Software Testing MTAT.03.159
Lab 6: Automated GUI testing

Inst. of Comp. Science, University of Tartu

February 2015

- Submission deadline: Lab reports must be submitted within seven days. For example, if your lab takes place on Tuesday, then you have to submit your report no later than following Monday, 23:59 hours.
- Late Policy: 50% deducted for every 24 late hours.
- Group: There should be maximum **two** members in a group. Answers should be your own group work, explained in your own words.
- Maximum amount of points is ten (10).

1 Introduction

In software engineering, Graphical user interface (GUI) testing is the process of ensuring proper functionality of the GUI for a given application and making sure it conforms to its written specifications [1]. The testing is usually performed with the help of a variety of test cases. However, it is possible to automate this activity with the help of different tools. This lab will specifically focus on Sikuli.
Sikuli is a freeware tool that uses screenshots to automate actions/steps. Sikuli Script is a Jython and Java library that automates GUI interaction using image patterns to direct keyboard/mouse events. The core of Sikuli Script is a Java library that consists of two parts: java.awt.Robot, which delivers keyboard and mouse events to appropriate locations, and a C++ engine based on OpenCV, **which searches given image patterns on the screen (**Test creator will provide screenshot of the image being looked for. If Sikuli detects an image pattern, it will execute the action it is programmed to do). The C++ engine is connected to Java via JNI and needs to be compiled for each platform. On top of the Java library, a thin Jython layer is provided for end-users as a set of simple and clear commands [2]. SikuliX is java API based on Sikuli.

2 Learning objectives

The exercise aims at giving an understanding of automated GUI testing. The specific learning goals are to gain a basic insight of Sikuli and to learn why and how properly built automated tests (autotests) can help you during regression testing. This will be achieved by using a predefined project – so called skeleton code, where necessary methods for building the automated tests are provided.

3 Preparation (Individual Work)

The lab is divided into 2 parts: Set up your workspace and get familiar with Sikuli. In the second part, which will be graded, you will automate testing of a sample program written in Java.

Execute the following steps:

- Follow the document "**Installation**"
- Once you have set up the work environment, go through the examples below and try to understand them (same methods are also used in the skeleton code or Autotest for students.zip)

*****

NOW INSTALL THE SOFTWARE

WHEN DONE CONTINUE WITH THE INSTRUCTIONS

*****

4 Examples

Below are examples how you can use the methods from the skeleton code. You can always create more methods on your own but the reporting logic should stay the same.

Example 1. How to click on an image with custom similarity
Example 2. How to click on an image with default similarity
Example 3. How to copy text to clipboard and compare it
Example 4. How to write text to the given location
Example 5. How to test if image exists on the screen with retries
Example 6. How to click on a picture periodically
Example 7. How to call out right click on the screen

Following methods are copied from the skeleton project.

Similarity – Sikuli does pixel by pixel comparison, the similarity specifies how big the coverage between the found and the given image.

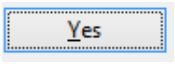
Example 1

```
/**
 * @param pictureUrl - location of the image - absolute or relative
 * @param similarity - from 0-1, sets the similarity of the image looked for
 * @returns
 */
click(String pictureUrl, Double similarity)
```

Method tries to click on the image with provided similarity, which can be from 0.0 to 1.0. When Sikuli fails to find the picture just lower the similarity. (Default similarity is 0.8)

Example 2

For example:

To click on 

(yes.jpg) image, method call-out should be:
click("yes.jpg");

```
/**
 * Calls out the click method with default similarity = 0.80
 * @param pictureUrl
 * @return
 */
click(String pictureUrl)
```

Calls out the click method with similarity 0.8

Example 3

```
/**
 * Tries to copy text to the clipboard and the compare it to predefined text
 * @param textToCompare
 */
private void compareTextToClipboards(String textToCompare)
```

Selects all the text and then compares it to the predefined text.
NB! In order for copying to work, Sikuli must click on the text area first.
Example 3 demonstrated in skeleton project as well

Example 4

```
/**
 * Clicks on the given pictures and tries to enter text
 * @param pictureUrl
 * @param text
 */
private void write(String pictureUrl, String text)
```

Clicks on the given picture and if possible, tries to write text into the given location
Example 4 demonstrated in skeleton project as well

Example 5

```
/**
 * Returns true if pictures was clicked, if not, then returns false
 * @param pictureUrl
 * @return
 * @throws FindFailed
 */
private boolean verifyIfExists(String pictureUrl)
```

Returns a Boolean depending if the image existed on the screen

NB! There also exists a method verifyIfExists(String pictureUrl, int retries), which checks the picture up to RETRIES times and returns true if found.

