

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Využití nástrojů pro testování grafického uživatelského rozhraní

Místo této strany bude
zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 24. dubna 2016

Jaroslav Klaus

Abstract

This paper deals with the use of software tools for testing graphical user interface. It compares some of the tools and describes the use of one of them in a way that fits into the subject KIV/OKS.

Abstrakt

Tato práce se zabývá využitím nástrojů k testování grafického uživatelského rozhraní aplikací. Srovnává některé nástroje k tomu určené a popisuje použití jednoho z nich tak, aby svou filosofií zapadal do předmětu KIV/OKS.

Obsah

1	Úvod	1
2	Testování softwaru	2
2.1	Požadavky	2
2.2	Specifikace požadavků na software	2
2.3	Kvalita softwaru	3
2.3.1	FURPS	3
2.4	Chyba, defekt, selhání	3
2.5	Testování softwaru	4
2.6	Úrovně testování	5
2.7	Testování uživatelského rozhraní	5
2.8	Automatizace testování	7
2.8.1	Monkey testování	7
3	Přehled nástrojů	8
4	Zvolené nástroje	9
4.1	Jubula	9
4.2	SikuliX	9
4.3	Robot Framework	10
5	Srovnání nástrojů	11
6	SikuliX	14
6.1	Instalace	14
6.2	Tvorba testů	15
6.3	SikuliX-IDE	16
6.3.1	První skript	17
6.4	Java API	23
6.4.1	První test	23
6.4.2	Sofistikovanější testy	24
7	Sada ukázkových testů a jejich scénářů	26
7.1	Rozdělení testů	27
7.2	Statické prvky	28
7.3	Převody	29
7.4	Vymazání	30

7.5	Zjištěné chyby	32
7.5.1	Chybné převody z (na) decimetry	32
7.5.2	Chybné zaokrouhlení	32
7.5.3	Převod záporné hodnoty	32
7.5.4	Tlačítko Vymaž	33
8	Problémy práce se SikuliX	34
8.1	Rozlišení obrazovky	34
8.2	Rozměry screenshotů	35
8.3	Ukazatel myši	35
8.4	Nespolehlivé OCR	36
8.5	Nefunkčnost některých metod, tříd	36
8.5.1	Pozice a velikost okna	36
8.5.2	Stav aplikace	37
8.5.3	Spuštění aplikace	37
9	Monkey testy	38
9.1	Hloupá opice	38
9.2	Zpola inteligentní opice	38
9.3	Inteligentní opice	39
9.4	Monkey test pomocí SikuliX	39
10	Automatizace nástrojem	41
11	Závěr	43
11.1	Statistika	43
	Literatura	44
	Příloha A	46

1 Úvod

2 Testování softwaru

Na začátek je potřeba vysvětlit některé pojmy z oblasti testování. Budeme vycházet hlavně z [Roudenský – Havlíčková, 2013], [Patton, 2002] a [Herout, 2016].

2.1 Požadavky

Požadavky zachycují přání zákazníka na funkcionalitu softwaru. Dělí se na dvě skupiny:

- **Funkční** – popisují funkčnost služby vykonávané systémem, tedy co má vykonávat. Patří sem např.:
 - Uživatel bude moci vytvořit záznam pro nového zákazníka.
 - Systém automaticky odhlásí uživatele po 3 minutách nečinnosti.
- **Mimofunkční** – popisují určité vlastnosti systému, či omezující podmínky. V podstatě říkají, jaký by systém měl být. Sem patří např.:
 - Modul „Správa klientů“ bude dostupný pouze uživatelům s rolí správce.
 - Systém bude použitelný při zátěži 1000 uživatelů.

Je vhodné, aby se testeři zabývali i požadavky, neboť mohou již v rané fázi vývoje zachytit ty chybně formulované (nekonzistentní, neproveditelné, nekompletní, netestovatelné, nejednoznačné, více požadavků zapsaných jako jeden apod.).

2.2 Specifikace požadavků na software

Funkční i mimofunkční požadavky zákazníka, jejich analýza a dokumentace a všeobecný popis systému se zapisuje do dokumentu nazvaného specifikace požadavků na software. Na základě tohoto dokumentu probíhají následující fáze vývoje, proto je jeho správnost velmi podstatná.

K tomuto dokumentu se poté vztahuje i testování, konkrétně funkční testování, které kontroluje, zda software vyhovuje a splňuje požadavky zákazníka.

2.3 Kvalita softwaru

Kvalita softwaru je velmi obtížně definovatelný pojem. Pro její definici vzniklo několik norem. Ty jsou dnes zastaralé či nekonzistentní, proto jsou nahrazeny jednotným systémem norem ISO/IEC 25000-25099 v rámci projektu SQuaRe (Software Quality Requirements and Evaluation).

Např. norma ISO/IEC 25010 říká, že kvalita softwaru je míra, do jaké softwarový produkt splňuje stanovené a implicitní potřeby, je-li používán za stanovených podmínek.

2.3.1 FURPS

Dnes nejčastějším modelem kvality softwaru je tzv. FURPS, který vytvořila společnost Hewlett-Packard. Ten kvalitu popisuje pomocí těchto pěti charakteristik:

- **Functionality** (Funkčnost) – soubor požadované funkcionality, schopností a bezpečnostních aspektů systému.
- **Usability** (Použitelnost) – snadnost použití, konzistence, estetika, dokumentace apod.
- **Reliability** (Spolehlivost) – četnost a závažnost selhání, doba bezporuchového běhu, správnost výstupů, zotavení atd.
- **Performance** (Výkonnost) – odezva systému, výkon za různých podmínek, požadavky na systémové prostředí.
- **Supportability** (rozšiřitelnost/podporovatelnost) – škálovatelnost, udržovatelnost, testovatelnost, snadnost konfigurace.

Většinou se setkáme s modelem FURPS+, který navíc přidává kategorie jako omezení návrhu, požadavky na implementaci, požadavky na rozhraní a požadavky na fyzické vlastnosti.

2.4 Chyba, defekt, selhání

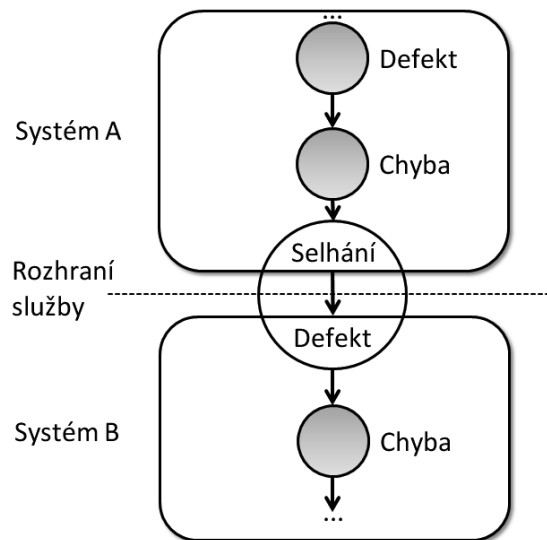
Během vývoje softwaru se do dokumentů či zdrojových kódů dostávají defekty, způsobující chyby a selhání. Tyto pojmy je velmi důležité rozlišovat. V následujících odstavcích bude jejich význam vysvětlen.

Selhání nastává v případě, že jeden nebo více výstupních stavů aplikace se odlišuje od stavu správného (nesplňuje specifikaci, případně specifikace nebyla kompletní nebo jednoznačná).

Právě toto odchýlení od očekávaného stavu se nazývá chybou. Původ chyby se nazývá defekt a označuje se tak vada v kódu či datech. Nejčastěji jej způsobí programátor chybou v kódu, špatným návrhem, nedostatečně či nesprávně pochopenou specifikací, nebo záměrnou sabotáží.

Jednotlivé pojmy na sebe tedy navazují následovně (viz obrázek 2.1): defekt (aktivace) → chyba (šíření) → selhání (příčina)...

Obrázek 2.1: Šíření chyby mezi systémy. Selhání systému A je pro příjemce jeho služby (systém B) externím defektem, který může vést k chybě a ta poté opět k selhání. Zdroj [Roudenský – Havlíčková, 2013]



2.5 Testování softwaru

Definice pojmu testování softwaru se více či méně liší téměř v každé publikaci. V knize [Roudenský – Havlíčková, 2013] je testování definováno jako proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů. Je zde důležité slovo spouštění, které z testování vyřazuje metody statické analýzy. Obecněji se však i tyto metody do testování řadí a jejich použití bývá ve vývoji softwaru vhodné.

Záměrem testování již není nalézat defekty, jak tomu bylo dříve. Nyní je jím získat informaci o kvalitě softwaru a o tom, jak moc splňuje požadavky zákazníka.

[Patton, 2002] uvádí řadu axiomů (obecně přijímaných pravd, které se nemusejí dokazovat) o testování. Čtyři nejdůležitější z nich jsou:

- **Žádný program není možné otestovat kompletně** – počet vstupů, výstupů a cest, které vedou skrze software je příliš velký.

- **Testování softwaru je postavené na riziku** – nemožnost otestování všech případů vede k tomu, že je možné nezachytit defekt ve scénáři, který se netestoval. Je proto důležité správně odhadnout možná rizika a testováním je minimalizovat.
- **Testování nikdy nemůže prokázat, že chyby neexistují** – testováním pouze můžeme dokázat, že chyby existují, ale jelikož není možné software kompletně otestovat, existenci chyb vyloučit nelze.
- **Čím více chyb najdeme, tím více jich v softwaru je** – kvůli tomu, že programátoři mívají špatné dny nebo dělají často stejné chyby, se chyby vyskytují ve skupinách. To znamená, že objevení jedné chyby zvyšuje pravděpodobnost dalších takových podobných chyb.

2.6 Úrovně testování

Během vývoje softwaru se testování nechá rozdělit do různých skupin, např. podle toho, kdy se testování provádí, jakým způsobem se provádí, jak se k testované aplikaci přistupuje, či jaká část aplikace se podrobuje testům. Rozlišují se např. tyto čtyři fáze:

- **Testování jednotek** (Unit Testing) – testování provádí obvykle sám programátor, který se snaží prokázat správnost fungování jednotlivých jednotek (nejmenších testovatelných součástí aplikace).
- **Integrační testování** (Integration Testing) – testuje se zapojení výše uvedených jednotek do aplikace. Správná funkčnost jednotek nezaručuje správnou funkčnost výsledné aplikace.
- **Systémové testování** (System Testing) – testuje se, zda aplikace splňuje požadavky zákazníka. Patří sem nejen funkční testování, ale i mimofunkční.
- **Akceptační testování** (User Acceptance Testing, UAT) – zde provádí testování sám zákazník a neomezuje se pouze na testování jako takové, ale na splnění před dohodnutých kritérií (dokumentace, manuály apod.).

2.7 Testování uživatelského rozhraní

Podle [Patton, 2002] má každá aplikace uživatelské rozhraní (UI – user interface). V dnešní době je dělíme do dvou skupin, grafické (GUI – graphical

user interface) a z příkazové řádky (CLI – command line interface). V obou případech UI slouží pro interakci a komunikaci s uživatelem. Je to prostředek pro zadávání vstupních data a získávání výstupů.

Použitelnost UI znamená, nakolik funkční a efektivní je práce uživatele s programem. K tomu se vztahuje pojem ergonomika – zabývá se návrhem a konstrukcí předmětů s ohledem na to, aby se s nimi pracovalo jednoduše. Vzhledem k použitelnosti se tedy bude testovat, zda je aplikace těžko srozumitelná, obtížně se s ní pracuje, je pomalá, nebo ji zákazník nebude považovat za vyhovující.

