Automated Localisation Testing in Industry with Test*

Mireilla Martinez[1], Anna I. Esparcia[1], Urko Rueda[1](✉), Tanja E.J. Vos[2], and Carlos Ortega[3]

[1] Universidad Politecnica de Valencia, Camino de vera s/n, Valencia, Spain
{mimarmu1,aesparcia,urueda,tvos}@pros.upv.es
[2] Open Universiteit, Valkerburgerweg 177, Heerlen, The Netherlands
tanja.vos@ou.nl
[3] Indenova, Carrer Dels Traginers 14, Valencia, Spain
cortega@indenova.com
http://www.testar.org

Abstract. Test* is a testing tool that automatically and dynamically generates, executes and verifies test sequences based on a tree model that is derived from the software User Interface through assistive technologies. Test* is an academic prototype that we continuously try to transfer to companies to get feedback about its applicability. In this paper we report on one of these short experiences of using Test* in industry at the Valencian company Indenova. We applied the tool to check the localisation quality of a secure web platform that encapsulates a set of applications as services.

Keywords: Automated testing · Localisation · Technology transfer

1 Introduction

In previous work [4] we have presented an approach to automated testing of software applications from their User Interface (UI). Test*[1] automatically and dynamically generates test sequences which are executed and verified to reveal quality issues of the software under test. The tool is based on a tree model that is derived from the UI through the Operating System' Accessibility API[2]. From that API we can get access to the set of widgets that compose the UI of the target application (e.g. buttons, text-fields, menu bars) and the properties of each widget that characterise their appearance in the screen (e.g. screen position, size, whether it is enabled or not). From the UI model Test* is able to compute

[1] Previously known as TESTAR or Testar, and available as open source at http://www.testar.org.

[2] https://msdn.microsoft.com/en-us/library/windows/desktop/ff486375(v=vs.85).aspx.

© IFIP International Federation for Information Processing 2016
Published by Springer International Publishing AG 2016. All Rights Reserved
F. Wotawa et al. (Eds.): ICTSS 2016, LNCS 9976, pp. 241–248, 2016.
DOI: 10.1007/978-3-319-47443-4_17

a set of feasible actions (user events like left clicks and typing texts) to automate the interaction, so do the testing, with the software interface. No test cases are recorded and the tree model is dynamically inferred for every state[3], this implies that tests will run even when the GUI changes. This reduces the maintenance problem that threatens other GUI testing techniques like Capture and Replay [3] or Visual testing [1].

The Test* tool has been developed in the context of the EU FITTEST project that finished in 2014. First, it was evaluated in experimental conditions using different real and complex software applications like MS Office suite (running it 48 hours we detected 14 crash sequences). Subsequently, and with the purpose of getting a better understanding about the applicability of the tool in an industrial environment, we continuously try to apply Test* in companies to get feedback about its applicability and help companies to obtain solutions to the problems they face. In [5] results are described of transferring and evaluating the tool within 3 different companies on 2 desktop applications and one web application. In this paper we report on yet another short experience of using Test* in industry at the Valencian company Indenova[4].

2 Test*

To automate test generation, execution and verification, Test* performs the steps as is shown in Fig. 1: (1) start the SUT (System Under Test); (2) obtain the GUI's *State* (a widget tree[5]); (3) derive a set of sensible actions that a user could execute in a specific SUT's state (i.e. clicks, text inputs, mouse gestures); (4) select one of these actions (random or using some search-based optimisation criteria); (5) execute the selected action (through Java Robot[6] class); (6) apply the available oracles to check (in)validness of the new UI state. If a fault is found, stop the SUT (7) and save a re-playable sequence of the test that found the fault. If not, keep on testing if more actions are desired within the test sequence.

Using Test*, you can start testing immediately from the UI without the traditional requirement of specifying test cases, which are commonly provided manually with some degree of tool support. Based on the information gathered from the Accessibility API tests are generated by selecting an action to execute in the UI (e.g. left click a button with the title "Ok"). The action selection mechanism mainly drives how the test cases are generated, which can be performed randomly (select any suitable action for the current UI) or using a more advanced approach to increase the effectiveness of tests like the work in [2]. Without specifying anything, Test* can detect the violation of general-purpose system requirements through implicit oracles like those stating that the

[3] The Graphical User Interface at a particular time.

