# Towards a Model-Driven Testing Framework for GUI Test Cases Generation from User Stories

Maria Fernanda Granda[1][a], Otto Parra[1][b] and Bryan Alba-Sarango[1][c]
[1]*Department of Computer Science, Universidad de Cuenca, Av. 12 de Abril s/n, Cuenca, Ecuador*
*{fernanda.granda, otto.parra, bryan.albas}@ucuenca.edu.ec*

Keywords: Test Cases, User Stories, Requirements, GUI-based Testing, Model-Driven Testing.

Abstract: In the software testing stage, it is possible to benefit from combining the requirements with the testing specification activities. On the one hand, the specification of the tests will require less manual effort, since they are defined or generated automatically from the requirements specification. On the other hand, the specification of requirements itself will end up having a higher quality due to the use of a more structured language, reducing typical problems such as ambiguity, inconsistency, and inaccuracy. This research proposes a model-based framework that promotes the practice of generating test cases based on the specification of Agile user stories to validate that the functional requirements are included in the final version of the user interfaces of the developed software. To show the applicability of the approach, a specification of requirements based on user stories, a task model using ConcurTaskTree, and the Sikulix language are used to generate tests at the graphical interface level. The approach includes transformations; such as task models in test scripts. Then, these test scripts are executed by the Sikulix test automation framework.

## 1 INTRODUCTION

To react to the changing software development market in a more efficient way, the adoption of Agile development practices is gaining momentum (Kassab 2015). The Agile methodology is an iterative and incremental approach to software development, where the requirements and solutions evolve over time according to the need of the stakeholders. How to test the application to seek evidence that the functionality requested by end users or stakeholders is provided by the application now emerges as an issue. However, designing and executing test cases is very time-consuming and error-prone task when done manually and frequent changes in requirements reduce the reusability of these manually written test cases. According Latiu et al. (Latiu, Cret, and Vacariu 2013), automatic testing based on Graphical User Interfaces (GUIs) may be a good alternative because it is more accurate, reliable and efficient.

The existing methods to generate test cases from user stories have not been widely accepted in practice, because they require substantial human participation or because the results obtained have a very low accuracy. Also, not all the testers have prior knowledge of the working of the system which leads to difficulties in designing the test cases to match the requirements.

In this work, we consider the version integrated of two methodologies to develop software such as Agile and Model-driven Development. This version is called Agile Model-driven Development (AMDD) (Alfraihi, Lano, and Kolahdouz-rahimi 2018). On the one hand, Model-Driven Engineering is a well-known software development paradigm which provides many benefits to develop suitable solutions of software. On the other hand, Agile Methods are a good paradigm to gain a better understanding of requirements (Grangel and Campos 2019).

In this paper, we aim at providing an approach to accommodate the following issues:

- How to generate test cases from user stories and that they to adapt to the evolution of the requirements in an easy way?

[a] https://orcid.org/0000-0002-5125-8234
[b] https://orcid.org/0000-0003-3004-1025
[c] https://orcid.org/0000-0001-9418-9489

- How to transform the test cases into an executable script so that tester can minimize the effort to run them?
- How to simulate the interactions between the user and the GUI so that it can be tested alone?

In order to answer these questions, we are proposing a model-driven framework to generate test cases suitable for GUI-based testing from the requirements provided as user stories.

Our proposal is as follows:

a. We derived a task-based test model based on ConcurTaskTree (Paterno, Mancini, and Meniconi 1997) by parsing the user stories to describe the test scenarios with abstract test cases.
b. The concrete test cases are generated semiautomatic from the test scenarios.
c. Once the user stories are modified, a new set of test cases could be generated again.
d. At last, the test cases are transformed into test script in Sikulix [4] language that is a standardized test language for GUI-based testing.

The rest of this paper is structured in 5 sections. Section 2 introduces the related works. Section 3 presents the background about user stories, task-based test model and the language used for automating the GUI scripting. Section 4 shows how the test cases are generated. In section 5, the conclusions and future work are summarized.

## 2 RELATED WORK

In the requirements engineering field, several techniques for testing requirements had been proposed. Specifically, we consider GUI-based testing to check if the requirements previously defined in the software development life cycle have been included in the software product already implemented. In this context, we describe several works reported by related literature.

