

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Testování grafických uživatelských rozhraní

BAKALÁRSKA PRÁCA

Andrej Gajdoš

Brno, 2012

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Andrej Gajdoš

Ďakujem doc. RNDr. Jiřímu Barnatovi, Ph.D. za odborné vedenie bakalárskej práce a poskytovanie cenných rád pri jej spracovávaní.

Zhrnutie

Táto bakalárska práca sa zaoberá testovaním grafických užívateľských rozhraní. Popisuje metódy a nástroje určené na testovanie užívateľských rozhraní aplikácií. Súčasťou práce je zvoliť vhodnú metódu a nástroj na otestovanie GUI nástroja DiVinE takým spôsobom, aby testovanie mohlo byť zahrnuté do repozitára projektu.

Kľúčové slová

testovanie, užívateľské rozhranie, skript, testovací prípad, traceability matrix, požiadavky, zachyt' /prehraj, test monkeys, Sikuli, DiVinE, EFG

Obsah

1	Úvod	3
2	Testovanie GUI	5
2.1	Požiadavky kladené na aplikáciu	5
2.2	Testovacie prípady	6
2.3	Odvodenie testovacích prípadov	7
2.3.1	Manuálne odvodenie testovacích prípadov	7
2.3.2	Automatizované odvodenie testovacích prípadov	8
2.4	Techniky testovania GUI	9
2.4.1	Vykonanie testovania GUI	10
3	Nástroje určené na testovanie GUI	13
3.1	Plne automatizované nástroje	13
3.1.1	GUITAR	14
3.2	Nástroje využívajúce zdrojový kód testovanej aplikácie	15
3.3	Nástroje využívajúce rozpoznávanie obrazu	16
3.3.1	Sikuli	17
3.4	Porovnanie nástrojov	18
4	Testovanie GUI nástroja DiVinE	19
4.1	Popis užívateľského rozhrania nástroja DiVinE	19
4.2	Požiadavky kladené na GUI nástroja DiVinE	19
4.3	Testovacie prípady určené na GUI nástroja DiVinE	19
4.4	Prepísanie testovacích prípadov do testovacích skriptov	20
4.4.1	API vizualizačných skriptov	20
4.4.2	Testovacie skripty	21
4.5	Vyhodnotenie testovania GUI nástroja DiVinE	30
4.6	Zahrnutie testovacích skriptov do repozitára projektu DiVinE	31
5	Záver	33
A	Obsah priloženého CD	37
B	Časť zoznamu požiadaviek na GUI nástroja DiVinE	39
C	Časť testovacieho prípadu	41
D	Testovací skript	43
E	Časť traceability matrix	47

1 Úvod

V dnešnej dobe je prirodzené, že produkty musia prejsť kontrolou kvality a procesom testovania pred ich použitím. Testovanie je široko uznávané ako kľúčová dôvera kvality, čo platí aj pre softwarové produkty a ich UI. Prvé aplikácie používali ako UI príkazový riadok. Užívatelia si pamätali príkazy, ktoré zadávali do príkazového riadku a pomocou nich ovládali aplikáciu. V súčasnosti väčšina aplikácií používa GUI¹, ktoré uľahčuje ovládanie aplikácie na rozdiel od CLI².

Ak má užívateľské prostredie prívlastok grafické, znamená to, že obsahuje grafické prvky alebo komponenty, ktoré sú tiež v angličtine nazývané *widgets*, ako napríklad panel nástrojov, dialógové okná, tlačidlá, textové polia, ikony, a iné. GUI komponenty slúžia na sprístupnenie logiky aplikácie, aby prístup k tejto „logike“ vyzeral užívateľsky prívetivo a aby komponenty interagovali s užívateľom. Jednotlivé prvky GUI reagujú na udalosti zadané od užívateľa a predstavujú problém pre návrhárov testov, pretože užívateľské prostredia sú zložité a tým sa stáva testovanie časovo a technicky náročné.

Cieľom tejto bakalárskej práce bolo popísať existujúce metódy a nástroje určené na testovanie grafických užívateľských rozhraní a následne aplikovať vhodnú metódu a nástroj na otestovanie GUI nástroja DiVinE tak, aby bolo možné testovanie zahrnúť do repozitára projektu. V rámci teoretickej časti som popísal problematiku testovania GUI, vhodné techniky a nástroje určené na testovanie GUI. V praktickej časti som napísal testovacie prípady, ktoré som prepísal do testovacích skriptov.

Výsledkom práce sú testovacie skripty, ktoré je možné zahrnúť do repozitára projektu DiVinE. K výsledku práce tiež patrí zoznam požiadaviek na GUI nástroja DiVinE, z ktorých vychádzajú testovacie prípady prepísané do podoby testovacích skriptov. Na záver bola zhotovená *traceability matrix* pomocou ktorej boli skontrolované požiadavky.

Teoretickú časť tvoria dve kapitoly. V prvej kapitole je popísaná problematika testovania, existujúce metódy a techniky, ktoré sa dajú použiť na testovanie GUI. Druhá kapitola je venovaná nástrojom určených na testovanie GUI.

Praktická časť pozostáva z jednej kapitoly, v ktorej je testované GUI nástroja DiVinE. Na začiatku sú napísané požiadavky, ktoré sú následne prepísané do testovacích prípadov. Väčšina testovacích prípadov je prepísaná do podoby testovacích skriptov. Na záver je zhotovená *traceability matrix*, podľa ktorej je zistené ktorá požiadavka je overená testovacím prípadom.

1. Graphical User Interface – grafické užívateľské rozhranie

2. CLI – Command Line Interface

2 Testovanie GUI

Testovanie GUI sa môže zdať na prvý pohľad jednoduché. Nie je potrebné písať žiadny zdrojový kód, jednoducho stačí kliknúť na grafický element UI a užívateľ vidí či sa aplikácia správa podľa očakávania. Takýto prístup by mohol fungovať s veľmi jednoduchým grafickým prostredím pozostávajúcim z niekoľkých tlačidiel a formulárov. Na druhej strane mnoho aplikácií, ktorých GUI sa môže zdať na prvý pohľad jednoduché, môžu obsahovať mnoho spôsobov interakcie s užívateľom a s nimi spojené kombinácie udalostí, ktoré je neefektívne testovať manuálne.

Memon [1] definoval GUI ako množinu widgetov $W = \{w_1, w_2, \dots, w_n\}$, kde každý widget predstavuje grafický prvok ako napríklad ikonu alebo textové pole. Každý widget má množinu vlastností $P = \{p_1, p_2, \dots, p_n\}$, čo môže byť farba, veľkosť, font a vlastnosti majú množinu hodnôt $V = \{v_1, v_2, \dots, v_n\}$, ako napríklad červená, výška, 8pt. Niektoré vlastnosti a ich hodnoty nemusia závisieť od nastavenia GUI aplikácie, ale od systémových parametrov. Widgets je možné rozdeliť na statické a animované. Statické elementy nemenia svoje vlastnosti spojito v určitom čase na základe udalosti, naopak animované elementy tieto vlastnosti meniť môžu. Komponenty UI, ktoré majú svoje parametre s určenými hodnotami v určitom čase môžu byť definované ako stav GUI. Takto môže každá udalosť, ktorá je asociovaná s GUI predstavovať funkciu, ktorá GUI dostane z jedného stavu do druhého.

Tak ako užívateľ interaguje s GUI, tak sa aj mení stav alebo logika aplikácie. Keď užívateľ vykoná udalosť na UI, napríklad kliknutím na tlačidlo, zavolá sa časť zdrojového kódu aplikácie, alebo sa sprístupní časť logiky aplikácie pomocou obslužnej metódy udalosti, ktorá sa v angličtine nazýva *event handler*. Udalosť je základná jednotka interakcie s GUI a aby mohol užívateľ používať aplikáciu, potrebuje vykonať udalosť alebo sekvenciu udalostí.

Testovanie GUI zahŕňa vykonanie sady úloh, porovnanie výsledkov z vykonaných úloh s očakávanými výsledkami a schopnosti zopakovať rovnakú sadu úloh viackrát s odlišnými udalosťami vyvolanými vstupnými perifériami, ako sú klávesnica alebo myš. GUI môže robiť testovanie zložitejšie kvôli nežiaducim udalostiam zadaných od užívateľa a viac spôsobov zadania vstupov pre aplikáciu.

2.1 Požiadavky kladené na aplikáciu

Na každú aplikáciu sú stanovené požiadavky, ktoré musí spĺňať. Môžu to byť typické požiadavky známe z bežných aplikácií, alebo vybrané požiadavky zo špecifikačného dokumentu, ktorý vo väčšine prípadov popisuje daný systém aj samotné požiadavky naň kladené.

Aby sa pri vývoji na niektoré požiadavky nezabudlo, je vhodné použiť *traceability matrix* [2], ktorá predstavuje dokument v tvare tabuľky. V tejto tabuľke sú zachytené vzťahy medzi dvoma dokumentmi. Najľavejší stĺpec predstavuje identifikátory jedného

2. TESTOVANIE GUI

dokumentu, môžu to byť napríklad testovacie prípady. Identifikátory pre horný riadok tabuľky môžu predstavovať jednotlivé požiadavky kladené na aplikáciu. Pomocou tejto matice sa dá určiť aké požiadavky na aplikáciu sú splnené a pomáhajú identifikovať nové úlohy pri plánovaní testovania [2].

Existuje viac druhov požiadaviek kladených na aplikáciu medzi ktoré patria napríklad systémové požiadavky, funkčné požiadavky, požiadavky výkonu. Preto bola z traceability matrix odvodená *requirements traceability* [3], ktorá sleduje vzťahy medzi požiadavkami.

Matice je taktiež možné využiť pri analýze testovateľnosti. Tsung-Hsiang Chang et al. [4] vytvorili maticu, kde jeden dokument predstavuje prvky GUI a druhý dokument predstavuje vlastnosti, ktoré je možné otestovať pomocou nástroja Sikuli. Pomocou tejto matice sa dá určiť, ktoré vlastnosti GUI komponent sa dajú testovať pomocou daného nástroja a podľa toho je možné prispôbiť testovanie aplikácie.

2.2 Testovacie prípady

Na testovanie aplikácie je vhodné použiť testovacie prípady, pričom ich navrhnutie je dôležité kvôli odhaleniu čo najviac chýb. Testovací prípad je dokument, ktorý popisuje určitú činnosť, ktorú je treba otestovať. Termín testovací prípad (v angličtine *test case*) popisuje konkrétne akcie alebo jednotlivé kroky, ktoré sú vykonané s určitou softwarovou komponentou a očakávané výsledky po vykonaní jednotlivých krokov [5].

Testovacie prípady, ktoré nejakým spôsobom spolu súvisia sa zoskupujú do *testovacej sady*, ktorá zvyčajne obsahuje dodatočné informácie, ako napríklad konfiguráciu, na ktorej sa testovalo alebo záver z výsledkov jednotlivých testov. Aké testovacie prípady budú do testovacej sady zaradené záleží na voľbe testera [6].

Každý testovací prípad je svojim spôsobom spojený s *test oracle*. Test oracle je mechanizmus, ktorý rozozná či testovací prípad skončil úspešne. Memon [7] rozdelil test oracle na dve časti. *Oracle information*, ktorá je použitá ako očakávaný výsledok a *oracle procedure*, ako mechanizmus na zistenie či sa oracle information zhoduje s výstupom testu. V užívateľských rozhraniach môže byť očakávaný výsledok snímka obrazovky, alebo pozícia a titulka okna. Test oracle je možné rozdeliť na viac druhov, napríklad *heuristic oracle*, ktorá poskytuje nejakú radu alebo návod na základe kontextu, alebo *statistic oracle*, ktorá využíva štatistické charakteristiky. „Ľudský úsudok“ je tiež druh test oracle, ktorý funguje tak, či stav aplikácie vyzerá korektne podľa úsudku testera [8].

Pri tvorbe testovacích prípadov sú nápomocné *testovacie scenáre*. Testovací scenár popisuje hypotetický príbeh ovládania aplikácie, ktorý je nápomocný pri odhade, alebo zmýšľaní o zložitosti aplikácie [9]. Tento „príbeh“ by mal byť dôveryhodný, motivujúci, mal by sa dať ľahko vyhodnotiť a nemal by byť založený na detailoch.

2.3 Odvodenie testovacích prípadov

Aby mohli testerí vytvoriť dobrú sadu testovacích prípadov, musia si byť istí, že ich sada testov pokryje všetku funkcionálnosť systému a všetky úlohy samotného užívateľského rozhrania.