Dobré uživatelské rozhraní by mělo mít následující vlastnosti:

- **Dodržuje standardy nebo zásady** – pokud aplikace poběží na specifické platformě, je možné, že pro tuto platformu existují předepsané standardy a normy na vzhled a chování aplikací (tzv. look and feel), které je vhodné dodržovat.
- **Je intuitivní** – míry intuitivnosti softwaru jsou např.: Je UI zřetelné, vhodně rozložené? Nepřekáží vám v tom, co s ním hodláte dělat? Nachází se funkce a odpovědi tam, kde je očekáváte? Je jasné, co bude další akce v pořadí? Není příliš komplikované (neobsahuje moc funkcí)?
- **Je konzistentní** – uživatelé si zvyknou na určitý způsob ovládání a očekávají, že provedou stejnou akci velmi podobně i v jiném programu. Proto je důležité věnovat pozornost nejen konzistenci v rámci naší aplikace, ale i vůči ostatním. Pozor bychom si měli dávat na klávesové zkratky, výběry z nabídek, umístění tlačítek, výstupy aplikace apod.
- **Je flexibilní** – uživatelé mají rádi možnost volby. I v jednoduchých programech typu kalkulačka bývá na výběr alespoň ze dvou rozložení – standardního a vědeckého. To ovšem znesnadňuje testování, proto by voleb nemělo být příliš mnoho.
- **Je pohodlné** – software by neměl uživateli práci s ním nijak zneprůjemňovat. Je vhodné zaměřit se na náležitost (měl by působit dojmem, který odpovídá jeho záměru), ošetření chyb (upozornění před kritickou operací, návrat o krok zpět aj.), rychlost zpracování (chybové hlášky zobrazovat dostatečně dlouho, uvést odhadovaný čas do dokončení).
- **Je správné** – ověřuje se to, zda UI dělá to, co má. Pozornost by se měla věnovat marketingovým rozdílům (v aplikaci nejsou oproti marketingovým dokumentům funkce navíc, nebo chybějící funkce), jazyku

a pravopisu, nesprávným médiím (různorodé ikony, zvuky a jiné části UI).

- **Je užitečné** – přispívají dané funkce zvýšení užité hodnoty aplikace? Pomůže uživateli v jeho práci s aplikací? Nepodstatné funkce znamenají zbytečné testování navíc a pro zákazníka mohou být špatné.

2.8 Automatizace testování

Důvodem automatizace testování je zvýšení jeho efektivity. Manuální návrh, provedení a analýza testů jsou časově náročné a často velice monotónní práce. Testeři mohou jednak ušetřit čas, a jednak se sníží pravděpodobnost chybného vykonání testu z důvodu ztráty koncentrace testera v důsledku jednotvárné činnosti. Dalším přínosem je průběžně se stále zvětšující sada regresních testů. Ty slouží k otestování, zda nově přidaná funkcionality či úprava aplikace nezpůsobuje chyby v již funkčních částech.

Funkcionální testování UI je jednou z částí testování UI, která se nechá automatizovat. Existuje celá řada nástrojů, které umožňují simulování klikání a psaní na klávesnici. Další částí vhodnou pro automatizaci je kontrola, zda GUI obsahuje všechny prvky, a jestli je chování prvků konzistentní a stejné, jako požadované. V obou těchto disciplínách je pro testovací nástroj důležité, aby zvládal klikat a psát do aplikace, ale zároveň vyhodnocoval její stav (vstup je možné zadávat až tehdy, když je na to aplikace připravená, apod.).

2.8.1 Monkey testování

Specifickou oblastí testování GUI a automatického testování je tzv. monkey testování (testování za pomoci „cvičené opice“). To vychází z „Nekonečného opičího teorému“, [Patton, 2002], [Wikipedia, 2016], který říká: „Pokud bychom měli po nekonečnou dobu k dispozici nekonečně mnoho opic, které by neustále náhodně psaly na nekonečně mnoho klávesnic, statisticky vzato by mohly nakonec napsat všechny Shakespearovy hry.“

Nejde o automatizaci za účelem usnadnění práce a zvýšení efektivity, ale za účelem napodobit možné chování potenciálního uživatele. Přes všechno naše snažení a testování určitě v aplikaci zůstaly chyby, které by odhalil případný uživatel podobným nezasvěceným klikáním a psaním. Tomu je však možné alespoň částečně předejít pomocí monkey testování.

3 Přehled nástrojů

Existuje celá řada nástrojů pro testování GUI. V této kapitole následuje jejich přehled a výčet některých jejich vlastností. V tabulce 3.1 je uveden název nástroje, jeho licence resp. cena, jazyk, ve kterém se testy píší, platforma, na které nástroj funguje a která GUI je nástroj schopen testovat. Z bezplatných multiplatformních nástrojů byly vybrány tři a ty podrobněji prozkoumány a porovnány, viz následující kapitola.

Tabulka 3.1: Přehled nástrojů

Název	Licence/Cena	Skriptovací jazyk	Platforma	Jazykové omezení
AutoIt AutoIT [2015]	Freeware	BASIC-like	Windows	-
AutoHotKey [Mallet, 2015]	GNU GPLv2	AutoHotKey	Windows	-
AutoKey [Daniels, 2011]	GNU GPLv3	Python	Linux	-
SikuliX [Sikuli, 2015] [Hocke, 2015]	MIT License	Python, Ruby	Windows, Linux, Mac	-
Jubula [Jubula, 2015]	EPL 1.0	Drag & Drop, Java	Windows, Linux, Mac	Java, HTML, .NET, iOS
Robot Framework [Framework, 2015]	Apache License 2.0	Natural-like	Windows, Linux, Mac	Podle pluginů (Java, web, Android, iOS, ...)
Squish [Squish, 2015]	cca 2400 €/osoba	Python, JavaScript, Ruby, Perl, Tcl	Windows, Linux, Mac	-
eggPlant [eggPlant, 2015]	nedostupná, vázaná na stroj	SmartTalk, Drag & Drop, pomocí rozhraní eggDrive např. Java, C#, Ruby	Windows, Linux, Mac	-
UFT [UFT, 2015]	nedostupná	VBScript, Drag & Drop	Windows, Linux, Mac	-
Rational Functional Tester [RFT, 2015]	3300 \$/osoba	Nahrávání akcí	Windows, Linux	-
Ranorex [Ranorex, 2015]	690 €	C#, VisualBasic, nahrávání akcí	Windows	-
SilkTest [SilkTest, 2015]	nedostupná	C#, VisualBasic, Java	Windows	-
TestComplete [TestComplete, 2015]	889 €/stroj	Python, VBScript, JScript, C#Script, DelphiScript, C++Script, nahrávání akcí	Windows	-

4 Zvolené nástroje

Vzhledem k požadavkům na nástroje, které vyplývají z vazby na předmět KIV/OKS, jako je bezplatnost, schopnost fungování nezávisle na OS nebo podpora testování programů vytvořených technologií Java a webových aplikací, jsem z výše zmíněných vybral nástroje Jubula, SikuliX a Robot Framework. Každý z nástrojů bude stručně charakterizován a bude následovat podrobnější srovnání.

4.1 Jubula

Jubula je nástroj, který vznikl a je vyvíjen v rámci IDE Eclipse. Do projektu přispívá také firma BREDEX GmbH, která vytváří i tzv. standalone verzi, což je program, který je možné používat samostatně bez IDE Eclipse. Navíc obsahuje navíc některé nespécifikované funkce a nemusí být licencována pod EPL 1.0, jako je tomu u verze pro IDE Eclipse.

Pro tvorbu testovacích skriptů byla používána metoda Drag & Drop, popř. se akce určovaly klikáním na různé nabídky. V jedné z posledních verzí bylo vydáno Java API a skripty je tak možné psát pomocí jazyka Java. Mezi podporovaná testovaná rozhraní patří Java Swing, SWT, JavaFX, HTML a iOS. Výhodou této aplikace je také možnost její integrace do ostatních programů pro organizaci testování.

4.2 SikuliX

Sikuli (nověji SikuliX) je nástroj, který vznikl jako projekt skupiny User Interface Design Group na MIT, což odpovídá i jeho licenci – MIT License. Nyní jeho vývoj převzal Raimund Hock (aka RaiMan) společně s open-source komunitou.

Při tvorbě skriptů je možné využít pro SikuliX vlastní jazyk podobný přirozené angličtině, nebo některý ze zavedených, jako je Python, Ruby, Java, Jython, JRuby, Scala, Groovy, Clojure a další. Nástroj není omezený na určitá testovaná rozhraní, protože k identifikaci GUI používá rozpoznávání obrazu podle vzoru¹, dokáže simulovat ovládání myši a klávesnice nebo rozpoznávat text v obrázcích². Výhodou této aplikace je proto její nezávislost

¹Pomocí OpenCV, <http://opencv.org/>

²Pomocí Tesseract OCR, <https://github.com/tesseract-ocr>

vůči testovanému rozhraní. Cenou za to je pravděpodobné snížení její rychlosti. Použití SikuliX se neomezuje pouze na testování, ale je možné pomocí něj i automatizovat činnosti.

4.3 Robot Framework

Robot Framework je nástroj založený na pluginech a je open-source. Vývoj podporuje společnost Nokia Networks.

Základ nástroje, tzv. core framework, je vytvořený v jazyce Python. Knihovny je možné psát v jazyce Python nebo Java a samotné skripty pak v jazyce podobném přirozené angličtině. Díky dodržování jistého formátování je pro člověka velmi přehledný. Mezi podporovaná testovaná rozhraní patří např. Android, iOS, Java Swing, webové aplikace, databáze a aplikace vytvořené pro OS Windows. Výhodou této aplikace je možnost si chybějící modul pro testování určitého rozhraní vytvořit a používat.

5 Srovnání nástrojů

Pro srovnání nástrojů byl vytvořen návrh multikriteriálního hodnocení, který se snaží nástroje hodnotit z různých úhlů a vytvořit tak komplexní klasifikaci. Každé z hodnocených částí je možné přiřadit vlastní váhu. Ta určuje důležitost hodnotícího kritéria pro každého jedince a tím napomáhá výběru vhodného nástroje. V obrázku 5.1a je návrh ukázán a je vidět výsledek pro mnou zvolené váhy, viz obrázek 5.1b. Jako nejvhodnější se jeví použití nástroje SikuliX. Dále se budu věnovat jednotlivým hodnotícím kritériím.

Možnost vytváření testů je jedno z nejdůležitějších kritérií vzhledem k vazbě na předmět KIV/OKS. Hlavním požadavkem bylo, aby bylo možné testy tvořit v jazyce Java. Dále jsem vybral několik skriptovacích jazyků a metod.

Podpora testovaných rozhraní byla dalším z rozhodujících kritérií. Hlavními platformami měly být aplikace vytvořené pomocí jazyka Java a webové aplikace. Opět jsem přidal některé další běžné platformy. Nástroj by měl být též multiplatformní, proto je jedním z kritérií podpora operačních systémů.

Reportování výsledků testů, složitost jejich tvorby a jejich přehlednost může napomoci vývojáři diagnostikovat případnou chybu. Také je přínosné znát stav obrazovky a to zajistí screenshot. Díky tomu se stává vývoj jednodušší, a proto jsem toto kritérium také zařadil do hodnocení.

Dále jsem přidal kritérium univerzálnosti nástroje s poněkud individuálním ohodnocením. To je zde myšleno tak, co obecně nástroj dokáže, ale co není podstatné z pohledu předmětu KIV/OKS. Například Jubula je čistě testovací nástroj, ale SikuliX se dá použít navíc pro automatizaci pracovních postupů.

Posledním kritériem je vhodnost nástroje pro účely předmětu KIV/OKS. Jedná se hlavně o to, jak zapadá do konceptu výuky, jak je práce s ním složitá a jaké má nároky na studentovy znalosti.

Vysvětlivky termínů z dále použitých tabulek:

- **Natural-like** – víceméně okleštěný přirozený jazyk založený na angličtině,
- **Složitost tvorby** – míní se tím složitost přípravy reportu a v tabulce vyšší počet bodů znamená jednodušší tvorbu pro tvůrce skriptů

Z uvedeného multikriteriálního hodnocení je zřejmé, že vybrané nástroje jsou v podstatě vyrovnané. To ostatně potvrzuje i jejich poměrné zastoupení

(a) Multikriteriální hodnocení

Kritéria multikriteriálního hodnocení			
Možnosti vytváření skriptů	Jubula	RobotFramework	SikuliX
Java	Ano	Ano	Ano
Python	Ne	Ano	Ano
Ruby	Ne	Ano	Ano
Drag & Drop	Ano	Ne	Ne
Natural-Like	Ne	Ano	Ano
Body	2	4	4
Podpora testovaných rozhraní			
SWT	Ano	Ano	Ano
Java Swing	Ano	Ano	Ano
JavaFX	Ano	Ne	Ano
.NET	Ano	Ano	Ano
HTML	Ano	Ano	Ano
Android	Ne	Ano	Ano
iOS	Ano	Ano	Ano
Body	6	6	7
Podporované operační systémy			
Linux/Unix	Ano	Ano	Ano
Windows	Ano	Ano	Ano
Mac	Ano	Ano	Ano
Body	3	3	3
Reportování výsledků testů			
HTML	Ano	Ano	Ano (pomocí JUnit a Ant, resp. HTML Test Runner a unittest)
XML	Ano	Ano	Ano (pomocí JUnit a Ant, resp. HTML Test Runner a unittest)
Složítost tvorby (čím vyšší, tím snazší)	4	4	1
Přehlednost reportu	4	4	3
Body	10	10	6
Screenshot při chybě	Ano	Ano	Ano
Body	1	1	1
Univerzálnost nástroje	2	3	5
Vhodnost nástroje pro účely KIV/OKS	4	3	4

mezi uživateli. SikuliX byl zvolen též po diskuzích s vedoucím práce a to pro jeho vlastnost naprosté nezávislosti na testovaném rozhraní. Tato vlastnost se významně hodí v předmětu KIV/OKS, protože je možné dát nástroj do kontrastu s nástrojem Selenium. Jinými slovy řečeno SikuliX je principiálně odlišný nástroj, což není možné říci o např. Jubule.

