ístup je zde procedurální a jazyk nepodporuje vytvá ení objekt a t íd. Dále jsou v jazyce podporovány pouze 4 typy prom nných: integer, znakové et zce, pole a systémové prom nné. Pole je nutné deklarovat v etn pouřlité velikosti, nejsou tedy dynamická. Navy-uje to výkonnost, ale je zde op t omezení, kdy je t eba nutné hlídat chyby zp sobené nesprávným využíváním index . Dovolím si tvrdit, fle s vý-e uvedeným budou souhlasit programáto i, kte í by tento nástroj využívali.

10.1.7.5 Cena

Plná verze produktu vyjde na 249 USD. Pokud chce uřlivatel využívat i mofnost stahovat aktualizace po dobu 2 let, pak je tato cena o 50 USD vy-í. Lite verze nástroje stojí 99 USD a 119 USD.

Pro tento nástroj jsem se rozhodl na základě vlastních zkušeností a znalosti tohoto nástroje z mého působení v Komerční bance na pozici Test koordinátor. Zde byl tento nástroj používán pro přípravu testovacích dat.

Výše uvedený nástroj se stal velmi populárním v oblasti vykonávání administrativních úkolů. Přestože je toto jeho nejznámější využití, lze díky němu zautomatizovat cokoli v prostředí Windows. Velmi obsáhlý skriptovací jazyk, jež je založen na syntaxi jazyk Basic, umí simulovat stisky kláves a kliky myši.

První verze AutoIt vznikla v roce 1998. Důvodem k jeho vzniku byla automatizace administrativních činností v rámci instalace software. V roce 1999 AutoIt tým vydal verzi, jež v sobě zahrnovala i funkce pro práci s okny a další funkce. Poté byla vydána i další verze a stále se doplňovaly nové funkce. V roce 2001 byl kompletní kód pro AutoIt kompletně přepsán do jazyka C++ a vývoj se zastavil do roku 2003, kdy byla vydána první beta verze AutoItv3. Po více než dalších 100 beta verzích vývojář i AutoIt vydali v lednu roku 2004 AutoIt v3. V lednu roku 2005 byla vydána verze 3.1.0, jež v sobě obsahovala i možnost vytváření GUI. Tato verze byla nejvýznamnější, neboť vynesla AutoIt do popředí mezi skriptování ve Visual basic, dávkové soubory a další populární skriptovací jazyky.

10.2.2 Vybavení

Tento software oproti nástroji WinTask v sobě obsahuje velice rozsáhlou knihovnu funkcí. Stejně jako ve WinTask je v něm obsažena možnost pro nahrávání jednotlivých testů, i zde dochází k automatickému generování kódu. V nástroji je obsaženo i vlastní vývojové prostředí. Toto prostředí je mnohem komfortnější nežli u předchozího nástroje. Dochází v něm ke zvýraznění syntaxe a proměnných, automatickému formátování zdrojového kódu a k dispozici je i náhled dostupných funkcí. Velice mě potěšil balíček služeb sloužící k práci s databází SQLite [11]. Proto jsem se rozhodl informace o průběhu testu zapisovat právě do této databáze.

Výhodou této databáze je nenáročnost a vysoká rychlost zápisu záznamů. Databáze je uchovávána ve formě textového souboru a není zde nutnost mít nainstalovaný databázový server. To mě nám vyhovuje, testujeme desktopovou aplikaci a nejsme omezeni rychlostí zápisu do databáze. Pro přístup do databáze využívám podpořený nástroj nainstalovaný do prohlížeče Mozilla Firefox. [12]

