Automated Testing at the GUI level

Hands-on do it yourself session

4th of September 2017

A-TEST, FSE, Paderborn

Ramón de Vries, Floren de Gier and Tanja E. J. Vos

Contents

1 Objective of this Session

The objective of this session is to get first impressions of the challenges associated with GUI testing. You will be using a given System Under Test (SUT) with several faults. Your task is to setup a test which finds the faults within this SUT. To achieve this, you will use TESTAR which automatically stresses GUI-based applications and can – once setup correctly – detect specific types of errors. Your goal is to find as many errors as possible and reproduce them using TESTAR.

2 What is GUI Testing?

GUI Testing is a testing technique where one tests the SUT solely through its Graphical User Interface (GUI). The only way to find errors is to thoroughly observe the status of the GUI throughout the test. This type of testing is usually carried out manually, where a tester just follows a previously written "test case" and verifies whether the application responds to all inputs as expected. On the one hand GUI testing is a relatively straightforward process, since one does not need to read or test the source code of the SUT. On the other hand it is quite laborious, time consuming and, well... boring...

Therefore, in this assignment we will try to automate GUI testing by using a tool called *TESTAR* [1]. TESTAR is a random testing or *fuzzing* tool. It automatically generates input sequences for the SUT by clicking and typing on the controls of the GUI. It is able to recognize when the SUT behaves *unusual* and reports this to you, the tester.

3 Handson step by step

3.1 Manually test the SUT

Before using TESTAR, you will test the SUT manually, to get an impression of potential faults. Double click the file "calc - Shortcut" on the Desktop and test the application. How many and what type of failures can you find?

3.2 Starting up TESTAR

Now let us start up TESTAR to see what it can do. Navigate to and execute the file *testar.bat*. The GUI of TESTAR starts up. Basically this is a dialog that enables us to configure the values that are present in the test.settings file. These settings define values that tell TESTAR about the SUT we want to test, and define details about how we want to do that.

[1] TESTAR is a result of the european project FITTEST `http://crest.cs.ucl.ac.uk/fittest/`

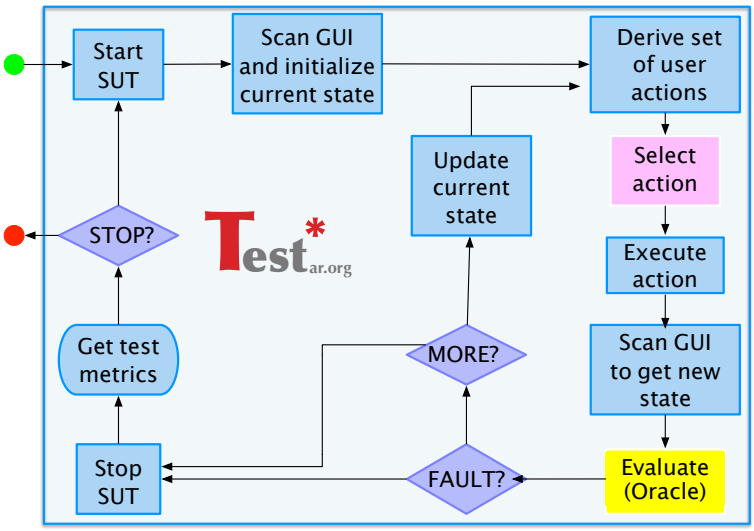


Figure 1: TESTAR test cylce

Let us start by clicking on the *SPY* button (the one with the magnifying class). This one enables us to spy the buttons of the SUT and see all the information that there is... Hover the buttons and look for yourself.

3.3 The SPY Mode

The SPY mode allows you to inspect the controls of the GUI. In the spy mode you can hit Shift + 1 to see what actions TESTAR chooses from. If you hit Shift + 3 and hover over an element it will show detailed information about that element. This way you can find the titles of the elements. To go back to less information just hit Shift + 2 .

To see a display of the widget tree there is Shift + 4 .

To stop TESTAR, hit Shift + ↓ . You can find more shortcuts in the appendix of this assignment. It is a good idea to become familiar with this Mode and its shortcuts.

