Automated Testing of Web Applications with TESTAR

Lessons Learned Testing the Odoo Tool

Francisco Almenar, Anna I. Esparcia-Alcázar(✉), Mirella Martínez, and Urko Rueda

Research Center on Software Production Methods (PROS), Universitat Politècnica de València, Camino de vera s/n, 46022 Valencia, Spain
{falmenar,aesparcia,mmartinez,urueda}@pros.upv.es
http://www.testar.org

Abstract. The TESTAR tool was originally conceived to perform automated testing of desktop applications via their Graphical User Interface (GUI). Starting from the premise that source code is not available, TESTAR automatically selects actions based only on information derived from the GUI and in this way generates test sequences on the fly. In this work we extend its use to web applications and carry out experiments using the Odoo open source management software as the testing object. We also introduce novel metrics to evaluate the performance of the testing with TESTAR, which are valid even when access to the source code is not available and testing is only possible via the GUI. We compare results obtained for two types of action selection mechanisms, based on random choice and $\mathcal{Q}$-learning with different parameter settings. Statistical analysis shows the superiority of the latter provided an adequate choice of parameters; furthermore, the results point to interesting areas for improvement.

Keywords: Automated GUI testing · Testing metrics · Testing web applications · $\mathcal{Q}$-learning

1 Introduction

TESTAR is an automated tool that performs testing via the GUI, using the operating system's Accessibility API to recognise GUI controls and their properties, and enabling programmatic interaction with them. It derives sets of possible actions for each state that the GUI is in and selects and executes appropriate ones, thus creating a test sequence on the fly. In previous work we have shown how TESTAR has been successfully applied to various commercial desktop applications [1,2,4,6], allowing automated testing of not just the GUI but of all the functionality that is accessible via the GUI, including e.g. databases.

In this work we report the first application of TESTAR to test a web application, namely the Odoo open source enterprise resource planning (ERP) system.

F. Sarro and K. Deb (Eds.): SSBSE 2016, LNCS 9962, pp. 218–223, 2016.
DOI: 10.1007/978-3-319-47106-8_15

Testing web applications poses challenges that differ from those of desktop applications. For instance, web latency must be taken into account. Hence, the test automation tool must wait for the GUI to react before executing the next action. Also, we must avoid testing the *browser* rather than the application; for instance, we must filter out the search bar or the bookmarks.

We run experiments in three phases or iterations, refining the process after each phase. We used $\mathcal{Q}$-learning with different parameter combinations as the action selection mechanism, and compare them using random action selection as a baseline. For the comparison we have introduced four new metrics that evaluate the quality of the testing; these metrics take into account that the source code of the software under test (SUT) is not available.

The rest of this paper is structured as follows. Section 2 explains the two main decisions taken by the human tester when testing with TESTAR, namely the action selection mechanism and the testing protocol. Section 3 introduces the metrics used for quality assessment of the testing procedure. Section 4 summarises the experimental set up, the results obtained and the statistical analysis carried out; it also highlights the problems encountered. Finally, in Sect. 5 we present some conclusions and outline areas for future work.

2 TESTAR Settings

The two main inputs for the human tester in TESTAR are the choice of an action selection mechanism and the protocol. We briefly describe these below.

Action selection. We have employed the *$\mathcal{Q}$-learning* algorithm to guide the action selection process. $\mathcal{Q}$-learning is a model-free reinforcement learning technique in which an agent, at a state S, must choose one among a set of actions A available at that state. By performing an action $a \in A$, the agent can move from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score which measures the utility of executing a given action in a given state). The goal of the agent is to maximize its total reward, since it allows the algorithm to look ahead when choosing actions to execute. It does this by learning which action is optimal for each state. The action that is optimal for each state is the action that has the highest long-term reward.

Our version of the $\mathcal{Q}$-learning algorithm is governed by two parameters: $maxReward$ and $discount$. Depending on how these are chosen the algorithm will promote exploration or exploitation of the search space. The $maxReward$ parameter determines the initial reward unexplored actions have; so, a high value biases the search towards executing unexplored actions. On the other hand, $discount$ establishes how the reward of an action decreases after being executed. Small $discount$ values decrease the reward faster and vice versa.

TESTAR protocol. A TESTAR custom protocol is a Java class that allows extending the basic functionality in order to implement complex action sets,

specific filters and sophisticated oracles. Successive iterations allow the human tester to observe the problems encountered in the testing process and improve the protocol. In this work, three such iterations were carried out.