[4] www.indenova.com/.

[5] Test* uses the Operating System's Accessibility API, which has the capability to detect and expose a GUI's widgets, and their corresponding properties like: display position, widget size, title, description, etc.

[6] https://docs.oracle.com/javase/8/docs/api/java/awt/Robot.html.

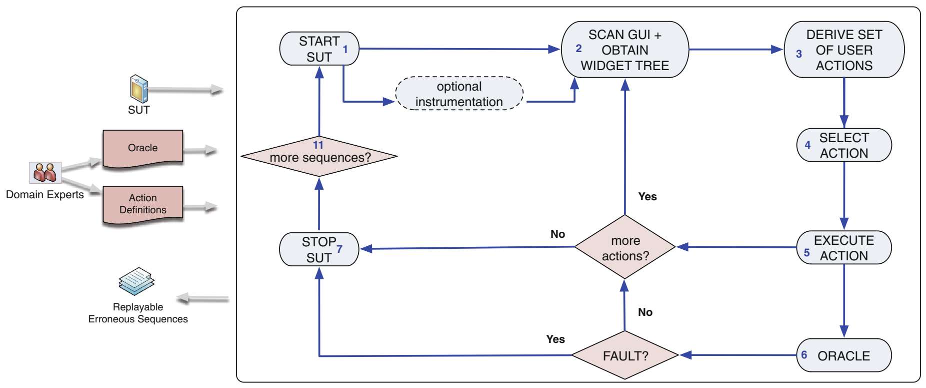

Fig. 1. Test* testing flow

SUT should not crash, the SUT should not find itself in an unresponsive state (freeze) and the GUI state should not contain any widget with suspicious words like *error*, *problem*, *exception*, etc.

This is a very attractive feature for companies because it enables them to start testing immediately and refine the tests as we go.

3 Indenova and the SUT eSigna

Indenova is a Valencian ICT company that provides ERP (Enterprise Resource Planning) solutions for companies. Their initial clients are based in Spain. But throughout the years, Indenova has gained new clients in Latin America. Testing at Indenova is mainly manual and basically done at the system acceptance test level. Written requirements are used for the design of system test suites. They would like to have more tests automated, but currently in the company there is a lack of time and people with knowledge about test automation.

Becoming aware of Test* Indenova is very interested to see how they can start test automation, so they provided access to their eSigna product. It is a web platform that securely integrates and provides access to applications as services enabling users to perform specific processes inside their organisations. Thus, eSigna is a base component in which concrete services can be plugged-in as required by each particular project. Those services are independent from each other, but they are interconnected to share information in real time.

4 The Industrial Experience

During the investigation we have measured the following *effectiveness and efficiency* aspects of Test* for testing the localisation quality of eSigna:

1. Number of failures (wrongly translated words) observed after executing Test* on eSigna

2. Time needed to set-up the test environment and get everything running
3. Lines Of Code (LOC) and time needed for UI actions definition, oracles design and stopping criteria setup.
4. Time for running Test* to reveal localisation issues on eSigna.

The project has been carried out in a fashion that allowed us to perform iterative development of Test*. The process included the following steps which were repeated several times to yield the final setup:

1. Planning: Implementation of Test Environment, consisting of planning and implementing the technical details of the test environment for Test*, as well as the anticipating and identifying potential fault patterns in the Error Definition.
2. Implementation: Consisting of implementing the Test* protocol consisting of: Oracles to implement the detection of the errors defined in the previous step; Action Definition to define the action set from which Test* selects; and the Implementation of stopping criteria that determine when sufficient testing has been done by Test*.
3. Testing and Evaluation: Run the tests.

4.1 Planning the Testing: What Do We Want to Test

One of the immediate problems that Indenova faces with eSigna, localisation to Latin America community, fits perfectly with Test* capabilities. The tool enables not only to detect stability problems for free, like crashes and exceptions, but it also allows to systematically analyse the UI in the search of wrongly translated texts.