(b) Výsledek multikriteriálního hodnocení

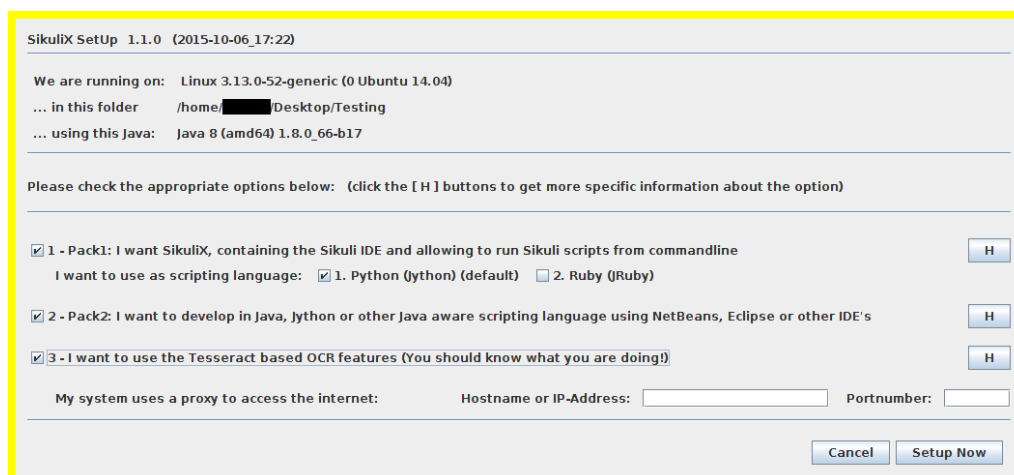
Váhy	Vlastní váha	Celková váha	Váha v %
Možnosti vytváření skriptů	4	0,10	10,3
Podpora testovaných rozhraní	8	0,21	20,5
Podporované operační systémy	3	0,08	7,69
Reportování výsledků testů	5	0,13	12,8
Screenshot při chybě	4	0,10	10,3
Univerzálnost nástroje	7	0,18	17,9
Vhodnost nástroje pro účely KIV/OKS	8	0,21	20,5
Součet	39	1	100
Celkové zhodnocení nástrojů	Jubula	RobotFramework	SikuliX
Možnosti vytváření skriptů	0,21	0,41	0,41
Podpora testovaných rozhraní	1,23	1,23	1,44
Podporované operační systémy	0,23	0,23	0,23
Reportování výsledků testů	1,28	1,28	0,77
Screenshot při chybě	0,10	0,10	0,10
Univerzálnost nástroje	0,36	0,54	0,90
Vhodnost nástroje pro účely KIV/OKS	0,82	0,62	0,82
Celkové skóre	4,23	4,41	4,67

6 SikuliX

6.1 Instalace

Po stažení balíčku započne instalace jeho spuštěním¹. Je ukázána instalace v Linuxu, avšak instalace ve Windows je obdobná. V průběhu máme na výběr různé možnosti, jak chceme nástroj používat, viz obrázek 6.1. Např. zda chceme používat SikuliX-IDE a Python nebo Ruby, jestli budeme používat jiné IDE a Javu a zda chceme používat OCR funkce. Zaškrtneme všechna políčka kromě *Ruby (JRuby)* a klikneme na *Setup Now*. Jsme dotázáni, zda chceme balíčky stáhnout, nebo ukončit instalaci. Zvolíme *Yes*. Další dotaz je na verzi Jythonu, kterou chceme použít, s upozorněním, že může nastat problém se znaky v kódování UTF-8. Opět zvolíme *Yes*. Začne vytváření souborů a měla by se otevřít dvě okna jako na obrázku 6.2, obě potvrdíme tlačítkem *OK*. Pokud vše proběhne v pořádku, vzniknou v adresáři soubory podobné těmto² *runsikulix*, *SetupStuff*, *SikuliX-1.1.0-SetupLog.txt*, *sikulixapi.jar*, *sikulix.jar*.

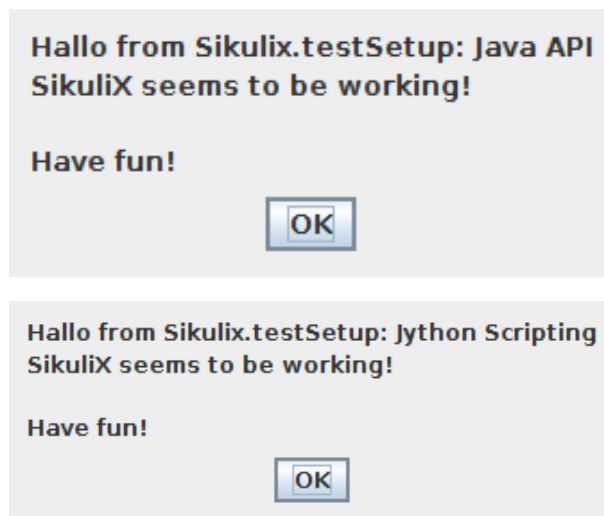
Obrázek 6.1: Instalace SikuliX



¹Je potřeba instalace JRE nebo JDK 6 a vyšší, v linuxové distribuci balíky *libopencv-core2.4*, *libopencv-imgproc2.4*, *libopencv-highgui2.4*, *libtesseract3* a *wmctrl* [Hocke, 2015]

²Může se lišit na různých OS

Obrázek 6.2: Test instalace



6.2 Tvorba testů

Budeme pracovat s webovou aplikací *Převodník* dostupné z url `http://oks.kiv.zcu.cz/Prevodnik/Prevodnik`. V té se vyskytují záměrné chyby a bude sloužit jako cvičná aplikace pro tuto práci.

Pro tvorbu testů pomocí SikuliX jsou nejdůležitější snímky (screenshoty) řídicích prvků, které bude SikuliX hledat a případně používat k některým akcím. Je tedy vhodné si nejprve aplikaci spustit, vybrat příslušné prvky a vytvořit jejich snímky. Při jejich tvorbě se doporučuje preciznost a přesnost, neboť v jistých situacích mohou nastat problémy, které budou zmíněny později.

Pokud je snímků více, je vhodné je třídit do adresářů. To není nutné, ale zlepšuje to čitelnost kódu a usnadňuje práci s nástrojem. Adresáře mohou např. sdružovat snímky prvků, které jsou si nějakým způsobem podobné (tlačítka, textová pole, výběrové seznamy, chybové hlášky, apod.). Stejně tak je vhodné snímky pojmenovávat na základě toho, co obsahují (vstupní textové pole, label Vstup, apod.).

Dle [Hocke, 2013] SikuliX interně používá třídu `ImageIO` z Javy. Podporované formáty jsou tedy `bmp`, `wbmp`, `jpg`, `jpeg`, `png` a `gif`. Vzhledem k vlastnostem se doporučuje používat formát `png`.

Pokud je vytvářený test jednoduchý a není potřeba většího množství testů, je jednodušší vytvořit jej pomocí SikuliX-IDE. Pokud však chceme aplikaci testovat podrobněji a psát velké množství testovacích případů, je vhodnější použít některý z podporovaných programovacích jazyků a využít tak jeho možnosti jako nadstavbu nad SikuliX.

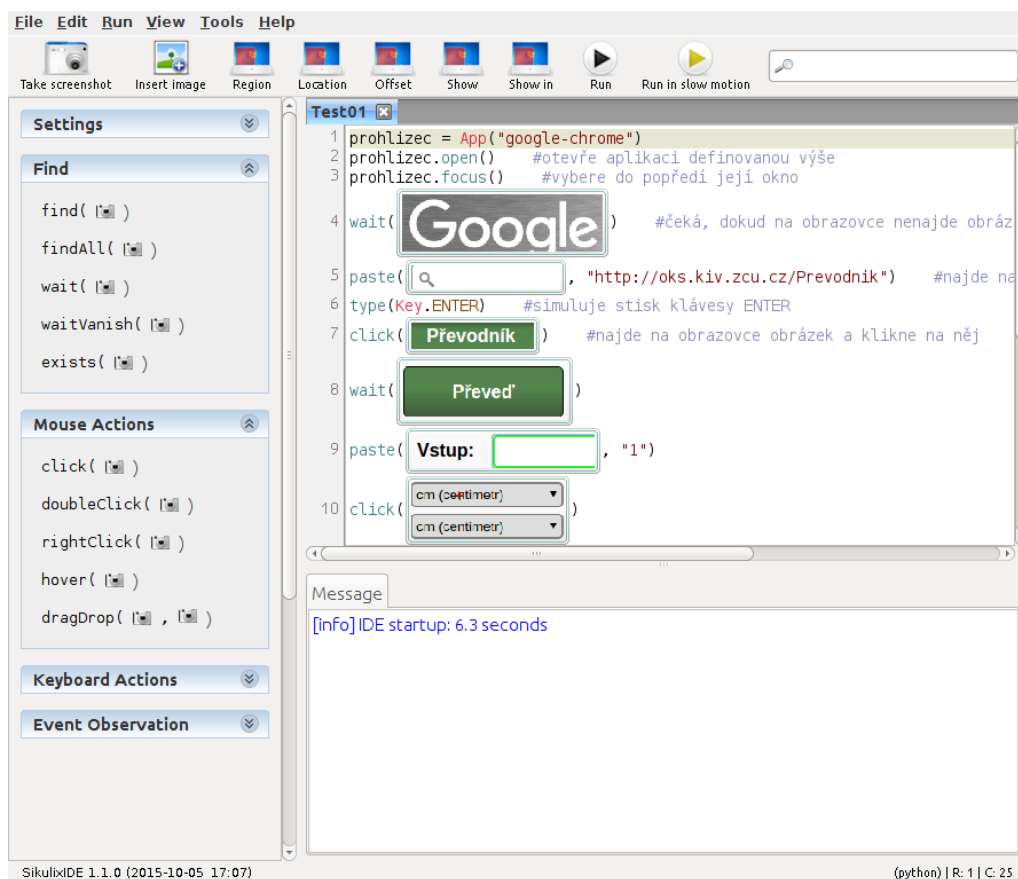
6.3 SikuliX-IDE

Dle [Hocke, 2015] je možné SikuliX-IDE spustit různými způsoby:

1. Spuštěním souboru *SikuliX.app* (Mac) nebo *SikuliX.exe* (Windows),
2. dvojklikem na soubor *runsikulix* (Linux) nebo *runsikulix.cmd* (Windows),
3. z příkazové řádky příkazem
`java -jar cesta/k/sikulix.jar [volitelné parametry]`

Po spuštění vypadá IDE jako na obrázku 6.3. Jako parametry se v metodách, ve kterých je to možné, ukazují obrázky vzorů, podle kterých se na obrazovce nástroj orientuje, případně cesta k nim.

Obrázek 6.3: SikuliX-IDE



6.3.1 První skript

Skript se připravuje v SikuliX-IDE, které je vidět na obrázku 6.3. Kód, který je vidět v 6.1, nemusí být v IDE identický, ale cesta k obrázku může být nahrazena jeho názvem, pokud je tato možnost zvolena v nastavení nástroje. K tvorbě jsou v IDE užitečné pomůcky, které se nacházejí v levém a v horním panelu.

