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

Bakalářská práce

Využití nástrojů pro testování grafického uživatelského rozhraní

Plzeň 2016 Jaroslav Klaus

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 8. 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

2 Testování softwaru

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

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 nahrazovány 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

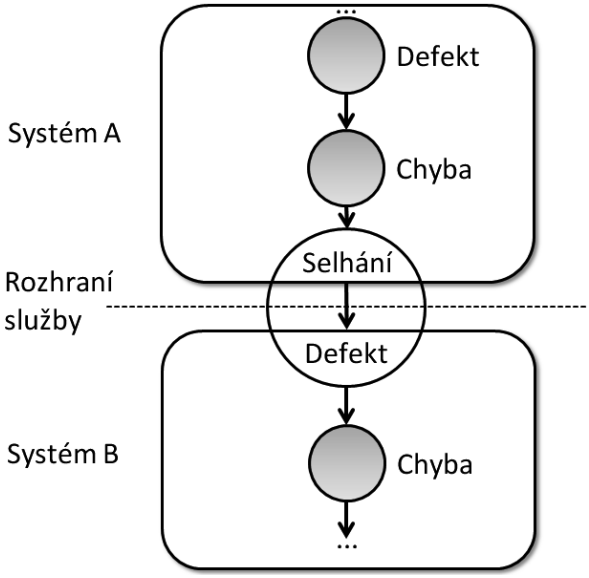
Během vývoje softwaru se do dokumentů či zdrojových kódů dostávají defekty, způsobující chyby a selhání. Tyto pojmy je velmi důležité rozlišovat. V následujících odstavcích bude jejich význam vysvětlen.

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 testovaní 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 kritérii (dokumentace, manuály apod.).

2.7 Testování grafického uživatelského rozhraní

3 Přehled nástrojů

V této kapitole následuje přehled nástrojů a 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 jsem si vybral tři a ty podrobněji prozkoumal a porovnal, 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 nespecifikované 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 vzoru[1], dokáže simulovat ovládání myši a klávesnice nebo rozpoznávat text v obrázcích[2]. Výhodou této aplikace je proto její nezávislost

[1]Pomocí OpenCV, `http://opencv.org/`

[2]Pomocí 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)
Složitost 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

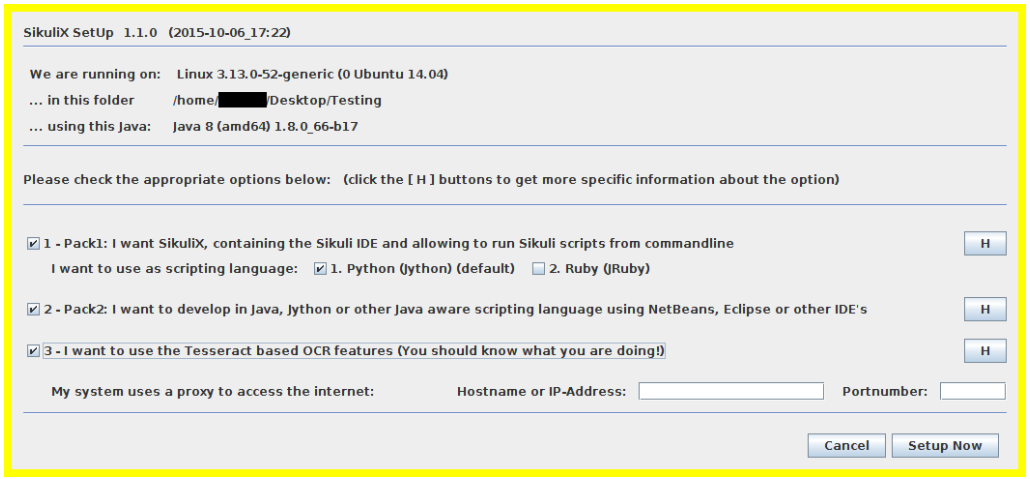
Váhy	**Vlastní váha**	**Celková váha**	**Váha v %**
Možnosti vytváření skriptů	4	0,10	10,3
Podpora testovaných rozhraní	8	0,21	20,5
Podporované operační systémy	3	0,08	7,69
Reportování výsledků testů	5	0,13	12,8
Screenshot při chybě	4	0,10	10,3
Univerzálnost nástroje	7	0,18	17,9
Vhodnost nástroje pro 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 spuštěním[1]. 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 obrázu 6.2, obě potvrdíme tlačítkem *OK*. Pokud vše proběhne v pořádku, vzniknou v adresáři soubory podobné těmto[2] *runsikulix, SetupStuff, SikuliX-1.1.0-SetupLog.txt, sikulixapi.jar, sikulix.jar*.