GUI niektorých aplikácií sa môže zdať na prvý pohľad jednoduché, ale napríklad program Microsoft WordPad obsahuje viac ako 300 GUI operácií [10]. Preto je ťažké si predstaviť, kde sa môžu aplikácie „zaseknúť“. V testovacích prípadoch určených na GUI môže mierna modifikácia grafického prvku aplikácie výrazne narušiť štruktúru jednotlivých krokov, ktoré s týmto grafickým elementom súvisia bez toho aby sa menila funkcionálnosť aplikácie. Napríklad premiestnenie položky v menu môže výrazne narušiť štruktúru testovacích krokov v niektorých testovacích prípadoch.

Na určenie veľkosti domény a popisu testovaného systému sa dá použiť metóda *Model-based* testovanie, ktorá popisuje funkcionálnosť systému AUT¹. Model abstraktné popisuje vlastnosti a správanie AUT. Testovacie prípady, ktoré sú odvodené od tohto modelu sú abstraktné a preto nemôžu byť priamo vykonané. Aby mohli byť požadované testovacie prípady vykonané, je ich potrebné odvodiť z abstraktných testovacích prípadov [11]. Testovacie prípady sa dajú odvodiť pomocou heuristík, alebo sa dajú použiť diagramy určené na návrh aplikácie.

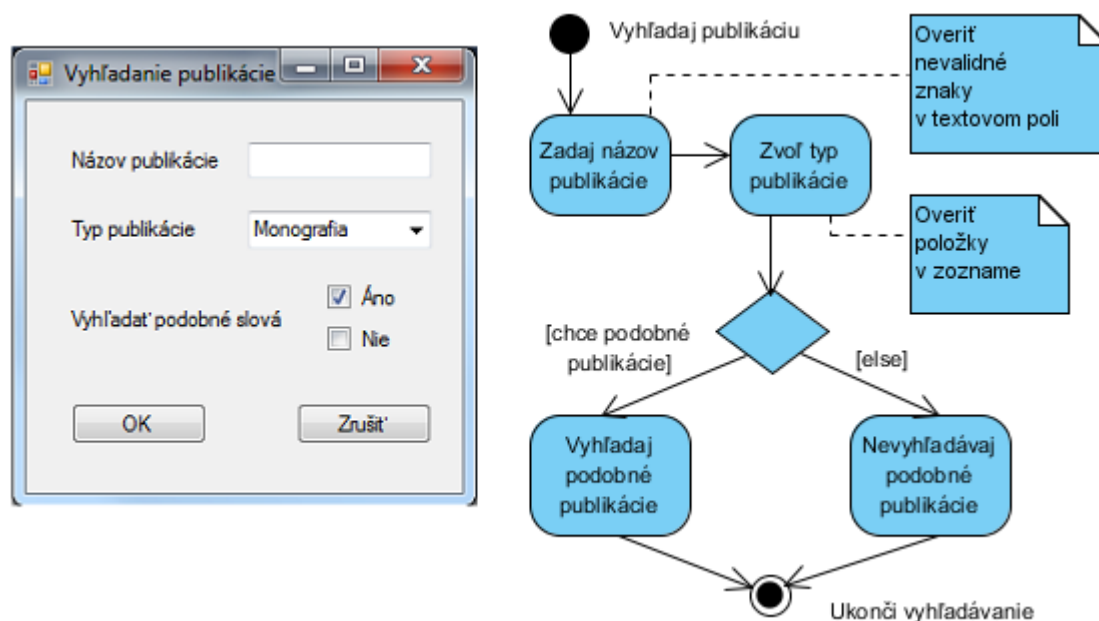
2.3.1 Manuálne odvodenie testovacích prípadov

Vieira et al. [12] popísali ako je možné odvodiť testovacie prípady z *UML diagramov*. Nasledovná časť textu popisuje odvodenie testovacích prípadov z návrhových diagramov, preto vychádza vo väčšej miere z tejto práce.

Diagramy prípadov použitia sa dajú použiť na popis vzťahu medzi testovacími prípadmi, ktoré sú pre aplikáciu špecifické a užívateľ, ktorý interaguje s aplikáciou na základe týchto testovacích prípadov. Diagramy aktivít umožňujú modelovať logiku aplikácie zachytenú testovacím prípadom. Sada diagramov aktivít celkovo predstavuje správanie, ktoré je pre aplikáciu špecifické. Dôvod použitia diagramov aktivít je predpoklad, že vyjadrujú ako môžu byť funkcionality aplikácie vykonané jednotlivo za sebou.

Pre účely testovania je potrebné stávajúce modely upraviť tak, aby boli zachytené informácie potrebné pre generovanie testovacích prípadov. Medzi tieto zmeny patrí premenovanie názvov aktivít v diagramoch aktivít podľa správania užívateľa, ktorá vedie k zlepšeniu definície aktivít užívateľa a zmenšenie abstrakcie modelu aplikácie. Druhá zmena znamená pridanie poznámok, napríklad v tvare dodatočných požiadaviek a vstupných dát k jednotlivým činnostiam v diagrame. Z upraveného diagramu aktivít je možné odvodiť sekvencie testovacích krokov, ktoré sa dajú spojiť do testovacieho prípadu.

1. AUT – Application Under Test



Obr. 2.1: Na obrázku je znázornená časť GUI a upravený návrhový diagram aplikácie. Pomocou upraveného diagramu je možné manuálne odvodiť testovacie prípady.

2.3.2 Automatizované odvodenie testovacích prípadov

Testovacie prípady sa dajú odvodiť algoritmicke, pričom je vyžadovaná výrazne menšia časť úsilia ako pri manuálnom odvodení testovacích prípadov. Medzi tieto modely patrí model odvodený pomocou dôkazov, logického programovania alebo *event-flow* model [11]. Z týchto modelov je *event-flow* model (ďalej iba EFG) vhodný na testovanie GUI, ktorý v nasledovnej časti bližšie popíšem.

Memon [13] v kapitole *Testing Graphical User Interfaces* popísal GUI ako EFG a EIG². Nasledovná časť textu je zameraná na popis týchto modelov a je z väčšej časti prevzatá z časti tejto kapitoly, ktorá popisuje tieto modely.

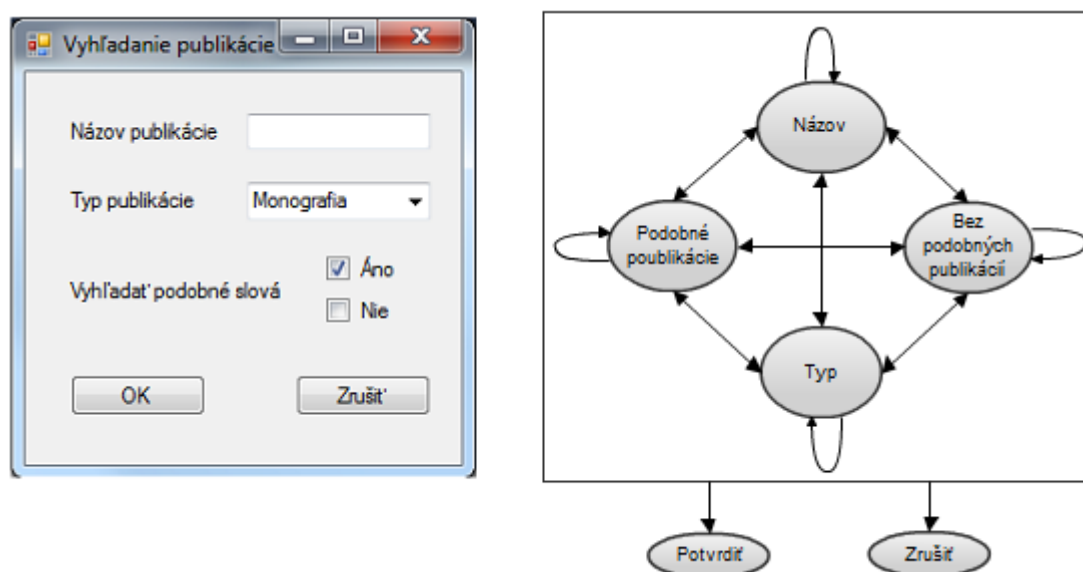
EFG reprezentuje všetky možné sekvencie udalostí, ktoré môžu byť vykonané na UI. V EFG je GUI ako graf v ktorom vrcholy reprezentujú udalosti a hrany reprezentujú vzťahy medzi udalosťami. Na zostrojenie EFG je potrebné identifikovať tieto vzťahy na základe typu udalosti. Napríklad udalosť otvorí dialógové okno, ktoré následne drží interakciu nad celým UI. Naopak môže nastať udalosť, ktorá otvorí nemodálne okno, ktoré ponúka ďalšie udalosti a neodrží interakciu nad celým UI aplikácie.

Prechádzaním EFG môžu byť testovacie prípady vytvorené rýchlo a automatizovane. Pri tomto spôsobe vytvárania testovacích prípadov treba brať do úvahy, že EFG je zložený z mnoho udalostí a môže obsahovať mnoho chybných udalostí. Napríklad udalosť

2. EIG – Event-interaction model

sa môže odvolať na obslužnú metódu, ktorá ale vôbec neinteraguje so zdrojovým kódom. EFG môžu byť použité pre generovanie testovacích prípadov, ktoré odhalia veľké množstvo chýb, avšak tieto grafy sú rozsiahle a poskytujú veľké množstvo testovacích prípadov.

Kvôli veľkému počtu udalostí, ktoré neinteragujú s logikou aplikácie vznikajú grafy s veľkým počtom uzlov a hrán, preto Memon definoval *event-interaction* model (ďalej iba EIG), ktorý je odvodený z EFG. Uzly v tomto grafe predstavujú udalosti, ktoré priamo interagujú s logikou aplikácie.



Obr. 2.2: Na obrázku je znázornená časť GUI a jeho EFG. Stavy *Potvrdiť* a *Zrušiť* môžu nastať po ľubovoľnom stave v štvoruholníku, ktorý reprezentuje zoskupené stavy.

2.4 Techniky testovania GUI

Na testovanie aplikácií sa dá aplikovať mnoho testovacích techník. V nasledovnej časti popíšem techniky, ktoré sú vhodné pri testovaní GUI.

Medzi tieto techniky patrí *Data-driven* testovanie, pomocou ktorej testovací skript využíva databázu z ktorej získava dáta. Takto je skript spustený opakovane podľa množstva dát v databáze. Táto technika je vhodná na overenie nekorektných dát, ktoré môžu byť zadane komponentám GUI.

Ďalšia technika sa nazýva *regresné testovanie*, ktorá sa používa na odhaľovanie nových chýb, ktoré sa môžu vyskytnúť počas vývoja nových verzií aplikácie, ako napríklad vylepšenie aplikácie, konfiguračné zmeny alebo záplaty systému. Hlavný úmysel regresného testovania je, aby vykonané zmeny v systéme nespôsobili nové chyby a ne-

2. TESTOVANIE GUI

zasahovali do inej časti systému. Pri regresnom testovaní je užitočné zhromažďovanie informácií z užívateľských profilov aktuálnej verzie aplikácie. Zhromaždené informácie pomáhajú určiť, či aplikácia funguje podľa očakávania a môžu byť použité v budúcom testovaní nasledovnej verzie aplikácie.

Pri odhaľovaní chýb je taktiež užitočná technika *Test Monkeys*. Táto technika alebo nástroj vykonáva náhodné akcie, ktorým môže priradiť množstvo odlišných vstupov. Pretože vykonanie všetkých akcií a vstupov je vykonaných automatizovane, môžu byť vykonané všetky možné kombinácie udalostí na GUI testovanej aplikácie.

Existuje viac druhov *Test Monkeys*, ktoré sa líšia tým v akom rozsahu vyberajú akcie a ich vstupy. Napríklad *Smart Monkey Testing* získava údaje z užívateľských profilov a na základe nich vyberá akcie a vstupy na testovanie [14]. Pri testovaní GUI ponúka *Test Monkeys* nasledovné výhody. Pretože nevie nič o GUI, môže byť ľahko použiteľný pri ďalších projektoch a môže byť použitý v akomkoľvek štádiu vývoja. Pri nájdení chyby vie určiť časť zdrojového kódu v ktorom chyba nastala. Na druhej strane, môžu náhodne vykonané akcie zasiahnuť do nastavení GUI, prípadne neočakávane zmeniť stav aplikácie, čo môže viesť k nepredvídateľným výsledkom.

Naopak tento prístup môže byť vhodný pri testovaní GUI aplikácií určených pre smartphony, pričom mnoho týchto aplikácií nedovolí užívateľovi meniť nastavenia GUI. *Test Monkeys* dokážu nájsť mnoho chýb, ale nevedia ju rozoznať, preto je tento spôsob testovania vhodný skôr ako dodatočné testovanie [15].