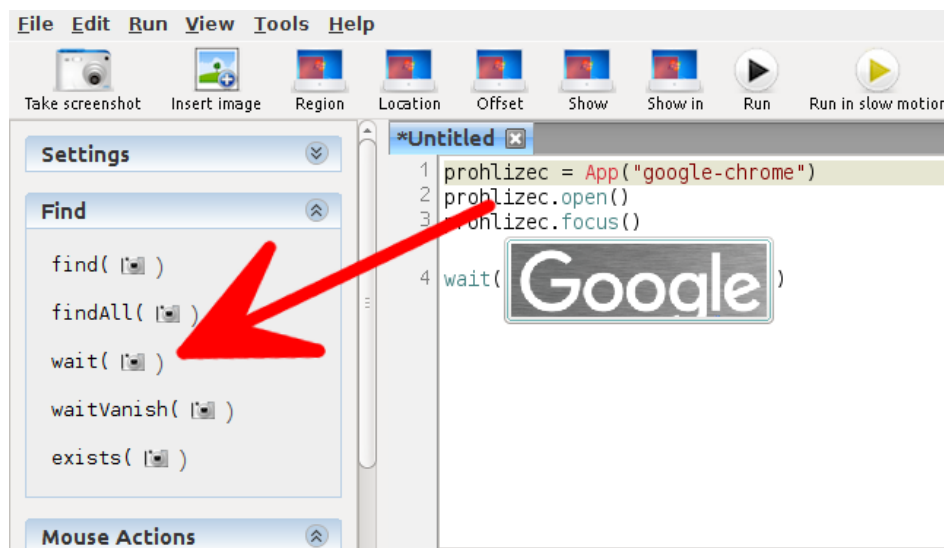
Pro spuštění aplikace (např. webového prohlížeče) se do příslušné metody zadá cesta k jeho spustitelnému souboru. Pokud se používá OS Linux, stačí zadat terminálový příkaz pro spuštění (např. "google-chrome", "firefox", apod.).

Skript pracuje tak, že se otevře prohlížeč, který přejde na adresu `http://oks.kiv.zcu.cz/Prevodnik`. Klikne na odkaz *Převodník*, do vstupního pole vloží *1* a stiskne *Převěd*. Z pole s výsledkem přečte text a porovná jej s předpokládanou hodnotou *2,54*. Pokud si odpovídají, objeví se dialogové okno s potvrzením, jestliže ne, zobrazí se chybová hláška. Obdobně je tomu v následující části, kde se pouze kontroluje existence obrázku.

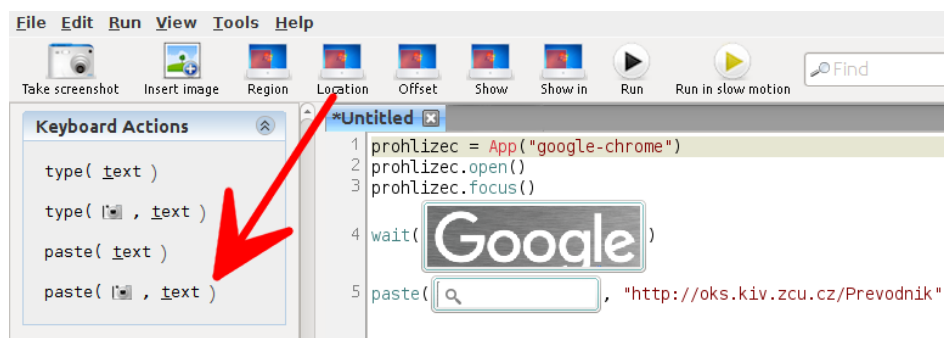
Pro vytvoření skriptu postupujeme následovně:

1. Spustíme SikuliX-IDE jednou z výše uvedených metod.
2. Napíšeme kód pro otevření a vybrání okna prohlížeče do popředí:

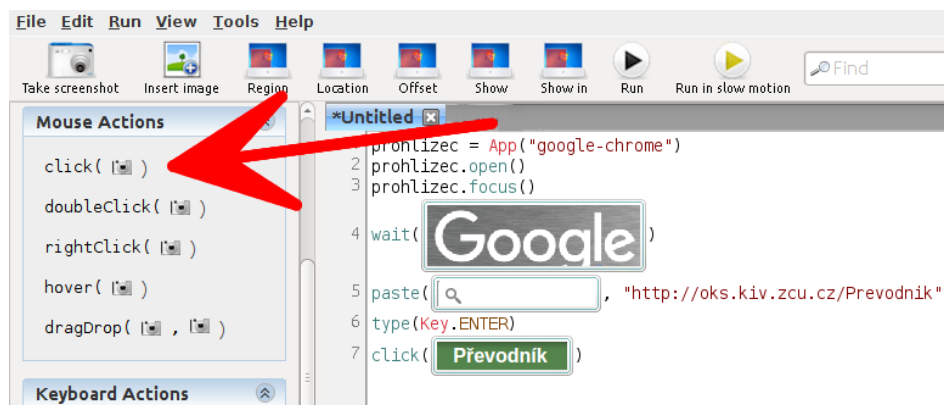
```
prohlizec = App("google-chrome")
prohlizec.open()
prohlizec.focus()
```
3. Spustíme prohlížeč.
4. V SikulixIDE v levém menu klikneme na `wait([obrázek])` a provedeme screenshot statické části aplikace. Skript tedy bude vypadat přibližně takto:



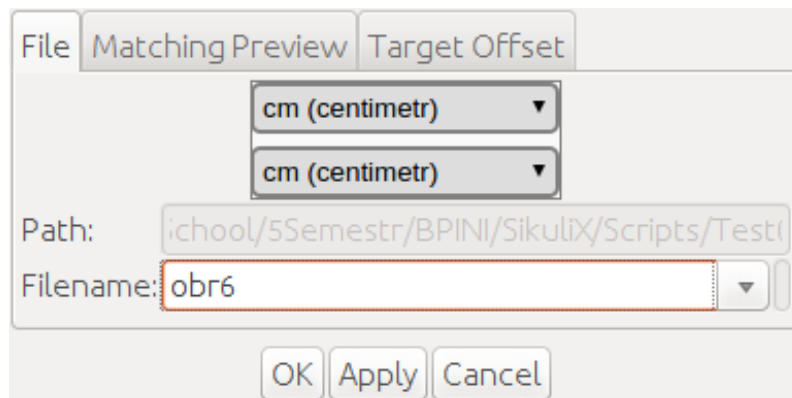
5. V levém menu otevřeme podmenu *Keyboard Actions* a vybereme `paste(, text)`. Nyní provedeme screenshot políčka pro URL adresu a jako druhý parametr funkce zadáme URL adresu aplikace *Převodník* ("http://oks.kiv.zcu.cz/Prevodnik"). Přidaná část tedy vypadá přibližně takto:



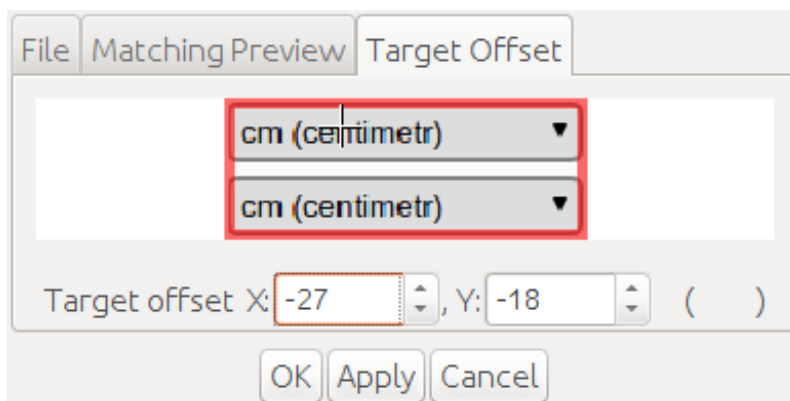
6. Na další řádce zadáme příkaz pro stisknutí klávesy Enter:
`type(Key.ENTER)`
7. V prohlížeči otevřeme webovou stránku s aplikací *Převodník*. V SikuliX-IDE klikneme v levém menu na `click()` a provedeme screenshot záložky, která odkazuje na *Převodník*.



8. Klikneme na záložku aplikace *Převodník*. V SikuliX-IDE opět vybereme `wait([image])` a uděláme snímek tlačítka *Převod*.
9. Dále vybereme `paste([image], text)` a vytvoříme snímek vstupního pole. Snímek musí obsahovat popisek *Vstup*:, aby byl jednoznačně identifikovatelný. Také musí obsahovat část vstupního pole dost velkou na to, aby ve středu snímku bylo toto pole, nikoli jeho okolí. Vložení hodnoty se provádí do středu snímku, což se ovšem dá upravit, jak si ukážeme v dalším kroku. Jako druhý parametr zadáme hodnotu "1".
10. Vybereme možnost `click([image])` a vytvoříme screenshot obou výběrových seznamů. Nyní klikneme na obrázek seznamů v SikuliX-IDE a otevře se okno:



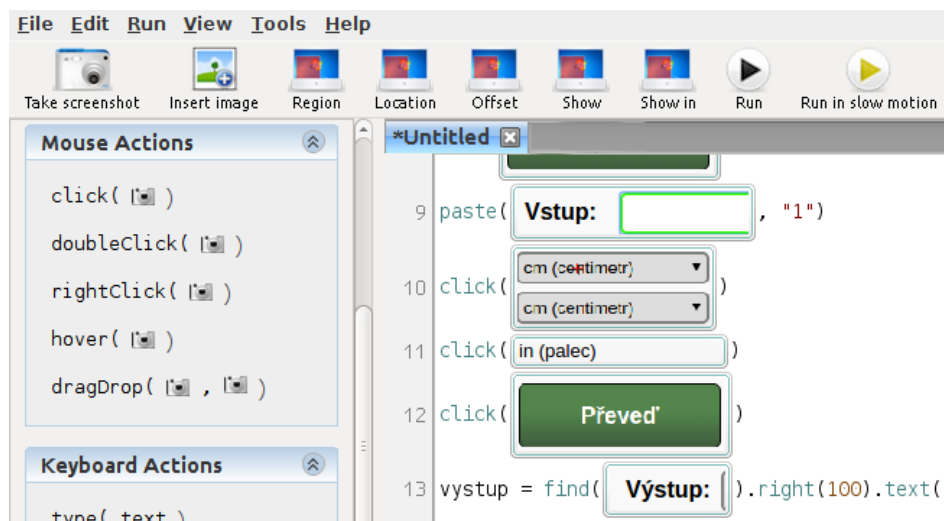
Klikneme na *Target Offset*, které vypadá následovně:



Zde buď zadáme příslušné hodnoty do vstupních políček, nebo jen vybereme v obrázku požadované místo. To nám zajistí, že SikuliX neklikne do středu obrázku, ale na zvolené místo v něm. Potvrdíme tlačítkem *OK* a pokračujeme v tvorbě skriptu.

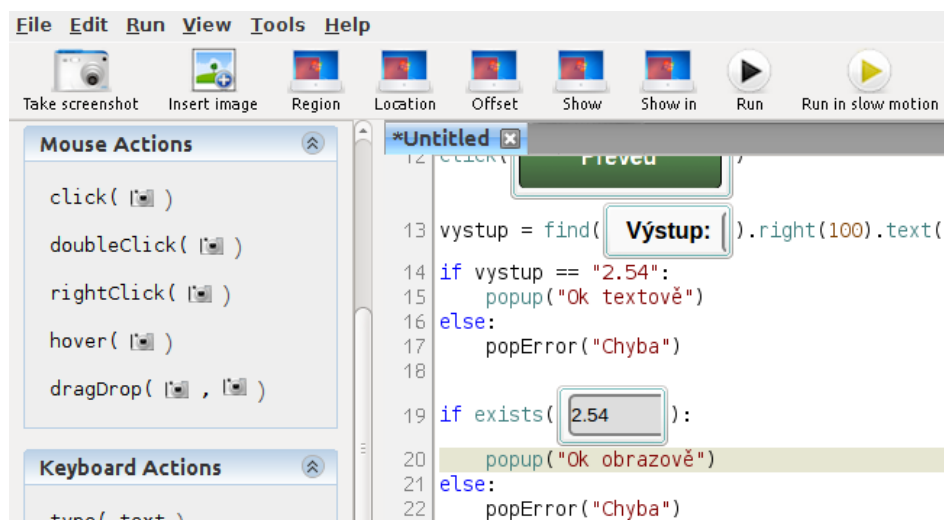
11. Nyní nastává problém. Potřebujeme vytvořit screenshot jedné položky ve výběrovém seznamu, ale SikuliX-IDE nám k tomu nedává možnost. Řešením je např. v prohlížeči výběrový seznam rozbalit, provést screenshot celé obrazovky pomocí tlačítka *PrintScreen* a v nějakém grafickém editoru vyříznout pouze tu část, kterou chceme (*in (palec)*). Tu poté uložíme do složky s vytvářeným skriptem a pouze doplníme následující kód: `click("nazev-snimku.png")`
12. Na novou řádku napíšeme kód `click()` a do závorek zkopírujeme výše použitý obrázek tlačítka *Převéd.*
13. Na další řádce si vytvoříme proměnnou *vystup* a přiřadíme do ní text z výstupního textového pole. To se provede následujícím způsobem:
`vystup = find("label-vystup.png").right(100).text()`
 Funkce *right()* říká, že pracujeme s oblastí širokou 100 pixelů napravo od nalezeného obrázku (popisku *Výstup:*). Funkce *text()* provede OCR³ a vrátí zjištěný text.