í testované aplikace. V následujícím kroku si opice pro svoji další potěbu uloží údaje o umístění okna aplikace a o jeho šířce a výšce. Tyto údaje jsou poté použity pro generování náhodných souřadnic, které určují místo kliknutí. Každé takové kliknutí je uloženo do databáze. Pro tyto účely jsem vytvořil jednoduchou tabulku *steps*. Tato tabulka obsahuje 6 sloupců. Ve sloupcích *date* a *time* jsou uloženy o datu aase, kdy byl konkrétní krok proveden. Ve sloupci *step* je uložena informace o pořadí úkonu v daném spuštění testu. Ve sloupcích *x* a *y* jsou uloženy souřadnice použité pro kliknutí. Opice nemůže v rámci testu ukončit aplikaci, tím je myšleno klasickým způsobem odepovědí na stisknutí ikony pro ukončení. Pokud však dojde k nějaké chybě, která způsobí, že je okno uzavřeno nebo neaktivní, pak je tato informace uložena do sloupce *description*. Stejně tak je uložena i informace o tom, že daný krok byl v pořádku proveden.

Tato tabulka byla vytvořena pomocí SQL dotazu:

```
CREATE TABLE steps (
date VARCHAR(20),
time VARCHAR (20),
step INT,
x VARCHAR (10),
y VARCHAR (10),
description VARCHAR (100),
PRIMARY KEY (date, time)
)
```

Data jsou pak v tabulce uložena následujícím způsobem:

date	time	step	x	y	description
29:06:2011	00:41:09:517	15	360	380	OK
29:06:2011	00:41:10:059	16	450	210	OK
29:06:2011	00:41:10:597	17	170	370	OK
29:06:2011	00:41:11:185	18	350	200	OK
29:06:2011	00:41:11:738	19	510	390	OK
29:06:2011	00:41:12:289	20	450	270	OK
29:06:2011	00:41:12:843	21	190	390	OK
29:06:2011	00:41:13:385	22	410	430	OK
29:06:2011	00:41:14:438	23	190	230	Error occured in previous step
29:06:2011	00:41:14:976	24	160	60	OK
29:06:2011	00:41:15:589	25	30	250	OK
29:06:2011	00:41:16:127	26	450	120	OK
29:06:2011	00:41:16:697	27	80	260	OK

Obr.10.2.3 – způsob uložení v databázi (zdroj: Victor Espinoza)

Jedná se o open source projekt, jeřl je k dispozici zdarma a to i pro komer ní ú ely. Je zde ovšem mofnost dal-í vývoj tohoto nástroje sponzorovat dobro innými dary.

10.2.4.2 Knihovna

Tento nástroj disponuje skute n bohatou základní knihovnou, jeřl pokrývá mnoho oblastí, od práce se soubory, přes matematické funkce po práci se zvukem a zvukovým záznamem.

10.2.4.3 Podpůrné nástroje

K tomuto nástroji je nabízena řada dal-ích nástroj , jeřl obohacují například práci se zdrojovými kódy. Jako příklad mohu uvést plugin Tidy, jeřl upravuje strukturu zdrojového kódu. Obsařen je i nástroj pro rychlý návrh formulářů.

10.2.4.4 Přenositelnost

Vytvořený skript je možné vygenerovat do souboru EXE. Poté je možné jej spustit i tam, kde není nástroj nainstalovaný.

10.2.5 Zápory

10.2.5.1 Omezená modifikovatelnost vývojového prostředí

Jednotlivé řásti vývojového prostředí nejsou tvořeny plovoucími panely, lze je maximálně změnit i zv t-ít. Dále se mi při práci neosv d ěla práce s knihovnami funkcí, přepínání mezi oknem nápovědy a editoru je nepohodlné.

10.2.5.2 Omezené uplatnění

Tento nástroj je možné vyuřlít pouze v prostředí OS Windows.

10.2.5.3 Zobrazení prvků

V některých případech může dojít k chyb ěm, jeřl může nastat například nená tením rozm ěrů i pozice okna, nebo v době, kdy program tyto údaje pořladuje, není je-t pot ebné okno dostate n vykresleno. S tímto je třeba při návrhu testů počítat a je nutné o-et ět to přímo v kódu.