In the context of the generation of test cases from agile user stories, Rane (Rane et al. 2017) have developed a tool to derive test cases from natural language requirements automatically by creating UML activity diagrams. However, their work requires of the Test Scenario Description and Dictionary to execute the test case generation process. The authors developed a tool that uses NLP techniques to generate functional test cases from the free-form test scenario description automatically.

Elghondakly et al. (Elghondakly, Moussa, and Badr 2015) proposed a requirement based testing approach for automated test generation for Waterfall and Agile models. This proposed system would parse 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. (Finsterwalder 2001), in his job reports how he is using automated acceptance tests for interactive graphical applications. However, according 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. For each user story the customer specifies acceptance tests as well. These tests are implemented and run frequently during the development process.

Tao, C. et al. (Tao, Gao, and Wang 2017) proposes 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 the authors, a unified automation infrastructure is not offered with the existing test tools. In addition, there is lack of well-defined mobile test scripting method to deal with the massive multiple mobile test running. Therefore, test automation central control is needed to support behaviour-based testing or scenario-based testing at multiple levels.

Ramler et al. (Ramler, Klammer, and Wetzlmaier 2019), describe the introduction of 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 (Medhat Kamal, Darwish, and Elfatatry 2019) presents a test-case generation model to build a testing suite for webpages using its HTML file. The proposed model has two branches. The first one

[4] http://sikulix.com/

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.

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

# 3 BACKGROUND

## 3.1 User stories

In the software development life cycle (SDLC), the requirements elicitation is a crucial stage because functional (and no functional) requirements are defined. Interviewing to the stakeholders is a typical technique to obtain the requirements. The result of this process are the user stories which are an increasingly popular textual notation to capture requirements (Lucassen et al. 2016) in the agile software development.

The term "user stories" was coined by Beck and Andres (Beck and Andres 2004) and it refers to the description of the tasks of the users by means of a template. Figure 1 shows the elements of the template, however, the last element (SO THAT I CAN) is optional.

AS A <user role>
I WANT TO <perform some task>
SO THAT I CAN <reach some goal>

Figure 1. Template to define a user story.

Moreira (Moreira 2013) describes the hierarchy of requirements within an Agile context incorporating some concepts: themes, epics, user stories and tasks (Figure 2).

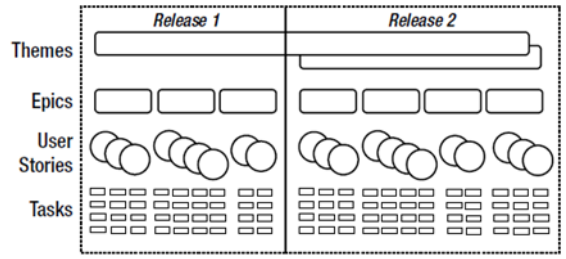


Figure 2. Hierarchy of requirements types within an Agile context (taken from (Moreira 2013).

According Moreira (Moreira 2013), themes are top-level objectives that may span multiple releases and products. Themes should be decomposed into epics that can be applied to a specific product or release. Epics are the parent of multiple user stories and are roughly equivalent to a feature or very large story that encapsulates a large piece of functionality. Tasks are the children of user stories and are equivalent to an incremental decomposition of the user story.

The acceptance criteria are an important attribute of a user story. Each user story should have its own unique set of acceptance criteria (Moreira 2013). Acceptance criteria answer the question, "How will I know when I'm done with the story?" They do this by providing functional and non-functional information that helps set boundaries for the work and establishes pass/fail criteria for testers to establish the test cases that are used to test a user story.

## 3.2 Task model

A task model is a description of the process a user takes to reach a goal in a specific domain. Task models are amongst the most commonly used models during interactive systems design.

Typically, ConcurTaskTree (CTT) (Paterno, Mancini, and Meniconi 1997) is used to describe in a graphical way the sequence of steps to do a task. In our job we will be using CTT as the task modelling notation. Figure 3 shows some task types in a CTT model.

| TYPE | DESCRIPTION |
|---|---|
| **Interaction Task** | Represents user interaction with the system. |
| **Application Task** | Represents tasks that must be performed by the system. |
| **User Task** | Represents user decision points. |
| **Abstraction Task** | Represents abstract tasks (i.e. the combination of subtasks into a higher level task) |

Figure 3. Task types in a CTT model[1].

We use the concepts of themes, epics, user stories and tasks in order to obtain the task model. For example: in the context of using a text editor such as Notepad, a theme could be "Managing documents in Notepad", an epic could be "As a user can create a document to write an essay", a user story could be "As a user I want to enter text in the document"; and finally, some tasks could be "As a user I want to type text in the document", "As a user I want to copy text in the document" and so on.

