Exercise 9: GUI-based System Testing

Unit-Tests enable efficient and automated testing of implemented functional behavior on code level. Nevertheless, on system level testing via the user interface is often the only option – especially for applications with a large number of complex graphical user interfaces (GUI), which sometimes even mix UI code and business logic etc. The objective of these exercises is to provide an overview of different approaches for automated GUI testing and show advantages and disadvantages of individual approaches.

9.1 GUI Testing: Abbot Example (2 points)

First, record a test script. Use the following software to complete the exercises:

Abbot 1.3.0 (http://abbot.sourceforge.net/doc/overview.shtml) plus Costello Script Editor.

JCalculator 0.4a (www.sourceforge.net/projects/javcalc), a simple calculator implementation, is used as an example to illustrate automated GUI testing.

Instructions

Use the script editor *Costello* to create simple automated GUI test, which adds two numbers and checks the result. Please read the user guide *Getting Started with Costello* (http://abbot.sourceforge.net/doc/user-guide.shtml) and proceed as follows:

- ☐ Unpack the archive *JCalc0.04a.zip* containing the application to be tested. Start the script editor via *abbot.bat*. Create a new script (*File > New Script…*) and set following options: *Test > Use Separate VM* and *Test > AWT Mode*.
- ☐ Provide the information to start the application: At *Insert your launch information here* you have to specify *JCalc* at *Target Class Name* and the path to the unpacked archive file at *Classpath*. Note, a correct input is indicated by changing the text color of the class name from red to black. Now you are able to start the test script. E.g., select the step *Terminate* in the test script and run the script until this step (*Test > Run to Selection*).
- ☐ Use the function *Capture > All Actions* to record a sequences of actions (i.e., mouse clicks). Do a simple calculation such as *1 + 2*. Complete the sequence by clicking on the button = to display the result. Switch to the script editor, which will finish the recording. Your test script should now include the sequence of your mouse clicks.
- ☐ Move the mouse pointer to the results field of the calculator application and press *Shift+F1*. The script editor should now show the hierarchy of the GUI elements of the calculator in the tab *Hierarchy,* with the *JTextField instance* selected. The tab *Properties* shows all attributes of the text field including the calculation result *(text)*. Select this text and add a check to your test script by clicking the button *Assert equals*. (Note: The button changes its function if you press *Shift* or *Ctrl*!)

☐ Review and run your test script. Save the script as *JCalcTestAdd.xml*. Have a look into the XML script file to see how your actions are specified.

9.2 Write Your Own GUI Tests (2 points)

What's behind the recorded test scripts? Goal of this exercise is to implement test scripts in Java like those you have previously recorded. Use the framework *Abbot*, which is also the basis for the test scripts created with *Costello*.

Instructions

The test scripts recorded in the previous exercise should now be implemented as JUnit test cases. The framework *Abbot* is used to access the GUI elements of the *JCalculator* application. Please read the user guide *Getting Started with the Abbot Java GUI Test Framework* (http://abbot.sourceforge.net/doc/overview.shtml) and proceed as follows:

☐ Create a new project in Eclipse named SQE09-*Lastname*_AbbotJCalc to collect all results of the following steps. Add the class *JCalc* to this project.

☐ Add the unit test class *JCalcTest*. Derive the test class from *ComponentTestFixture* available in *abbot.jar*.

☐ Create an instance of the calculator's GUI in the *setUp()* of the test class, just like *JCalc* does in the method *main()*.

☐ Additionally, provide references to all GUI elements required for testing (e.g., references to the *JButton* instances of the calculator's number buttons and the results field). Thereby, use the instance of *ComponentFinder* you get via *getFinder()* and appropriate matchers:

```
JButton button1 = (JButton) getFinder().find(new ClassMatcher(JButton.class) {
    public boolean matches(Component c) {
      return super.matches(c) && ((JButton) c).getText().equals("1");
    }
});
```

☐ Write a test method *testAdd()* mirroring the test you created in the previous exercise. Use a *JButtonTester* to issue the clicks on the instances of *JButton*.

```
JButtonTester bt = new JButtonTester();
bt.actionClick(button1);
```

☐ Check the result in the *JTextField* with a JUnit assert.