3 Metrics

Finding appropriate metrics for assessing the quality of the testing has been a long standing issue. For instance, [3] defines a number of metrics for GUI testing, but these imply having access to the code of the software under test (SUT); one of the strengths of TESTAR is precisely not relying on the assumption that this is the case. However, this also implies that specific metrics must be defined. In this work they were chosen as follows:

- **Abstract states.** This metric refers to the number of different states, or windows in the GUI, that are visited in the course of an execution.
- **Longest path.** Any automated testing tool must ensure the deepest parts of the GUI are tested. To measure whether the tool has just stayed on the surface or it has reached deeper, we define the longest path as the longest sequence of non-repeated (i.e. excluding loops) consecutive states visited.
- **Minimum and maximum coverage per state.** We define the *state coverage* as the rate of executed over total available actions in a given state/window; the metrics are the highest and lowest such values across all windows. This allows us to know to what extent actions pertaining to states were explored.

A consequence of not having access to the source code is that the metrics given above can be used to compare the efficiency of different testing methods, but not to assess the overall goodness of a method in isolation, because we do not know the global optima for each metric; for instance, we cannot know exactly how many different states there are.

4 Experiments and Results

4.1 Odoo - The Software Under Test (SUT)

Odoo[1] is an open source Enterprise Resource Planning software consisting of several enterprise management applications that can be installed or not depending on the user needs. It can be used to create websites, manage human resource (HR), finance, sales, projects and others. Odoo has a client-server architecture and uses a PostgreSQL database as a management system. Once deployed, we installed the mail, calendar, contacts, sales, inventory and project applications in order to test a wide number of options.

[1] See https://github.com/odoo/odoo for Odoo's *git* repository and issue tracker, including a manual with instructions on how to deploy the server and its requirements.

4.2 Procedure

In order to test Odoo with TESTAR a server version of Odoo must first be deployed[2]. Then TESTAR must be configured by supplying the URL that accesses the Odoo client and the browser that will be used to launch it. Next, we run TESTAR in *spy mode*; this uncovers possible problems with items that may not be detected well, such as emergent windows. In addition, it helps detecting undesired actions that might be performed by TESTAR that may bring problems such as involuntary file deletion. A number of parameters must also be set up, which are given in Table 1. With these settings and a first version of the TESTAR *protocol*[3] we carried out three iterations of the testing process, improving the protocol each time so as to remove the problems encountered.

Table 1. Experimental set up. We carried out three iterations involving the five sets. After each iteration the results obtained were used to refine the TESTAR protocol so as to better adapt it to the application.

Set	Max. actions per run	Runs	Action selection algorithm	Parameters	
				maxReward	*discount*
Q1	1000	30	$\mathcal{Q}$-learning	1	0.20
Q20	1000	30	$\mathcal{Q}$-learning	20	0.20
Q99	1000	30	$\mathcal{Q}$-learning	99	0.50
Q10M	1000	30	$\mathcal{Q}$-learning	9999999	0.95
RND	1000	30	random	N/A	N/A

4.3 Statistical Analysis

We run the Kruskal-Wallis non parametric test on the results for the five sets. In iteration 3 the test shows that all the metrics have significant differences among the sets. Running pair-wise comparisons confirms this finding; results for all sets are given in Fig. 1, which shows how random selection can outperform some of the other sets. This highlights the importance of an adequate choice of parameters when using $\mathcal{Q}$-learning for action selection.

4.4 Issues Encountered

Several issues arose when testing Odoo with TESTAR. The first one relates to the delays induced by network latency, which is to be expected in any web application. This can be circumvented via the TESTAR GUI, which allows the human tester to select the time to wait between actions. In addition, we have

[2] See the source install tutorial available from https://www.odoo.com/documentation/8.0/setup/install.html.

[3] For more details the reader is referred to the tutorial available from www.testar.org.