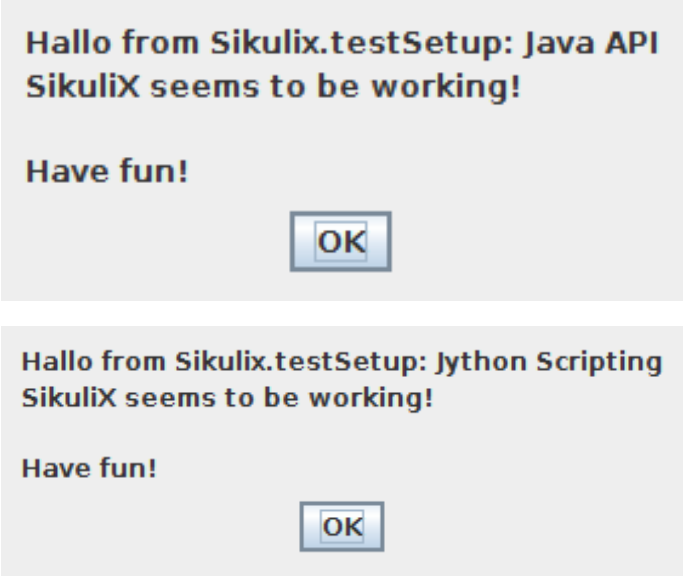
Obrázek 6.1: Instalace SikuliX



[1] Je 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]

[2] Může se lišit na různých OS

Obrázek 6.2: Test instalace



6.2 Tvorba testů

Pro tvorbu testů pomocí SikuliX jsou nejdůležitější snímky (screenshoty) řídících 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 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.

