TESTAR: Tool Support for Test Automation at the User Interface Level

Tanja E.J. Vos, Universidad Politecnica de Valencia, Valencia, Spain
Peter M. Kruse, Berner and Mattner Systemtechnik GmbH, Berlin, Germany
Nelly Condori-Fernández, Vrije Universiteit van Amsterdam, Amsterdam, The Netherlands
Sebastian Bauersfeld, Universidad Politecnica de Valencia, Valencia, Spain
Joachim Wegener, Berner and Mattner Systemtechnik GmbH, Berlin, Germany

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. In earlier works the authors presented the TESTAR tool, an automated approach to testing 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. In this paper they report on their experiences obtained when transferring TESTAR in three different industrial contexts with decreasing involvement of the TESTAR developers and increasing participation of the companies when deploying and using TESTAR during testing. The studies were successful in that they reached practice impact, research impact and give insight into ways to do innovation transfer and defines a possible strategy for taking automated testing tools into the market.

Keywords: Capture and Replay Tools, Graphical User Interface (GUI), Input Sequences, TESTAR

INTRODUCTION

Testing software applications at the Graphical User Interface (GUI) level is a very important testing phase to ensure realistic tests. The GUI represents a central juncture in the application under test from where all the functionality is accessed. Contrary to unit or interface tests, where components are operated in isolation, GUI testing means operating the application as a whole, i.e. the system’s components are tested in conjunction. This way, it is not only possible to discover flaws within single modules but also faults arising from erroneous or inefficient inter-component communication. However, it is difficult to test applications thoroughly
through their GUI, especially because GUIs are designed to be operated by humans, not machines. Moreover, they are inherently non-static interfaces, subject to constant change caused by functionality updates, usability enhancements, changing requirements or altered contexts. This makes it very hard to develop and maintain test cases without resorting to time-consuming and expensive manual testing.

Capture replay (CR) tools (Singhera, Horowitz, & Shah, 2008; Nguyen, Robbins, Banerjee, & Memon, 2014) rely on the UI structure and require substantial programming skills and effort. The idea behind CR is that of a tester developing use cases and recording (capturing) the corresponding input sequences, i.e., sequences of actions like clicks, keystrokes, drag and drop operations. These sequences are then replayed on the UI to serve as regression tests for new product releases. These tools implicitly make the assumption that the UI structure remains stable during software evolution and that such structure can be used effectively to anchor the UI interactions expressed in the test cases. Consequently, when test cases are evolved, adapted, parametrized or generalized to new scenarios, the maintenance cost can get real high and the competence required from programmers can become an obstacle (Leotta, Clerissi, Ricca, & Spadaro, 2013). This has severe ramifications for the practice of testing: instead of creating new test cases to find new faults, testers struggle with repairing old ones, in order to maintain the test suite (Grechanik, Xie, & Fu, 2009). For software applications, the UIs change all the time and hence make the CR method infeasible. Furthermore, new generations of applications are increasingly able to adapt their own layout to a target screen (e.g., small mobile or large desktop) and its user profile (e.g., different screen layouts for novice vs. advanced users). Consequently, CR tools are sometimes referred to as Shelfware and CR tool vendors are accused of trying to sell them as the silver bullet (Kaner, 2002). Due to this maintenance problem, companies return to manual regression testing which results in less testing being done and faults that still appear to the users.

Visual testing tools (Yeh, Chang, & Miller, 2009; Alegroth, Nass, & Olsson, 2013) take advantage of image processing algorithms to simulate the operations carried out manually by testers on the UI making UI testing as simple as that carried out step by step by humans. These visual testing approaches simplify the work of testers as compared to the structural testing approaches. However, they do rely on the stability of the graphical appearance of the UI, and require substantial computational resources for image processing. Changes to the application often also involve changes to the UI, hence also threatening the visual approach. Visual clues in the UI might mislead the image recognizer of visual testing tools, which are correspondingly subject to false positives (wrong UI element identification) and false negatives (missed UI elements).

In this work we present a Model-Based Testing approach implemented in a tool we call TESTAR (Test Automation at the user inteRface level) (Bauersfeld & Vos, 2012a, 2012b; Bauersfeld, de Rojas, & Vos, 2014; Bauersfeld, Vos, Condori-Fernández, Bagnato, & Brosse, 2014; Vos, 2014). (TESTAR was formally known as the Rogue User.) TESTAR automatically generates and executes test cases based on a tree model that is automatically derived from the UI through the accessibility API. Since this structure is built automatically during testing, the UI is not assumed to be fixed and tests still run even though the UI evolves, which reduces the maintenance problem that threatens the approaches mentioned earlier. This paper presents the tool, together with evaluative studies done in industry to evaluate the approach (see Table 1 for an overview of the industrial studies presented).

A first early study is done at Berner & Mattner (B&M), a German specialist in systems engineering, development and testing of complex software, electronic and mechanical systems. Their software TESTONA implements the classification tree method (Grochtmann & Grimm, 1993). The product, formerly known as Classification Tree Editor (Lehmann & We-