Lab Instructions

- Submission deadline: Lab reports must be submitted within seven days. For example, if your lab takes place on a Tuesday, then you have to submit your report no later than on the following Monday, 23:59 hours.
- Late submission policy:
  - 50% of the total marks deducted for submission up to 24 hours late.
  - 100% of the total marks deducted for submission more than 24 hours late.
- Group: There should be a maximum of two members in a group. Answers should be your own group work, explained in your own words. If you work in a group of two students, make sure to mention the names of both students in the submitted lab report.
- Maximum amount of points is ten (10).

0. Installation

This lab requires virtual machine. Preinstalled Windows 10 with necessary programs is provided with the ova file. In the desktop of the virtual machine, you find a folder called Testar, which contains Testar application and two SUTs: for in lab task and homework task.

Virtual machine:

1. Install VirtualBox (if you don’t have it already): [https://www.virtualbox.org/](https://www.virtualbox.org/)
2. Inside VirtualBox choose **File -> Import Appliance -> choose the ova file**, then click **Next -> Import**
3. Wait until import is finished, then choose the imported machine and click Start
4. You can read more about ova file importing from here: [https://www.techjunkie.com/ova-virtualbox/](https://www.techjunkie.com/ova-virtualbox/)
1. Introduction

Purpose of the lab is to learn a way of GUI testing that can help testers to find critical GUI faults before complete test suite is finished. In this lab, we are using tool called Testar, which generates test scripts automatically, executes them and possibly finds critical faults.

a) Useful vocabulary

- **API** – application programming interface
- **GUI** – graphical user interface.
- **SUT** – system under test.
- **Scriptless** testing actually means, that scripts are generated automatically by the tool. Term ‘scriptless’ is more convenient to use than to say every time ‘testing, where scripts are generated automatically by the tool’.
- A test **oracle** is a mechanism that determines whether system executed correctly for a test case.
- **Widget** is a graphical object in the GUI and also a node in the widget tree. Some examples, that are widgets: button, container, textbox, etc.
- **UIA** (Microsoft UI Automation) is an API that allows manipulating user interface elements of another application
- **False positive** – test is marked as failed, when in reality it has passed
- **False negative** – test is marked as passed, when in reality it has failed

b) Automated GUI testing tools

Today, state of the art GUI testing tools are Capture & Replay and Visual testing tools.

**Capture & Replay tools**

These tools record sequences, that user executes manually, and then makes them into scripts. The script can be used as automation later on. For example, **Selenium**.

Advantage of these tools are that they are simple and easy to use. User must execute test case manually, record that, and it can be replayed later in regression testing as an automated test case. Disadvantages are that, whenever user interface changes, scripts break and that causes a maintenance problem.

**Visual testing tools**

To solve the maintenance problem, which came with Capture & Replay tools, Visual testing tools use image recognition to find areas to click on. So, when the interface of the program changes and the widget, that is used in the test, is moved to somewhere else, the Visual testing tool finds this area and the script would still work. Example tool is **Sikuli**.

These tools are easy to understand and no high programming skills are needed in order to develop test scripts. They also solve some of the maintenance problem, which came, if the user interface changed. However, it does not solve all the problems. For example, if some button is moved inside same menu, everything is correct and nothing breaks. But, when this button is moved to another menu, then the maintenance problem occurs, because the tool does not know to search this image of the button from another menu. There is also problem when appearance of the widget changes (ex. Button icon).
Although, Visual testing tools solve part of the maintenance problems, there are still some issues with maintaining scripts.

**Scriptless testing tools**

The name ‘Scriptless’ comes from the **Testar** tool testing approach, however it is not really a correct way of describing this approach. In this context, ‘scriptless’ actually means that, test scripts are generated automatically by the tool. Because these kind of tools do not need scripts, there are no maintenance issues regarding to script management. These tools use random testing, and due that, user can almost instantly start executing tests. This kind of testing is called robustness testing or monkey testing, which will test if the SUT is robust and does not have any failures (for example “<your app> has stopped working” window appears). This way, crashes and application hangings can be found.