2.4.1 Vykonalie testovania GUI

Testovanie GUI môže byť vykonané pomocou „ľudskej inteligencie“ alebo pomocou počítačových nástrojov. Každé testovanie by malo byť postavené na ľudskej inteligencii, aj keď nie je praktické aby sa zaoberala opakujúcimi sa a nekreatívnyimi úlohami, ktoré môžu byť vykonané automatizovane.

Manuálne testovanie znamená testovanie aplikácií bez pomoci automatizovaných nástrojov. Tento spôsob testovania je časovo náročný a je vyžadované dodatočné ľudské úsilie. Manuálne testovanie je vhodné pre testovanie použiteľnosti, pre otestovanie vysoko rizikových oblastí systému, alebo pre akceptačné testy. Tento spôsob má tiež svoje obmedzenia čo sa týka ľudskej dôkladnosti, ktorá nemusí byť pri testovaní dostatočujúca. Taktiež je náročné udržať „krok“ manuálneho testovania pri projektoch, ktoré sú vyvíjané v rýchlom tempe, kde sa logika aplikácie mení takmer každý deň.

Automatizované testovanie je spôsob testovania za pomoci automatizovaných nástrojov. Tieto nástroje dokážu simulovať človeku podobnú interakciu s aplikáciou ako napríklad pohyb myši alebo stisk klávesy. Pre každé rozsiahlejšie GUI sa vyžaduje automatizované testovanie, čo ale nie je ľahká úloha. Techniky, ktoré sa používali v minulosti pri testovaní CLI, sa ťažko prenášajú do GUI. Medzi tieto techniky patrí napríklad zachyt'/prehraj.

Technika *zachyt'/prehraj* funguje tak, že sú zachytené udalosti pri používaní apli-

kácie, ktoré sú uložené do súboru vo vhodnom formáte. Každá zachytená udalosť je „namapovaná“ a následne uložená vo forme testovacieho skriptu, kde sekvencia týchto udalostí predstavuje testovací prípad. Takto vytvorený testovací prípad môže byť automaticky spustený pomocou namapovaných testovacích skriptov bez dodatočného úsilia. Výhodou tohto spôsobu je rýchlosť vytvorenia testovacích prípadov, následné prevedenie a opakovania vytvorených testov. Nevýhoda tejto metódy spočíva v dodatočnom úsilí pri zmene logiky alebo dizajnu aplikácie, kde musí byť v niektorých prípadoch prepísaná veľká časť testovacích skriptov [16, 17].

Automatizované testovanie GUI je v niektorých prípadoch efektívnejšie a spoľahlivejšie, ponúka možnosť väčšieho pokrytia testov a je nákladovo lacnejšie ako manuálne testovanie. Na druhej strane mnoho nástrojov určených na automatizované testovanie GUI vyžaduje vytvorenie testovacích skriptov, ktoré musia byť pri zmenách testovanej aplikácie upravené a je potrebné ich ladiť ako každý zdrojový kód programovacieho jazyka.

3 Nástroje určené na testovanie GUI

V súčasnosti sú dostupné nástroje, ktoré sú určené pre určité platformy a pre určitý typ aplikácií, medzi ktoré patrí napríklad CompuWare TestPartner, Mercury Interactive Tools, IBM Rational Test Tools, Segue a ďalšie. Ich cieľom je zrýchliť vývojový cyklus aplikácií a nájsť čo najviac chýb v testovanom systéme. Umožňujú vytvárať a spravovať testy, určiť typ testu ako napríklad regresný, funkčný alebo GUI, zobrazíť chyby a dodať správne opravy o chybách.

Mnoho organizácií tieto nástroje nepoužíva, lebo sa ťažko aplikujú na ich produkty, alebo sú drahé. Aby sa niektoré organizácie vyhli manuálnemu testovaniu, začali vyvíjať vlastné testovacie nástroje alebo používajú niekoľko testovacích nástrojov na rôzne účely pre potreby testovania.

Nástroje určené na testovanie GUI majú jeden spoločný cieľ a to automatizované ovládať aplikáciu, čo nie je ľahká úloha. Podľa toho ako tieto nástroje pristupujú k danej problematike sa dajú rozdeliť do troch skupín, ktoré v nasledovnej časti tvoria tri podkapitoly.

3.1 Plne automatizované nástroje

Do prvej skupiny patria plne automatizované nástroje, kde sa vyžaduje minimálne úsilie pri testovaní aplikácie. Tieto nástroje tvoria testovacie prípady automatizovane, ktoré následne vykonajú. Celý postup testovania funguje na základe znalosti zdrojového kódu. Medzi tieto nástroje patrí test monkeys, alebo nástroj GUITAR, ktorému venujem jednu podkapitolu. Medzi test monkeys patrí napríklad *monkey event generator*¹, ktorý je zahrnutý v Android SDK². V prípade použitia týchto nástrojov stačí spustenie testu a akým spôsobom sa vykoná je možné nastaviť podľa parametrov testovania.

Medzi hlavné výhody nástrojov tejto skupiny patrí rýchlosť prevedenia testov a minimálne úsilie pri tvorbe a prevedení samotných testov. Medzi hlavné nevýhody patrí to, že tieto nástroje vytvárajú testovacie prípady automatizovane, z čoho vyplýva že nemusia pokryť veľkú časť testovanej aplikácie a tak výsledky testov nemusia byť dostatočne dôveryhodné. Medzi ďalšie nevýhody patrí neschopnosť testovania komponent GUI, ktoré sú naprogramované pomocou iných programátorských knižníc ako testovací nástroj podporuje.

1. <http://developer.android.com/guide/developing/tools/monkey.html>

2. Software development kit – sada nástrojov určená na vývoj software

3. NÁSTROJE URČENÉ NA TESTOVANIE GUI

3.1.1 GUITAR

Nástroj GUITAR³ je *open-source* výskumný projekt určený na automatizované testovanie GUI. Podporuje testovanie aplikácií vytvorených v *.NET*, *Java Swing* a *Java SWT*. Nástroj GUITAR sa skladá zo štyroch zabudovaných nástrojov, ktoré sa spúšťajú pomocou dávkových súborov a svoju činnosť vykonávajú pomocou JVM⁴.

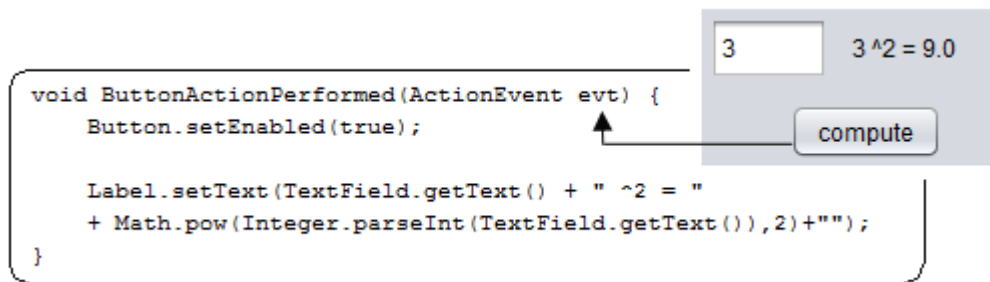
Prvý nástroj je *GUI Ripper*, ktorý automatizovane ovláda a interaguje s GUI testovanej aplikácie pre potreby nájdenia vzťahov medzi komponentami GUI. Výstup tohto nástroja sú artefakty, ktoré obsahujú informácie o testovaných widgetoch.

Ďalší nástroj je *Model Converter*, ktorý spracuje artefakty z nástroja GUI Ripper a prevedie ich do EFG testovanej aplikácie vo forme XML⁵ súboru. Na webovej stránke projektu⁶ sú grafické ukážky EFG. V skutočnosti nástroj nepodporuje prevod XML formátu do značkových formátov určených na grafickú reprezentáciu a grafické ukážky vytvorili vývojári nástroja GUITAR pomocou vlastných nástrojov.

Ďalší nástroj je *Test Case Generator*, ktorý prechádza EFG za účelom vytvorenia sekvencie udalostí, ktoré je možné vykonať na UI a tak vytvorí testovací prípad. Užívateľ má možnosť zvoliť maximálny počet a dĺžku vygenerovaných testovacích prípadov.

Posledným nástrojom je *Replayer*, ktorý spustí testovacie prípady na testovanej aplikácii a ako výstup sú informácie o vykonaných testovacích prípadoch.

Na ukážku je naprogramovaná jednoduchá aplikácia v jazyku Java. GUI aplikácie pozostáva z hlavného okna v ktorom sú komponenty textové pole, tlačidlo a *label*. Aplikácia funguje tak, že stlačením tlačidla je v komponente *label* zobrazený text predstavujúci „vypočítané číslo na druhú zadané v textovom poli“.



Obr. 3.1: Na obrázku je znázornené GUI aplikácie, ktoré je otestované nástrojom GUITAR. Časť zdrojového kódu reprezentuje udalosť stlačenia tlačidla, v ktorej sa vykoná výpočet hodnoty v textovom poli a je zobrazený v komponente *label*.

3. http://sourceforge.net/apps/mediawiki/guitar/index.php?title=GUITAR_Home_Page

4. JVM – Java Virtual Machine

5. eXtensible Markup Language – rozšíriteľný značkový jazyk

6. http://sourceforge.net/apps/mediawiki/guitar/index.php?title=The_EFG_galaxies

Predpokladá sa, že táto jednoduchá aplikácia je súčasťou zložitejšej aplikácie, v ktorej programátor zabudol ošetriť validáciu komponenty textového poľa. Ak je v textovom poli zadaná iná hodnota ako číslo a je zavolaná obslužná metóda tlačidla, časť zdrojového kódu v ktorom prebieha výpočet je vyhodnená výnimka `NumberFormatException()`. Vyhodenie výnimky znamená, že sa aplikáciou nepodarilo hodnotu textového reťazca previesť na číslo.

Aplikácia je otestovaná pomocou nástroja GUITAR, ktorej úlohou je zadať textovému poľu nekorektnú hodnotu, zavolať obslužnú metódu tlačidla a odhaliť vyhodnenú výnimku. Nástroj Replayer odhalil chybu v aplikácii, ktorú zapísal do logovacieho súboru. Časť logovacieho súboru so zachytenou výnimkou vyzerá podľa nasledovného výpisu:

```
... ERROR edu.umd.cs.guitar.util.GUITARLog - Uncaught exception
java.lang.NumberFormatException: For input string: ""
```

Podľa logovacieho súboru je možné zistiť, že výnimka bola vyhodnená pri zadaní prázdneho textového reťazca. V logovacom súbore je tiež výpis komponent a okno, ktoré boli zobrazené pri vykonaní testovacieho prípadu. V logovacom súbore sú tiež zaznamenané informácie z výstupov predchádzajúcich nástrojov. Tieto informácie v tvare artefaktov sú pre užívateľa nečitateľné. Preto nemusí byť v niektorých prípadoch jednoduché zistiť čo spôsobilo chybu pri vykonaní testovacieho prípadu.

Medzi výhody nástroja GUITAR patrí to, že na základe znalosti zdrojového kódu dokáže vygenerovať veľké množstvo testovacích prípadov, ktoré dokáže následne vykonať. Tento systém je automatizovaný, čím sa dá aplikácia otestovať za krátku dobu. Medzi hlavné nevýhody patrí to, že pri písaní tejto práce je nástroj vo vývojovej fáze *alfa*, čo znamená, že nástroj je v etape testovania a je nestabilný. Medzi ďalšie nevýhody patrí to, že celý projekt nie je zdokumentovaný a v prípade nájdenia chyby je možné vyhľadať chybu len pomocou artefaktov z jednotlivých nástrojov.

Nástroj GUITAR je v skorej fáze vývoja, zatiaľ nie je zdokumentovaný, nemá vlastné UI a je nestabilný. Aj napriek týmto nedostatkom má podľa môjho názoru potenciál stať sa v budúcnosti vhodným nástrojom na dodatočné testovanie GUI.

3.2 Nástroje využívajúce zdrojový kód testovanej aplikácie

Do druhej skupiny patria nástroje, ktoré využívajú skriptovacie alebo iné programovacie jazyky, pomocou ktorých nástroj simuluje správanie užívateľa. Tieto skripty je možné vytvoriť manuálne, alebo sa dajú vytvoriť pomocou techniky zachyt'/prehraj. Pre názornosť predpokladajme, že jeden testovací krok predstavuje funkciu v danom programovacom jazyku. Táto funkcia predstavuje udalosť vykonanú na komponente GUI. Táto komponenta slúži ako argument funkcie. Tento argument pre funkciu je definovaný podľa atribútov v zdrojovom kóde testovanej komponenty.