**Table 1.** A comparison of software tools for testing.

| Tool Features | AutoIt | RobotFramework | Squash | SikuliX |
|---|---|---|---|---|
| Type of license | Freeware | Open source | Commercial, a payment is required for use it | Open Source |
| Supported platform | Microsoft Windows | Operating system and application independent | Microsoft Windows | Microsoft Windows, MacOs, Linux |
| Type of applications | Desktop applications | Web testing, Swing, SWT, GUIs, databases. | Web apps, applications based on Kubernetes | Desktop and Web applications |
| Used technology | Regular expressions | Keyword and data oriented | JUnit native code, keywords-driven approach | Uses image recognition to control GUI elements. |
| Language | Visual Basic and C# | Python and Java | Jira | Python, Java and Ruby |
| Automation method | Record/playback to automate process | Acceptance-level test automation | Template-based automation | Workflow automation scripts |

## 3.3 Language for GUI scripting

In the related literature about software tools to test the different paths in the testing process, we found several alternatives, between them: AutoIt [5], RobotFramework[6], Squash[7] and SikuliX.

In order to select the tool to use in the process, we did a comparison of features of each one. The results of this comparison are included in **Table 1**. According to these results, we selected SikuliX for testing the different paths in our proposal. SikuliX automates screens tests of desktop computer running Windows, Mac or some Linux/Unix by using scripts. It uses image recognition powered by OpenCV to identify GUI components. Additionally, SikuliX is open source, it does not require any payment for its use.

# 4 PROPOSED APPROACH

This research intends to encourage and support both requirements and testing areas, by generating test scripts from user stories or at least foster the alignment of such test cases with requirements.

In this section is described the proposed approach (see Figure 4) by means of the following steps: (1) Requirements specification (i.e. user stories) that serves as a basis for the (2) test model derivation (i.e. task model). Then, (3) tests scenarios with the test cases are generated automatically by applying the algorithm for path analysis in the test model, which can be further (4) refined by the tester to add the Graphical User Interface (GUI) locators and assign values to variables. (5) The test scripts (i.e. Sikulix language) are generated automatically from the test cases. Finally, (6) these test scripts are executed against the system GUI under test generating a test report. To illustrate and discuss the suitability of the approach, we applied it on the Notepad application of Microsoft. This application was selected because this is a common and well-known application for readers, which facilitates the explanation of the approach.

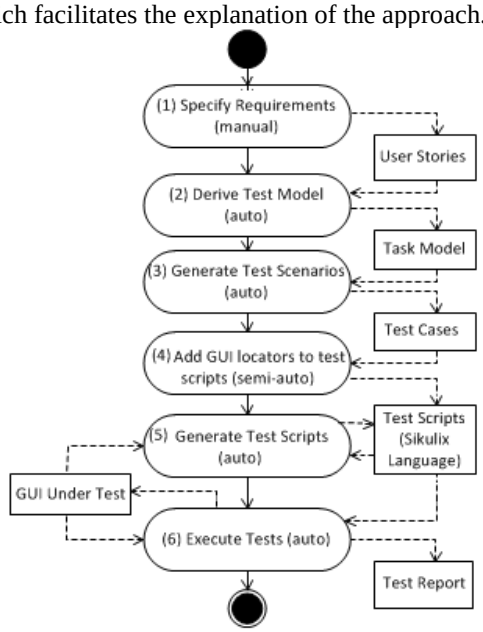


Figure 4. Proposed approach (UML activity diagram).

## 4.1 Step 1: Specifying requirements using user stories

The first task is the requirements specification using user stories that usually involves the intervention of requirements engineers, stakeholders and eventually testers. User stories follow a standard predefined format (Wautelet et al. 2014) to capture three aspects of a requirement: (1) who wants the functionality; (2)

[5] https://www.autoitscript.com/site/

[6] https://robotframework.org/

[7] https://www.squash.io/

what functionality the end users or stakeholders want the system to provide; and (3) why the end users and stakeholders need this functionality. This latter aspect is optional. In this context, we check all user stories in order to confirm that each user story is written according the aforementioned template. An excerpt of the user stories defined for the use of Notepad application of Microsoft is shown in Figure 5.