³Optical Character Recognition – optické rozpoznání znaků



14. V další části provedeme kontrolu zjištěného textu. Ten má obsahovat výslednou hodnotu – 2.54. Vložíme do programu následující úsek kódu:

```
if vystup == "2.54":
    popup("Ok textově")
else:
    popError("Chyba")
```
15. Další následuje „optická“ kontrola podle screenshotu. Potřebujeme si připravit do výstupního textového pole správný výsledek (hodnotu 2.54). To lze např. udělat tak, že provedeme převod hodnoty 2.54 mezi stejnými jednotkami. Jakmile se nám toto podaří, uděláme snímek výsledku a uložíme jej ke skriptu. Zkopírujeme kód z bodu 16. a upravíme if-větev tak, že smažeme porovnání proměnné s hodnotou a napíšeme jako podmínku následující: `exists("nazev-snimku.png")`.



16. Jako poslední bod je zavření prohlížeče. To se provede příkazem `prohlizec.close()`.

Tímto postupem by měl vzniknout skript téměř totožný s tím, který je uveden v kódu 6.1. Nebude obsahovat komentáře a mohou se lišit názvy snímků.

Kód 6.1: První skript

```
#prohlizec je promenna
prohlizec = App("google-chrome")
prohlizec.open()          #otevře aplikaci definovanou
                           #vyse
prohlizec.focus()         #vybere do popredi její okno
#ceka, dokud na obrazovce nenajde obrazek
wait("obr1.png")
#najde na obrazovce obrazek a vlozi do neho text
paste("obr2.png", "http://oks.kiv.zcu.cz/Prevodnik")
type(Key.ENTER) #simuluje stisk klavesy ENTER
#najde na obrazovce obrazek a klikne na nej
click("obr3.png")
wait("obr4.png")
paste("obr5.png", "1")
#klikne o 27px vyse a 18px vlevo od nalezeného
#obrazku
click(Pattern("obr6.png").targetOffset(-27,-18))
click("obr7.png")
click("obr4.png")
#prectete text z casti, která je vpravo od nalezeného
#obrazku 100px široka, do promenne vystup
vystup = find("obr8.png").right(100).text()
if vystup == "2.54":
    #pokud rozpoznany text souhlasí se zadáním,
    #otevře se vyskakovací okno
    popup("Ok textove")
else:
    popError("Chyba")      #jinak se zobrazí chybové
                           #okno

if exists("obr9.png"):
    #pokud na obrazovce existuje obrazek, otevře
    #se vyskakovací okno
    popup("Ok obrazove")
else:
```

```
        popError("Chyba")
prohlizec.close()      #ukonci aplikaci
```

6.4 Java API

Dále bylo zkoumáno Java API, které SikuliX poskytuje. Pro jeho použití je potřeba mít při překladu a spuštění nastavený v classpath *sikulixapi.jar*. Toho docílíme např. tak, že použijeme v příkazové řádce dvou příkazů

```
javac -cp sikulixapi.jar:. Test01.java
```

```
java -cp sikulixapi.jar:. Test01
```

Syntaxe, kterou SikuliX v Java API využívá, je velmi podobná té v SikuliX-IDE.

6.4.1 První test

První test s použitím Java API, viz kód 6.2, je téměř identický s tím, který byl vytvořen pomocí SikuliX-IDE.

Kód 6.2: První test Java API

```
import org.sikuli.basics.Settings;
import org.sikuli.script.*;
import javax.swing.*;

public class Test01 {

    static Screen s;
    static App prohlizec;

    public static void main(String [] args) {
        Settings.OcrTextSearch = true;
        Settings.OcrTextRead = true;

        s= new Screen();
        prohlizec = new App("google-chrome");
        prohlizec.open();
        prohlizec.focus();

        try {
            s.wait("obr1.png");
            s.paste("obr2.png");
            s.type(Key.ENTER);
        }
```

```

s.click("obr3.png");
s.wait("obr4.png");
s.paste("obr5.png", "1");
s.click(new Pattern("obr6.png").targetOffset(
    -27,-18));
s.click("obr7.png");
s.click("obr4.png");
String t = s.find("obr8.png").right(
    100).text();
if (Double.parseDouble(t) == 2.54) {
    JOptionPane.showMessageDialog(null, "Ok" +
        " textove");
} else {
    JOptionPane.showMessageDialog(null, "Chyba");
}
if (s.exists("obr9.png") != null) {
    JOptionPane.showMessageDialog(null, "Ok" +
        " obrazove");
} else {
    JOptionPane.showMessageDialog(null, "Chyba");
}
prohlizec.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

6.4.2 Sofistikovanější testy

S využitím knihoven *JUnit* a *Log4j* (ani jedna z těchto knihoven není pro běh SikuliX bezprostředně nutná) byly vytvořeny čtyři testy, viz kód 11.1. První test skončí negativně, druhý pozitivně, třetí pozitivně a čtvrtý negativně.

Knihovna *JUnit* byla použita z toho důvodu, že nám pomůže jednak s organizací testů a jejich spouštěním, a jednak s jejich vyhodnocováním. Dále obsahuje metody pro vyhodnocování a porovnávání hodnot, tzv. *asserty*.

Log4J je knihovna, která slouží k logování informací do souborů. Umožňuje vlastní konfiguraci výstupních souborů a spoustu dalších funkcí. Použita byla z toho důvodu, že během testování je vhodné zaznamenávat prováděné činnosti z důvodu jednoduššího zjištění případného selhání testu. SikuliX poskytuje informace o tom, kam klikal či psal. Ty vypisuje na standardní

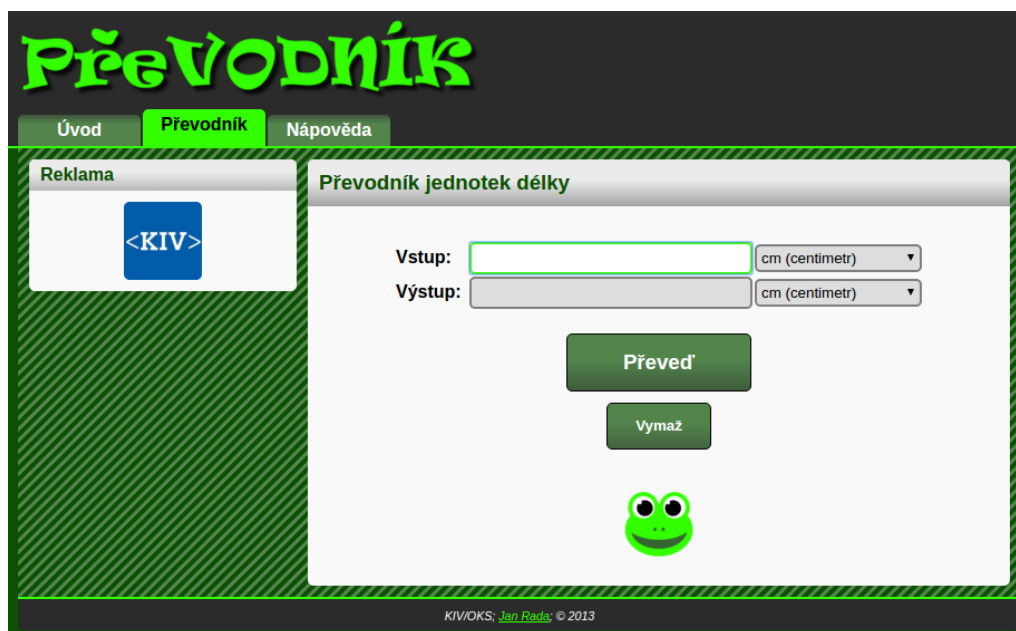
výstup, avšak poskytuje metodu, které se předá instance *Logger*, který poté SikuliX použije pro logování. Následuje ukázka zapisovaných informací, formátovaných vlastní konfigurací loggeru, viz 11.2.

```
18.04.2016 12:58:45 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[info] runcmd: lsb_release -i -r -s  
18.04.2016 12:58:46 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] App.focus: [1:PreVODNIK]  
18.04.2016 12:58:50 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(925,298)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:50 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(1128,298)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:51 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(1128,422)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:51 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(1128,333)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:52 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(1127,457)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:52 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(960,400)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:53 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(960,587)@S(0)[0,0 1920x1080]  
18.04.2016 12:58:58 [ERROR - TS03VymazaniJavaFX.  
TC03_01_02PrevodChyba()] - Vstupni vyberovy seznam nema  
defaultni hodnotu  
18.04.2016 12:58:59 [INFO - TS03VymazaniJavaFX.invoke0()] -  
[log] CLICK on L(661,198)@S(0)[0,0 1920x1080]
```

7 Sada ukázkových testů a jejich scénářů

Jak již bylo zmíněno dříve, existuje webová aplikace *Převodník* dostupná z url <http://oks.kiv.zcu.cz/Prevodnik/Prevodnik> se záměrnými chybami. Pro potřeby práce a pro názornou ukázkou odlišnosti přístupu k testování webové a desktopové aplikace byla vytvořena téměř identická aplikace *Převodník* pomocí technologie JavaFX, jejíž kód se nachází na přiloženém CD. Následující sada testů a jejich scénářů se vztahuje k těmto aplikacím, jejichž vzhled je ukázán na obrázcích 7.1 a 7.2.

Obrázek 7.1: Webová aplikace Převodník



Ze scénářů i z přiloženého zdrojového kódu je patrné, že přístup k testování obou aplikací je totožný. Testy vypadají stejně jak pro desktopovou aplikaci, tak pro webovou. Jediné rozdíly nastávají ve způsobu spouštění aplikací, zacházení s nimi a v cestě k použitým řídicím screenshotům. Příčinou je to, že se jedná o téměř identicky vypadající a chovající se aplikace. Dále budou uvedeny vždy jen stručné ukázky testů s vyznačenými případnými odlišnostmi.

Obrázek 7.2: Aplikace Převodník vytvořená pomocí JavaFX

7.1 Rozdělení testů

Scénáře byly rozděleny do tří částí a každá část poté může obsahovat další skupiny, které sjednocují tematicky si blízké testy. Struktura scénářů tedy vypadá přibližně takto:

1. Statické prvky
 - 1.1. Přítomnost prvků
 - 1.2. Editovatelnost polí
 - 1.3. Úplnost výběrových seznamů
2. Převody
 - 2.1. Happy Day Scenario
 - 2.2. Stejně jednotky
 - 2.3. Varianty Vstup OK
 - 2.4. Vše na metr
 - 2.5. Vše z metru
 - 2.6. Varianty Vstup CHYBA
 - 2.7. Všechny vstupy na všechny výstupy
 - 2.8. Hraniční hodnoty
3. Vymazání

Testovací případy mají předponu *TC* za níž následuje číslo testovacího případu a jeho název. Pokud tedy např. testujeme úplnost výstupního výběrového seznamu, název bude *TC01_03_02VystupniVyberovySeznam*. Číslo testovacího případu se tvoří dle jeho příslušnosti do části (TestSuite) ve výčtu výše. Patří do *Úplnost výběrových seznamů* a tomu odpovídá první část čísla, *01_03*. Označení *02* je pořadové číslo testu v rámci dané kategorie.

7.2 Statické prvky

Scénáře v této části pouze zkontrolují, zda testovaná aplikace obsahuje všechny prvky, jako např. tlačítka či vstupní pole. Dále se zkoumá, zda je vstupní pole editovatelné a výstupní pole nikoli. Poté se zjistí, zda jsou ve výběrových seznamech obsaženy všechny položky.

Následující úsek kódu 7.1 demonstruje, v jakém duchu se testují statické prvky aplikace.

Kód 7.1: Test přítomnosti statického prvku

