TESTAR - from academic protoype towards an industry-ready tool for automated testing at the User Interface level

Urko Rueda, Tanja E.J. Vos, Francisco Almenar, Mirella Oreto, and Anna Esparcia Alcazar

Universidad Politecnica de Valencia, Spain,
Camino de vera s/n, 46022 Valencia
{urueda,tvos,fraalpe2,mimarmu1,aesparcia}@pros.upv.es
https://staq.dsic.upv.es/webstaq/

Abstract. Testing applications with a Graphical User Interface (GUI) is an important, though challenging and time consuming task. The state of the art in the industry are still capture and replay tools, which may simplify the recording and execution of input sequences, but do not support the tester in finding fault-sensitive test cases and leads to a huge overhead on maintenance of the test cases when the GUI changes. While search-based test case generation strategies are well researched for various areas of testing, relatively little work has been done on applying these techniques to an entire GUI of an application. In this paper we present the tool TESTAR, an automated search-based approach to test applications at the GUI level whose objective is to solve part of the maintenance problem by automatically generating test cases based on a structure that is automatically derived from the GUI.

Keywords: Automated, Search-Based, Testing, User Interface level

1 Introduction

Testing software applications at the Graphical User Interface (GUI) level, is a very important testing phase to ensure realistic tests from a user's perspective. From the GUI, the product functionality is accessed, which poses the GUI as a natural access point towards customer acceptance testing.

Nowadays, a substantial part of GUI testing tools at industry are still based on the Capture and Replay (C/R) technique [1], which requires significant human intervention to produce application interactions. These interactions (i.e. clicks, keystrokes, drag/drop operations) are recorded by the C/R tool and used as regression tests for new product releases. However, a main concern in C/R is that of evolvable software forcing testers to fix old test cases. This is a critical maintenance problem of the technique, which makes companies to return to manual regression testing. A more advanced technique, Visual testing [2] [3],

takes advantage of image processing algorithms to simulate step by step human interactions. Though visual approaches simplify the work of testers, they are slow, imprecise (prone to false positives with wrong UI element identification, and false negatives with missed UI elements), and also rely on the GUI stability.

With TESTAR (Test Automation at the user inteRface level) [4] [5], we present a Model-Based Testing approach for testing that automatically generates and executes test cases based on a tree model (automatically derived from the GUI through the Accessibility API). Since the GUI is not assumed to be fixed tests still run even though the GUI evolves, which reduces the maintenance problem that threatens the techniques mentioned earlier.

Next, the paper presents the TESTAR tool engine and its Search-Based capabilities, together with industrial cases performed to evaluate the approach.

2 TESTAR engine for Automated User Interface testing

TESTAR uses the Operating System's Accessibility API, which has the capability to detect and expose a GUI's widgets, and their corresponding properties[1]. Fig. 1 shows a sample TESTAR widget inspection over a simple *Calculator* application. The tool enables programmatic interaction with the identified widgets. It derives sets of possible actions for each state that the GUI is in and automatically selects and executes appropriate ones in order to drive the tests. In completely autonomous and unattended mode, the oracles can detect faulty behaviour when a system crashes or freezes. Besides these free oracles, the tester can easily specify some regular expressions that can detect patterns of suspicious titles in widgets that might pop up during the executed tests sequences. For more sophisticated and powerful oracles, the tester can enrich the default (Java-based) protocol that is used to drive the tests.

A basic testing cycle with TESTAR can easily be setup by: **(1)** install the Systen Under Test (SUT), setting the running command, and customize the testing environment for the SUT (i.e. time to wait for the SUT to start, desired length of a test run, etc.) **(2)** inspect the widgets of the SUTs and decide on how to fine-tune the SUT testing environment (i.e. do not click any button with the title "disconnect", "close" or "exit", etc.) **(3)** add simple oracles based on suspicious titles containing words like "Error", "Exception" or "Problem". **(4)** Define custom actions like a login with "user" and "pass" to a system, etc.).

Once the the setup is ready, the customizable testing protocol (described in the following) is put in scene and performs the steps as is shown in Fig. 2: starting the SUT, obtaining the GUI's *State* (a widget tree with properties detail), deriving the sensible actions from a SUT's state (i.e. clicks, text inputs, mouse gestures), selecting and executing actions, applying oracles to check invalid states, and driving the test runs in concordance with the test settings specified by the tester (i.e. time limit, number of faults found). Additionally, TESTAR stores any suspicious sequence to a dedicated directory, which can be later replayed and carefully inspected and analysed.