As previously indicated, the initial clients from Indenova were from Spain, but gradually they have expanded to Spanish speaking South American countries. One of the problem encountered is that there are differences between the Castilian Spanish spoken in Spain and the different Latin American Spanish. Although it is not a problem of not being able to understand what is meant, some of the clients from Columbia and Peru just have complained about the usage of Castilian words. For example:

English	Castilian Spanish	Latin American Spanish
Mobile phone	Móvil	Celular
Holiday	Festivo	Feriado
Computer	Ordenador	Computadora

Since the implementation is not based on dictionaries and the Castilian Spanish is hard-coded, there is no other way than test the application to find the words that need to be changed for the other countries. This is a tedious and boring job.

4.2 Implementing the Test* protocol

Test* has the flexibility to adapt its default behaviour for specific needs. We will describe next how did we setup the tool to automatically verify localisation problems on *eSigna* product. We refer to the steps in the testing flow (Fig. 1):

1. START SUT - Set *eSigna* activation: it will tell Test* how to start/run the application. Being a web application, it consists of a command line *BROWSER URL* where *BROWSER* is the path and executable of an available web browser (i.e. Internet Explorer) and *URL* the entry point for the *eSigna* web application.
2. DERIVE SET OF USER ACTIONS - Set suitable actions: from the space of candidate actions that the user could perform over the product UI we are interested in (1) actions that will enable an automatic login to *eSigna* and (2) actions which are not interesting for our localisation verifying objective (i.e. web browser actions, a logout button, an administration panel in *eSigna*, etc.)
3. SELECT ACTION - Set test algorithm: the tool provides several strategies to generate a test (e.g. picking a random action each time). We are interested in exercising as much of the UI as possible to verify any potential localisation issues. We selected the Q-Learning algorithm from previous work [2].
4. ORACLE - Set localisation oracles: verifying the localisation correctness of *eSigna* for a target language can be straightforward performed by defining a list of taboo words that should not appear in the UI. This list can be easily defined in Test* UI through Java regular expressions (i.e. .*[mM][óo]vil.*—.*[fF]estivo.*—.*[oO]rdenador.*).
5. FAULT?/more actions? - Set the stopping criteria: the tool offers different approaches to stop a test, including a fixed time for execution, a fixed length for the number of UI actions to be executed or a self-made stopping criteria through a Java based protocol class (check next point). We made use of the last option to establish that we have tested enough when no more new UI is being exercised by our tests.
6. Advanced setup editing the tool' test protocol: Test* provides a Java class composed of a method for each task in the testing cycle presented in Fig. 1. Concretely, we implemented the automated login inside the task *START SUT*, non-interesting actions filtering inside the task *DERIVE SET OF USER ACTIONS* and the stopping criteria in the *more actions?* check point.

Once Test* was setup for automated localisation verification we just had to wait for the tool test reports. Following the testing flow of Test* it would first activate eSigna, perform an automated login and repeat a cycle of <*select and execute action*, *verify localisation problems*, *check stopping criteria*>.

4.3 Testing and Evaluation

Our context multilingual scenario consisted of one target language, *Latin American Spanish*, as this was the first concern on eSigna testing with Test*. We account in Table 1 (LOC = Lines Of Code; time in minutes) for metrics that measure the effort required for our solution on automated verification of localisation issues.

Setting up Test* for eSigna is an easy process that consists on providing the command line that would activate the product. Actions configuration would require some effort though as we would like Test* to perform automated tests without user intervention. Thus, we first need to analyse eSigna authentication process to provide the proper actions once the product has been activated. Additionally, we wanted to maintain our tests in relevant UI parts of eSigna, for example disabling/filtering non interesting actions like closing the browser, log-out of eSigna, etc. Yet, 35 lines of code and 10 min were enough for Test* to perform automated tests over eSigna. We acknowledge that future enhancements on Test* would enable a more efficient configuration of actions (we used version 1.1a of the tool).