Traditional way is to take requirements, generate test suite, automate it with scripts and test these requirements. With the scriptless testing, it is other way around, starting with automated random tests with implicit oracles, and then, step by step add some requirements. This way, it is possible to find critical faults with minimal effort. For example, tester sets up the initial configuration on Friday and lets the tool execute tests over the weekend.

**c) GUI and widgets**

GUI contains graphical objects, called widgets. An example is shown in Figure 1. Green dots and yellow rectangle represent individual widgets. Green dots are widgets, that have actions, that can be executed and a yellow rectangle represents a node in a widget tree.
Figure 1: Widgets

Widgets form a hierarchy called widget tree. The root of the tree is the whole window of an application, as shown in Figure 2.
All the widgets have properties like title, role, path etc. Property, which we will be using later in this lab, is widget title, used by Testar filters and oracles. The widget tree and properties of each widget form a GUI state. If user executes some action, for example click a button, it will cause a state change. GUI with
widgets, actions and state changes, form sequences, that can be used to test the user interface. And to
do that, an oracle is needed, which tells whether the state is correct or not after each change. A test
oracle is a mechanism that determines whether system executed correctly for a test case. In this lab, we
use Testar as our testing tool and this tool allows us to write more specified oracles in addition to implicit
ones, which determine failures such as crashes and hangings.

d) Introduction to Testar

Testar (also written as Test* following the logo of the tool) is a tool for automated testing at the User
Interface (UI) level of software applications. Testar is neither a Capture-and-Replay tool nor a Visual-
based testing tool. Testar does not record or need scripts. It uses accessibility technologies to access
User Interface and generates scripts automatically. [1]

Testar uses the following test cycle, which is illustrated in Figure 3:

1. Start System Under Test (SUT)
2. Get the specific state of the GUI - all the widgets and their properties
3. Check that state for failures described in test oracles
   a. if no failure found - continue
   b. if failure found - save test sequence and exit program
4. Check for stopping criteria (number of actions executed, etc.)
   a. if stopping criteria achieved - exit program
   b. if not found - continue with loop
5. Derive possible actions and filter out unwanted actions, specified by user
6. Select one action
7. Execute that action (which brings the GUI to another state)
8. Repeat from point 2
Figure 3: Testar test cycle [2]
2. In lab tasks

In this part, we are using buggy calculator app as our SUT and learn the basics of Testar. The buggy calculator app is provided in the ‘suts’ folder inside the lab package folder. This app is faulty and your job is to find out sequences that lead to failures. The specification that was used to develop calculator app is in Appendix A. This documentation helps to understand, what the calculator should do and then determine which outputs are failures, false positives and false negatives.

Figure 4 shows, how the example calculator application should behave, when input is processed. It means, whenever input is received, it can be either well-formed or not well-formed. If the input is not well formed, then the application must notify user with a defined message. And, if the input is well formed, result should be displayed. Exceptions, colored in red, should not be seen by the end user.

![Calculator input processing flow diagram](image.png)

**Figure 4: Calculator input processing flow**
a) Making first tests with Testar

Now, let’s configure Testar settings and make a first automated test.

1. Use testar.bat file to run Testar. Close all unnecessary windows and hide desktop items (right click on desktop -> view -> show desktop items) to minimize the risk of Testar executing other actions than the actions from the SUT.
2. Configure SUT connector to run “Part1_calculator” from the “suts” folder. An example is shown in Figure 5.

![Image of Testar settings](image)

**Figure 5: Example of Testar**

3. Now, we need to configure Filters to disable actions, that we don’t want Testar to execute, like closing or minimizing SUT or opening other external programs. To do that, we need to know titles of these widgets and we must write regular expression to match these titles, in the “Filters” tab.
“Disabled actions by widgets’ TITLE property (regular expression):” field. To find out these titles, do the following:

3.1. First, know the panic combination to close Testar: \textbf{SHIFT} + ↓
3.2. Start Testar in Spy-Mode (Big button with magnifying glass)
3.3. We want to disable all actions on the top window bar: closing the program, minimizing, maximizing and system menu (top left icon). Hover mouse over one of these buttons.
3.4. Press \textbf{SHIFT} + 3, to reveal more details about that widget. Find out the widget title. Example is shown in Figure 6, where title for closing the program is “Sule”. The title “Sule” means close in Estonian and it depends on the OS language.