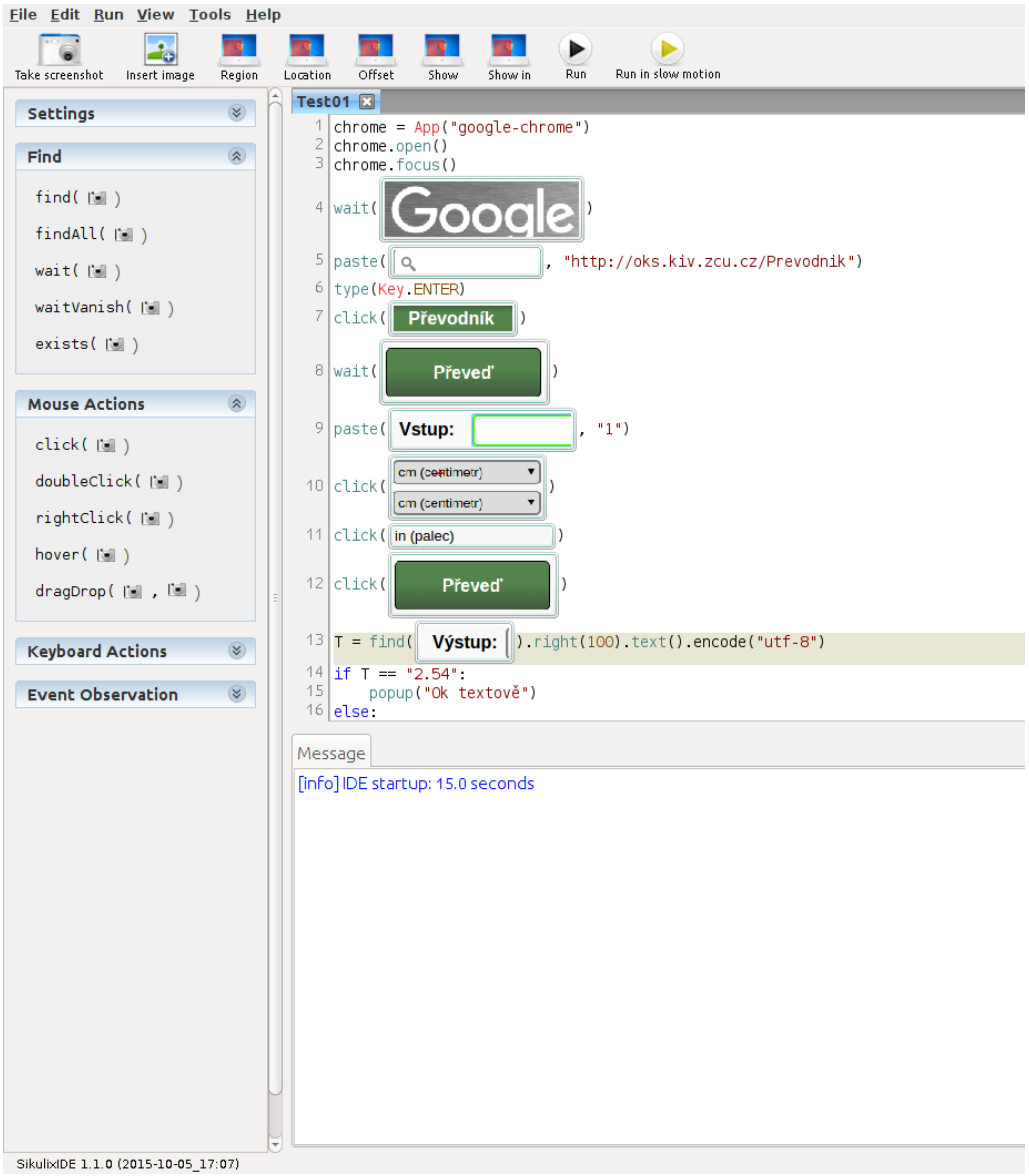
6.3 SikuliX-IDE

Spustit SikuliX-IDE je možné různými způsoby [Hocke, 2015].

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 [volitelne 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.

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, není v IDE identický, ale cesta k obrázku je vždy nahrazena jeho náhledem. K tvorbě jsou v IDE užitečné pomůcky, které se nacházejí v levém a v horním panelu.

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 *Převeď*. 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.

Kód 6.1: První skript

```
prohlizec = App("google-chrome")
prohlizec.open()        #otevre aplikaci definovanou
                        #vyse
prohlizec.focus()       #vybere do popredi jeji 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 nalezeneho
#obrazku
click(Pattern("obr6.png").targetOffset(-27,-18))
click("obr7.png")
click("obr4.png")
#precte text z casti, ktera je 100px vpravo od
#nalezeneho obrazku
T = find("obr8.png").right(100).text()
if T == "2.54":
        #pokud rozpoznany text souhlasi se zadanym,
        #otevre se vyskakovaci okno
    popup("Ok textove")
else:
    popError("Chyba")    #jinak se zobrazi chybove
                         #okno

if exists("obr9.png"):
        #pokud na obrazovce existuje obrazek, otevre
        #se vyskakovaci 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 9.1. První test skončí negativně, druhý pozitivně, třetí pozitivně a čtvrtý negativně.

7 Sada ukázkových testů a jejich scénářů

Pro potřeby práce a pro názornou ukázku odlišnosti přístupu k testování webové a desktopové aplikace byla vytvořena téměř identická aplikace Převodník pomocí technologie JavaFX. Následující sada testů a jejich scénářů se vztahuje k této aplikaci a k té dostupné z url `http://oks.kiv.zcu.cz/Prevodnik/Prevodnik`.

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 řídícím 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.

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 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. Stejné 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

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.

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.

7.4 Vymazání

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

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. Chování odpovídá převodu z (na) palce.

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.

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.

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.

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 screenshoty pro různá rozlišení, písma nebo velikosti stránek, případně budete testovat pouze s jedním daným rozlišením, písmem nebo velikostí 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.

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.

8.4 Nespolehlivé OCR

SikuliX poskytuje možnost rozpoznání textu v obrázcích. Tato funkcionalita 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 funkcionalita 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).

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 části zakryje, nebo pokud taková část vůbec neexistuje.

8.5.3 Spuštění aplikace

Metody pro spuštění aplikace, které SikuliX poskytuje, fungovaly bez problémů v OS Linux, avšak v OS Windows nastal problém. SikuliX nebylo schopné otevřít aplikaci pomocí příkazu

```
java -jar cesta/k/aplikaci.jar
```

Tento 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");
```

9 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.

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/`.

Hocke, R. *SikuliX* [online]. 2015. [cit. 4.10.2015]. Dostupné z: `http://www.sikulix.com/`.

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/`.

Příloha A

Kód 9.1: 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");
    }
  }
}
```