Nástroje druhej skupiny sú v súčasnosti najrozšírenejšie, sú to nástroje s vlastným vývojovým prostredím, až po jednoduché utility vo forme pluginov pre vývojové prostredia

3. NÁSTROJE URČENÉ NA TESTOVANIE GUI

a webové prehliadače. Medzi hlavné nevýhody patrí dodatočné úsilie pri tvorbe testov a zmene logiky alebo dizajnu aplikácie, kde v niektorých prípadoch je potrebná zmena veľkej časti testovacích skriptov. Tieto nástroje je možné rozdeliť podľa toho, či sú určené na testovanie webových alebo desktopových aplikácií. Ďalšia nevýhoda je podobná ako v nástrojoch prvej skupiny, čo znamená podpora testovaných technológií. Z internetových zdrojov je možné zistiť podrobný popis nástrojov a ich jednotlivé porovnania. Na nasledujúcej tabuľke je porovnanie niekoľkých nástrojov určených na testovanie internetových aplikácií. Informácie v tabuľke sú prebraté z jedného z internetových zdrojov.⁷

	WATIR	Selenium	Rational Functional Tester
Podpora prehliadačov	IE, FF, Safari	FF, Safari, Chrome, Opera, časť IE	IE, FF
Zachyt' / prehrať	Watir Recorder	Dostupný v IDE	✓
Operačný systém	Windows (IE, FF), Linux (FF), Mac (Safari, FF)	Mac, Windows, Linux	Mac, Windows, Linux
Výsledná správa testov	×	Zachytená obrazovka, logovacie súbory	✓
Programovací jazyk	Ruby	Vlastný jazyk Selenese, export do C++, Java, Ruby, Python, HTML	Java, VB, .NET
Cena	Zdarma	Zdarma	Záleží od licencie

3.3 Nástroje využívajúce rozpoznávanie obrazu

V minulosti vznikli rôzne projekty, ktoré boli určené na automatizované ovládanie GUI za pomoci rozpoznania obrazu, ale neboli určené na testovanie. Medzi tieto projekty patrí napríklad Triggers alebo VisMap. Mnoho podobných projektov v minulosti „trpelo“ dlhým spracovaním obrazu, pričom získať užitočné informácie zo systémovej obrazovky s rozlíšením 1024×768 pixelov trvalo niekoľko sekúnd [16]. Preto neboli tieto nástroje v minulosti rozšírené. V súčasnosti je bežný výpočtový výkon dostačujúci na získanie potrebných informácií zo systémovej obrazovky.

Do tretej skupiny patria nástroje, ktoré využívajú techniku *black-box* testovania, čo

7. https://spreadsheets.google.com/pub?key=tWKHlKxj3s8_1zisIRIJBjg

znamená, že testovanie prebieha bez potreby znalosti zdrojového kódu testovanej aplikácie. Tieto nástroje sa z veľkej časti podobajú nástrojom druhej skupiny, tiež používajú programovacie jazyky a testovacie skripty. Líšia sa hlavne spôsobom, akým vyhodnotia zhodu s testovanou komponentou. Využívajú počítačové videnie (v angličtine *computer vision*), čo je opak počítačovej grafiky, ktorá vytvára obrazové dáta z informácií popisujúcich zachytené objekty, zatiaľ čo počítačové videnie získava informácie zo zachyteného obrazu [18]. V týchto nástrojoch je ako argument funkcií v skriptoch použitá „zachytená systémová obrazovka“ alebo takzvaný *screenshot*. Zhoda so screenshotom a testovanou komponentou je vykonaná na základe algoritmov počítačového videnia.

V prípade použitia týchto nástrojov medzi hlavné výhody patrí to, že nie je potrebná znalosť zdrojového kódu testovanej aplikácie. Pomocou nástrojov tretej skupiny sa dajú otestovať aplikácie, ktoré využívajú ľubovoľné technológie na ľubovoľnej platforme. Medzi hlavné nevýhody patrí dodatočné úsilie pri zmenách GUI testovanej aplikácie a v niektorých prípadoch závislosť systému, na ktorom sú vykonané testovacie skripty. Aj keď sa dajú testovacie skripty spustiť na ľubovoľnej platforme, nastáva problém pri systémových zmenách, ako napríklad iná téma systémových okien. Medzi ďalšie nevýhody patrí neočakávané systémové zmeny ako napríklad vyvolané systémové okno, ktoré môže prekryť práve testovanú komponentu. Medzi ďalšie nevýhody patrí rýchlosť vykonania testovacieho prípadu, ktorá je výrazne menšia ako v nástrojoch prvej a druhej skupiny.

Medzi nástroje, pomocou ktorých sa dá otestovať GUI pomocou počítačového videnia patrí Eggplant, RoutineBot, Sikuli, Ranorex, T-Plant Robot, iMacros.

3.3.1 Sikuli

Pre potreby testovania nástroja užívateľského rozhrania DiVinE som zvolil nástroj Sikuli, ktorý v tejto časti stručne popíšem. Nástroj Sikuli slúži na testovanie GUI pomocou rozpoznania obrazu. Obsahuje API⁸ a IDE⁹ potrebné na písanie vizualizačných skriptov, ktoré automatizovane ovládajú GUI bez potreby znalosti zdrojového kódu testovanej aplikácie.

Skripty je možné vytvoriť pomocou API vizualizačných skriptov a syntaxe programovacieho jazyka Jython, ktorý dovoľuje spustiť kód programovacieho jazyka Python pomocou JVM. Jazyk Jython obsahuje všetky moduly, ktoré sú štandardnou súčasťou distribúcie jazyka Python a dovoľuje importovať ľubovoľnú triedu vytvorenú v programovacom jazyku Java.

Akcie udalostí vykonané pomocou myši alebo klávesnice sú vo vizualizačných skriptoch implementované pomocou triedy *Robot* v jazyku Java. Medzi nevýhody tohto nástroja patrí to, že akcie vyvolané skriptami držia interakciu nad systémom a nedá sa ovládať. Jediná možnosť je prerušiť beh skriptu pomocou klávesovej skratky Ctrl+Shift+C.

8. Application programming interface – rozhranie pre programovanie aplikácií

9. Integrated development environment – vývojové prostredie

3. NÁSTROJE URČENÉ NA TESTOVANIE GUI

Funkcie na porovnávanie obrazu sú implementované pomocou OpenCV¹⁰ knižníc počítačového videnia, v jazyku C++. OCR¹¹ algoritmy sú implementované pomocou knižnice tesseract-ocr.¹²

3.4 Porovnanie nástrojov

Podľa môjho názoru sa dá pomocou nástrojov druhej a tretej skupiny zaručiť určitá dôvera kvality voči testovanej aplikácii. Aký nástroj na testovanie tester zvolí, záleží od toho, akú aplikáciu je potrebné otestovať. Ak je potrebné otestovať komplexný systém, kde je potrebné vykonať veľké množstvo testovacích prípadov, je vhodné použiť nástroje druhej kategórie. Ak nie je potrebné vykonať stovky testovacích prípadov, je vhodné použiť nástroje tretej kategórie. V niektorých prípadoch je vhodné použiť kombináciu oboch skupín nástrojov. Výber nástrojov záleží od mnoho faktorov ako napríklad cena, nároky na výpočtový výkon, predpokladané zmeny v GUI testovanej aplikácie, podporované technológie. Nástroje prvej kategórie je vhodné použiť v každom prípade, ale sú vhodné skôr na dodatočné testovanie.

10. <http://code.opencv.org/projects/OpenCV/wiki/WikiStart>

11. Optical character recognition – optické rozpoznanie znakov

12. <http://code.google.com/p/tesseract-ocr/>

4 Testovanie GUI nástroja DiVinE

4.1 Popis užívateľského rozhrania nástroja DiVinE

Nástroj DiVinE slúži na formálny popis modelu pomocou vlastnej syntaxe *DVE*. GUI nástroja DiVinE umožňuje vizuálne zobraziť model v *Sequence Chart* paneli. Panel výstupu slúži na výpis informácií zo zabudovaných nástrojov. Ostatné panely slúžia na sledovanie stavu zásobníka, premenných a jednotlivých prechodov počas simulácie. Hlavné okno UI sa skladá z lišty menu, panela nástrojov, stavovej lišty a šiestich panelov. GUI obsahuje ďalšie komponenty ako zaškrŕavacie polia, textové polia, tlačidlá a dialógové okná určené na otvorenie, uloženie súboru a otvorenie palety farieb.

UI je možné rozdeliť do dvoch stavov. V prvom stave je možná editácia modelu, pričom je možné zobraziť panel editora a panel výstupu. Druhý stav nastane pri spustení simulácie modelu, pričom pri spustenej simulácii nie je možná editácia modelu. V tomto stave je možné zobraziť všetkých šesť panelov. Zabudované nástroje určené na popis a verifikáciu modelu je možné použiť v ľubovoľnom stave GUI.

4.2 Požiadavky kladené na GUI nástroja DiVinE

Na každé užívateľské prostredie sú stanovené požiadavky, ktoré musí spĺňať. GUI nástroja DiVinE tiež musí spĺňať určité požiadavky, ktoré sú napísané v podobe zoznamu. V tomto prípade ich hlavný účel je, aby po vykonaní jednotlivých testovacích prípadov boli poznačené požiadavky, ktoré sú splnené. Po dokončení všetkých testovacích prípadov je možné zistiť, na ktoré požiadavky sa zabudlo a upraviť, prípadne vytvoriť nové testovacie prípady. Na GUI nástroja DiVinE je napísaných 48 požiadaviek, ukážka zoznamu niektorých požiadaviek je v prílohe B. Požiadavky sú rozdelené do nasledovných kategórií:

- Obecné požiadavky, ktoré vo väčšej miere vychádzajú z požiadaviek bežných desktopových aplikácií
- Požiadavky na obecné klávesové skratky pre prácu s textom
- Požiadavky na jednotlivé GUI komponenty, ktoré obsahuje nástroj DiVinE
- Požiadavky na jednotlivé časti GUI, ktoré umožňujú prácu so súbormi, nastaveniami aplikácie, simuláciu a vizualizáciu modelu

4.3 Testovacie prípady určené na GUI nástroja DiVinE

Na testovanie GUI nástroja DiVinE sú odvodené testovacie prípady manuálne, kde každý testovací prípad je určený na určitú požiadavku, alebo podobnú skupinu požiadaviek.

4. TESTOVANIE GUI NÁSTROJA DiVINE

Neviem dopredu určiť, ktoré požiadavky je možné otestovať pomocou zvoleného nástroja. Preto som zvolil možnosť manuálneho spísania testovacích prípadov, pričom väčšina testovacích prípadov bude v ďalšej časti prepísaná do podoby testovacích skriptov. Pri tvorbe testovacích prípadov sa berie ohľad na počet testovacích krokov, aby bolo možné pomocou jedného testovacieho prípadu otestovať čo najviac požiadaviek bez zbytočných krokov v testovacom prípade. Štruktúra testovacích prípadov je nasledovná:

- Identifikátor
- Názov
- Účel
- Podmienky nutné k spusteniu testovacieho prípadu
- Jednotlivé kroky
- Očakávané výsledky z jednotlivých krokov
- Vykonanie testu, ktoré vypovedá o tom, ako dopadol testovací prípad
- Poznámky s dodatočnými komentármi testovacieho prípadu

V testovacích prípadoch je niekoľko krokov zoskupených do jedného kroku. Dôvod zoskupenia je, že niektoré sekvencie udalostí ako stlačenie sekvencie klávesových skratiek nepredstavujú udalosti, ktoré je potrebné zdokumentovať. Celkovo bolo napísaných 20 testovacích prípadov. Ukážka testovacieho prípadu je v prílohe C.

4.4 Prepísanie testovacích prípadov do testovacích skriptov

4.4.1 API vizualizačných skriptov

Testovacie skripty vo väčšej miere pozostávajú z objektov a funkcií API vizualizačných skriptov, preto v nasledovnej časti predstavím triedy, ktoré sú v skriptoch použité.

Triedy `Settings`, `App` a `Env` dovoľujú volať metódy, ktoré umožňujú meniť prednastavené hodnoty správania a nastavenia niektorých funkcií, taktiež umožňujú získať informácie o prostredí v ktorom sú skripty spustené.

Trieda `Screen` reprezentuje systémovú obrazovku. Jej metódy dovoľujú získať informácie o systémovej obrazovke. Každá inštancia triedy `Screen` má svoj identifikátor. Je to z dôvodu, že skript môže byť spustený súčasne na viacerých systémových obrazovkách.

Trieda `Region` predstavuje oblasť na systémovej obrazovke v tvare pravouhlého štvoruholníka. Má štyri atribúty `x`, `y`, `w`, `h` ktoré predstavujú dva body. Atribúty `x`, `y` predstavujú bod ľavého horného bodu oblasti a atribúty `w`, `h` predstavujú bod pravého dolného bodu oblasti.