Example 6

```
/**
 * Tries to click on given image with given similarity for X retires
 * @param pictureUrl
 * @param similarity
 * @param retires
 */
public boolean click(String pictureUrl, Double similarity, int retires)
```

Tries to click on the given image. Returns true if found. Useful when picture is moving

Example 7

```
void rightClick()
```

Makes a right click on the screen. NB! Position must be previously set. To set position, click on the location before with click method.

Where to use Sikuli? [3]

- Sikuli provides extensive support to automate flash objects (for example)
- Sikuli can automate Web as well as Windows application
- Sikuli uses visual match to find elements on screen
- Sikuli has a simple API
- Sikuli does not use source code to navigate (meaning it can survive code refactoring)

5 Practical Part

Following figure describes the lab practical work flow that is followed by the explanation.

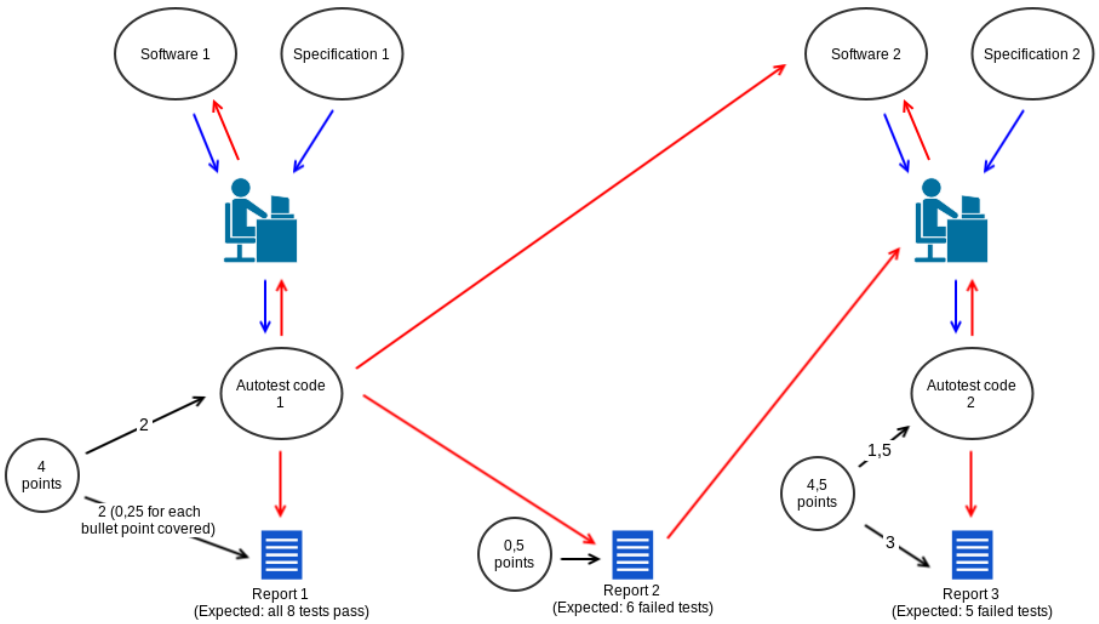


Figure 1 Lab work order

Autotest code 1 – Autotest code developed by you in the first iteration	Autotest code 2 – Autotest code developed by you in the second iteration

For the lab's practical assignment, you will have to build an autotest basing on the skeleton project, imported in "Installation" part. You will be given 2 software versions with specifications. The mentioned software is developed by the "TabMaker" company. Version 2 introduces new functionality, what changed core functionality of the program. Difference can be found when comparing specification 1 and specification 2. Your assignment is to automate this regression testing phase.

First iteration is to build a set of test cases to cover ALL the points in the specification 1. Run it against Software1.jar.

Second iteration is to run the same set of test cases against Software2.jar - this run should discover all failures introduced by the new functionality.

Note: There can be also failures due to the specification changes (although this means that the autotest has a defect, which must be fixed in order to get a proper autotest result).

If the previous part is done, you call out your code in iteration 3 **if**-block.

From Specification 2 you will find small changes/updates, to find out these changes you must compare Specification 2 to Specification 1.

To summarize the tasks:

1. *In first iteration, build a set of test cases to run against software1.jar*
    1. *Cover each specification point presented in Specification1*
    2. *1 specification point must produce 1 report row (**example below shows how to do add a result of the test to the report file**) – There must be 8 rows in report file corresponding to Specification1*
    3. *Save the report – you have to submit it later **NB! All tests must pass***
2. *Run the test cases against Software2.jar*
    1. *If done properly in the previous development cycle, autotest should find all mistakes (that were introduced manually for this lab) during the automated regression testing*
    2. *Report 2 must include 8 rows (6 tests must fail)*
    3. *If all the bugs were not found, go back to point 1*
3. *Fix iteration 2 faults against specification 2 (thus, it becomes iteration 3)*
    1. *Don't forget to save Report 1, Report 2 or test code*
    2. ***New or changed specification points must pass***
    3. *Report 3 must include 9 rows (marking the functionality passed or failed)*
    4. *Report 3 must detect 5 failed tests (You can't make them pass, because there is no access to the source code)*