3.5. Find out all the titles on the top window bar.
3.6. After finding out the titles, close Testar by pressing \textbf{SHIFT} + ↓ or with SUT close button.
3.7. Open “Filters” tab and write regular expressions, that match these titles, into the “Disabled actions by widgets’ TITLE property (regular expression):” field. Use “|” as a separator. To test regular expressions, visit \url{https://regex101.com/}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Widget Title (surrounded by red square)}
\end{figure}
4. In order to find errors, oracles must be defined. Open “Oracles” tab and write regular expressions “.*[rR]untime.*”, which will match all strings, which contain “runtime” or “Runtime”. This means, it will match “RuntimeException” for example.

5. Set the number of sequences to three and sequence actions to 15. Let other settings to be the same as in Figure 5.

6. Open “Time Settings” tab and set “Action Duration” and “Action Wait Time” to 0.2 seconds, “SUT Startup Time” to 10 seconds. In more complicated programs, where actions can take more time to finish, it is important to set these values regarding to the program.

7. Now, let’s run Testar. Click on “Generate” button, next to the Spy Mode button, and let Testar complete three sequences. It is possible (but unlikely), that you find sequence leading to runtime exception.

8. After Testar has finished, we can examine if something suspicious was found. Look inside “output\sequences_suspicioustitle” folder, if this folder contains some sequence, remember their number.

9. Go to “output\graphs” folder and open sequence that you found suspicious, or if you didn’t find any, just open sequence with highest number.

10. Double click on “offline_graph_conversion.bat”, which will generate different graphs inside the same folder. If none was created, there is probably error with system variables in your system – make sure you had Graphviz installed like described in chapter 0.

11. Open “graph_<number>_scrshotted.dot.svg” with a web browser and analyze the steps, that lead to “Pass” or “Failure”. Example, shown in Figure 7, shows, when input field is empty and “DEL” button is pressed, Runtime Exception appears.

![Figure 7: Screenshotted graph of failure](image-url)
b) Run tests and fill in the report

Read documentation in Appendix A and reconfigure Testar, according to that. Main settings, needed configuring, are Filters and Oracles. After configuring Testar, run some tests and analyze output. Look in the unexpected close folder too and analyze if there are failures or not. After finding failures, fill out the report in Appendix B. Reported failures should not be equivalent.

Testar is still a prototype tool and this version doesn’t have automated failure report, which sums up equivalent failures, so that the user doesn’t have to look through each one of them. Due that, after finding failures with some functions, it might be useful to disable the related widget in the Filters tab, but be careful not to disable widgets too easy handed, otherwise you might not find all the failures.

Note: Testar saves all sequences by default, including the ones made with the Spy-mode – if you close the program in spy-mode then you can find that sequence in the unexpected close folder (which is not a failure).
3. Homework task

In this part you have to use Testar to test an automated teller machine (ATM). Specifications for the ATM can be found in Appendix C and template for report in Appendix D. Remember if you don’t use Testar filters smart way, then you will find the same failures multiple times, which increases time spent for review.

Homework report consist of two parts: describing configured settings and a failure report.

In the first part, you have to describe what and why you configured and copy the value of the setting. In the what and why part, say clearly what (filters, oracles, number of actions, etc.) you configured, and in the last column, copy the exact value from the setting.

In the failure report, you must describe the failure with words and a picture; describe how to regenerate it; write down a combination of settings, which allowed you to find this failure; and submit a picture of screenhotted graph. The idea is that the lab assistants should be able to find these failures if they copy these settings to Testar. So make sure, when reporting a failure, you describe the settings that allowed you to find this failure. Your report should look like the given template.