Trieda `Pattern` predstavuje uloženú rastrovú grafiku v `.png` formáte. Každá inštancia triedy `Pattern` má atribút v tvare textového reťazca, ktorý predstavuje názov uloženého súboru. Trieda `Match` predstavuje oblasť na systémovej obrazovke a inštancie sú vytvorené ako návratová hodnota metód určených na vyhľadanie požadovaného objektu. Inštancie triedy `Match` majú rovnaké atribúty `x`, `y`, `w`, `h` ako objekty triedy `Region`. Navyše majú atribúty `score` a `target`. Atribút `score` môže nadobudnúť hodnotu 0 až 1, ktorá predstavuje veľkosť zhody s objektom `Pattern` a atribút `target` predstavuje bod, na ktorý sa majú zamerať funkcie, ktoré vykonávajú akcie pomocou kurzora myši.

Trieda `Location` predstavuje bod na systémovej obrazovke pozostávajúci zo súradnice `x`, `y`. Triedy `Key` a `KeyModifier` reprezentujú konštanty kláves.

V triede `Pattern` sú najčastejšie používané metódy `similar()` a `targetOffset()`. Po zavolaní metódy `targetOffset()` objektom triedy `Pattern` je návratová hodnota nový objekt triedy `Pattern`, s odlišným bodom na ktorý sa majú zamerať funkcie, ktoré vykonávajú akcie pomocou kurzora myši. Po zavolaní metódy `similar()` je taktiež návratová hodnota nový objekt triedy `Pattern`, ale podľa veľkosti zhody, ktorá je daná vstupným argumentom metódy.

Každý testovací skript je uložený vo forme adresára. Adresár obsahuje `.png` súbory použité v skripte, zdrojový kód vo forme `.py` súboru a súbor v značkovacom jazyku HTML¹, ktorý slúži na prezentáciu zdrojového kódu spoločne s rastrovou grafikou.

IDE nástroja Sikuli dovoľuje graficky zobrazíť objekty triedy `Pattern` a `Region` spolu so zdrojovým kódom skriptu. Objekty triedy `Pattern` môžu byť v skripte vytvorené vložením cesty k fyzickému súboru na disku, alebo pomocou zabudovaného nástroja, ktorý dovoľuje vytvárať objekty triedy `Pattern` zachytením systémovej obrazovky. Zabudovaný nástroj tiež dovoľuje volať najpoužívanejšie metódy `similar()` a `targetOffset()`. Na obrázku 4.1 je náhľad zabudovaných nástrojov IDE nástroja Sikuli.

4.4.2 Testovacie skripty

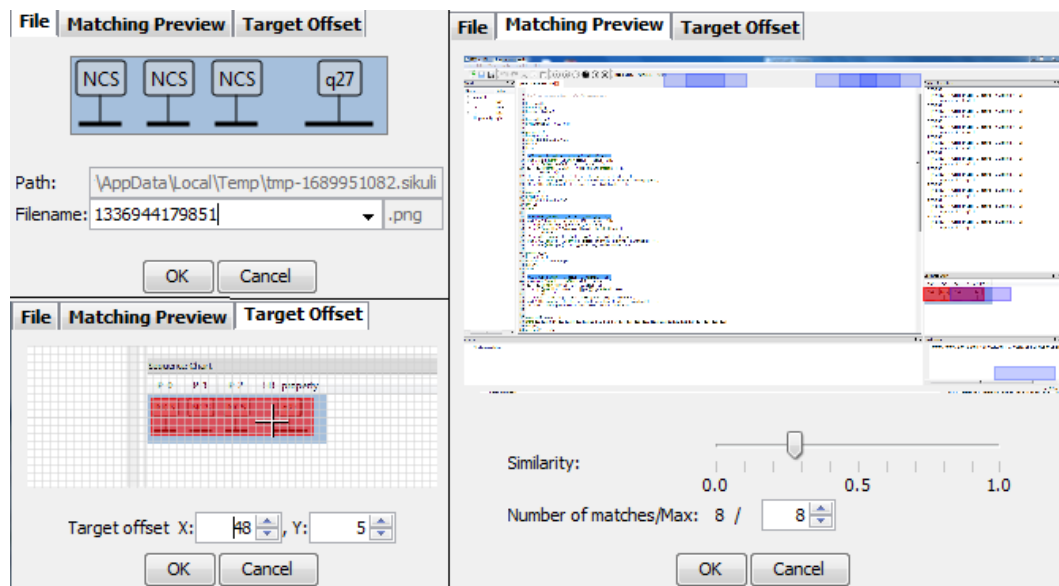
Medzi najpoužívanejšie funkcie v testovacích skriptoch zameraných na testovanie GUI nástroja DiViNE patrí `click()`, `type()` a `wait()` a `exists()`. Ukážka testovacieho skriptu je v prílohe D.

Funkcia `click()` môže mať ako vstupný argument objekt triedy `Pattern`, `Match`, `Region`, `Location` alebo textový reťazec (ďalej v texte PSMRL) a vykoná kliknutie ľavým tlačidlom myši na vstupný argument. Ak je zvolený argument textový reťazec na systémovej obrazovke, je vyhľadaný text pomocou funkcií rozpoznania znakov zo systémovej obrazovky, ktoré sa pri tvorbe testovacích skriptov neosvedčili. V aktuálnej verzii API je možnosť vyhľadania znakov skúšobná, preto sa skôr odporúča vyhnúť sa jej.

Funkcia `type()` má ako vstupný argument textový reťazec alebo konštanty kláves, ktoré dovoľuje zadať do aktuálne zameranej GUI komponentu. Funkcia môže mať ešte

1. HTML – HyperText Markup Language

4. TESTOVANIE GUI NÁSTROJA DIVINE



Obr. 4.1: Na obrázku sú znázornené tri náhľady reprezentujúce interné nástroje IDE nástroja Sikuli. Obrázok *vľavo hore* definuje objekt na ktorý sa majú volať metódy `similar()` a `targetOffset()`. Obrázok *vpravo* reprezentuje nástroj, ktorý dovoľuje volať metódu `similar()` a podľa posuvníka je nastavená hodnota vstupného argumentu. Náhľad dovoľuje zobraziť celú systémovú obrazovku, v ktorej sú vyznačené oblasti zhodujúce sa s požadovaným objektom. Obrázok *vľavo dole* dovoľuje volať metódu `targetOffset()` na požadovaný objekt. Ako náhľad slúži časť systémovej obrazovky, kde sa nachádza požadovaný objekt. Zabudované nástroje vygenerujú v tomto prípade zdrojový kód v nasledovnom tvare: `Pattern("tmp-1689951082").similar(0.28).targetOffset(48, 5)`

voliteľný argument PSMRL, pomocou ktorého je možné definovať zameranú GUI komponentu. Funkcie `click()` a `type()` majú voliteľný vstupný argument klávesovú konštantu z triedy `KeyModifier`, ktorá môže byť zavolaná pri vykonaní funkcie.

Funkcia `wait()` môže mať vstupný argument objekt triedy `Pattern`, textový reťazec alebo celočíselnú hodnotu, ktorá reprezentuje čas vyjadrený v sekundách. Ak má funkcia definovaný argument čas, spustený skript pozastaví svoju činnosť po dĺžku zadáných počet sekúnd. Ak je vstupný argument objekt triedy `Pattern` alebo textový reťazec a zároveň celočíselná hodnota, funkcia pozastaví činnosť skriptu pokiaľ nie je v objekte triedy `Region` zobrazený zadáný objekt po dobu definovaného času. Ak nie je zadáný čas, je použitá prednastavená hodnota v nastaveniach `getAutoWaitTimeout()`.

Funkcia `exists()` je podobná funkcií `wait()`, vstupný argument objekt triedy `Pattern` alebo textový reťazec je povinný. Funkcia slúži na overenie, či je vstupný argument viditeľný v objekte triedy `Region`. Pri nájdení objektu je návratová hodnota funkcií `exists()` a `wait()` objekt triedy `Match`. V prípade neúspechu funkcia `exists()`

vráti nulový objekt `None`, `wait()` vyhodí výnimku nenájdenia objektu `FindFailed`.

V testovacích skriptoch sú tiež často použité funkcie `doubleClick()`, `rightClick()` a `mouseMove()`, ktoré sa od funkcie `click()` odlišujú akciou, ktorú vykonajú pomocou kurzora myši. Funkcia `doubleClick()` vykoná dvojkliknutie ľavým tlačidlom myši, `rightClick()` vykoná kliknutie pravým tlačidlom myši a `mouseMove()` vykoná pohyb myši.

Prvý testovací prípad je zameraný na overenie či sa otvorí aplikácia, má správny názov titulkový okna a či sa zobrazí nápoveda. V skripte je použitá funkcia `Env.getOs()`, ktorej návratová hodnota je popis operačného systému na ktorom je spustený skript. Ak je skript spustený na operačnom systéme Windows, skript otvorí aplikáciu DiViNE dvojkliknutím na jej ikonu, skontroluje titulku okna, otvorí nápovedu a podľa titulkového okna nápovedy skontroluje či sa otvorila. Následne skript vykoná rovnakú činnosť, ale aplikácia sa otvorí cez ponuku programov v položke „Štart“. Ak je skript spustený na inom operačnom systéme, otvorí sa dialógové okno so správou, že tento skript nie je určený pre daný operačný systém. Je to z toho dôvodu, že akcie vykonané v skripte sú odlišné v iných operačných systémoch.

Dialógové okno s požadovanou správou je možné vytvoriť pomocou funkcie `popup()`, ktorá má ako vstupný argument textový reťazec a je to jeden zo spôsobov interakcie užívateľa so skriptami, pričom pri zavolaní tejto metódy je otvorené dialógové okno a po jeho zatvorení skript pokračuje vo svojej činnosti. V tomto prípade je zavolaná funkcia `exit()` a skript ukončí svoju činnosť. V skripte je pre overenie titulkového okna vhodné použiť textové reťazce, ale rozpoznanie znakov v aktuálnej verzii nástroja nie je spoľahlivé. Preto sú pre overenie titulkového okna použité objekty triedy `Pattern`.

V tomto skripte bola nájdená chyba nástroja DiViNE. Pri inštalácii nástroja, je možnosť zvoliť vytvorenie ikony aplikácie na pracovnej ploche. Po zvolení tejto možnosti, ikona na ploche nie je vytvorená.

Druhý skript slúži na overenie, či aplikácia vytvorí nový súbor, zobrazí varovnú správu pri pokuse zavretia aplikácie ak súbor nebol uložený, súbor následne uloží a uloží viac súborov naraz. V skripte sú jednotlivé akcie vykonané pomocou klávesových skratiek, položiek v lište menu a položiek panela nástrojov.


V skripte je použitá metóda `Settings.setAutoWaitTimeout()`, ktorá v skripte nastaví dobu vyhľadávania funkciám, ktoré vykonávajú akcie na *PRLSM* objekty. V testovacích prípadoch nastanú situácie kedy prednastavená doba vyhľadávania nie je vyhovujúca, preto je zmenená. Overenie uloženia súboru je vykonané na základe titulkového súboru v paneli editora a titulkového hlavného okna aplikácie. V druhom skripte je použitá funkcia `paste()`, ktorá vykonáva akciu „vložiť“ do aktuálne zameranej GUI komponenty. Povinný vstupný argument funkcie `paste()` je textový reťazec, ktorý má byť vložený a voliteľný vstupný argument je *PSMRL*, ktorý definuje komponentu do ktorej má byť vložený textový reťazec.

Metódy, ktoré používajú argumenty *PSMRL* môžu vyhodíť výnimku `FindFailed`, ktorá znamená, že nebol nájdený vstupný argument. Odchytením výnimky je možné

4. TESTOVANIE GUI NÁSTROJA DiVINE

na túto chybu reagovať vhodným spôsobom a v niektorých prípadoch zvoliť alternatívnu možnosť akcie s testovanou aplikáciou. API obsahuje pre tento druh výnimky metódu `setFailedResponse()`, ktorá v skripte nastaví ako sa majú metódy vysporiadať s danou výnimkou. Vstupné argumenty môžu byť konštanty `ABORT`, `SKIP`, `PROMPT`, `RETRY`. Predvolená hodnota je `ABORT`, čo znamená, vyhodenie výnimky a jej následné vypísanie na konzolu. Hodnota `SKIP` znamená, že nenastane žiadna akcia a metódy vrátia návratovú hodnotu `None`. `RETRY` znamená, že objekt bude hľadaný pokiaľ ho funkcia nenájde. `PROMPT` znamená vytvorenie dialógového okna, ktoré ponúka predchádzajúce tri možnosti ako sa majú funkcie vysporiadať s práve vyhodnotenou výnimkou.

