UI-Test: A Model-Based Framework for Visual UI Testing– Qualitative and Quantitative Evaluation

Bryan Alba(✉), Maria Fernanda Granda(✉), and Otto Parra(✉)

Computer Science Department, Universidad de Cuenca, Av. 12 de Abril s/n, Cuenca, Ecuador
{bryan.albas,fernanda.granda,otto.parra}@ucuenca.edu.ec

Abstract. During the testing stage in the software development life cycle, developers can take advantage of combining the requirements specification with the testing specification. This will allow the specification of the tests to require less manual effort, since, they can be defined or generated automatically from the requirements specification. The requirements specification will thus be based on a more structured language, gaining in quality and reducing ambiguity, inconsistency, and inaccuracy. In this research work, the UI-Test model-based methodological framework and its tool support are proposed. Both of these can generate evidence based on the specification of agile user stories that are used in the validation of the functional requirements that must be included in the final version of the user interfaces of the developed software. Our proposal makes use of two model transformations to obtain the test scripts from user stories that will be applied in the process using SikuliX for automated visual UI testing. The results of the empirical evaluation of the effectiveness and user experience of the framework and its tool support suggest that the UI-Test tool can benefit testers by confirming that the actions proposed in the user stories can be run on the UIs.

Keywords: UI-Test · Visual UI testing · User stories · Test scripts · Model-based testing · Testing framework

1 Introduction

In the field of Software Engineering, agile software development methodologies have gained momentum because of their benefits to software developers in the software development life cycle (SDLC) [1]. In this context, organizations are increasingly deploying agile methodologies in their software development projects in order to efficiently produce higher quality software in a shorter period of time. This methodology enables a software developer to be more flexible and responsive to changing environments and customer demands. However, it has been pointed out that agile methods largely ignore issues of designing the user interface (UI). To some extent this is understandable: agile processes are highly iterative and incremental, while traditional approaches to user interface design have been big-bang with heavy reliance on upfront design [2].

R. Ali et al. (Eds.): ENASE 2021, CCIS 1556, pp. 328–355, 2022.
https://doi.org/10.1007/978-3-030-96648-5_15

The Agile Model-Driven Development (AMDD) method, which combine Model-driven Development (MDD) and Agile Development [3] has many benefits for developing suitable solutions by means of a better understanding of the requirements [4]. In the agile method there is no requirements freezing, they are obtained using a face-to-face conversation with the stakeholders, and the solutions change over time based on the requirements [5]. In this context, developers have a problem: how to test the software to seek evidence that the requirements specified by the stakeholders are satisfied by the software. Our proposal considers visual testing for both user interface types: Graphical User Interface (GUI) and Web User Interfaces (WUI) and it checks if the requirements previously defined in the software development life cycle have been included in the implemented software product. However, designing and executing test cases is very time-consuming and error-prone when done manually and frequent changes in the requirements reduce the reusability of these manually written test cases.

In the software development life cycle, there are several available techniques for requirements testing when the software engineers require them. In our research work, we use the requirements specification obtained from the stakeholders in order to describe user stories [2]. We then derive a task-based test model using ConcurTaskTree (CTT) [6] by applying a parsing process to the user stories to describe test scenarios with abstract test cases. The next step generates concrete test cases by means of a semiautomatic process in test scenarios. At this point the test cases are transformed into test script by applying model transformations and SikuliX language[1], which is a standardized test language for UI-based testing.

Two research approaches were used to evaluate the proposed UI-Test tool: a qualitative evaluation to measure the user experience quality of the proposal with seventeen subjects, and a quantitative evaluation to measure the effectiveness of the test cases generated by the tool.

The contributions of this paper are: (1) an extension of the UI-Test, a model-based testing framework described in [7] and its tool support, (2) an empirical evaluation of the framework which was applied in order to assess effectiveness and user experience in two scenarios: using (i) GUI and (ii) WUI, described in this paper.

The rest of this paper is structured in 5 sections. Section 2 presents the background of related topics and also introduces related works. Section 3 includes the description of the UI-Test framework and the tool support implemented to test the proposed approach. In Sects. 4 and 5, the empirical evaluations are described. Section 6 contains a discussion of the results. Section 7 summarizes the threats to validity and Sect. 8 includes our conclusions and our plans for future work.

2 Background and Related Work

Within Agile methods, user stories are mostly used as primary requirements artefacts and units of functionality of a software project [8]. Typically, a user story includes three components (Fig. 1): (1) a short description of the user story used for planning (Who?), (2) conversations about the user story to discover the details (What?), and (3) acceptance criteria (Why?) [6].

[1] http://sikulix.com/.

Component	Describes
As a [user/stakeholder]	Who?
I want to [requirement]	What?
So that [motivation]	Why?

Fig. 1. A user story template.