Example of reporting:

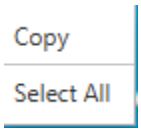


Figure 2 contextMenu.png

```
Sikuli.click("image.png");
sikuli.rightClick();
// If succeeded, then add PASSED line to report file
boolean succeeded = sikuli.verifyIfExists("contextMenu.png");
if (succeeded) {
      reportFile.add("PASSED", "Tab 3",
                  "Context menu is present");
} else {
      reportFile.add("FAILED", "Tab 3",
                  "Context menu is not present");
}
```

Example report:

Result	Tab	Comment
PASSED	Tab 1	HTML editor had all the elements
PASSED	Tab 1	Entered text format was the same
PASSED	Tab 1	Text format selection was the same
PASSED	Tab 2	Koala picture is the same
PASSED	Tab 3	Text length was the same
PASSED	Tab 3	Context menu is present
PASSED	Tab 4	Tab was collapsable
PASSED	Tab 4	Default color was white

6 Report

Make a ZIP/RAR file which contains the following:
Include 3 report files (named accordingly by the following naming **schema**), each generated by the autotest.

- Report1.pdf – generated when the autotest ran against version 1 (must contain 8 results)
- Report2.pdf – generated when the autotest ran against version 2 (must contain 8 results)
- Report3.pdf – generated when the autotest ran against version 2, where defects have been fixed in the autotest framework (must contain 9 results)
- Autotest.java - file with autotest code
- Resources folder with the pictures
- Info.pdf – Team member's names. (even if you are working alone)

NB! NO CODE OR REPORT = NO POINTS
NB! Do NOT submit the whole project

Autotest can be exported from **Export – General – Archive file** when clicking on a .java file.

7 Grading

- **No code or report = no points**
- Late Policy: 50% deducted for every 24 late hours
- The zip package that will be submitted must contain the following elements:
    - Working Autotest code + pictures in the resources folder

- Report 1,2,3 with results

You'll get following points for Reports:

- Report 1 gives 2 points maximum – 0,25 for each specification point covered correctly (Report must contain 8 rows in total)
- Report 2 gives 0.5 points (Report must contain 8 rows in total)
- Report 3 gives 3 points (Report must contain 9 rows and contain correct results for the functionality)

You'll get following points for the Autotest code:

- Iteration 1 – 2 points (if code produces correct amount of report rows)
- Iteration 2 – 1.5 points (if code produces correct amount of report rows)

Example:

If you produce 8 rows for the Report 1, then you get 2 points for the code but you may get less than 2 points for the Report, if the functionality test is not passed / is passed when it should not.

8 Tips and tricks

- If Sikuli can't find an image, lower the similarity or take a new identifiable picture
- Sometimes desktop background / icons can disturb the image identifying process – the solution is black background without icons on desktop
- Running the same test with different resolution will work unless the image is rendered somehow differently (which usually happens and your tests won't work in a machine with different resolution)
- If picture is moving, then Sikuli might have problems finding it – test the picture periodically with 1 second sleep
- You cannot move the mouse when Sikuli is running.
- Sikuli clicks in the center of the image by default. Consider this while taking an image.

9 Sikuli support

Detailed info about what and how can be found from the following links:
https://github.com/RaiMan/SikuliX-2014/wiki/Usage-in-Java-programming
http://www.sikulix.com/quickstart.html

10 Questionnaire (EXTRA point for just this)

Please fill in the questionnaire at https://www.surveymonkey.com/s/PPFCF9P at the end of the lab session or within 1 week after the lab has ended.

This is needed for Rasmus Sõõru's BSc thesis. If you decide to fill in the virtual questionnaire, so please also **add your name** and **answer to all of the questions (same goes for the sheet filled in at the lab)** to get the extra point. Only Rasmus will look at the individual answers. The course instructor will receive the list of students who answered and an anonymized summary from Rasmus.

Please answer during 1 week period (the sooner the better).

11 References

[1] Matthew Haughn. "GUI testing". WhatIs.com®. Feb. 2014. Web. 15 March 2015, http://whatis.techtarget.com/definition/GUI-testing-graphical-user-interface-testing
[2] Sikuli Doc Team. "How Sikuli works". doc.sikuli.org, Sikuli X 1.0 documentation. Nov. 2012. Web. 10 February 2015, http://doc.sikuli.org/devs/system-design.html
[3] 2007-2013 Software Testing help, Selenium Vs Sikuli. Retrieved from http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/09/sikuli17.jpg