V druhom skripte je zatvorené okno aplikácie pomocou tlačidla na titulke okna, ktoré je závislé od témy operačného systému. V skriptoch, ktoré sú závislé od systémových okien sú použité iné možnosti požadovanej akcie. Systémové tlačidlá sa môžu líšiť od prostredia v ktorom je skript spustený, preto ak daný vzor systémového tlačidla nie je nájdený, skript na túto výnimku reaguje stlačením klávesovej skratky podľa nasledovného výpisu:

```
# Try close application
try:
    click()
except FindFailed:
    type(Key.F4, KeyModifier.ALT)
type(Key.RIGHT + Key.RIGHT + Key.ENTER) # Cancel dialog
```

Tretí skript slúži na overenie či aplikácia otvorí, obnoví a uloží naraz dva súbory. V skripte boli použité metódy a funkcie z predchádzajúcich skriptov. Overenie, či bol uložený súbor je podľa titulky okna v paneli editora.

Štvrtý skript je zameraný na akcie „vystrihni“, „vložiť“, „vrátiť späť“ a „prerobiť“. Akcie sú vykonané pomocou klávesových skratiek, položiek v lište menu a položiek v paneli nástrojov pri editácii súboru v paneli editora. Funkcia vyber všetok text je overená pomocou nasledovného výpisu:

```
OSType = Env.getOS() # Get the type OS
if OSType == OS.WINDOWS:
    type('a' , KeyModifier.CTRL)
    type('c' , KeyModifier.CTRL)
    openApp("notepad.exe")
    wait(0.2) # Wait for opening notepad
    type('v' , KeyModifier.CTRL)
    popup("Before closing this window please check source code
    in notepad and then close notepad")
else:
    popup("CTRL+A keyboard shortcut wasn't tested")
```


Všetok text je vybratý z panela editora, následne je otvorený program poznámkový blok do ktorého je vložený text. Po vložení textu je otvorené dialógové okno s výzvou, aby užívateľ skontroloval text v poznámkovom bloku a následne poznámkový blok zavrel aj s dialógovým oknom. Ak je skript pustený na inom systéme, zobrazí sa dialógové okno so správou, že daná funkcia nebola otestovaná.

Ďalej je použitá funkcia `dragDrop()`, ktorá vykonáva akciu myši „chyť a pusť“. Má tri vstupné argumenty, prvé dva sú PSMRL a sú povinné. Prvý argument predstavuje objekt na ktorom sa má vykonať akcia „chyť“ a druhý argument objekt na ktorom sa má vykonať akcia „pusť“. Tretí argument môže byť klávesová konštanta z triedy `KeyModifier`. Aj keď akcia „chyť a pusť“ sa väčšinou používa na premiestnenie objektu, v tomto prípade bola funkcia použitá na označenie textu pomocou myši.

Piaty skript slúži na overenie akcií „kopíruj“ a „vložiť“, „overwrite mode“ a zobrazenie čísla riadkov. V tomto skripte boli použité metódy a funkcie z predchádzajúcich skriptov.

Šiesty skript je zameraný na funkciu vyhľadávania v zdrojovom kóde. Ak je požadovaný text v zdrojovom kóde nájdený, tak je v zdrojovom kóde zobrazený a v stavovej lište je zobrazená pozícia hľadaného prvku. Ak prvok nie je nájdený, v stavovej lište je zobrazená správa „No Match“. Tieto informácie sú overené pomocou skriptu a taktiež je otestovaná funkcia „Case Sensitive“ a vyhľadanie regulárneho výrazu.

V tomto prípade je nájdená chyba správy v stavovej lište. Ak nie je nájdená hľadaná položka, v stavovej lište je zobrazená správa „No Match“. Následne je vo vyhľadávaní zadaný text, ktorý je nájdený, ale správa „No Match“ je stále v stavovej lište zobrazená. Medzi ďalší nedostatok patrí to, že zvýraznenie syntaxe nájdeného textu je nevýrazné.

Siedmy skript slúži na overenie či sú položky v paneli nástrojov zobrazené podľa nastavení užívateľa a tiež je otestovaná funkcia zalamovania textu. Funkcia zalamovania textu je otestovaná tak, že je zmenšené hlavné okno aplikácie pomocou funkcie `dragDrop()`. Je otvorený naposledy otvorený súbor a ak je zobrazený horizontálny posuvník znamená to, že text nie je zalomený.

Ôsmy skript slúži na overenie, či sú pri spustení simulácie zobrazené všetky panely. Zobrazenie panelov je overené podľa titulkov jednotlivých panelov. V tomto prípade je odhalená chyba použiteľnosti. Kurzor myši v režime „text“ znamená, že užívateľ má možnosť zadať znakové informácie v UI. Počas spustenej simulácie je kurzor myši nad panelom editora v režime „text“ napriek tomu, že užívateľ nemá možnosť editovať zdrojový kód.

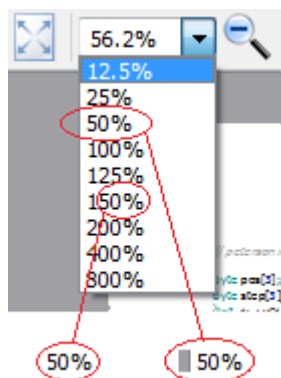
Deviaty skript je zameraný na overenie funkcie nahradenia nájdeného textu v zdrojovom kóde. Pred vykonaním tohto skriptu je potrebné aby v okne nahradenia boli všetky zaškrtávacie polia odškrtnuté. Deviaty skript vychádza vo väčšej miere zo šiesteho skriptu. V skripte boli otestované funkcie nahradenia jednej položky a všetkých položiek. V tomto prípade sa tiež prejavili chyby spomenuté v šiestom testovacom skripte.

Desiaty skript je zameraný na náhľad tlaču a zobrazenia systémového okna na tlač súboru. Náhľad tlaču umožňuje zobraziť dokument zdrojového kódu v rozličných polohách a veľkostiach. Niektoré zmeny sa funkciám v skriptoch nepodarilo rozlíšiť, preto

4. TESTOVANIE GUI NÁSTROJA DiVINE

je potrebné vykonané zmeny náhľadu „odsledovať“ pri vykonávaní skriptu. V tomto skripte bola použitá funkcia `getLocationFromPSRML()`, ktorá má ako vstupný argument objekt PSRML a návratová hodnota je bod na systémovej obrazovke, ktorý predstavuje objekt triedy `Location` vstupného argumentu. V skripte je z návratovej hodnoty funkcie `getLocationFromPSRML()` vykonaná funkcia `click()` v slučke.

V jednom z krokov v skripte je zmenšenie náhľadu tlaču na 50 %. Testovací krok je vykonaný pomocou funkcie `click()` a ako argument je zvolená rastrová grafika reprezentujúca text v komponente „rozbaľovací výpis“. V niektorých prípadoch, kde je text použitý ako argument rastrovej grafiky je potrebné rastrovú grafiku reprezentovať pomocou iných farebných prechodov alebo tvarov.



Obr. 4.2: Na obrázku je znázornená chyba, ktorá môže nastať v dôsledku zle zvolenej rastrovej grafiky predstavujúcej text. Hore je znázornená komponenta „rozbaľovací výpis“, ktorá predstavuje veľkosť náhľadu tlaču. Dole sú zvolené dve grafiky, reprezentujúce požadovanú položku 50%. V „rozbaľovacom výpise“ je znázornené, ktorá položka je funkciou rozoznaná podľa zvolenej grafiky.

Testovacie skripty č. 11, 12 a 13 slúžia na overenie či sa správne vykoná simulácia modelu. V skripte č. 11 je overenie simulácie pomocou položiek v lište menu, taktiež je overená možnosť uloženia stavu modelu počas simulácie a následné importovanie stavu. Na overenie v akom stave je simulácia slúži vizuálne zobrazenie modelu v *Sequence chart* paneli. V tomto skripte je overené či je v paneli výstupu zobrazená správa po spustení verifikácie modelu, ďalej je overená možnosť prázdneho panela výstupu.

V skripte č. 12 je vykonaná rovnaká simulácia modelu použitím klávesových skratiek. V tomto prípade je nájdená chyba funkčnosti klávesovej skratky. Položky v lište menu je možné štandardne otvoriť pomocou klávesovej skratky ALT + „Začiatkové písmeno názvu položky“. Požadovanú položku *Simulation* nie je možné otvoriť pomocou klávesovej skratky ALT+S.

V skripte č.13 je vykonaná simulácia modelu pomocou položiek v lište nástrojov. Pomocou skriptu je tiež overené, či sa zobrazí správa po prerušení verifikácie modelu.

Skripty č. 14, 15 a 16 sú zamerané na grafickú vizualizáciu modelov v *Sequence Chart* paneli. V každom z týchto skriptov je použitý iný model a jednotlivé kroky simulácie sú vykonané pomocou klávesových skratiek. Po spustení simulácie modelu je panel *Sequence chart* oddelený od hlavného okna aplikácie a následne zväčšený podľa veľkosti systémovej obrazovky.

Veľkosť systémovej obrazovky je možné zistiť pomocou funkcie `getBounds()`, kto-

rej návratová hodnota je objekt v tvare pravouhlého štvoruholníka, ktorého hodnoty x , y , $width$ a $height$ predstavujú veľkosť systémovej obrazovky. Následne je v skripte spustená simulácia a po každom kroku je overená vizualizácia modelu. Po prejdení myšou nad jednotlivými stavmi je v grafe možné zobraziť predchádzajúce stavy. V skripte je uložený bod posledného stavu v grafe a pomocou funkcie `mouseMove()` je vykonaný pohyb myšou nad každým stavom o -60 bodov na ose y od posledného stavu. Pri každom premiestnení myši o -60 bodov je overené či je zobrazený požadovaný graf.

V skripte č. 16 je overené či sú zobrazené posledné stavy simulácie. V grafe je tiež možné zvýrazniť predchádzajúce stavy kliknutím alebo dvojkliknutím na požadovaný stav. Obe možnosti sú v skripte overené. Po vykonaní niekoľkých prechodov simulácie nie sú viditeľné posledné stavy modelu na systémovej obrazovke. Spodnú časť panela *Sequence Chart* je možné zobraziť pomocou akcie „skrolovania“ myši, použitím vertikálneho posuvníka alebo pomocou klávesových skratiek.

V skripte je použitá akcia „skrolovania“ pomocou funkcie `wheel()`, ktorá má tri vstupné argumenty. Prvý vstupný argument je PSRML, ktorý predstavuje objekt v ktorom má byť vykonaná akcia. Druhý vstupný argument je konštanta `WHEEL_DOWN`, alebo `WHEEL_UP`, ktoré predstavujú smer „skrolovania“, buď je to smer nahor alebo nadol. Tretí vstupný argument je celočíselná hodnota, ktorá predstavuje koľkokrát má byť vykonaná akcia. V skripte je akcia „skrolovania“ vykonaná, pokiaľ nie sú v grafe na systémovej obrazovke viditeľné posledné stavy podľa nasledovného výpisu:

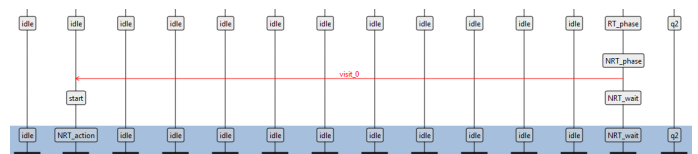
```
# Scroll down while don't find last states
```

```
lastState = None
```

```
while not lastState:
```

```
    wheel( , WHEEL_DOWN, 1)
```

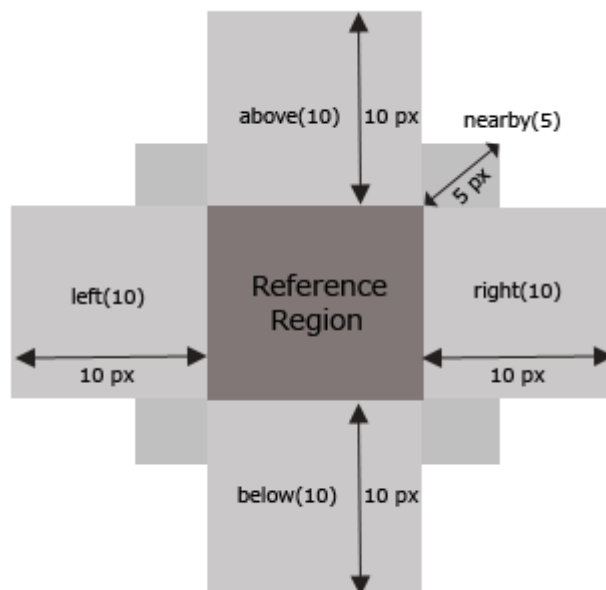
```
    lastState = exists(
```



V skripte nastane situácia vizualizácie modelu, v ktorej je zobrazených viac uzlov, ktorých grafická reprezentácia sa od seba neodlišuje a je potrebné vykonať akciu len na jednom z týchto uzlov. Táto situácia sa dá vyriešiť viacerými spôsobmi. Jednou z možností je zvoliť grafickú reprezentáciu celého panela *Sequence chart* na ktorú je zavolaná metóda `targetOffset()`, pomocou ktorej je možné zvoliť bod požadovaného objektu. Ďalšia možnosť je použiť metódy `nearby()`, `above()`, `below()`, `right()`, `left()`.