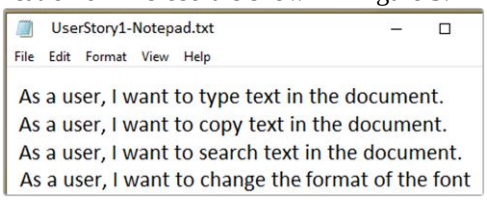


Figure 5. An excerpt of user stories for Notepad application.

## 4.2 Step 2: Deriving Task Model

GUI-Test is a tool support that is developed in Java programming language, using Eclipse platform[8] with the aim of supporting our framework. Using this tool, when the user stories specification is complete, it follows the derivation of the test model (Figure 6).

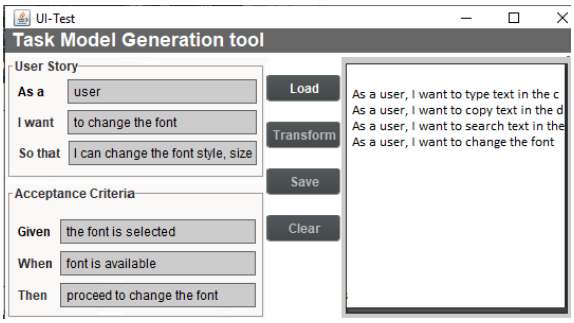


Figure 6. The main user interface of the tool support.

This step is an iterative process: each user story is translated to a task model using the CTT syntax by means of XML (Figure 7). This derivation process is based on relations established between the user stories specification and the syntax of the task model. It is possible to make an association of the user stories concepts with the task model syntax and some of the keywords made available by the Notepad application. These keywords are related with the main menu and its options (File, Edit, Format, etc.) and these keywords permit to describe the steps required to do an action. For instance, the sequence of commands "Format" and "Font" permits change the text font, font style and size of the text in the document.

By using CTT to define a task model, a XML file is obtained. This file describes the tasks included in the model following the syntax defined in CTT.

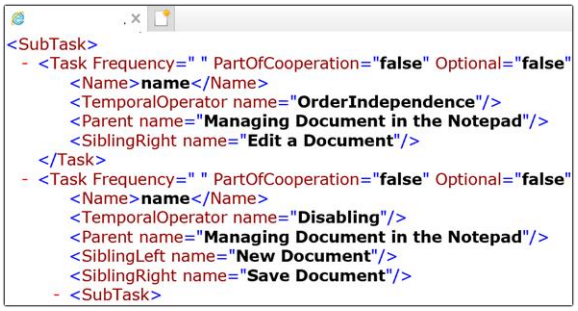


Figure 7. An excerpt of the XML definition of a task model using CTT syntax.

Therefore, when the task model is obtained as a result of this process using GUI-Test, it has the same format specified by means of CTT syntax (Figure 8).

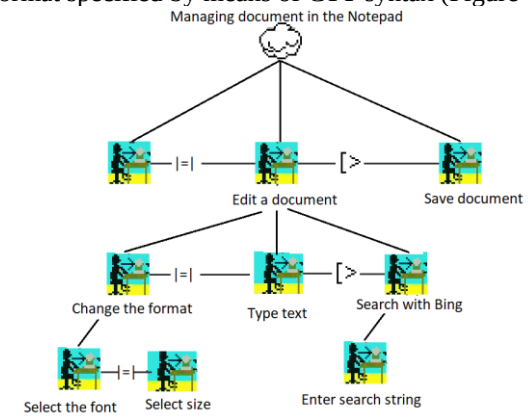


Figure 8. Using CTT to describe a task of the Notepad example.

Additionally, we obtain a second result of this process, it is a tree (a hierarchical data structure) containing the information of each node of the task model. Each node of the tree is defined by three fields: (a) the task to do ("Edit a Document"); (b) if each node has children, the reference to each child; (c) the relationship with other node of the tree. For default the relationships are created as interleaving (|||), since tasks can be performed in any order. However, the tester could change them by editing the CTT model, e.g. the relationship between "Type Text" and "Search with Bing" ([>) included in the Figure 8 was modified to indicate that you must enter the text first before using the Bing Application.

## 4.3 Step 3: Generating Test Scenarios

The next step comprises in the generation of test scenarios. This step is based in the definition of different paths obtained as a result of apply two basic operations in the tree: enumerating (to traverse the tree) and searching (to find a specific node).