Table 1. Efficiency

Setup environment	Actions		Oracles		Stop criteria		Test run
Time	LOC	Time	LOC	Time	LOC	Time	Time
1	35	10	0	5	9	2	>60

Oracles did not require any lines of code, but just a regular expression with the full list of unwanted localised product words (e.g. Móvil, Festivo, Ordenador). From Indenova, we acquired a full list of more than 30 words that the Latin American community had issued to the company in the past. This list contained wrongly used Spanish words (e.g. Móvil instead of Celular). Thus, we defined the regular expression for the words in the list that would enable Test* to check the quality of eSigna with respect to its localisation.

The stopping criteria was easily implemented taking into account how much of the UI was being exercised (user events) by the test. We named this *UI space exploration*, where the full UI space is composed of every particular and different[7] screen window that the application might show to the user. We forced to stop the tests when no more UI space was being explored by the last 100 executed actions. In other words, when there was no new UI window already exercised by the test.

Finally, using the configuration just described we let Test* run a test for almost an hour. The tool was able to report localisation issues on 2 words from the list in the first 5 min of execution. Both words were confirmed by the company as they were already aware that they were incorrectly localised. Other words were not reported, but Indenova indicated that such words were not part of

[7] Two windows are considered different if there is almost (a) one widget not present in both windows or (b) a widget with different properties (e.g. text or size)) in each window.

the product. We also observed that new UI space was explored after an hour of execution, which could reveal additional issues in the localised product. We expect a direct relation between the UI space exploration (coverage) and the effectiveness achieved on localisation verification of software products, but this should be analysed in a further study.

We would like to make some final considerations. We acknowledge that the verification of localisation issues has been traditionally performed using other alternatives, for example through text finding utilities like *grep* command on Linux hosts or a general purpose text editor with file searching features. A main efficiency problem of these approaches is that we cannot safely distinguish between texts used in the source code, and text that is mainly appearing in the UI: users will never complain on texts that they do not see in the User Interface.

Moreover, more complex products like eSigna might make more difficult to check localisation issues when the source code is spread over several (virtual) machines (perhaps targeting different operative systems), databases, or even legacy systems. In this sense, Test* provides a central setup place from which products localisation can be verified.

Additionally, we decided to stop our tests after an hour of execution tough we could have allowed it to run for longer. If checking localisation issues is performed manually by a human (interacting with the UI) then Test* is helpful once it is setup correctly, as it can operate without human supervision. Although Test* is a general purpose testing tool, we have presented how it can be used to verify that a software product has the quality levels expected by a company like inDenova.

5 Conclusions and Further Work

We have presented a short experience of transferring an academic prototype from the university to the industry, for testing software applications at the UI level. Indenova is a Valencian ICT company that provides ERP solutions to other companies. We applied the prototype Test* for testing localisation issues in eSigna product, which targets the Latin American countries. eSigna is a secure web platform composed of integrated web services.

The automation level achieved by the prototype and its potential for testing software products made Indenova consider the integration of Test* into their testing processes. They used the prototype for performing smoke testing, which would provide early feedback of the quality of developed product versions.

As further work, we will improve localisation testing in the prototype by including dictionaries. We would also like to further investigate the effectiveness of the presented localisation testing solution, concretely its relation to the test' UI space coverage.

Acknowledgement. This work was partly funded by the SHIP project (EACEA/A2/UHB/CL 554187) and the PERTEST project (TIN2013-46928-C3-1-R). Test* was funded by the EC within the context of the FITTEST project, ICT-2009.1.2 no. 257574 (2012–2015).

References

1. Alegroth, E., Nass, M., Olsson, H.H.: Jautomate: a tool for system- and acceptance-test automation. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST), pp. 439–446, March 2013
2. Bauersfeld, S., Vos, T.: A reinforcement learning approach to automated GUI robustness testing. In: Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE 2012), pp. 7–12. IEEE (2012)
3. Nguyen, B.N., Robbins, B., Banerjee, I., Memon, A.M.: GUITAR: an innovative tool for automated testing of GUI-driven software. Autom. Softw. Eng. **21**(1), 65–105 (2014)
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. 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. Int. J. Inf. Syst. Model. Des. **6**(3), 46–83 (2015)