10.2.5.4 Český jazyk

Po zkušenostech s nástrojem Wintask jsem i AutoIt podrobil testu, abych zjistil, jakým způsobem se vypo řádá s jistými specifickými znaky českého jazyka. Věchny české znaky nejsou v editoru podporovány, což může v některých případech ěnit potířle.

vání aplikací, ale svou bohatou výbavou vystačí i na pokročilou administraci celých počítačových sítí, což je vlastně i původní důvod jeho vzniku. Na tomto nástroji mě zaujala bohatost knihovny funkcí a i funkce samotné. Nebo jednotlivé funkce mají v každém případě kromě povinných parametrů i parametry nepovinné. Jako příklad bych uvedl vestavěnou funkci *MouseClicked*. Zde jsou povinnými parametry:

- Tlačítko, jehož stisk simulujeme
- Počet stisků
- Souadnice pro stisknutí tlačítka myši

Nepovinným, ale pro účely náhodného testování zajímavým parametrem je rychlost pohybu myši. Ta může být nastavena od plynulého pohybu po okamžitě umístění na místo kliknutí. Dále je zde kromě klasického stisku kláves možnost simulovat i jejich držení i jednoduše definovat stisknutí klávesy *n* kolikrát za sebou.

Obsaženy jsou i *While* i *For* cyklus, zahrnuta je i funkce pro práci s objekty. Pro řízení toku programu jsou v jazyku použity mimo jiné i *Switch* konstrukce a konstrukce *Select*. Při samotné tvorbě skriptu můžete použít náčrtávací, který by měl být součástí každého vývojového prostředí.

Tento nástroj mě osobně velice potěšil. Pro vývoj je účinný, bohatý na funkce a programovací jazyk je rychle osvojitelný. Díky dalším dostupným podpůrným nástrojům je snadné vytvořit přehledný a kvalitní kód. Přesto bych jej doporučil pokročilejším testerům se znalostmi programování.

10.3.1 Představení

Tento nástroj slouží k automatizaci různých opakovaných činností. Jeho využití je skutečně rozmanité. Používá se k simulaci práce s myší, stisk kláves, podepisování emailů, k zálohování a mnoha dalšími činnostem. Kromě těchto činností je používán i k automatizaci testování a příprav testovacích dat.

První beta verze nástroje byla vydána v listopadu roku 2003. Důvodem jeho vzniku bylo obohacení již zmíněného nástroje AutoIT o práci s klávesovými zkratkami. Sloučení těchto dvou nástrojů se nezdařilo a proto se autor AutoHotkey Chris Mallet rozhodl svůj projekt dále rozvíjet a to odděleně od vývoje AutoIt.

V roce 2010 oficiálně vznikl projekt zastávající další vývoj tohoto nástroje. Na jeho úpravách se může podílet každý dobrovolník.

10.3.2 Vybavení

Z vybraných nástrojů je AutoHotkey, což se týká výbavy podpůrnými nástroji, nejchudší.

Tento nástroj jako jediný z vybraných neumí automaticky automatizovat vybrané činnosti využitím nahrávacího módu. Disponuje ovšem nástrojem pro identifikaci oken a grafických prvků, což je pro účely, ke kterým byl nástroj vybrán dostačující. Jako editor skriptů slouží editor textových dokumentů.

10.3.3 Vytvoření testu

Test začíná spuštěním testované aplikace. Poté stejně jako v případě AutoIt dojde k získání pozice, šířky a výšky okna. Díky těmto údajům jsou poté vygenerovány dvojice souřadnic, které se poté využívají ke kliknutí v aplikaci. I v tomto případě jsou souřadnice ukládány do externího souboru ve formátu CSV.

```
1; 451; 28
2; 231; 218
3; 401; 106
4; 327; 25
5; 491; 87
6; 493; 127
7; 197; 116
8; 213; 217
9; 312; 212
10; 24; 62
```

Obr. 10.3.3 – formát CSV (zdroj: Victor Espinoza)

Tento log je úplně prostý, obsahuje pouze nejnutnější údaje. První číslice značí číslo kroku. Druhé číslo je horizontální souřadnice a poslední číslo je hodnota vertikální souřadnice.