3.4 The GENERATE-TEST mode

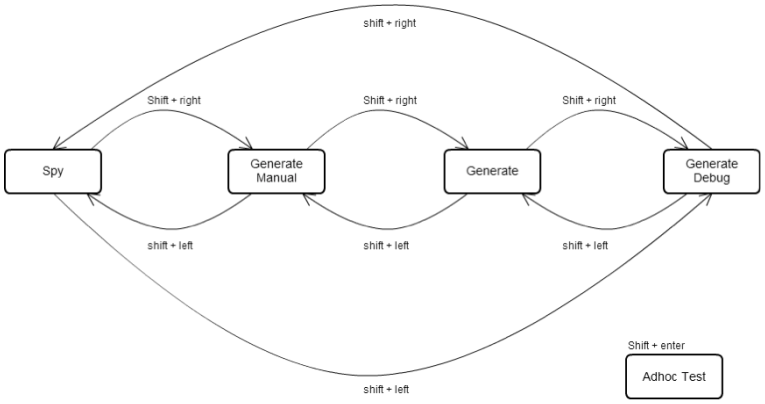
In this mode, the TESTAR tool carries out automated testing following the test cycle depicted in Figure 1.

Basically, it derives a set of possible actions for the current state that the GUI of the SUT is in. Then, it automatically selects and executes an action from this set which makes the SUT go to a new GUI state. This new state is evaluated with the available oracles. If no fault is found, again a set of possible actions for the new state is derived, one action is selected and executed, etc. This continues until a fault has been found or until a

stopping criteria is reached. With the right test set up all you will need to do is to wait for your tests to finish.

The default behaviour includes random selection of actions and implicit oracles for the detection of the violation of general-purpose system requirements: like that the SUT should not crash, the SUT should not find itself in an unresponsive state (freeze) and the UI state should not contain any widget with suspicious titles like *error*, *problem*, *exception*, etc.

Now let us do some automated testing. From the SPY mode you can go to various GENERATE-TEST modes by hitting Shift + Arrow Right. The Figure below shows how they relate:



You can interact during the tests at any time, or even stop them through the *panic* keyboard shortcut (Shift + ↓). The main working modes that apply for the running tests are presented next, and you can switch between them through the keyboard shortcuts Shift + ← (or Shift + →).

- **Generate**. The default operation mode. Runs tests as specified by the test set up.
- **GenerateManual**. You will take over TESTAR control and manually perform some actions during a test. This is particularly interesting if you want to force the test to move to a concrete part of the UI and/or if you want to move the test out of the current UI.
- **GenerateDebug**. Similar to the *Generate* mode, but you will be able to display the UI actions (Shift + 1) and color codes are applied during the test:

  green for UI actions that the test can execute,

  red for the current UI action being executed, and

  (alpha) blue for UI actions that were already executed.

- **Slow motion**. Hit Shift + Space to (de)activate a delay between the executed UI actions, which will aid to supervise the test execution of

more critical UI parts.

Run some tests. These are monkey test. TESTAR can *see* the controls of the SUT's GUI and can automatically detect possible actions. It randomly selects and executes these actions.

In the tab "Time Settings" you can set the action duration and the time that TESTAR waits after executing each action. Play with the settings and observe what happens.

Until now you should have familiarized yourself a bit with the tool. TESTAR does not only generate random sequences, it also records every action it executes and can thus replay sequences. The directory "output/sequences" contains all sequences that the tool generated.

☞ HANDSON: Record 1 sequence of at max 30 actions and then replay it. To replay it you can use the Replay button and specify the path to the sequence.

3.5 Design a Test Oracle

The default behavior of TESTAR is not very useful. In order to detect faults, TESTAR needs a so-called *Test Oracle*. The Oracle tells whether a specific state is correct, faulty or suspicious. TESTAR comes with a default Oracle which detects when the application crashes – meaning when it is not running or does not react anymore. Although this will detect simple faults it will not detect others. If you played with the SUT a bit, you probably detected a few faults on your own, such as dialogs with exception messages. To detect such faults one can simply scan the current GUI state for specific words, such as "exception" or "error" or "NullPointerException", etc. The tab "Oracle" has a field "Suspicious Titles" in which you can write Java Regular Expressions. TESTAR will apply these expressions after the execution of each action in order to find potential matches.