[1] Properties like: display position, widget size, ancestor widgets, etc.

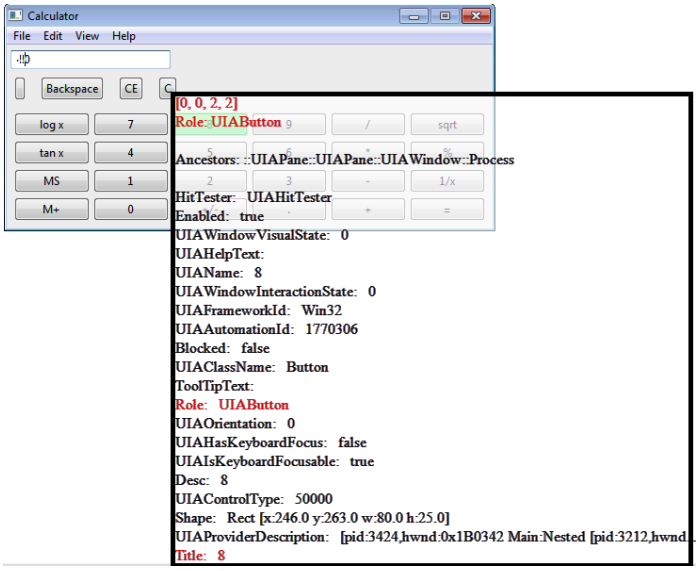

Fig. 1. TESTAR widget inspector for a desktop Calculator application

Protocol customization is performed through an end-user compilable Java class file, which contains the following methods (each coinciding with a phase in Fig. 2) that can be edited:

```
// initial setup before starting SUT test
01. void initialize(Settings settings)
// clean-up tasks for new test runs
02. void beginSequence()
// any action to be taken during SUT execution
03. SUT startSystem()
// step-by-step STATE of the SUT, with an attached ORACLE
04. State getState(SUT system)
// determines the STATE ORACLE verdict
05. Verdict getVerdict(State state)
// the set of available ACTIONs from a SUT's STATE
06. Set<Action> deriveActions(SUT system, State state)
// which ACTION should be PERFORMED next (i.e. random, Search-Based)
07. Action selectAction(State state, Set<Action> actions)
// runs an ACTION from a SUT STATE, with return code (success?)
08. boolean executeAction(SUT system, State state, Action action)
// determines the stopping criteria
09. boolean moreActions(State state)
// finishing tasks for an ending test run
10. void finishSequence(File recordedSequence)
// determines whether to continue SUT testing (additional runs)
11. boolean moreSequences()
```

2.1 Applying SBSE for intelligent automated testing

The default TESTAR *Action* selection mechanism is random (randomly selecting an action from a set of actions available in a concrete SUT's state). To try to

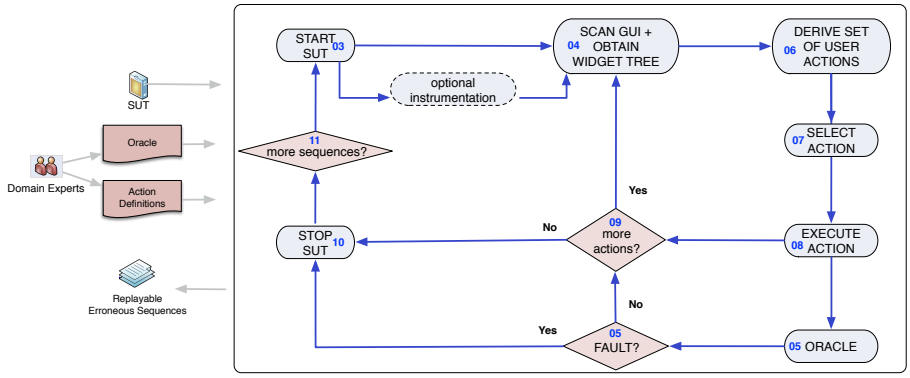

Fig. 2. TESTAR testing flow

enhance the capabilities of TESTAR search-based techniques are investigated to intelligently guide which actions to execute over the SUT's GUI.