10.3.4 Klady

10.3.4.1 Cena

Jedná se o open source software pro OS Windows, pořizovací náklady jsou nulové.

10.3.4.2 Velikost

Velikost software po instalaci nepřekračuje 7.5MB.

10.3.4.3 Přenositelnost

Jednotlivé skripty lze zkonvertovat do spustitelných EXE souborů, je-li umožněn jejich spuštění i bez nutnosti mít software nainstalovaný.

10.3.5 Zápory

10.3.5.1 Programovací jazyk

Tento zápor je založen na mém osobním pocitu, používaný jazyk mi přijde velice nepraktický a jeho jazykové konstrukce jsou zbytečně složitě.

10.3.5.2 Editor

Pokud chce uživatel vyvíjet za pomoci tohoto nástroje složitější skripty, pak jistě nemůže vystačit s textovým prohlížečem jako editorem. Editor do společnosti Scite v instalačním balíku ze záhadného důvodu není obsažen.



PDF
Complete

*Your complimentary
use period has ended.
Thank you for using
PDF Complete.*

[Click Here to upgrade to
Unlimited Pages and Expanded Features](#)

Když jsem zjistil možnosti tohoto nástroje a možnosti,

které přináší, rozhodně bych se nepřiklonil k používání jiného nástroje. Vzhledem k tomu, že pro účely náhodného testování není v tomto nástroji žádná zvláštní vlastnost, jež by umožnila efektivnější testování, využil bych nástroj odlišný.

Použitý skriptovací jazyk se mi zdá velice kostrbatý a práce s ním pro mě nebyla pohodlná.

Analýza výsledků jednotlivých testů. Každou opici jsem spustil třikrát. V první pokusu měla vykonat 1000 náhodných akcí a v druhém 3000 náhodných akcí. Tyto testy byly spuštěny zejména kvůli získání přehledu o délce testu a také z důvodu zjištění možných nedostatků vytvořených opic. Poté jsem spustil testy, které dle odhadu měly trvat přibližně 8 hodin.

11.1 Výsledky

Nástroj	Počet provedených operací	Byla objevena chyba	Doba trvání testu	Typ použité opice
WinTask	1000	Ne	7 minut 24 sekund	Polointeligentní opice
WinTask	3000	Ne	23 minut 33 sekund	Polointeligentní opice
WinTask	60000	Ano	7 hodin 12 minut 34 sekund	Polointeligentní opice
AutoIt	1000	Ne	10 minut 7 sekund	Hloupá opice
AutoIt	3000	Ne	30 minut 23 sekund	Hloupá opice
AutoIt	60000	Ano	8 hodin 14 minut 32 sekund	Hloupá opice
AutoHotkey	1000	Ne	20 minut 2 sekundy	Hloupá opice
AutoHotkey	3000	Ano	57 minut 41 sekund	Hloupá opice
AutoHotkey	25000	Ano	7 hodin 40 minut 17 sekund	Hloupá opice

Tab 11.1 – zaznamenání výsledků testů

11.2 Popis provedených testů

Každý nástroj byl spuštěn k otestování aplikace Microsoft Calculator Plus celkem třikrát. Rozdíl mezi jednotlivými testy byl počet okeřávaných akcí. Za akci je v případě hloupé opice považováno kliknutí myši, u polointeligentní opice to kromě kliknutí myši může být navíc odeslání vstupu z klávesnice. K testování byly v případě nástrojů AutoIt a AutoHotkey použity hloupé opice. V případě WinTasku se jednalo o opici polointeligentní.

uace kterou jsme se snažili vyvolat právě spuštěním

testovacích opic, se povedlo vyvolat v některých nástrojích. Tímto chováním je myšlena situace, kdy došlo například k zavření okna aplikace nebo okno bylo neaktivní (neodpovídalo).