Example: `.*[Ff]aultystring.*|SomeOtherFaultyString`

This expression will make TESTAR look for the string "Faultystring" (upper- or lowercase) anywhere on the screen in any position as well as for the exact match "SomeOtherFaultyString". If TESTAR encounters such a string, it will complain and save the corresponding sequence under "output/sequences_suspicioustitle".

☞ HANDSON: Write regular expressions for the errors that you have encountered in the SUT. Run some tests. Did you find any error with your oracles?

3.6 Adjust TESTAR's Behavior

You might have observed, that from time to time, TESTAR executes "undesirable" actions. The tool minimizes or even terminates the application, which is not optimal for the testing process. Even worse: If you observed the tool's output you might have noticed that it detects a "crash" whenever it closes the main window. Obviously the tool does not know that closing the main window terminates the application. In the tab "Filters" you can find the setting "Click Filter" which is similar to the "Suspicious Titles" field. TESTAR will ignore all actions that exercise control elements whose title matches the given expression.

Example: `.*Backspace.*|.*CE.*|.*View.*`

This expression will ignore clicks to all control elements whose titles contain the given strings.

👉 HANDSON: Configure TESTAR such that it does not close or minimize the window anymore. In addition disallow clicks to the "Open File" menu item, to prevent TESTAR to "go wild" on the operating system's files.

Use the Spy Mode to see whether your ClickFilters have an effect ("Shift + 1" to visualize the actions)!

3.7 The test.settings file

As indicated before, the GUI of TESTAR is basically a dialog that enables us to configure the values that are present in the test.settings file.

If you want you can also edit these directly. For now, to illustrate we refer to Figure 2, where you can see part of the test.settings file for the calculator application. All the settings are explained in detail in the manual also.

3.8 👉 Find as many Faults as possible and reproduce them

Now that you know how to use the tool your task is to setup a longer test, e.g. 30 sequences with a length of 50 actions (whatever you please). Run the tool and observe its output! Does it find and report the faults? Can you replay the erroneous sequences and reproduce the errors? If the tool does undesirable things, improve your setup and restart the test. At the end of this task you should have a folder with several erroneous sequences that can be replayed and expose errors of the SUT.

```
######################################################################
# TESTAR mode
#
# Set the mode you want TESTAR to start in: Spy, Generate, Replay
######################################################################

Mode = Spy

######################################################################
# Connect to the System Under Test (SUT)
#
# Indicate how you want to connect to the SUT:
#
# SUTCONNECTOR = COMMAND_LINE, SUTConnectorValue property must be a
# commandline that starts the SUT. It should work from a Command Prompt
# terminal window (e.g. java - jar SUTs/calc.jar ).
# For web applications, follow the next format: web_browser_path SUT_URL.
#
# SUTCONNECTOR = SUT_WINDOW_TITLE, then SUTConnectorValue property must be
# the title displayed in the SUT' main window. The SUT must be manually
# started and closed.
#
# SUTCONNECTOR = SUT_PROCESS_NAME: SUTConnectorValue property must be the
# process name of the SUT. The SUT must be manually started and closed.
######################################################################

SUTConnector = COMMAND_LINE
SUTConnectorValue = java -jar "C:\\Users\\Tania\\Desktop\\calc.jar"

######################################################################
# Sequences
#
# Number of sequences and the length of these sequences
######################################################################

Sequences = 6
SequenceLength = 100

######################################################################
# Oracles based on suspicious titles
#
# Regular expression
######################################################################

SuspiciousTitles = .[eE]rror.*|.*[eE]xception.

######################################################################
# Actionfilter
#
# Regular expression. More filters can be added in Spy mode,
# these will be added to the protocol_filter.xml file.
######################################################################

ClickFilter = .[cC]lose.*|.*[cC]errar.*|.*[fF]ile.*|.*[aA]rchivo.*
            |.*Minimize.*|.*[mM]inimizar.
```


Figure 2: Part of the test.settings file

3.9 👉 Play with the Tool and Test Microsoft Paint or Notepad

Obviously, what we have done so far was rather simplistic and does not find very complex faults. In addition the SUT was very simple as well. However, TESTAR can test more complex SUTs. You can change the path to another SUT if you like to see what happens when you test something like:

- Microsoft Paint
- Notepad

Be careful, though, the tool might perform dangerous and unexpected actions!