```
@Test
public void TC01_01_02VstupniTextovePole() {
    if (run) {
        try {
            assertTrue("Vstupni textove pole neexistuje",
                s.find(pngs + "labely/vstupLabel.png").
                    right().grow(0, 20).exists(pngs +
                        "textovaPole/vstupniTextovePole.png") !=
                    null);
        } catch (FindFailed | AssertionError e) {
            s.capture().save("errors", screenshotName());
            logger.error(e.getMessage());
            fail(e.getMessage());
        }
    } else {
        logger.error("Setup failed");
        fail("Setup failed");
    }
}
```

7.3 Převody

V této části jsou zpracované funkční testy konkrétních převodů. Nejprve se provedou testy Happy Day Scenario – scénář, kdy vše dopadne podle očekávání. Poté se zkontrolují převody mezi stejnými jednotkami, převody s možnými korektními i nekorektními vstupy, převody mezi všemi jednotkami a nakonec převody s hraničními hodnotami.

Část kódu v 7.2 ukazuje, jak se mohou tvořit testy převodů.

Kód 7.2: Test převodu

```
@Test
public void TC02_04_01PrevodCmNaM() {
    if (run) {
        try {
            s.find(pngs + "labeled/vstupLabel.png").right().
                grow(0, 20).click(pngs + "textovaPole/" +
                    "vstupniTextovePole.png");
            s.paste("1");
            Match hledani = s.find(pngs + "labeled/" +
                "vstupLabel.png").right().grow(0, 20).find(
                pngs + "vyberoveSeznamy/vstupniVyberovy" +
                "Seznam.png");
            hledani.click();
            hledani.below().click(pngs + "vyberove" +
                "Seznamy/vstupModryCm.png");
            hledani = s.find(pngs + "labeled/vystupLabel" +
                ".png").right().grow(0, 20).find(pngs +
                "vyberoveSeznamy/vystupniVyberovySeznam" +
                ".png");
            hledani.click();
            hledani.below().click(pngs + "vyberove" +
                "Seznamy/vystupM.png");
            s.click(pngs + "tlacitka/tlacitkoPreved.png");
            s.wait(pngs + "tlacitka/tlacitkoPreved.png",
                5);
            assertTrue("Ocekavano: 0.01, zjisteno neco " +
                "jineho", s.find(pngs + "labeled/vystup" +
                "Label.png").right(200).grow(0, 10).exists(
                pngs + "vystupy/vystup0_01.png") != null);
        } catch (FindFailed | AssertionError e) {
            s.capture().save("errors", screenshotName());
            logger.error(e.getMessage());
        }
    }
}
```

```

        fail(e.getMessage());
    }
} else {
    logger.error("Setup failed");
    fail("Setup failed");
}
}
}

```

7.4 Vymazání

V této části se testuje funkčnosti tlačítka *Vymaž*. Otestuje se případ, kdy se nevyskytla chybová hláška, i ten, kdy se vyskytla.

Kód 7.3 je jednou z možností, jak testovat správnou funkčnost tlačítka *Vymazat*.

Kód 7.3: Test funkčnosti tlačítka *Vymazat*

```

@Test
public void TC03_01_02PrevodChyba() {
    if (run) {
        try {
            s.find(pngs + "labeled/vstupLabel.png").right().
                grow(0, 20).click(pngs + "textovaPole/" +
                    "vstupniTextovePole.png");
            s.paste("abc");
            Match hledani = s.find(pngs + "labeled/" +
                "vstupLabel.png").right().grow(0, 20).
                find(pngs + "vyberoveSeznamy/vstupni" +
                    "VyberovySeznam.png");
            hledani.click();
            hledani.below().click(pngs + "vyberove" +
                "Seznamy/vstupM.png");
            hledani = s.find(pngs + "labeled/vystupLabel" +
                ".png").right().grow(0, 20).find(pngs +
                "vyberoveSeznamy/vystupniVyberovySeznam" +
                ".png");
            hledani.click();
            hledani.below().click(pngs + "vyberove" +
                "Seznamy/vystupM.png");
            s.click(pngs + "tlacitka/tlacitkoPreved.png");
            s.wait(pngs + "tlacitka/tlacitkoPreved.png",
                5);
        }
    }
}

```

```

    assertTrue("Nenalezeno upozorneni o chybe", s.
        exists(pngs + "chyby/chybaNeplatneCislo") !=
            null);
    s.click(pngs + "tlacitka/tlacitkoVymaz.png");
    s.wait(pngs + "tlacitka/tlacitkoPreved.png",
        5);
    assertTrue("Vstupni pole neni prazdne", s.
        find(pngs + "labeled/vstupLabel.png").right().
        grow(0, 20).exists(pngs + "textovaPole" +
            "/vstupniTextovePole.png") != null);
    assertTrue("Vystupni pole neni prazdne", s.
        find(pngs + "labeled/vystupLabel.png").
        right().grow(0, 20).exists(pngs +
            "textovaPole/vystupniTextovePole.png") !=
            null);
    assertTrue("Vstupni vyberovy seznam nema" +
        "defaultni hodnotu", s.find(pngs +
            "labeled/vstupLabel.png").right().grow(0, 20).
        exists(pngs + "vyberoveSeznamy/vstupni" +
            "VyberovySeznam.png") != null);
    assertTrue("Vystupni vyberovy seznam nema" +
        "defaultni hodnotu", s.find(pngs +
            "labeled/vystupLabel.png").right().grow(0,
            20).exists(pngs + "vyberoveSeznamy/" +
            "vystupniVyberovySeznam.png") != null);
    assertTrue("Nalezeno upozorneni o chybe", s.
        exists(pngs + "chyby/chybaNeplatneCislo") ==
            null);
} catch (FindFailed | AssertionError e) {
    s.capture().save("errors", screenshotName());
    logger.error(e.getMessage());
    fail(e.getMessage());
}
} else {
    logger.error("Setup failed");
    fail("Setup failed");
}
}
}

```

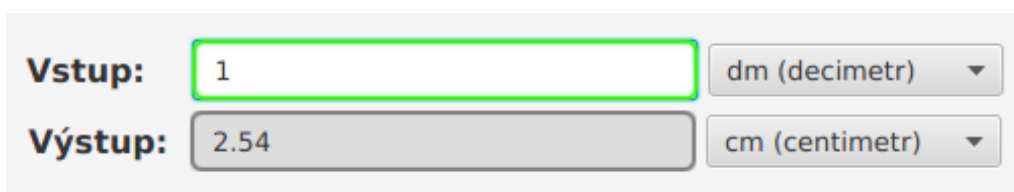
7.5 Zjištěné chyby

Během testování aplikace bylo zjištěno několik chyb. Jak již bylo řečeno dříve, tyto chyby jsou v aplikaci zaneseny záměrně.

7.5.1 Chybné převody z (na) decimetry

Pokud provádíme převod z (případně na) decimetry, dostaneme nesprávný výsledek, viz 7.3. Chování odpovídá převodu z (na) palce.

Obrázek 7.3: Chybný převod z decimetru na centimetr

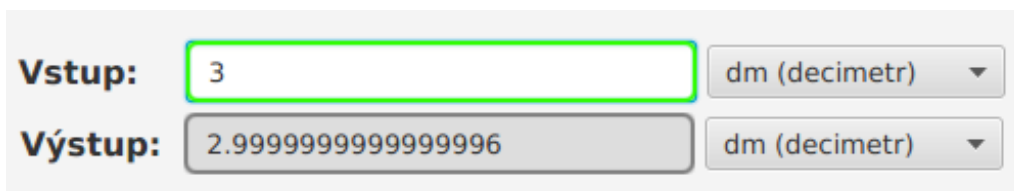


The screenshot shows a web interface for unit conversion. It has two rows: 'Vstup:' (Input) and 'Výstup:' (Output). The 'Vstup:' row has a text input field containing the number '1' and a dropdown menu set to 'dm (decimetr)'. The 'Výstup:' row has a text output field containing '2.54' and a dropdown menu set to 'cm (centimetr)'. The input field for '1' is highlighted with a green border.

7.5.2 Chybné zaokrouhlení

Dále u jednotek decimetry i palce v situaci, kdy jsou použity jak na vstupu, tak na výstupu, je hodnota 3 převedena přibližně na 2.9999996, viz 7.4.

Obrázek 7.4: Chybné zaokrouhlení při převodu mezi decimetry



The screenshot shows the same unit conversion interface. The 'Vstup:' row has an input field with '3' and a dropdown set to 'dm (decimetr)'. The 'Výstup:' row has an output field showing '2.9999999999999996' and a dropdown set to 'dm (decimetr)'. The input field for '3' is highlighted with a green border.

7.5.3 Převod záporné hodnoty

Při zadání záporné hodnoty pro převod se zobrazí chybová hláška o záporném čísle, avšak převod se i tak provede, viz 7.5.

Obrázek 7.5: Převod záporné hodnoty

The screenshot shows a web interface for a unit converter. At the top, a red-bordered box contains the text "Nelze převést" (Cannot convert) in bold red font, followed by a red square bullet point and the text "Zadané číslo je záporné" (The entered number is negative). Below this, there are two rows of input fields. The first row is labeled "Vstup:" (Input) and contains a text box with the value "-1" and a dropdown menu set to "cm (centimetr)". The second row is labeled "Výstup:" (Output) and contains a text box with the value "-1.0" and a dropdown menu set to "cm (centimetr)".

7.5.4 Tlačítko Vymaž

Tlačítko *Vymaž* nenastaví všem komponentám výchozí hodnoty. Pouze vymaže obsah vstupního pole. Výstupní pole a výběrové seznamy nadále obsahují hodnoty z posledního převodu, viz 7.6.

Obrázek 7.6: Použití tlačítka Vymaž

The screenshot shows the same unit converter interface after the "Vymaž" button has been clicked. The "Vstup:" (Input) field is now empty and highlighted with a green border. The "Výstup:" (Output) field still contains the value "-1.0". The dropdown menu for the input unit is now set to "dm (decimetr)" and the dropdown menu for the output unit is set to "in (palec)". Below the input fields, there are two green buttons: "Převést" (Convert) and "Vymaž" (Clear).

8 Problémy práce se SikuliX

Během tvorby testů je možné narazit na různé problémy. Ty, které byly zjištěny během vytváření této práce, jsou zde uvedeny, popsány a je k nim nastíněno možné řešení.

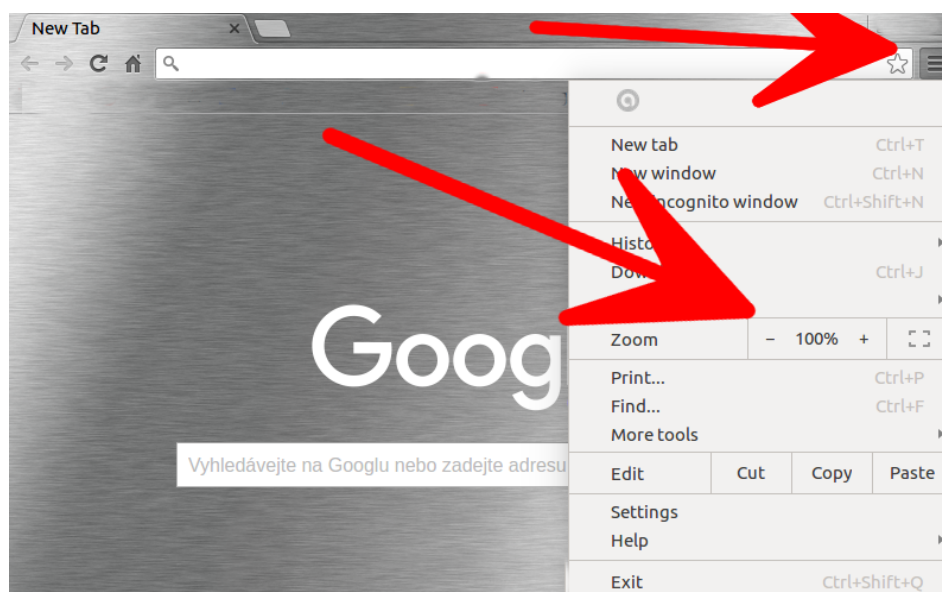
8.1 Rozlišení obrazovky

Jelikož se SikuliX v aplikaci orientuje podle screenshotů, je v danou chvíli závislé na rozlišení, ve kterém byl snímek pořízen. To je z důvodu, že ještě není implementována funkce, která by měla tento problém odstranit. Pokud se změní rozlišení, nebude daný prvek nalezen, ačkoli bude možné pouhým okem zjistit, že ve skutečnosti přítomen je. Stejný problém nastává i pokud se změní např. písmo nebo velikost webové stránky s aplikací a podobnými změnami vzhledu.

Jedním z možných řešení je, že si uděláte screenshots pro různá rozlišení, písma nebo velikosti stránek, případně budete testovat pouze s jedním daným rozlišením, písmem nebo velikostí stránky.

Nejvhodnější je nastavit v prohlížeči 100% velikost stránky. Toho se v Google Chrome nechá docílit např. tak, že klikneme na ikonku *Menu* a položku *Lupa* (příp. *Zoom*) nastavíme na 100%, jak je ukázáno na obrázku 8.1.

Obrázek 8.1: Nastavení 100% velikosti stránky



8.2 Rozměry screenshotů

Pokud se v testu hledá prvek v závislosti na pozici jiného, je možné, že nebude nalezen. Důvodem mohou být rozdílné rozměry screenshotů v kombinaci s použitými metodami hledání – první prvek nalezneme, ale screenshot druhého je větší, než prohledávaná oblast vymezená rozměry prvního, tudíž nemůže být nalezen. Obrázek 8.2 demonstruje problém. Pokud budeme hledat červeně vyznačený obrázek, který je vyšší než snímek, ve kterém hledáme, nebude nalezen.

Na snímku je situace, kdy se někde v záložce *Reklama* nalézá ikona *KIV*. První snímek obrazovky ukazuje výřez záložky *Reklama*. Tento snímek je vysoký x, y pixelů. Druhý, červeně orámovaný snímek již ukazuje hledané logo *KIV* (ve stejném měřítku), ale výška tohoto snímku je (chybně / zbytečně) o 10 pixelů na každou stranu větší, než první snímek.

Obrázek 8.2: Nesprávné rozměry screenshotů



8.3 Ukazatel myši

Při pořizování screenshotů je vhodné vyvarovat se umístění ukazatele myši ve snímané oblasti. Ten se totiž v průběhu testu v této oblasti vyskytovat nemusí a hledaný prvek by tak nemusel být rozpoznán.

Stejně tak je důležité uvědomit si, že na některých platformách se simulováním pohybu myši a klikáním mění pozice ukazatele. To může vyústit v problém, pokud je ukazatel umístěn přes hledaný prvek, který tím pádem pravděpodobně nebude rozpoznán. Oba problémy shrnuje obrázek 8.3.

Obrázek 8.3: Screenshot s ukazatelem myši



8.4 Nespolehlivé OCR

SikuliX poskytuje možnost rozpoznání textu v obrázcích. Tato funkcionality je však v experimentální fázi a na jejím vývoji se stále pracuje. Je tedy nespolehlivá a pro naše účely nevhodná. Text v obrázku buď vůbec nebyl nalezen (pokud se jednalo např. o jedinou číslici), nebo byl špatně rozpoznán (záměna O a 0, získána pouze část textu, apod.).

Možným řešením je tedy nasimulovat korektní výstup, provést screenshot a ten použít pro obrazové rozpoznání správného výsledku.

8.5 Nefunkčnost některých metod, tříd

Java API SikuliX poskytuje třídy a metody pro práci s aplikacemi a jejich okny. Tyto metody a třídy mají však občas jiné než očekávané chování. Vzhledem k téměř nulové dokumentaci je toto celkem velký problém.

8.5.1 Pozice a velikost okna

Konkrétním příkladem je např. to, pokud bychom chtěli testování omezit pouze na okno aplikace. SikuliX je schopno okno s aplikací najít podle (části) jejího titulku a přenést jej do popředí. Už ale není schopno získat rozměry a pozici tohoto okna, ačkoli metody pro tyto funkce existují.

Postup, kterým se dá tato funkcionality nahradit, je následující. Aplikaci najdeme podle jejího titulku a necháme ji přenést do popředí. Poté je SikuliX schopné získat pozici a rozměry okna, které je v popředí (má tzv. focus), jak je ukázáno v následujícím kódu.