Osobně se domnívám, že mohlo dojít například k obtížné výpočetní operaci, o které jsem se s vyvoláním chyby při výpočtu faktoriálu z příliš vysokého čísla nebo z jiného důvodu.

K odhalení skutečného důvodu chyby je ovšem třeba dále analyzovat logy jednotlivých opic.

11.4 Záznam průběhu testu

Zaznamenávání průběhu náhodného testování je dle mého nedílnou součástí testu, bez které by samotný test neměl vůbec žádnou vypovídací hodnotu. Vyděl bychom pouze, že chyba je v software obsažena, ale neměli bychom vůbec žádné informace o způsobu, jak tuto chybu znovu nasimulovat a tudíž ani jak ji odstranit. Důležitější je tedy zaznamenat všechny provedené operace a jejich parametry. Uchovat si informaci o pořadí a pořadí provedených kroků. Mít uloženy typ operace, její potřebné parametry a jejich výsledek.

U mnou vytvořených opic jsou využívány 2 způsoby uchování provedených akcí. V rámci prvního způsobu zaznamenávání se ukládá:

- Pořadí provedeného kroku
- Horizontální a vertikální souřadnice kroku

Pokud jako zapisovací médium slouží soubor, jsou pořadované údaje zapsány tak, aby odpovídaly zápisu ve formátu CSV. Jinak je ukládáno do databáze a uložená data jsou poté ve formátu CSV vyexportována.

Druhý způsob zaznamenávání průběhu testu je opět pomocí externího souboru a také ve formátu CSV. Ale jsou ukládány odlišné informace:

- Pořadí provedeného kroku
- O jaký typ akce se jedná
- Prvek, s nímž byla pořadovaná akce provedena

Chyby a jejich analýza

Protestní krok takový způsobem, aby opatrně došlo k nalezení chyb.

V případě náhodného testování musíme k reprodukci chyby nahlédnout do logů a na základě analyzovaných údajů nasimulovat nalezenou chybu.

Zde jsem se zamyslel, zda je opravdu třeba analyzovat log pro reprodukci chyby. Zda není možné nějakým způsobem i tuto činnost zautomatizovat a pouze v případě potřeby se do logů podívat. V případě mnoha vytvořených opic si toto dovedu představit.

Pouze jako příklad si představme, že opice ukládá všechny provedené akce do externího souboru. V tomto souboru jsou jednotlivé akce popsány ve formátu CSV. Pro příklad můžeme použít tento zápis:

```
1;button;8
```

Podobný zápis jsem ufil i vytvořeně polointeligentní opice. První hodnota (1) označuje pořadí kroku v rámci daného testu. Druhý parametr popisuje prvek (button), se kterým opice manipulovala. V tomto případě se jedná o tlačítko. Poslední hodnota označuje identifikátor grafického prvku. V ukázce se jedná o tlačítko, které na výstup kalkulačky vypisuje číslo 8.

Uchování prvních testů je důležité a přináší nám další informace a například i možnost chybu znovu nasimulovat. Proto je důležité logování věnovat stejnou pozornost jako při implementaci požadovaného chování opice.

i vyufití nástroje do-lo k chybovému stavu. P i

používání opic vytvořených ve zbylých dvou nástrojích do-lo vždy k předčasnému ukončení testu z důvodu zavření testované aplikace během testu.

Ze samotné povahy této testovací metody bych monkey testing použil i jako doplňkovou metodu k otestování aplikace. Tuto metodu bych využil jako doplněk k manuálním a automatizovaným testům.

Opici je třeba mít spuštěnou několik hodin. Proto je ideální vytvořenou opici spustit přes noc. Z toho nám vyplývá i efektivní využití zdrojů, neboť od spuštění testu po jejich ukončení není potřeba přítomnost testera. Ten poté ráno, po ukončení činnosti opice, začne analyzovat vygenerované výsledky a záznamy v logu.