Note: Describe steps for regenerating the failure as minimal as possible.
4. Grading

1. You can get maximum of 10 points of this lab.
2. 1 point for attending the lab.
3. Up to 9 points from homework task:
   3.1. Up to 2 points for describing what, how and why you configured (1p for filters, 1p for oracles).
   3.2. Up to 7 points for failure report (Each unique failure 1.4 p).
   3.2.1. If you fail to fill any of the requested columns, you will lose points.

Example solutions doesn’t give you any points, so it’s not necessary to report them.

5. Links and references

Useful Links:

1. [https://regex101.com](https://regex101.com)
   Page for testing regular expressions.
2. [https://testar.org/](https://testar.org/)
   Official Testar webpage.
   Testar hands-on manual. Keyboard shortcuts and test settings are explained on pages 26 – 31.
4. [https://chrome.google.com/webstore/detail/full-page-screen-capture/fdpohaocaelchifmmbbtknoalccl?hl=en](https://chrome.google.com/webstore/detail/full-page-screen-capture/fdpohaocaelchifmmbbtknoalccl?hl=en)
   Chrome tool for capturing whole webpage.
5. [https://www.techjunkie.com/ova-virtualbox/](https://www.techjunkie.com/ova-virtualbox/)
   How to import ova file in VirtualBox.
6. [https://drive.google.com/open?id=1e84lg4jHJtWBtXTAQTERrWFfmgFP2jD](https://drive.google.com/open?id=1e84lg4jHJtWBtXTAQTERrWFfmgFP2jD)
   Example screenshot of graph for homework report.

References:

Appendix A – Document for calculator application

1) About this document

This document provides description what the system should do, explanations of abbreviations and symbols used in the calculator app, and guide which was used in developing process. All these together should help in the testing process.

2) Development guide for developers

The system must be written in Java using JDK 8. If errors should occur, corresponding exception must be thrown and handled. Some examples of exceptions:

- ArithmeticException - when an exceptional condition has occurred in an arithmetic operation
- RuntimeException - any exception which occurs during runtime
- UnsupportedOperationException - when some of the functionalities are not implemented
- TimeoutException - when some process takes too much time

3) System Requirements

<table>
<thead>
<tr>
<th>REQ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>The system must handle decimal numbers.</td>
</tr>
<tr>
<td>REQ2</td>
<td>In addition to basic functionality (add, sub, multiply, divide), the system shall have functionalities for division with remainder, square root, power of, value of pi, and trigonometric functions: sin, cos, tan. These functions must be accessible from the main panel.</td>
</tr>
</tbody>
</table>
| REQ3  | Functionalities ‘square root’, ‘sin’, ‘cos’ and ‘tan’ should be displayed without parentheses in the input field. If user want to use these functions inside an expression, they must be surrounded with parentheses. (“5 + (sin 5)”)
| REQ4  | The system shall have delete function, which erases last character from the input. If the input is empty, nothing shall be done or displayed. |
| REQ5  | The system shall have functionality to clear the input field with one click. If the input field is empty, nothing shall be done. |
| REQ6  | The system must have evaluation function, accessible from main panel. Input for that function must be taken from the input field and outcome must be also displayed in the input field. If the input is not correct, the system shall notify about faulty input. |
| REQ7  | The system shall save the value that was calculated and automatically clear the input field if user starts typing another expression. |
| REQ8  | The system must allow user to insert last calculated value into new expression. |
| REQ9  | The system should have menu for functionalities that does not fit on the main panel. This menu should also contain link to online documentation. |
4) **Explanations of abbreviations and symbols**

- **Nan** - Not a number
- **SQRT** - square root
- **DEL** - function that erases last character from input
- **PI** - value of pi 3.14...
- **SIN** - trigonometric function of sin
- **COS** - trigonometric function of cos
- **TAN** - trigonometric function of tan
- **%** - division with remainder function