4. TESTOVANIE GUI NÁSTROJA DiVINE

Vo väčšine prípadov je efektívnejšie použiť druhú možnosť, kvôli menšej ploche na ktorej sú porovnávané grafické vzory. Tieto metódy je možné volať na objekty reprezentujúce oblasť na systémovej obrazovke. Vstupný argument je kladné číslo reprezentujúce rozsah novej oblasti, ktorá je zároveň návratová hodnota metódy. Metódy `above()`, `below()`, `right()`, `left()` zvolia oblasť v požadovanom smere, metóda `nearby()` zvolí oblasť vo všetkých smeroch podľa obrázku 4.3, ktorý je z časti prevzatý z dokumentácie API vizualizačných skriptov.²



Obr. 4.3: Na obrázku je znázornené ako funkcie `nearby()`, `above()`, `below()`, `right()`, `left()` zvolia požadovanú oblasť. Funkcia `nearby()` vráti novú oblasť spolu s oblasťou na ktorú bola funkcia volaná. Ostatné funkcie vrátia novú oblasť bez oblasti na ktorú bola funkcia volaná.

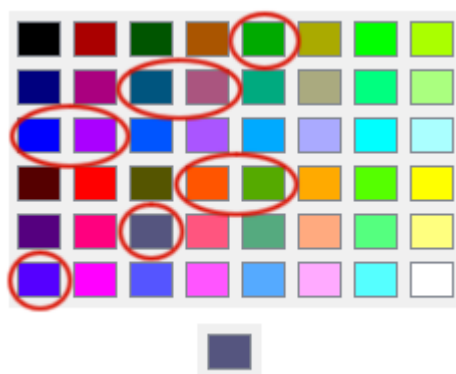
V skripte bola použitá funkcia `below()` na nájdenie bodu, v ktorom je umiestnený najbližší uzol pod prvým uzlom v prvom stave modelu podľa nasledovného výpisu:

```
getLocationFromPSRML(find(LTL_property q1) .below().find(q295)))
```

2. <http://sikuli.org/docx/region.html?highlight=above#Region.nearby>

Testovacie skripty č. 17, 18 a 19 sú zamerané na nastavenia aplikácie. V skripte č. 17 je otvorený súbor so zdrojovým kódom, ktorý slúži na overenie či sa prejaví zmena nastavenia aplikácie. Na začiatku je v skripte odsadený text komentára a podľa zdrojového kódu je overené, či je text odsadený. Následne je v nastaveniach panela editora zmenená hodnota veľkosti písma a odsadenia. V nastaveniach simulácie modelu je overené či sa dajú označiť zaškrŕavacie polia a či sú blokované komponenty vyznačené. Následne sú pomocou dialógového okna palety farieb zmenené farby určené na zvýraznenie syntaxe.

V tomto prípade je potrebné vhodne zvoliť vzor. Vzory líšiace sa od seba iba odtieňom farby sú považované za takmer rovnaký objekt aj napriek tomu, že veľkosť zhody je nastavená na maximálnu hodnotu.



Obr. 4.4: Na obrázku je znázornená paleta farieb, v ktorej sú vyznačené grafické vzory považované za rovnaké podľa zvoleného vzoru *dole*.

V tomto prípade chybné rozoznanie odtieňa farby nie je žiadnou prekážkou. Stačí zvoliť vhodný odtieň farby, ktorý sa nebude zhodovať s predchádzajúcim. Tento nedostatok je potrebné brať do úvahy pri tvorbe ďalších testovacích skriptov.

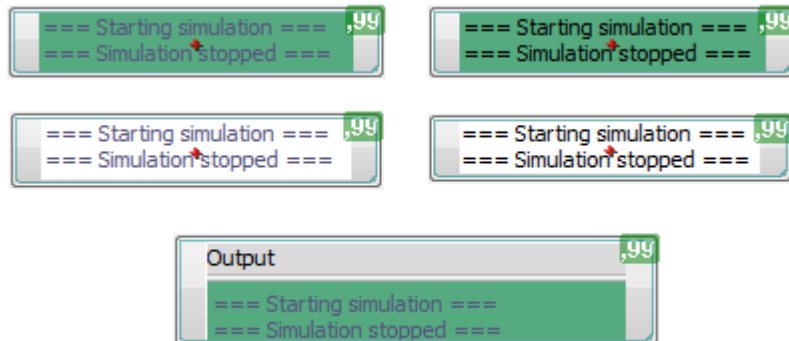
V ďalšom kroku je nastavené podčiarknutie textu, kurzíva a tučné písmo pre vybrané položky v zdrojovom kóde. V prípade overenia nastavení voľby písma je potrebné zvoliť položky, ktoré sa v zdrojovom kóde často vyskytujú, inak nie sú zmeny podľa vzoru badateľné.

Testovací skript č. 18 nadväzuje na skript č. 17, preto môže byť spustený len po jeho vykonaní. V skripte je zmenená farba písma a pozadia panela výstupu. V tomto prípade je chyba v nastaveniach aplikácie, kde položky *Foreground:*, *Background:* sú zamenené. Funkcie použité v skripte nie sú schopné rozoznať chybné nastavenia aplikácie. Preto je ako grafický vzor zvolená časť výpisu spolu s titulkou panela. V tomto prípade sú funkcie schopné rozoznať jednotlivé rozdiely medzi vzormi. Na obrázku 4.5 je zobrazené v akom prípade sú grafické vzory považované za rovnaké.

Pre veľkosť písma a odsadenia textu v paneli editora sú v skripte nastavené pôvodné hodnoty. Ďalej je v skripte zmenená farba pozadia číselných dátových položiek a farba písma komentárov. Následne je v zdrojovom texte overené či sa požadované zmeny prejavili v zdrojovom kóde a v paneli výstupu.

Testovací prípad č. 19 slúži na zadanie pôvodných hodnôt nastavenia aplikácie pred

4. TESTOVANIE GUI NÁSTROJA DiVINE



Obr. 4.5: Na obrázku je znázornených päť vzorov. Hore je skupina štyroch vzorov, ktoré sú považované za rovnaké. Ak je k jednotlivým vzorom pridaná dodatočná informácia formou farebného prechodu alebo tvaru, tak funkcie jednotlivé vzory rozoznajú. Dole je vzor s titulkou panela editora. Ak je vytvorená skupina vzorov, ako na obrázku hore spolu s titulkou panela, tak funkcie jednotlivé vzory od seba odlišia.

vykonaním testovacieho prípadu č. 17 a 18. Je to z toho dôvodu, že v nastaveniach nie je položka vrátenia pôvodných zmien v aplikácii.

Testovací prípad č. 20 je zameraný na tlačidlá titulného okna aplikácie a ostatné panely, ktoré neboli overené. Tento testovací prípad nebol prepísaný do skriptu, pretože závisí od nastavení systémovej lišty v operačnom systéme. Obsah panelov *Watch*, *Stack trace* a *Enabled Transitions* pozostáva z textových informácií, preto je v tomto prípade jednoduchšie overiť tieto informácie manuálne.

4.5 Vyhodnotenie testovania GUI nástroja DiVinE

Pred spustením testovacích skriptov bola vytvorená *traceability matrix*, kde vrchný riadok predstavuje požiadavky kladené na UI nástroja DiVinE a ľavý stĺpec predstavuje testovacie prípady. V *traceability matrix* je zaznamenané, ktoré požiadavky sú splnené daným testovacím prípadom. Časť *traceability matrix* je v prílohe E. Pomocou nástroja Sikuli sa nedajú overiť všetky požiadavky, preto je v *traceability matrix* zaznamenané, ktoré požiadavky je potrebné skontrolovať vizuálnou kontrolou pri vykonaní testovacieho prípadu.

Pomocou nástroja Sikuli je overená väčšina požiadaviek GUI nástroja DiVinE. Pri tvorbe a vykonaní testovacích prípadov boli nájdené následné nedostatky vo funkčnosti a použiteľnosti UI nástroja DiVinE.

Chyby funkčnosti

- Zvolená možnosť pri inštalácii vytvorenia ikony na pracovnej ploche nefunguje

- Chybné zobrazenie správy v stavovej lište pri vyhľadávaní a nahradení v zdrojovom kóde
- Chybné prístupnenie položky menu *Simulate* pomocou klávesovej skratky
- Zámena položiek *Foreground:* a *Background:* v nastaveniach panela výstupu

Chyby použiteľnosti

- Nevýrazné zvýraznenie syntaxe nájdenej položky v zdrojovom kóde
- Počas simulácie modelu je kurzor myši nad panelom editora v režime „text“

4.6 Zahnutie testovacích skriptov do repozitára projektu DiVinE

Zdrojový kód vizualizačných skriptov je napísaný v programovacom jazyku Jython, preto je na ich spustenie potrebný JVM. Skripty je možné spustiť vo forme príkazu bez potreby spustenia vývojového prostredia. Na ich spustenie sú potrebné dynamicky linkované knižnice a zdrojový kód určený na linkovanie knižníc a spustenie skriptu.

V prípade operačného systému Windows sú knižnice uložené v spoločnom adresári. Pred preložením zdrojového kódu a zlinkovaním knižníc je potrebné nastaviť cestu k adresáru s linkovanými knižnicami premennej prostredia JVM. Na nastavenie premennej prostredia je možné použiť dávkový súbor, ktorý pred spustením skriptu nastaví premennú prostredia, spustí skript a po skončení skriptu vráti premennú prostredia do pôvodného stavu. Potrebné knižnice a zdrojový kód je možné stiahnuť z webovej stránky projektu Sikuli³, dávkový súbor vo forme .cmd súboru môže byť zahrnutý do repozitára projektu DiVinE. Testovací skript je možné spustiť pomocou nasledovného príkazu:

```
cesta-k-súboru\runsikuli.cmd cesta-k-súboru\case.sikuli
```

V prípade operačného systému Linux sú knižnice uložené vo viacerých systémových adresároch. Na spustenie skriptu sú potrebné linkovacie knižnice nástrojov `wmctrl` a `OpenCV 2.1` knižnice `libcv 2.1`, `libcvaux 2.1`, `libhighgui 2.1`. Potrebné knižnice sa dajú stiahnuť z oficiálneho repozitára *Debian* balíkov. Spustenie skriptu v Linuxe je nasledovné:

```
java -jar cesta-k-súboru/sikuli-script.jar cesta-k-súboru/case.sikuli
```

3. <http://sikuli.org/download.shtml>

4. TESTOVANIE GUI NÁSTROJA DiVINE

V operačných systémoch Mac je postup podobný ako v prípade operačného systému Windows. Knižnice sú zahrnuté v jednom adresári, pričom sa líšia od jednotlivých distribúcií operačného systému. Na spustenie skriptu nebol vytvorený dávkový súbor, preto je pred ich spustením potrebné nastaviť premennú prostredia a spustiť skripty vo forme príkazu rovnakom ako v operačných systémoch Linux. Premennú prostredia je možné nastaviť podľa nasledovného tvaru:

```
export SIKULI_HOME=cesta-k-adresáru-s-knižnicami
```


5 Záver

Cieľom tejto bakalárskej práce bolo popísať vhodné techniky a nástroje určené na testovanie GUI. Na základe týchto znalostí bolo otestované GUI nástroja DiVinE. Testovanie bolo vykonané pomocou testovacích skriptov, ktoré boli vytvorené vo vývojovom prostredí nástroja Sikuli. Testovacie skripty boli odvodené z manuálne vytvorených testovacích prípadov.

Testovaním boli zistené nedostatky funkčnosti a použiteľnosti užívateľského prostredia nástroja DiVinE. Nedostatky sa týkajú iba užívateľského rozhrania a nezasahujú do funkčnosti aplikácie. V prípade zmeny užívateľského rozhrania treba brať do úvahy, že niektoré skripty budú musieť byť upravené. Niektoré skripty tiež interagujú s užívateľom a sú závislé od témy systémových okien, čo je v testovacích prípadoch zaznamenané.

Práca môže poslúžiť ako „vodítko“ pre niekoho, kto sa chce zaoberať testovaním GUI, ale nemá s danou problematikou žiadne skúsenosti. Pomocou testovacích skriptov sa dala otestovať veľká časť GUI nástroja DiVinE. V práci sú tiež zaznamenané nedostatky API, ktoré bolo použité v testovacích skriptoch.