[8] http://www.eclipse.org

In this case, we traverse the tree to generate test scenarios. For example, the first scenario is obtained when we traverse the tree starting in the root node (Managing Document in the Notepad), and then we visit the left node (Open Document). Other test scenario can be obtained when we start in the root node and then we visit the central node (Edit Document). Considering this last node as the root of the subtree, then the next node to visit is "Change the format" and the last node is "Select the font". In this traverse, we need to consider the relationship between nodes in order to define which will be the next node to visit. The relationship is demarcated by the temporal operations defined in ConcurTaskTree (Brüning and Forbrig 2011).

## 4.4 Step 4: Adding the GUI locators and variable values to test scripts

At this stage, there is the need to complete the test scripts generated in the previous phase with the locators (e.g. path to an image file or just plain text, which can be used as parameter GUI element image) used for selecting the target GUI elements.

Applications interfaces are formed by sets of elements, namely, buttons, message boxes, forms, links among other elements that allow to increase the User Interface (UI) interactivity. Each of these elements has a specific locator, which allows it to be recognized among all elements of the UI. During the GUI-based testing activity, these elements are used to locate a certain position defined by the test case. In order to automate the test script generation and execution, it is necessary to identify these locators to be able to use the respective GUI elements during the execution of the test. Additionally, in this step the value of required variables must be entered by the tester (e.g. text to write, text to search, etc.) using the tool support (Figure 9).


Figure 9. Interface for specifying variables in the tool.

## 4.5 Step 5: Generating Test Scripts

This generation process is based on relations established between the user story specification (see Column 1 in **Table 2**) and GUI elements (see Column 2 in **Table 2**) and Sikulix code (see Column 3 in **Table 2**). It is possible to make an association of the GUI concepts with the GUI-Test framework syntax and some of the keywords made available in the menus and the user interface of the Notepad application (see **Table 2**). Using Eclipse editor and the elements and functions of SikuliX, the code to apply GUI-based testing in Notepad application in order to evaluate our proposal is written and it is shown in Figure 10. The sentence "s.wait(1.0)" is used in order to load the application (e.g. Notepad) and that its interface is active to be able to execute the tests on its elements.

**Table 2.** Partial view of elements and functions of SikuliX.

| Task Type from User Story | GUI element | Generated Code |
|---|---|---|
| Start | | Screen s=new Screen(); |
| select/order/ filter | Button | s.click($locator) |
| | Element | |
| Edit | Text field | s.type($locator,"text"); |

```
7
8  public class Notepad {
9
10     public static void main(String[] args) throws FindFailed,
11
12         Screen s=new Screen();
13         s.find("textField.png"); //identify textField button
14         s.type("textField.png", "Notepad.exe");
15         s.wait(1.0); //wait for 1 second to show results
16         s.type(Key.ENTER);
17         s.click("File.png"); //click File option
18         s.wait(1.0); //wait for 1 second to show results
19         s.type(Key.DOWN); s.type(Key.DOWN); s.type(Key.DOWN);
20         s.wait(1.0); //wait for 1 second to show results
21         s.type(Key.ENTER);
```

Figure 10. An excerpt of source code to apply GUI-based testing in Notepad application.

The sequence of commands included in Figure 10 corresponds to the selection of the option "File" in the main menu of Notepad, and then, the option "Open", as shown in Figure 11.

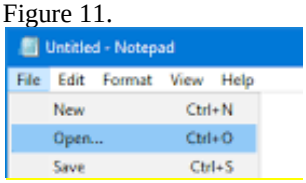


Figure 11. Menu "File" of Notepad.

## 4.6 Step 6: Executing test

Once the script is completely filled in, the tests are run and the test results are displayed, as shown in Figure 12. In this test, we load Notepad applications in Microsoft Windows 10 and then we load a file available in the hard disk of the computer, the test returned one result as expected and so, the test succeeded. On the other hand, if the visual object (image or text) cannot be found, Sikulix will stop the

script by raising an Exception FindFailed and so, the test failed (Figure 13).

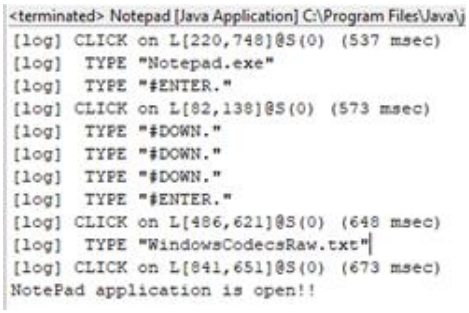


Figure 12. Results obtained in the execution of the test.