Další výhodou je využití opic pro stress testy. Kdy kromě chyb můžeme otestovat i stabilitu testované aplikace. Tím získáme další informace o robustnosti software a takové o tom, jakým způsobem se vyrovnává s chybami, za podmínek které přesahují normální využívání software.

náhodné testování a přiblížit mu jeho místo v oblasti testování software.

V úvodní části práce jsem nejprve představil úcel testování v obecné rovině. Vysvětlili jsme si základní pojmy a přínosy a rizika testování. V další části práce si můžete představit vlastní představu o jednotlivých typech testů a získat o nich základní informace.

V rámci rozdělení jednotlivých testů jsme se zabývali i tématem automatizace testování.

Stejně jako jiná technika testování má i tato svoje přednosti a své nedostatky. O tom jsem se mohl také dozvědět.

Tím jsme se jistě přiblížili hlavnímu tématu této práce, k náhodnému testování. Zabývali jsme se myšlenkou testovací opice a dozvěděli jsme se o jejích druzích. Díky zkušenostem Noela Nymana jsme získali představu a využívaní náhodného testování ve společnosti Microsoft. Projekt Luigiho Poderica nám umožnil vidět hloupou opici v akci.

Díky provedenému experimentu jsme mohli získat představu, co může taková opice vykonávat za chvilky a jaké nedostatky mohou ovlivnit efektivitu testů prováděných technikou náhodného testování.

V další části práce jsme tedy definovali vlastní požadavky na vlastnosti a schopnosti opice. Poté jsme definovali algoritmus, který po danou opici využije. Algoritmy jsme analyzovali dva - jeden pro hloupou a druhý pro polointeligentní opici. Následně jsme tyto opice vytvořili v nástrojích vybraných právě pro tyto účely.

Při samotném užití opic na vybranou aplikaci jsme měli výsledky a poté hodnotili průběh testů na základě obdržených výstupů.

V poslední části práce jsme se zamysleli nad nutností správně pečlivě zaznamenávat průběh testu, neboť právě toto je poté využíváno při analýze a reprodukci objevených chyb.

V kapitole *Zhodnocení experimentu* jsme dospěli k názoru, že opice je vhodné spouštět přes noc a také to, že díky testovacím opicím zároveň testujeme i stabilitu aplikace.

Protože se domnívám, že náhodné testování není v České republice dostatečně známý pojem, považuji za nejvyšší přínos své práce možnost přiblížit také i toto téma a uvést ho do problematiky náhodného testování. Po přečtení mé práce byste měli také mít představu o výhodách, nedostacích a možnostech náhodného testování. Dále byste si mohli odnést i určitě znalosti potřebné k opravě vlastních testovacích opic. Tímto znalostmi míním určitý postup k návrhu opice, její implementace ve vhodném prostředí i nástroji a poté i zpracování výstupů a analýzu chyb.



Monkey testing	Anglický výraz pro náhodné testování. Viz náhodné testování
Tester	Osoba provádějící testy
Testování černé skříňky	Způsob testování, kdy není znám vnitřní chod software
Testování bílé skříňky	Způsob testování, kdy je testerovi dostupný zdrojový kód testovaného software
Automatizované testy	Testy prováděné pomocí software simulující činnosti člověka nebo jiných systémů
Hloupá opice	Jedna z kategorií do kterých jsou rozdělovány nástroje pro náhodné testování.
Polointeligentní opice	Jedna z kategorií do kterých jsou rozdělovány nástroje pro náhodné testování.
Inteligentní opice	Jedna z kategorií do kterých jsou rozdělovány nástroje pro náhodné testování.
Microsoft Calculator Plus	Kalkulátor využitý pro účely experimentu prezentovaného v rámci této práce
WinTask	Nástroj sloužící k automatizaci činností a úkonů, v této práci využitý pro vytvoření polointeligentní opice
CSV	Datový formát přenášející data ve formě textu.
TaskWare	Společnost vyvíjející nástroj WinTask.
SQLite	Velmi rychlá a výkonná databáze.
Mozilla Firefox	Internetový prohlížeč.
Structured query language	SQL Dotazovací jazyk využívaný pro práci s daty
EXE	Formát spustitelného souboru.
AutoIt	Nástroj pro automatizaci úkonů v prostředí Windows. Po účely této práce využit k vytvoření hloupé opice.
AutoHotkey	Nástroj pro automatizaci úkonů v prostředí Windows. Po účely této práce využit k vytvoření hloupé opice.