5) **Information messages**

- NaN
- Faulty input
- Infinity
- Warning
- Be careful while opening external applications! It might be dangerous to open other applications while automatic tests are running.
Appendix B – report for in lab task

a) Configured settings and reasons for that

<table>
<thead>
<tr>
<th>Setting number</th>
<th>What (point out clearly) and why was done? Explain in your own words</th>
<th>How was done? Copy the value from the setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex1</td>
<td>Filtered “File” menu to prevent Testar closing the SUT and therefore report the sequence as unexpected close</td>
<td><em>.File.</em></td>
</tr>
<tr>
<td>Ex2</td>
<td>Configured Oracle to find failures with messages containing “runtime”</td>
<td><em>.runtime.</em></td>
</tr>
</tbody>
</table>

b) Failure report

<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
<th>How to regenerate</th>
<th>Found with settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEL button: Runtime Exception – when the input field is empty, click DEL button</td>
<td>Start the SUT -&gt; click DEL</td>
<td>Ex1, Ex2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C – specification for the ATM

1. The ATM shall have five types of accounts: regular, premium, business, senior and student.
   1.1. Regular account is the most basic account type, which allows client to deposit money to their account and withdraw it within the size of their assets (account balance cannot go lower than 0).
   1.2. Premium account, in addition to the same specifications as the regular account, has a function to take credit up to 100 units.
   1.3. Business account has functions: withdraw, deposit and take loan. If client withdraws more money than account has, then this type of account must give the client money for up to -1000 units (account balance can go up to -1000). Client can apply for two types of loans: fast loan and real estate loan. Fast loan is always approved and money is transferred to client account immediately. Real estate loan is bigger and therefore, it is approved only when the client has account balance higher than 1000 units.
   1.4. Senior account, in addition to regular account functionality, has a function to transfer money from pension funds.
   1.5. Student account, in addition to regular account functionality, has a function to take study loan. In order to apply for a study loan, a guarantor name is needed.

2. The system must be written in Java.
   2.1. If something fails, then exceptions must be thrown.
   2.2. All exceptions must be handled inside the system.
   2.3. The end-user only has to see a notification, not an error message.
      2.3.1. After the notification message, the user must have a possibility to continue from the last error-free state.

3. Information messages:
   3.1. You have withdrawed...
   3.2. Remaining balance...
   3.3. You have deposited...
   3.4. Increased balance...
   3.5. You can try to take credit...
   3.6. You have taken credit...
   3.7. Maximum credit sum can be...
   3.8. You had positive response for...
   3.9. Information
   3.10. Wrong sum
   3.11. Wrong input
   3.12. You cannot deposit negative amount
   3.13. You cannot take that much credit
   3.14. You don’t have enough money
   3.15. Input must be integer
Appendix D – report for homework task

a) Configured settings and reasons for that

<table>
<thead>
<tr>
<th>Setting number</th>
<th>What (point out clearly) and why was done? Explain in your own words</th>
<th>How was done? Copy the value from the setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex1</td>
<td>Filtered Close and Minimize button to prevent SUT from disappearing</td>
<td>.[Cc]lose.*</td>
</tr>
<tr>
<td>Ex2</td>
<td>Configured Oracle to find failures with messages containing “...fail...”</td>
<td>.<em>[fF]ail.</em></td>
</tr>
</tbody>
</table>

b) Failure report

<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
<th>How to regenerate</th>
<th>Found with settings</th>
<th>Link to graph picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex</td>
<td>Business account: Loan: When trying to take loan more than once, a failure message is shown</td>
<td>Business account -&gt; Loan -&gt; Ok -&gt; Loan -&gt; Ok -&gt; failure</td>
<td>Ex1, Ex2</td>
<td><a href="https://drive.google.com/open?id=1e84Ig4JHJtWBTxTAQTERrWFfmgF2jD">https://drive.google.com/open?id=1e84Ig4JHJtWBTxTAQTERrWFfmgF2jD</a></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>