Figure 13. Report about an error in the process of testing.

# 5 CONCLUSIONS AND FUTURE WORK

Based on the three issues faced within the test cases generation, this paper proposes a model-driven framework toward generating executable test cases for GUIs to assure that the functionality specified is performed through the different GUI actions of the application. It can cut the effort in testing GUI particularly when the process is evolving. To evaluate the approach, the Notepad application was choosing as an example through the two transformations: from Agile user stories requirements to a test model with abstract test scenario and from abstract test cases to executable test cases in Sikulix language. The transformations can be executed automatically. As the part of model driven testing project, the tool support is being developed. The tool will be able to execute the steps of the framework.

This automatic test case generation framework will reduce the effort needed, improving the quality test cases and the coverage of the requirements by the test cases generated from user stories. This work can find application in developments that use Agile methodologies for testing their products.

Naturally, we will continue our research focusing on the framework scalability, evaluation of the test cases coverage and measure the effort taken to create the test cases and the usability of the tool.

# ACKNOWLEDGMENTS

Fog Computing applied to monitoring devices used in assisted living environments; study case: platform for the elderly, winner of the Call for Research Projects DIUC XVII. Therefore, we thank to “Dirección de Investigación de la Universidad de Cuenca -DIUC” for its academic and financial support.

# REFERENCES

Alfraihi, Hessa, Kevin Lano, and Shekoufeh Kolahdouz-rahimi. 2018. ‘The Impact of Integrating Agile Software Development and Model-Driven Development : A Comparative Case Study’. In *SAM 2018*, Copenhagen, 229–45.

Beck, Kent;, and Cynthia Andres. 2004. 2 *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series)*.

Brüning, Jens, and Peter Forbrig. 2011. ‘TTMS: A Task Tree Based Workflow Management System’. In *BPMDS and EMMSAD*, Heidelberg, 186–200.

Elghondakly, Roaa, Sherin Moussa, and Nagwa Badr. 2015. ‘Waterfall and Agile Requirements-Based Model for Automated Test Cases Generation’. In *ICICIS*, Cairo, 607–12.

Finsterwalder, M. 2001. ‘Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment’. In *XP*, Villasimius, 114–17.

Grangel, R., and C. Campos. 2019. ‘Agile Model-Driven Methodology to Implement Corporate Social Responsibility’. *Computers and Industrial Engineering* 127(April 2018): 116–28.

Kassab, Mohamad. 2015. ‘The Changing Landscape of Requirements Engineering Practices over the Past Decade’. In *EmpiRE*, Ottawa, 1–8.

Latiu, Gentiana, Octavian Cret, and Lucia Vacariu. 2013. ‘Graphical User Interface Testing Optimization for Water Monitoring Applications’. In *CSCS 2013*, Bucharest, 640–45.

Lucassen, Garm;, Fabiano; Dalpiaz, Jan; van der Werf, and Sjaak; Brinkkemper. 2016. ‘The Use and Effectiveness of User Stories in Practice’. In *REFSQ*, Gothenburg, 205–22.

Medhat Kamal, M., Saad M. Darwish, and Ahmed Elfatatry. 2019. ‘Enhancing the Automation of GUI Testing’. In *ICSIE 2019*, Cairo, 66–70.

Moreira, Mario E. 2013. *Being Agile: Your Roadmap to Successful Adoption of Agile*. 1st ed. Apress.

Paterno, F, C Mancini, and S Meniconi. 1997. ‘ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models’. In *INTERACT ’97*,.

Ramler, Rudolf, Claus Klammer, and Thomas Wetzlmaier. 2019. ‘Lessons Learned from Making the Transition to Model-Based GUI Testing’. In *A-TEST*, Tallin, 22–27.

Rane, Prerana Pradeepkumar, Thomas L Martin, Steve R Harrison, and A Lynn Abbott. 2017. ‘Automatic Generation of Test Cases for Agile Using Natural Language Processing’.

Tao, Chuanqi, Jerry Gao, and Tiexin Wang. 2017. ‘An Approach to Mobile Application Testing Based on Natural Language Scripting’. In *SEKE 2017*, Pittsburgh, 260–65.

Wautelet, Yves, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. 2014. ‘Unifying and Extending User Story Models’. In *CAiSE 2014*, , 211–25.