2. FEWSTER, Mark; GRAHAM, Dorothy. *Software Test Automation : Effective use of test execution tools*. New York : ACM Press, c1999. 574 s.
3. FARREL-VINAY, Peter. *Manage Software Testing*. New York : Auerbach Publications, c2008. 573 s.
4. NYMAN, Noel. Using Monkey Test Tools. *STQE Magazine*. 2000, 1, s. 18-21.
5. LIBÍK, Jakub. *Zlepšování softwarových procesů v oblasti testování*. [s.l.], 2010. 95 s. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.
6. FIURÁNEK, Tomáš. *Návrh metodiky testování webových aplikací*. [s.l.], 2010. 78 s. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.
7. POUZAR, Lukáš. *Automatizované testování*. [s.l.], 2007. 38 s. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.
8. CHO, Kyoung Young; MITRA, Subhasish; MCCLUSKEY, Edward J. Gate Exhaustive Testing . *IEEE Transactions on Computers*. 2005, 11, s. 7-14.
9. PODERICO, Luigi. *Gui tester* [online]. m2000 [cit. 2011-05-07]. Gui Tester. Dostupné z WWW: <http://www.poderico.it/guiterster/index.html>
10. *Download details: Microsoft Calculator Plus* [online]. c2011 [cit. 2011-05-07]. Microsoft Download Center. Dostupné z WWW: <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=32b0d059-b53a-4dc9-8265-da47f157c091>
11. Hwaci. *About Sqlite* [online]. 2001 [cit. 2011-05-07]. About Sqlite. Dostupné z WWW: <http://www.sqlite.com/about.html>
12. *SQLite Manager :: Add-ons for Firefox* [online]. 2011 [cit. 2011-05-07]. Firefox . Dostupné z WWW: <https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>
13. TaskWare. *TaskWare WinTask Publisher* [online]. c1997 - 2011 [cit. 2011-05-07]. About WinTask - Who We Are. Dostupné z WWW: <http://www.wintask.com/about-taskware.php>
14. TaskWare. *WinTask Features* [online]. c1997 - 2011 [cit. 2011-05-07]. Features. Dostupné z WWW: <http://www.wintask.com/features.php>
15. AutoIt. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 21 July 2005, last modified on 16 February 2011 [cit. 2011-05-07]. Dostupné z WWW: <http://en.wikipedia.org/wiki/AutoIt>

17. *An AutoIt / AutoHotkey comparison* [online]. 2000 [cit. 2011-06-23]. An AutoIt / AutoHotkey comparison. Dostupné z WWW: <view-source:http://paperlined.org/apps/autohotkey/autoit_and_autohotkey.html>
18. ZALLAR, Kerry. *Ractical Experience in Automated Testing* [online]. 2000 [cit. 2011-06-23]. Ractical Experience in Automated Testing. Dostupné z WWW: <http://www.methodsandtools.com/archive/archive.php?id=33>
19. BOROVCOVÁ, Anna. *Testování webových aplikací*. Praha, 2008. 140 s. Diplomová práce. Univerzita Karlova v Praze.
20. MYERS, Glenford J. *The Art of Software Testing*. New Jersey : John Wiley & Sons, Inc, 2004. 151 s.
21. BEIZER, Boris. *Software Testing Techniques*. [s.l.] : [s.n.], 1990. 550 s. ISBN 0-442-20672-0.