3.10 Testing web applications

Bitrix24 is a complete suite of social collaboration, communication and management tools for your team.

We have already configured a test.settings file for TESTAR for bitrix24.

Start TESTAR up with the bitrix24 settings.

Starting `testar.bitrix24.com` we get a screen that asks us to login. Evidently, letting TESTAR guess values randomly will take us a long time to enter. We would like TESTAR to enter when it starts up the system. For this we can edit the protocol that TESTAR uses to implement the test-cycle from Figure 1. In the next section this is explained.

3.11 Editing the protocol

TESTAR offers a more detailed API in the form of a Java protocol. In the "General Settings" dialog you can click the "Edit Protocol" button to see the default source code that the tool uses to perform its tests. The source code allows you to write much more fine-grained oracles and definitions for drag and drop actions etc.

Read section 4 of the TESTAR user manual, there the different parts of the protocol are explained.

3.11.1 Editing the protocol to logon

For example, if we want to do the login to bitrix when we start up the system we need to edit the `start_System()` method (from section 4.3 of the TESTAR manual).

👉 Let us add a piece of code so we can do the login. The TESTAR has a user account with: username: `tvos@pros.upv.es` and password: `testarbitrix24`. Section 4.3 of the manual gives the following example:

```
new CompoundAction.Builder()
.add(new Type("my user"),0.1) //assume keyboard focus is
                              //on the user field
.add(new KeyDown(KBKeys.VK_TAB),0.5) //assume next focusable field is pass
.add(new Type("my user pass"),0.1)
.add(new KeyDown(KBKeys.VK_ENTER),0.5).build() //assume login is
                                               //performed by ENTER
.run(sut, null, 0.1);

return sut;
```

You need to make a new `CompoundAction` builder, that contains a series of actions that you want to execute when the system starts up. Note that when the username is being typed, it is assumed that keyboard focus is on the user field, but this may not be the case when you start up. You might need to add some TABs before to make sure that this assumption holds.

Also maybe you need to add some `Util.pause(`*number*`)` to give the website time to render completely before you execute some CompoundAction.

3.11.2 Editing the protocol to add an oracle

If we want to add oracles other than those that can be represented with the regular expressions, we can edit the `get_Verdict()` method (section 4.6 of the TESTAR manual).

Let us look at a small example, where we want to check in every state that the widgets that contains some text (i.e. its NativeRole is "UIAText") can fit a font size that is at least the `MINIMUM_FONT_SIZE`. This can be done by writing the following piece of code returning a `Verdict`.

```
Verdict getSmallTextVerdict(State state,
                            Widget w,
                            Role role,
                            Shape shape){

        final int MINIMUM_FONT_SIZE = 8; // px
        if (role != null
        && role.equals(NativeLinker.getNativeRole("UIAText"))
        && shape.height() < MINIMUM_FONT_SIZE
        )

        return new Verdict(Verdict.SEVERITY_WARNING,
                           "Not all texts have a size greater than "
                           + MINIMUM_FONT_SIZE + "px");
        else return Verdict.OK;
        }
```

Then we can call this method in a `get_Verdict` while loop for every state.

```
Role role;
String title;
Shape shape;

// apply the oracles to all widgets
for(Widget w : state){

        role = w.get(Tags.Role, null);
        title = w.get(Title, "");
        shape = w.get(Tags.Shape, null);

        // check for too small texts to be legible
        verdict = verdict.join(getSmallTextVerdict(state,

                                                   role,
                                                   shape)
                                                   );

}

return verdict;
```

☞ Try to add a piece of code that defines an oracle that checks whether every image on the screen (i.e. its NativeRole is "UIAImage") has an additional textual description according to the WAI guidelines for accesible webpages.