Börjesson et al. [16] porovnali dva nástroje určené na testovanie GUI využívajúce počítačové videnie. Jeden z nich je nemenovaný komerčný nástroj a druhý je nástroj Sikuli. Na porovnanie použili metriky, ako napríklad SLOC¹, doba vytvorenia a prevedenia skriptov. Oba nástroje sú porovnateľne vhodné na testovanie, pričom jeden nástroj „prevyšoval“ nad druhým len v niektorých špecifických prípadoch.

Tsung-Hsiang Chang et al. [4] popísali možnosti testovania pomocou nástroja Sikuli. V práci tiež zaznamenali priebeh testovania niekoľko verzií aplikácií, pričom hlavný úmysel bolo poukázať na počet testovacích skriptov a prípady, kedy je potrebné skripty upraviť v nasledovných verziách aplikácie.

Na túto prácu môžu nadväzovať ďalšie práce zaoberajúcimi sa testovaním GUI. Nástroje využívajúce počítačové videnie sa výrazne líšia prístupom k danej problematike od nástrojov využívajúcich zdrojový kód testovanej aplikácie.

Na základe metrík určených na odhad veľkosti, zložitosti zdrojového kódu testovacích skriptov môžu byť tieto nástroje medzi sebou porovnané a vyhodnotiť v akých prípadoch je vhodné použiť dané nástroje alebo techniku testovania GUI. Tiež je možné pomocou nástrojov otestovať viac verzií rovnakej aplikácie a vyhodnotiť pri akých zmenách testovaných aplikácií je vhodné použiť daný nástroj alebo techniku testovania GUI.

1. Source lines of code – počet riadkov zdrojového kódu

Literatúra

- [1] Atif M. Memon, Qing Xie. Designing and comparing automated test oracles for GUI-based software applications. *ACM Trans. Softw. Eng. Methodol.*, 16(1):9-10, Február 2007. ISSN 1049-331X. Dostupné z WWW: <<http://doi.acm.org/10.1145/1189748.1189752>>.
- [2] *Traceability matrix* [online], založené 29. 3. 2010, aktualizované 30. 3. 2012, [cit. 12. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Traceability_matrix>.
- [3] *Requirements traceability* [online], založené 214. 9. 2009, aktualizované 12. 4. 2012, [cit. 14. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Requirements_traceability>.
- [4] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. GUI testing using computer vision. *Proceedings of the 28th international conference on Human factors in computing systems* (CHI '10). ACM, New York, NY, USA, s. 1541. ISBN: 978-1-60558-929-9. Dostupné z WWW: <<http://doi.acm.org/10.1145/1753326.1753555>>.
- [5] *Test case* [online], založené 18. 1. 2011, aktualizované 4. 4. 2012, [cit. 12. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Test_case>.
- [6] *Test suite* [online], založené 10. 6. 2008, aktualizované 4. 4. 2012, [cit. 12. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Test_suite>.
- [7] Atif M. Memon, Qing Xie. Designing and comparing automated test oracles for GUI-based software applications. *ACM Trans. Softw. Eng. Methodol.*, 16(1):6-8, Február 2007. ISSN 1049-331X. Dostupné z WWW: <<http://doi.acm.org/10.1145/1189748.1189752>>.
- [8] *Oracle (software testing)* [online], založené 11. 11. 2008, aktualizované 20. 10. 2011, [cit. 12. 4. 2012], Wikipedia. Dostupné z WWW: <[http://en.wikipedia.org/wiki/Oracle_\(software_testing\)](http://en.wikipedia.org/wiki/Oracle_(software_testing))>.
- [9] *Scenario testing* [online], založené 17. 4. 2009, aktualizované 3. 4. 2012, [cit. 12. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Scenario_testing>.
- [10] Atif M. Memon, Martha E. Pollack, Mary Lou Soffa. Using a goal-driven approach to generate test cases for GUIs. *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, s. 264, 22-22 Máj 1999. ISBN 1-58113-074-0. Dostupné z WWW: <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=841016&isnumber=18169>>.

- [11] *Model-based testing* [online], založené 9. 11. 2009, aktualizované 12. 4. 2012, [cit. 13. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Model-based_testing>.
- [12] Marlon Vieira, Johanne Leduc, Bill Hasling, Rajesh Subramanyan, Juergen Kazmeier. Automation of GUI testing using a model-driven approach. *Proceedings of the 2006 international workshop on Automation of software test (AST '06)*. ACM, New York, NY, USA, s. 9-14. Dostupné z WWW: <<http://doi.acm.org/10.1145/1138929.1138932>>.
- [13] Atif M. Memon, Jaymie Strecker. Testing Graphical User Interfaces. *Encyclopedia of Information Science and Technology, Second ed.*, 2008, s. 3849-3850, ISBN-13: 978-1605660264.
- [14] *Monkey test* [online], založené 10. 11. 2005, aktualizované 24. 11. 2011, [cit. 13. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Monkey_test>.
- [15] Kanlin Li, Menggi Wu. *Effective GUI Testing Automation: Developing an Automated GUI Testing Tool*. Sybex, 2004, s. 41-43, ISBN-13: 978-0782143515.
- [16] Emil Börjesson, Robert Feldt. *Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry* [pdf, online]. Dostupné z WWW: <http://www.cse.chalmers.se/~algeroth/publications/Borjesson_Feldt_2012-01-16_Auto_System_test_using_GUI_interaction.pdf>.
- [17] Kanlin Li, Menggi Wu. *Effective GUI Testing Automation: Developing an Automated GUI Testing Tool*. Sybex, 2004, s. 21-41, ISBN-13: 978-0782143515.
- [18] *Computer vision* [online], založené 10. 11. 2011, aktualizované 8. 4. 2012, [cit. 13. 4. 2012], Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Computer_vision>.

A Obsah priloženého CD

V koreňovom adresári priloženého disku sú nasledovné súbory:

- *test-scripts.zip* – obsahuje zdrojové kódy testovacích skriptov a dávkový súbor *run.cmd* pre operačný systém Windows na nastavenie premennej prostredia JVM a spustenie skriptu. Dávkový súbor musí byť pred spustením v spoločnom adresári so súborom *sikuli-script.jar*, v podadresári musia byť dynamicky linkované knižnice. Súbor *sikuli-script.jar* a potrebné knižnice je možné stiahnuť z webovej stránky projektu Sikuli.¹
- *documents.zip* – obsahuje dokument požiadaviek na GUI nástroja DiVinE *poziadavky.odt*, testovacie prípady v súbore *testovacie-pripady.odt* a traceability matrix v súbore *traceability-matrix.ods*

1. <http://sikuli.org/download.shtml>

B Časť zoznamu požiadaviek na GUI nástroja DiVinE

Práca so súborom

- Vytvoriť nový súbor, uložiť súbor
- Otvoriť súbor
- Obnoviť zmenený súbor
- Vytlačiť súbor, zobrazíť náhľad tlače
- Zatvoriť súbor, zatvoriť aplikáciu

Editovanie súboru

- „Vrátiť späť“ a „prerobiť“ súbor
- Vystrihni, skopíruj, vlož text
- Režim prepisovania
- Vyhľadanie v texte
- Nahradenie v texte
- Kontrola syntaxe

Nastavenie aplikácie

- Voľba položiek v lište nástrojov
- Voľba zobrazených panelov
- Zalamovanie textu v paneli editora
- Zobrazenie čísla riadkov v paneli editora
- Voľba písma v paneli výstupu
- Voľba písma a odsadzovanie textu v paneli editora
- Voľba zvýraznenia syntaxe

C Časť testovacieho prípadu

ID:	#12
Názov:	Simulácia modelu
Účel:	Overenie či sa správne vykoná simulácia modelu pomocou klávesových skratiek
Podmienky:	Zobrazený panel výstupu
Kroky:	<ol style="list-style-type: none"> 1. Klikni v lište nástrojov na položku „Open“ 2. Dvakrát klikni na adresár „examples“ 3. Dvakrát klikni na súbor „beem-peterson.4.prop3.dve“ 4. Stlač F6, stlač F7, stlač CTRL+F8, stlač ALT+S, stlač R, stlač ENTER, stlač SHIFT+F5 5. Stlač štyrikrát šípku hore, stlač ENTER ... 9. Klikni na položku „Random Run...“ 10. Napíš „abcd.-1“, stlač ENTER ... 18. Klikni na položku „Clear Output“, stlač ENTER
Očakávaný výsledok:	<ol style="list-style-type: none"> 1. Dialógové okno s adresármi nástroja DiVinE je otvorené 2. V dialógovom okne je zobrazený zoznam súborov 3. Súbor „beem-peterson.4.prop3.dve“ je otvorený 4. Nenastane žiadna akcia 5. Simulácia modelu je spustená ... 9. Je otvorené dialógové okno 10. Je vykonaný náhodný krok ... 18. Panel výstupu je prázdny
Vykonanie testu:	Fail
Poznámky:	Nefunguje klávesová skratka ATL+S na sprístupnenie položky menu <i>Simulate</i>


D Testovací skript


```
# Test Case ID: 12
# Model simulation using shortcuts, check messages in output panel

setAutoWaitTimeout(6)

# Check chart in Sequence Chart panel
def checkChart(pattern):
    chart = exists(pattern)
    if not chart:
        popup("Occured error in Sequence chart")
        exit(1)

# Main function

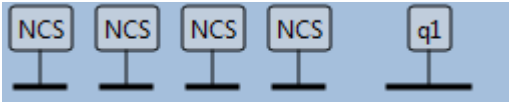
# Open beem-peterson.4.prop3.dve file
click( )

doubleClick( examples )

doubleClick( beem-peterson.4.prop3.dve )


# Try disabled buttons
type(Key.F6 + Key.F7)
type(Key.F8 , KeyModifier.CTRL)
type('s' , KeyModifier.ALT)
type('r' + Key.ENTER)
type(Key.F5 , KeyModifier.SHIFT)

# Start simulation
type(Key.UP + Key.UP + Key.UP + Key.UP + Key.ENTER)

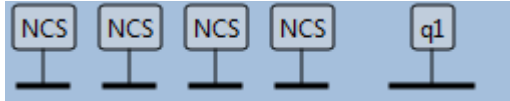
checkChart( );
```

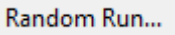
D. TESTOVACÍ SKRIPT

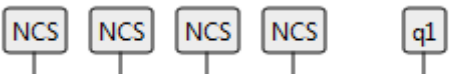
```
type(Key.F8 , KeyModifier.CTRL)

checkChart (  );

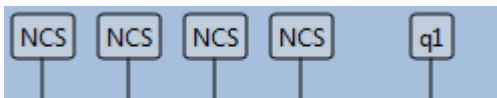
type('s' , KeyModifier.ALT)
type('r' + Key.ENTER)

checkChart (  );


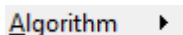
type('s' , KeyModifier.ALT)
click(  )
wait(0.1)
paste("abcd.-") # Try invalid data
paste("1")
type(Key.ENTER)

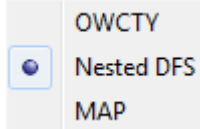
checkChart (  );

type(Key.F5 + Key.F6 + Key.F7)

checkChart (  );

# Choose Nested DFS algorithm
type('t' , KeyModifier.ALT)
type('a' + Key.DOWN + Key.RIGHT + Key.DOWN + Key.ENTER)

click(  )
mouseMove(  )

algorithm = exists(  )

if not algorithm:
    popup("Algorithm wasn't changed")
    exit(1)

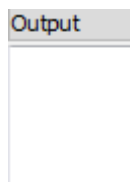
# Verify model
type('t' , KeyModifier.ALT)

type('v')
```

```
# Abort verification
type('t' , KeyModifier.ALT)
type('a' + Key.ENTER)
```

```
# Clear output panel
type('t' , KeyModifier.ALT)
wait(0.1)
click( Clear Output )
type(Key.ENTER)
```

```
ClearOutput = exists(
```



```
if not ClearOutput:
    popup("Output panel isn't clear")
    exit(1)
```


E Časť traceability matrix

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	2.1	2.2	2.3	2.4
1	×	✓							✓					
2			✓		👁		✓	✓		✓				
3					👁					✓				
4					👁					✓	✓	✓	✓	✓
5					👁					✓				
6					👁		👁	✓						
7										✓				
8														
9					👁									
10				✓										
11					👁		👁	✓						
12					👁		👁	✓		×		✓		
13							👁	✓				✓		
14														
15														
16														
17					👁		👁	✓						
18					👁		👁	✓						
19					👁									
20						✓								

- ✓ – požiadavka je splnená a skriptom overená
- ×
- 👁 – požiadavka je splnená a skriptom neoverená (vizuálne overiť pri vykonaní testu)