ConcurTaskTree belongs to the family of hierarchical task analysis notations, the most common approach to task analysis [6]. Using ConcurTaskTree as the task modelling notation has some advantages [10]: (i) the models are at a level of abstraction familiar to user interface designers/developers; (ii) testing will follow the anticipated use of the system; and, (iii) the cost incurred in developing the oracle is much reduced.

UIs are composed of graphical objects called widgets, such as buttons, text fields, menus, etc. Users interact with these widgets (e.g. press a button) to produce an action that modifies the state of the system [8]. This type of user interface is used in this research work for UI testing. Regarding the types of faults that can occur in an UI, we consider the classification presented by Lliet Lelli et al. [9], which considers two groups: user interface faults and user interaction faults.

According to the related literature, there are three generations of automated GUI-testing [9]: the first generation relies on GUI coordinates but is not used in practice due to unfeasible maintenance costs caused by fragility to GUI change. Second generation tools instead operate against the system's GUI architecture, libraries or application programming interfaces. Whilst this approach is successfully used in practice, it does not verify the GUI's appearance and it is restricted to specific GUI technologies, programming languages and platforms. The third generation, referred to as Visual GUI Testing (VGT), is an emerging technique in industrial practice with properties that mitigate the challenges experienced with previous techniques.

Visual UI Testing is an emerging technique in industrial practice and uses tools with image recognition capabilities to interact with the bitmap layer of a system, i.e. what is shown to the user on the computer monitor. SikuliX, JAutomate, etc. are some examples of tools that apply this type of testing. In this research work we use SikuliX [7], a standardized test language for visual UI-based testing (VGT) on GUI and WUIs. Alégroth et al. [10], in their study related with challenges, problems and limitations (CPL) of VGT in practice, their main conclusion is still that VGT is a valuable and cost effective technique with equal or even better defect-finding ability than manual testing.

In the following paragraphs, we describe several works reported by the related literature in the automated testing field.

In the context of the generation of test cases from agile user stories, Rane [11] developed a tool to derive test cases from natural language requirements automatically by creating UML activity diagrams. However, their work requires the Test Scenario Description and Dictionary to execute the test case generation process. The authors developed a tool that uses NLP (natural language processing) techniques to generate functional test cases from the free-form test scenario description automatically.

Elghondakly et al. [12] proposed a requirement based testing approach for automated test generation for Waterfall and Agile models. This proposed system parses functional

and non-functional requirements to generate test paths and test cases. The paper proposes the generation of test cases from Agile user stories but does not discuss any implementation aspects such as the techniques for parsing, or the format of the user stories that are parsed. This implementation does not follow a model-based approach.

Finsterwalder, M. [13], reports how he uses automated acceptance tests for interactive graphical applications. However, according to the author, it is difficult to automate tests that involve GUI intensive interactions. To test the application in its entirety, tests should actually exercise the GUI of the application and verify that the results are correct. In extreme programming (XP), the customer writes down small user stories to capture the requirements and specifies acceptance tests. These tests are implemented and run frequently during the development process.

Tao, C. et al. [14] propose a novel approach to mobile application testing based on natural language scripting. A Java-based test script generation approach is developed to support executable test script generation based on the given natural language-based mobile app test operation script. According to the authors, a unified automation infrastructure is not offered with the existing test tools. In order to deal with the massive multiple mobile test running, there is a lack of well-defined mobile test scripting methods, so that test automation central control is needed to support behaviour-based testing or scenario-based testing at multiple levels.

Ramler et al. [15], describe the introduction of Model-based Testing (MBT) for automated GUI testing in three industry projects from different companies. Each of the projects already had automated tests for the GUI but they were considered insufficient to cover the huge number of possible scenarios in which a user can interact with the system under test (SUT). MBT was introduced to complement the existing tests and to increase the coverage with end-to-end testing via the GUI.

Kamal [16] presents a test-case generation model to build a testing suite for webpages using their HTML file. The proposed model has two branches. The first one focuses on generating test cases for each web-element individually based on its type. The other branch focuses on generating test cases based on different paths between web-elements in the same webpage.

Coppola et al. [17] provide a detailed and fine-grained taxonomy of the modifications performed on the production code of Android apps, which may trigger the necessity for maintenance of test suites. The authors believe that the fragility issue – a problem that has already been explored extensively in the field of web applications – can seriously hinder large-scale use of automated testing for Android apps.

Silva et al. [18] describe an effort to develop tool support enabling the use of task models as oracles for model-based testing of user interfaces. The subject of interest in their research is the test oracle that will be used as a measure of the implementation quality.

Our contribution is a model-based framework to apply visual UI testing with the aim of checking if all the user story requirements of a software system are included in the final version (UI) of the developed software product. For this we use a task model, a parsing process and transformations using Java and SikuliX language.

Table 1 Summarizes the related work included in this section.