Submission Instructions

- Exercise 9.1: Submit the test script (e.g., *JCalcTestAdd.xml*). Add your name and ID in as comment in the script.
- Exercise 9.2: Submit the Eclipse project containing the source code and all relevant files. Include your name and ID in a comment at the beginning of the source files.
- **Note:** The Eclipse project has to be completely self-contained. Do not use absolute paths to reference any (external) libraries.

Exercise 10: GUI-based Testing for Triangle1st

The objective of this exercise is to experiment with automated testing for your own GUI.

10.1 Capture & Replay for Triangle1st (2 points)

The calculator used in the previous exercise is a simple example just to get the basic idea. Now, experiment with the user interface you developed for the application *Triangle1st* in the first exercise. Consider following hints:

- ☐ Use the script editor *Costello* to create simple automated GUI test as described in the previous exercise.
- ☐ If necessary, consider changing the implementation to make the user interface testable.
- ☐ Record a script that enters three sides of a valid triangle and checks the calculated area and perimeter.
- ☐ Record a script that checks that an error is indicated when no (or incorrect) data is entered. If a message box is used to indicate the error, make sure that the test waits for this box to appear and closes it before terminating.
- ☐ Review and run your test scripts. Save the script as *Triangle1st_CalcValid.xml* and *Triangle1st_ErrorMsg.xml.*

10.2 Implement Your Own GUI Test (3 points)

The test script calculating the perimeter and area recorded in the previous exercise should now be implemented as JUnit test cases. The framework *Abbot* should be used to access the GUI elements of the *Triangle1st* application directly. Note:

- ☐ Create a new project in Eclipse named SQE10-*Lastname*_AbbotTriangle1st to collect all results of the following steps. Add the code for the *Triangle1st* application and the unit test class *Triangle1stAbbotTest* (plus *abbot.jar* and *JUnit*).
- ☐ Follow the instructions given in the previous exercise on implementing a JUnit test case for the calculator example. If necessary, consider changing the implementation to make the user interface testable.
- ☐ Only the one test case for calculating the perimeter and area from valid inputs is required.

Submission Instructions

Submit a zip archive including the complete Eclipse project and all relevant additional files as specified. Adhere to the naming convention for projects: SQE##-*Lastname_Title* (## = exercise number, *Lastname* = your name, *Title* = exercise title).

- Exercise 10.1: Submit *Triangle1st_CalcValid.xml* and *Triangle1st_ErrorMsg.xml*. Add your name and ID in as comment in the script.
- Exercise 10.2: Submit the Eclipse project containing the source code and all relevant files (including those from the application *Triangle1st*). Include your name and ID in a comment at the beginning of the source files.

Exercise 11: Lessons Learned from Test Automation (1 p)

Learn about the strength and weaknesses of automated testing by reflecting about your observations and by reading about the experience others have made:

- Stefan Berner *et al.*: *Observations and Lessons Learned from Automated Testing.* Proc. of the 27th International Conference on Software Engineering, ICSE'05, May 2005, St. Louis, Missouri, USA.

Instructions

Read the conference paper and answer following questions. Document each of your answers in a short paragraph in the text file *exercise11.txt.*

1. Which kind of tests is able to detect more defects, manual tests or automated tests? Explain your answer for the testing approach described in the paper.
2. Why is it necessary to execute the automated tests frequently? Explain your answer in terms of the factors shown in *Figure 2* in the paper.
3. How can the maintainability of testware be facilitated or improved? Name and describe at least three ways.

Submission

- Submit the text file *exercise11.txt* containing your answers.

General Submission Instructions

- Add a comment at the beginning of every source code file containing your full name and your student ID number.
- The Eclipse projects have to be completely self-contained. Do not use absolute paths to reference libraries or otherwise the submitted exercise cannot be compiled without build errors and will not be accepted.
- Submit the complete Eclipse project including all relevant additional files as specified via packing in a Zip archive. Adhere to the naming convention for projects: SQE##-*Lastname_Title* (## = exercise number, *Lastname* = your name, *Title* = exercise title).
- Upload the archive at the specified submission link on the elearning platform http://elearning.fh-hagenberg.at/ until the specified date and time.