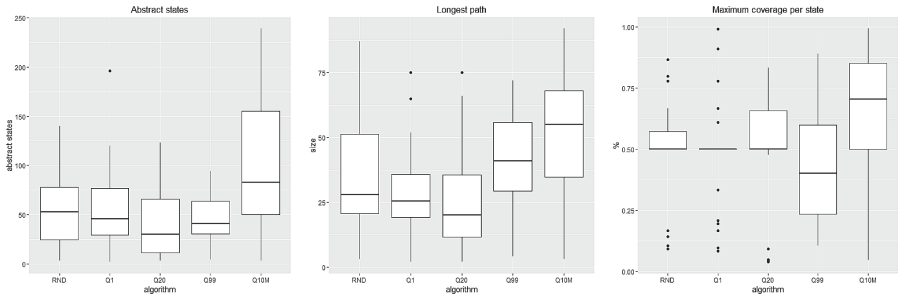

Fig. 1. Boxplots of the results obtained for 3 metrics in Iteration 3; Q10M beats the other options for these metrics, coming third in the remaining one (not shown here)

Table 2. Number of failures encountered per algorithm in the 3rd iteration. These failures coincide with known issues reported in https://github.com/odoo/odoo/issues

Set	Total failures	Unique failures
Q10M	3	1
Q99	0	0
Q20	6	2
Q1	2	1
RND	1	1

found that Odoo can display confirmation questions in the form of emerging windows that are not detected as a part of SUT by the accessibility API provided by Microsoft. This causes TESTAR to fail as it tries to find the SUT but is unable to, because the emerging window is in the foreground. Also, interactions coded via the CSS are usually not detected by the API, causing that actions available in emerging panels get mixed with those in the windows under them, which may cause the execution of unintended actions.

5 Conclusions

We have shown here the successful application of TESTAR to the automated testing of the Odoo management software - the first systematic experimentation of the testing tool to a web application. Two strategies for action selection were implemented within TESTAR: random and $\mathcal{Q}$-learning. Four metrics were defined in order to evaluate the performance. Statistical analysis reveals the superiority of the $\mathcal{Q}$-learning-based method, provided the parameters of the algorithm have been properly selected.

One metric we have not considered in the statistical analysis due to its low occurrence is the number of failures encountered, shown in Table 2.

Here we can see that although Q20 did not perform so well in the other metrics, it does on the other hand find the higher number of failures (which involve

stopping the execution and hence having a lesser chance of increasing the value of other metrics); this must also be taken into account when evaluating the different algorithms.

Further work will involve exploring three areas. One is related to the improvement of the metrics; for instance [5] refers to the lack of correlation between coverage and faults found, so we need to investigate metrics that are closer to the latter.

We will also study the possible interest of replacing the current accessibility API with a more suitable one that better supports dynamic webs. In particular, we will look at the open source tool Selenium; we think its API Selenium-WebDriver, www.seleniumhq.org, can help us fix the current problems we have found when applying TESTAR to web testing. Finally, we will introduce new, more complex, metaheuristics for action selection, as a substitute for the relatively simple $\mathcal{Q}$-learning algorithm.

Acknowledgments. This work was partially funded by projects **SHIP** (*SMEs and HEIs in Innovation Partnerships*, ref: EACEA/A2/UHB/CL 554187) and **PERTEST** (TIN2013-46928-C3-1-R).

References

1. Bauersfeld, S., de Rojas, A., Vos, T.: Evaluating rogue user testing in industry: an experience report. In: 2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), pp. 1–10, May 2014
2. Bauersfeld, S., Vos, T.E.J., Condori-Fernández, N., Bagnato, A., Brosse, E.: Evaluating the TESTAR tool in an industrial case study. In: 2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2014, Torino, Italy, p. 4, 18–19 September 2014
3. Memon, A.M., Soffa, M.L., Pollack, M.E.: Coverage criteria for GUI testing. In: Proceedings of ESEC/FSE 2001, pp. 256–267 (2001)
4. Rueda, U., Vos, T.E.J., Almenar, F., Martínez, M.O., Esparcia-Alcázar, A.I.: TESTAR: from academic prototype towards an industry-ready tool for automated testing at the user interface level. In: Canos, J.H., Gonzalez Harbour, M. (eds.) Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015), pp. 236–245 (2015)
5. Schwartz, A., Hetzel, M.: The impact of fault type on the relationship between code coverage and fault detection. In: Proceedings of the 11th International Workshop on Automation of Software Test, AST 2016, pp. 29–35. ACM, New York (2016). http://doi.acm.org/10.1145/2896921.2896926
6. Vos, T.E.J., Kruse, P.M., Condori-Fernández, N., Bauersfeld, S., Wegener, J.: TESTAR: tool support for test automation at the user interface level. IJISMD **6**(3), 46–83 (2015). http://dx.doi.org/10.4018/IJISMD.2015070103