```
Region window = App.focusedWindow();  
Location minCoord = window.getTopLeft();  
Location maxCoord = window.getBottomRight();
```

8.5.2 Stav aplikace

Dále nedokázalo indikovat, že aplikace skončila svůj běh. Pokud jsme tedy použili cyklus „testuj, dokud aplikace běží“, testování pokračovalo i v případě, že byla aplikace již ukončena.

Náhradním řešením tedy bylo vytvořit screenshot některé části aplikace, která se nemění a je vždy v aplikaci přítomna. Cyklus poté vypadá takto: „testuj, dokud najdeš tuto část aplikace“. To však není řešení absolutní, protože nebude funkční v případech, kdy se objeví např. dialogové okno, které tuto část zakryje, nebo pokud taková část vůbec neexistuje.

8.5.3 Spuštění aplikace

Metody pro spuštění aplikace `App.open("aplikace")`, `new App("aplikace").open()` a `App.run("příkaz")`, které SikuliX poskytuje, fungovaly bez problémů v OS Linux, avšak v OS Windows nastal problém. SikuliX nebylo schopné otevřít aplikaci pomocí ani jednoho z příkazů

```
App.open("java -jar cesta/k/aplikaci.jar");  
new App("java -jar cesta/k/aplikaci.jar").open();  
App.run("java -jar cesta/k/aplikaci.jar");
```

Týká se to pouze spouštění aplikace pomocí daného příkazu. Pokud by byla zadána cesta ke spustitelnému souboru (např.: `cesta/ke/spustitelnemu/souboru.exe`), vše proběhlo bez problémů i za použití těchto metod.

Problém jsem nebyl schopen za pomoci SikuliX vyřešit. Použil jsem proto metodu poskytovanou programovacím jazykem Java, konkrétně

```
Runtime.getRuntime().exec("java -jar cesta/k/aplikaci.jar");  
App application = new App("Titulek aplikace");  
application.focus();
```

9 Monkey testy

Princip monkey testování byl již zmíněn v kapitole 2.8.1. Nyní o něm zjistíme něco více a ukážeme si jeden konkrétní monkey test vytvořený pomocí SikuliX.

Dle [Patton, 2002] existují tři druhy monkey testů („cvičených opic“) lišících se svou inteligencí – co umí, jaké mají povědomí o aplikaci apod. Jejich vlastnosti a rozdíly jsou popsány v následujících částech.

9.1 Hloupá opice

Nejjednodušším případem cvičené opice je hloupá opice. Ta neví vůbec nic o testované aplikaci, jen náhodně kliká a píše na klávesnici. Software, který na počítači běží, není schopný rozlišit cvičnou opici od skutečného člověka, snad jen že opice by měla akce vykonávat rychleji.

Na první pohled se zdá, že takovýto přístup nemůže najít žádné chyby. Opak je ale pravdou, neboť jak se ukazuje, pokud máme dostatek času a pokusů, opice až překvapivě často vytvoří jakousi posloupnost akcí, která povede k havárii aplikace. Na tuto posloupnost nejspíše programátoři ani testéři vůbec nepomysleli, proto nebyla tato chyba nalezena dříve.

Další možnou chybou, kterou může i hloupá opice zjistit, jsou úniky paměti. Pokud totiž necháme opici pracovat přes noc, poběží software několik hodin (případně můžeme nechat běžet i několik dní) a případné problémy se tak mohou projevit.

9.2 Zpola inteligentní opice

Pokud hloupou opici doplníme o několik funkcí navíc, zvýšíme její inteligenci kvocient na zpola inteligentní opici. Jednou z těchto funkcí je zaznamenávání prováděných činností do souboru – logu. Díky tomu jsme poté schopni přesněji identifikovat, co se dělo těsně před selháním aplikace.

Další takovou funkcí je, aby opice pracovala pouze s testovanou aplikací. Pokud kliká a píše kamkoli na obrazovce, jednou by zvolila i možnost vypnutí počítače, čímž bychom ztratili drahý čas na testování. V jiném případě by zvolila možnost ukončit aplikaci, ale jelikož by si toho nebyla vědoma, pokračovala by v klikání a psaní na vše možné, co se vyskytuje na obrazovce. Zjištění, zda je ještě aplikace spuštěná, se řadí k těmto funkcím také.

Vhodná je i ta možnost, která dovolí opici aplikaci po selhání znovu spustit a pokračovat tak v testování.

9.3 Inteligentní opice

Dalším evolučním krok je inteligentní opice. Ta do klávesnice „nebuší“ zcela náhodně, ale uvědomuje si následující věci:

- kde se nachází,
- co na tomto místě může dělat,
- kam může přejít,
- kde již byla,
- jestli je to, co vidí, opravdu správné.

Opice si je tedy vědoma všech stavů aplikace a přechodů mezi nimi. Díky tomu dokáže s programem pracovat přibližně stejně, jako případný uživatel, jen o něco rychleji.

Inteligentní opice se však nemusí omezovat jen na hledání chyb, ale mohou také kontrolovat výsledky činností aplikace, správnost výstupů apod. Pokud je navíc schopna vykonávat konkrétní testovací případy, může nalézat poměrně velké množství chyb.

9.4 Monkey test pomocí SikuliX

Pro účely této práce byl vytvořen jeden monkey test prováděný nad aplikací *Kalkulačka*. Cvičená opice, která jej provádí, je zpola inteligentní – omezuje se na práci s oknem aplikace, zapisuje akce do logu a je si vědoma toho, zda je ještě aplikace spuštěná.

Z důvodů uvedených v kapitole 8.5, hlavně zhoršenou schopností SikuliX zjišťovat pozici a velikost okna aplikace a její stav, se nástroj jeví jako nevhodný pro tvorbu tohoto typu testů.

Konkrétní příklad v kódu 9.1 běží tak dlouho, dokud existuje statická část aplikace. Ve smyčce poté získá pozici a velikost jejího okna. Vygeneruje náhodná čísla pro souřadnice vykonávané akce a další číslo pro určení konkrétní akce (kliknutí, dvojkliknutí, kliknutí pravým tlačítkem, vložení textu případně stisknutí kláves).

Kód 9.1: Monkey test vytvořený pomocí SikuliX

```
@Test
public void MonkeyTest01() throws FindFailed {
    Random gen = new Random();
    while (s.exists(title) != null) {
        window = App.focusedWindow();
        Location minCoord = window.getTopLeft();
        Location maxCoord = window.getBottomRight();
        int random = gen.nextInt(5);
        int x = gen.nextInt(maxCoord.getX()), y = gen.
            nextInt(maxCoord.getY());
        while (x < minCoord.getX()) x = gen.nextInt(
            maxCoord.getX());
        while (y < minCoord.getY()) y = gen.nextInt(
            maxCoord.getY());
        switch (random) {
            case 0: // Click
                window.click(new Location(x, y));
                break;
            case 1: // Double-click
                window.doubleClick(new Location(x, y));
                break;
            case 2: // Right click
                window.rightClick(new Location(x, y));
                break;
            case 3: // Paste
                if (gen.nextInt(2) == 1) window.click(new
                    Location(x, y));
                String s = randomString(gen);
                window.paste(new Location(x, y), s);
                logger.info("PASTE \"" + s + "\" on [" + x +
                    ", " + y + "]");
                break;
            case 4: // Type
                if (gen.nextInt(2) == 1) window.click(new
                    Location(x, y));
                window.type(new Location(x, y),
                    randomString(gen));
                break;
        }
    }
}
```

10 Automatizace nástrojem

Podle [Hocke, 2015] je SikuliX primárně nástroj pro automatizaci činností. Je vhodný pro kohokoli, kdo provádí opakovaně některé monotónní činnosti jako denní práce s aplikacemi nebo webovými stránkami, hraní her, administrace IT systémů a sítí apod.

Zde ukážeme možnost automatizace takového procesu na konkrétním příkladu, viz kód 10.1. Jedná se o zapnutí a případné přihlášení (pokud jsme se během sezení již nepřihlašovali) do aplikace *TestLink* instalované podle postupu v podkladech k přednáškám KIV/OKS, viz [Herout, 2016].

Postup je ve své podstatě identický jako při vytváření testovacích případů. Jediným rozdílem je, že nebudeme používat knihovnu *JUnit*, ale skript vytvoříme jako samostatný program v Javě. Každý si ve skriptu musí upravit cestu k *Bitnami TestLink Stack Manager Tool*, login a heslo.

Kód 10.1: Automatizace spuštění a přihlášení do aplikace *TestLink*