In [6] re-inforcement learning (*Q-Learning*) was investigated. The main idea was to change the probability distribution over the sequence space, so that seldom executed actions will be selected with a higher likelihood than others, in order to favor exploration of the GUI. It was found that the technique helps to better explore the SUT's GUI to its extents, yet it does not become faster in crashing the application.

In [7] *Ant Colony Optimization* (ACO) with a metric called the *Maximum Call Tree* (MCT) for the adequacy criterion was studied, adopted from [8]. MCT is used to reduce the size of existing test suites, by instrumenting the SUT and extracting the *method call tree* for each generated GUI test sequence. These first experiments showed that the implementation worked, that the algorithm continuously improved the candidate solutions and eventually found a better sequence than the random strategy.

Both studies leave a lot of open questions for future research work. However, since for industry random seems to be working just fine, we are currently concentrating on transferring TESTAR to practice (see next Section).

3 Conclusions

TESTAR has been successfully applied in three different industrial context [9], obtaining feedback for improved innovation transfer and eventual market-adopting.

Currently, TESTAR is presented as one of the research results that are transferred to industry as part of the Spanish Software Testing Innovation Alliance[2]. This alliance brings together key Spanish stakeholders in software testing to jointly work to improve innovation and technology transfer from University to

[2] http://innovationalliance.eu

SME. The objective is to increase research impact in practice, education, business and also feedback into the research. Within this context, TESTAR is currently being evaluated at *PINEA* (for their cloud management system *Clickeen*[3]), *Indenova*[4] (a software technology provider for businesses and institutions) and *Sopra*[5] (a consultancy company that provides technology services and software).

Acknowledgments. This work was financed by the FITTEST project, ICT-2009.1.2 no 257574, the SHIP project (SMEs and HEIs in Innovation Partnerships) (reference: EACEA/A2/UHB/CL 554187) and the PERTEST project (TIN2013-46928-C3-1-R).

References

1. B. N. Nguyen, B. Robbins, I. Banerjee, and A. M. Memon, Guitar: an innovative tool for automated testing of GUI-driven software, Journal of Automated Software Engineering (ASE), vol. 21(1), pp. 65-105, 2014 (DOI:10.1007/s10515-013-0128-9)
2. T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: Using gui screenshots for search and automation", In Proceedings of the 22nd annual ACM Symposium on User Interface Software and Technology (UIST), pp. 183-192, New York, NY, USA, 2009 (DOI:10.1145/1622176.1622213)
3. E. Alegroth, M. Nass, and H. Olsson, "JAutomate: A tool for system- and acceptance- test automation", IEEE sixth international conference on Software testing, Verification and Validation (ICST), pp. 439-446, March 2013 (DOI:10.1109/ICST.2013.61)
4. S. Bauersfeld, and T.E.J. Vos, "Guitest: a Java library for fully automated GUI robustness testing", In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 330-333. New York, NY, USA, 2012 (DOI:10.1145/2351676.2351739)
5. S. Bauersfeld, and T.E.J. Vos, "A reinforcement learning approach to automated GUI robustness testing", In fast abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE), pp. 7-12, 2012.
6. S. Bauersfeld, and T.E.J. Vos, "User interface level testing with TESTAR; what about more sophisticated action specification and selection?", In Proceedings of the 7th edition of the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATToSE), L Aquila, Italy, July 2014, CEUR (in press).
7. S. Bauersfeld, S. Wappler, and J. Wegener, "A Metaheuristic Approach to Test Sequence Generation for Applications with a GUI", In Proceedings of the Third International Symposium on Search Based Software Engineering (SSBSE), pp. 10-12, Szeged, Hungary, September, 2011 (DOI:10.1007/978-3-642-23716-4)
8. S. McMaster, and A. Memon, "Call-stack coverage for gui test suite reduction", In IEEE Transactions on Software Engineering (TSE), vol. 34, pp. 99-115, 2008.
9. T.E.J. Vos, P. Kruse, N. Condori-Fernández, S. Bauersfeld, and J. Wegener, "TESTAR: Tool Support for Test Automation at the User Interface Level", In Proceedings of the International Journal of Information System Modelling and Design (IJISMD), 2015 (in press).

[3] http://www.clikeen.com
[4] https://www.indenova.com
[5] http://www.sopra