```
public static void main(String[] args) {
    Logger logger = LogManager.getLogger();

    Settings.MoveMouseDelay = 0;
    Debug.setLogger(logger);
    Debug.setLoggerAll("info");

    Screen s = new Screen();
    try {
        Runtime.getRuntime().exec("cesta/k/TestLink/manager/
            tool");
        App application = new App("TestLink");
        application.focus();
        s.wait("png/goToApp.png", 5);
        s.click("png/goToApp.png");
        if (s.exists("png/startServers.png") != null) s.
            click("png/startServers.png");
        application = new App("Chrome");
        application.focus();
        s.wait("png/access.png", 50);
        s.click("png/access.png");
        if (s.exists("png/login.png") != null) {
            s.find("png/login.png").below(20).click();
            s.paste("login");
        }
    }
}
```

```
s.click("png/login.png");
s.find("png/pass.png").below(20).click();
s.paste("password");
s.type(Key.ENTER);
}
} catch (IOException | FindFailed e) {
s.capture().save("errors", screenshotName());
logger.error(e.getMessage());
}
}
```


11 Závěr

Seznámil jsem se s některými metodami testování grafického uživatelského rozhraní a zjistil jsem některé důvody používání těchto metod. Dále jsem prozkoumal, které nástroje je k testování možné využít.

Vybral jsem tři programy, které jsem stručně popsal. Navrhl jsem multikriteriální hodnocení a provedl jejich podrobné porovnání. Výsledkem byl výběr jednoho programu, který použiji jako hlavní nástroj v této práci.

S aplikací SikuliX, kterou jsem zvolil předchozí činností, jsem se zběžně seznámil a vytvořil jeden test v prostředí SikuliX-IDE za použití vlastního jazyka SikuliX. Další čtyři testy jsem zhotovil pomocí Java API, které SikuliX nabízí a které budu využívat z důvodu vazby na předmět KIV/OKS.

Dále se hodlám zaměřit na podrobnější zkoumání používání nástroje. Také připravím sadu ukázkových testů a funkční scénáře.

11.1 Statistika

V této práci bylo vytvořeno 103 testů (každý pro webovou a JavaFX aplikaci, dohromady tedy 206 testů). Počet snímků potřebných pro vytvoření testů byl 74 (každý pro webovou a JavaFX aplikaci jak na Linux, tak na Windows, celkem tedy 296 snímků). Dále byl vytvořen jeden monkey test a jeden skript pro ukázkou automatizace činnosti. Dohromady bylo vytvořeno více než 330 souborů a napsáno více než 7350 řádek zdrojového kódu.

Literatura

- AUTOIT. *AutoIt* [online]. AutoIt Consulting Ltd, 2015. [cit. 4.10.2015].
Dostupné z: <https://www.autoitscript.com/site/autoit/>.
- DANIELS, K. W. *AutoHotKey* [online]. 2011. [cit. 4.10.2015]. Dostupné z:
<https://code.google.com/p/autokey/>.
- EGGPLANT. *eggPlant Functional* [online]. TestPlant, 2015. [cit. 4.10.2015].
Dostupné z: <http://www.testplant.com/eggplant/testing-tools/eggplant-developer/>.
- FRAMEWORK, R. *Robot Framework, Generic test automation framework for acceptance testing and ATDD* [online]. Robot Framework and Nokia Networks, 2015. [cit. 4.10.2015]. Dostupné z: <http://robotframework.org/>.
- HEROUT, P. *Přednášky z OKS* [online]. 2016. [cit. 13.4.2016]. Dostupné z:
<http://www.kiv.zcu.cz/~herout/vyuka/oks/prednasky/oks-1a4.pdf>.
- HOCKE, R. *SikuliX* [online]. 2015. [cit. 4.10.2015]. Dostupné z:
<http://www.sikulix.com/>.
- HOCKE, R. *Does sikuli work with most of the popular picture formats?* [online]. Parn Yin, 2013. [cit. 11.4.2016]. Dostupné z:
<https://answers.launchpad.net/sikuli/+question/241219>.
- JUBULA. *Jubula, Automated Functional Testing* [online]. Eclipse Foundation and BreDEX GmbH, 2015. [cit. 4.10.2015]. Dostupné z:
<http://www.eclipse.org/jubula/>.
- MALLET, C. *AutoHotKey* [online]. 2015. [cit. 4.10.2015]. Dostupné z:
<https://www.autohotkey.com/>.
- PATTON, R. *Testování softwaru*. Computer Press, 2002. ISBN 80-7226-636-5.
- RANOREX. *Ranorex* [online]. Ranorex GmbH, 2015. [cit. 4.10.2015]. Dostupné z:
<http://www.ranorex.com/>.
- RFT. *Rational Functional Tester* [online]. IBM, 2015. [cit. 4.10.2015].
Dostupné z: <http://www-03.ibm.com/software/products/cs/functional>.
- ROUDENSKÝ, P. – HAVLÍČKOVÁ, A. *Řízení kvality softwaru*. Průvodce testováním. Computer Press, 2013. ISBN 978-80-251-3816-8.

- SIKULI. *Sikuli* [online]. User Interface Design Group at MIT, 2015. [cit. 4.10.2015]. Dostupné z: <http://www.sikuli.org/>.
- SILKTEST. *SilkTest* [online]. Micro Focus, 2015. [cit. 4.10.2015]. Dostupné z: <http://www.borland.com/en-GB/Products/Software-Testing/Automated-Testing/Silk-Test>.
- SQUISH. *Squish, GUI Tester* [online]. froglogic GmbH, 2015. [cit. 4.10.2015]. Dostupné z: <http://robotframework.org/>.
- TESTCOMPLETE. *TestComplete* [online]. SmartBear Software, 2015. [cit. 4.10.2015]. Dostupné z: <http://smartbear.com/product/testcomplete/overview/>.
- UFT. *Unified Functional Testing* [online]. Hewlett Packard Enterprise Development LP, 2015. [cit. 4.10.2015]. Dostupné z: <http://www8.hp.com/cz/cs/software-solutions/unified-functional-automated-testing/>.
- WIKIPEDIA. *Infinite Monkey Theorem* [online]. Wikipedia, 2016. [cit. 18.4.2016]. Dostupné z: https://en.wikipedia.org/wiki/Infinite_monkey_theorem.

Příloha A

Kód 11.1: Další testy Java API

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.rules.ErrorCollector;
import org.sikuli.basics.Debug;
import org.sikuli.basics.Settings;
import org.sikuli.script.*;
import javax.swing.*;
import java.time.LocalDateTime;
import static org.junit.Assert.*;

public class Test01 {

    static Logger logger;
    static ErrorCollector collector;
    static Screen s;
    static App prohlizec;
    static boolean run;

    static {
        System.setProperty("log4j.configurationFile",
            "log-konfigurace.xml");
    }

    private String nazevScreenshotu() {
        LocalDateTime l = LocalDateTime.now();
        return l.getYear() + "-" + l.getMonthValue() +
            "-" + l.getDayOfMonth() + "-" + l.getHour() +
            "-" + (l.getMinute() < 10 ? "0" + l.
                getMinute() : l.getMinute()) + "-" + l.
                getSecond() + "-";
    }

    @BeforeClass
```

```

public static void setUpBeforeClass() {
    logger = LogManager.getLogger();

    Settings.OcrTextSearch = true;
    Settings.OcrTextRead = true;
    Debug.setLogger(logger);
    Debug.setLoggerAll("info");

    collector = new ErrorCollector();
    s = new Screen();
    prohlizec = new App("google-chrome");
    prohlizec.open();
    prohlizec.focus();
    run = true;
}

@AfterClass
public static void tearDownAfterClass() {
    JOptionPane.showMessageDialog(null, "Script" +
        " dokoncen");
    prohlizec.close();
}

@Before
public void setUp() {
    try {
        s.wait("png/addressBar.png", 10);
        s.click(new Pattern("png/addressBar.png").
            targetOffset(100, 0));
        s.paste("http://oks.kiv.zcu.cz/Prevodnik");
        s.type(Key.ENTER);
        s.wait("png/zalozkaPrevodnik.png", 5);
    } catch (Exception e) {
        run = false;
        s.capture().save("errors", nazevScreenshotu());
        logger.error(e.getMessage());
    }
}

@Test
public void testPorovnejText() {
    if (run) {

```

```

try {
    s.click("png/zalozkaPrevodnik.png");
    s.wait("png/tlacitkoPreved.png", 5);
    s.paste("png/vstup.png", "1");
    Match m = s.find("png/jednotky.png");
    m.setTargetOffset(-27, -18);
    m.click();
    s.findText("(metr)").click();
    s.click(new Pattern("png/jednotky.png").
        targetOffset(-27, 18));
    s.find("png/dm.png").click();
    s.click("png/tlacitkoPreved.png");
    String t = s.find("png/vystup.png").right(
        100).text();
    assertEquals(10, Double.parseDouble(t),
        0.01);
} catch (FindFailed | AssertionError e) {
    s.capture().save("errors",
        nazevScreenshotu());
    logger.error(e.getMessage());
    fail(e.getMessage());
}
} else {
    run = true;
    logger.error("setUp neuspesny");
    fail("setUp neuspesny");
}
}

@Test
public void testPorovnejObraz() {
    if (run) {
        try {
            s.click("png/zalozkaPrevodnik.png");
            s.wait("png/tlacitkoPreved.png", 5);
            s.paste("png/vstup.png", "1");
            s.click(new Pattern("png/jednotky.png").
                targetOffset(-27, -18));
            s.click("png/inch.png");
            s.click("png/tlacitkoPreved.png");
            assertTrue(s.exists("png/vysledek.png") !=
                null);
        }
    }
}

```

```

    } catch (FindFailed | AssertionError e) {
        s.capture().save("errors",
            nazevScreenshotu());
        logger.error(e.getMessage());
        fail(e.getMessage());
    }
} else {
    run = true;
    logger.error("setUp neuspesny");
    fail("setUp neuspesny");
}
}

@Test
public void testZkontrolujOdkazObrazekKiv() {
    if (run) {
        try {
            s.click("png/logoKiv.png");
            assertTrue(s.exists("png/zahlaviKiv.png") !=
                null);
        } catch (FindFailed | AssertionError e) {
            s.capture().save("errors",
                nazevScreenshotu());
            logger.error(e.getMessage());
            fail(e.getMessage());
        }
    } else {
        run = true;
        logger.error("setUp neuspesny");
        fail("setUp neuspesny");
    }
}

@Test
public void testChyba() {
    if (run) {
        try {
            s.wait("png/tlacitkoPreved.png", 5);
            s.paste("png/vstup.png", "1");
            s.click(new Pattern("png/jednotky.png").
                targetOffset(-27, -18));
            s.findText("(metr)").click();

```

```

        s.click("png/tlacitkoPreved.png");
        String t = s.find("png/vystup.png").right(
            100).text();
        assertEquals(100, Double.parseDouble(t),
            0.01);
    } catch (FindFailed | AssertionError e) {
        s.capture().save("errors",
            nazevScreenshotu());
        logger.error(e.getMessage());
        fail(e.getMessage());
    }
} else {
    run = true;
    logger.error("setUp neuspesny");
    fail("setUp neuspesny");
}
}
}
}

```

Kód 11.2: Konfigurace loggeru

```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Properties>
        <Property name="zprava">%d{dd.MM.YYYY HH:mm:ss}
            [%level - %logger{1}].%method() - %msg%n
        </Property>
        <Property name="souborLog">errors/log.txt
        </Property>
    </Properties>
    <Appenders>
        <Console name="obrazovka" target="SYSTEM_OUT">
            <PatternLayout pattern="${zprava}"/>
        </Console>
        <File name="toSouborLog" fileName=
            "${souborLog}" append="true" immediateFlush=
            "true">
            <PatternLayout pattern="${zprava}"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="INFO">

```



```
        <AppenderRef ref="toSouborLog"/>
    </Root>
</Loggers>
</Configuration>
```