Journal of Software Engineering Research and Development, 2020, 8:9, doi: 10.5753/jserd.2020.731
This work is licensed under a Creative Commons Attribution 4.0 International License..

Extraction of test cases procedures from textual use cases: is it worth it?

Erick Barros dos Santos[*] [**Federal University of Ceará** | ***erickbarros@great.ufc.br***]
Rossana Maria de Castro Andrade[†] [**Federal University of Ceará** | ***rossana@great.ufc.br***]
Ismayle de Sousa Santos [**Federal University of Ceará** | ***ismaylesantos@great.ufc.br***]
Lucas Simão da Costa [**Federal University of Ceará** | ***lucascosta@great.ufc.br***]
Thaís Marinho de Amorim [**Federal University of Ceará** | ***thaisamorim@great.ufc.br***]
Bruno Sabóia Aragão[‡] [**Federal University of Ceará** | ***bruno@great.ufc.br***]
Danilo Reis de Vasconcelos [**Federal Institute of Ceará** | ***danilo.reis@ifce.edu.br***]

Abstract Software testing plays a major role in software quality once it assures that the software complies with its expected behavior. However, this is an expensive activity and, consequently, companies usually do not perform testing activities on software projects due to the time required. These costs may be even higher in testing processes that rely on manual test execution only, which is both time-consuming and error-prone. One strategy commonly used to mitigate these costs is to use tools to automate testing activities such as test execution, test documentation, and test case generation. This paper presents an experience report in the context of a Test Factory about the use of a tool that partially automates the specification of test case procedures from textual use cases. This tool automatically retrieves use cases from the requirement management system, generates the test case procedures, requires inputs from the tester, and then sends the test cases to the test management system. This paper details how this tool was used in releases of an industrial software project through a proof of concept. We also performed a feasibility study with four test analysts from different projects to gather more data regarding its efficiency to support the test case documentation. The results indicate that the tool reduces the test specification time, and that the integration with both requirements and test management systems made our tool feasible in practice.

Keywords: *Software Testing, Test Generation, Test Factory*

1 Introduction

Software testing has an essential role in software quality assurance, allowing the discovery of bugs beforehand over the product life cycle (Myers et al., 2004). However, performing manual testing activities can be time-consuming and error-prone. Beyond that, mistakes in these activities (e.g., bad test coverage or error in testing effort estimation) may contribute to the appearance of test debts, i.e., technical debts related to software testing activities (Samarthyam et al., 2017; Aragão et al., 2019).

Aiming to reduce the mistakes and costs related to software testing, many companies have dedicated efforts to automate testing activities, such as the generation of test cases, test execution, and test reports (Garousi and Mäntylä, 2016). In spite of the advanced research on testing activities automation in the academy, the main concern in the industry is to improve the effectiveness and efficiency of the tests with the automation and use of techniques that are easy to use (Garousi and Felderer, 2017).

Besides that, many software development companies have hired test factories services. One of the advantages of a test factory is that it acts in software testing externally and independently from the development team (Andrade et al., 2017). Test factories can help to improve the quality of software by reducing the effort of testing activities from the development team. Test factories also have teams that work on several domains of systems, which can be allocated to work on different testing projects on demand. Software development organizations also have the benefit of outsourcing the selection of the testing team.

On the other hand, test factories have to cope with challenges related to the definition of testing processes (Aragão et al., 2017) and automation of test case execution (Vieira et al., 2018a). Additionally, the tight deadlines of software projects can hinder the process of an external company that offers the testing services. Thus, it is necessary to research the automation of testing activities.

Regarding the automation of activities, which is the focus of this paper, the literature still has few experience reports, especially in the context of a test factory. This sort of study is important since it provides evidence that knowledge from literature can support practitioners.

This paper's main objective is to report the experience on the test generation from use cases with an automated tool. In our previous work (Santos et al., 2019), we presented our first experience report on using a tool for the semi-automatic generation of test procedures based on use cases. The development of this tool was based on existing work in the software testing literature. We also conducted a proof of concept in the context of a test factory to assess the benefits of the tool during the testing process and reported five lessons learned from this experience. Afterwards, we intend to expand the previous report, also focusing on the acquired experience in

[*]Master Researcher scholarship - sponsored by CNPq (N^o 133464 / 2018-0).
[†]Researcher scholarship - DT Level 2, sponsored by CNPq (N^o 315543 / 2018-3).
[‡]Researcher scholarship - sponsored by Fundação de Cultura e Apoio ao Ensino, Pesquisa e Extensão.

the automatic generation of tests. This extension consists of improving the tool's functionalities, allowing users to define their own templates to extract data from the textual use cases. Another improvement in our tool was the inclusion of business rules in the test case generation process to increase the test coverage.

In this context, we plan to answer the following question: "Is it feasible to use a tool to generate test cases from textual use cases in the test process within a test factory?".

To answer this question, we expanded the efficiency proof of concept with more data regarding real releases of an industry software project. In this proof of concept, the specification of tests needed 65,38% less time than manual activity. We also conducted a feasibility study with test analysts from different projects and collected their feedback. All users needed less time to complete the specification task using the tool, but they also reported the need to improve its usability. For instance, the solution generates test procedures with unnecessary extra characters.

This paper is organized as follows: Section 2 discusses related work. Section 3 presents the methodology used for the development and proof of concept of our solution. Section 4 describes the environment of the test factory, its team profile, tools, and internal processes. Section 5 details the tool developed. Section 6 details the proof of concept conducted with our solution in a industrial context. Section 7 describes the feasibility study with users that was conducted. Section 8 summarizes the lessons learned during this study. Finally, Section 9 concludes the paper.

2 Related Work

In the literature, several work deal with the generation of test cases and procedures from use cases. For example, some approaches (Nogueira et al., 2019; Sneed, 2018) are based on Natural Language Processing (NLP) for the extraction of test cases and others on the generation of intermediate models to extract the necessary information (Some and Cheng, 2008; Massollar et al., 2012). Furthermore, studies (Gutiérrez et al., 2015; Jorge et al., 2018; Massollar et al., 2012; Yue et al., 2015) in the literature have performed evaluations and experience reports in the industry about test generation tools and approaches.

Some and Cheng (2008) offers an approach for generating test scenarios based on textual use cases, using a restricted language with tokens for preconditions, flows, steps, and conditional expressions. The first step in the approach consists of extracting information from structured texts to create a state machine called the Control Flow-Based State Machine (CFSM), in which transitions represent the steps, and states represent the actions and outputs. Use cases included in another use case compose the same CFSM. At the end of the generation process, a global CFSM is generated to link all use cases, which is traversed to generate the test scenarios. The paths in the model represent scenarios that can be generated with different coverage criteria. We use a similar concept in this paper to generate the test procedures, also requiring manual intervention to create the final tests. Another similarity is that we generate the scenarios by paths in the flows, but without generating intermediate models and with simplified selection criteria.

Massollar et al. (2012) present an automated model-based approach for generating test cases. The approach consists of specifying the use cases using specific patterns so that they are converted into UML[1] activity diagrams to represent the system's behavior. The goal of the activity diagram is twofolded - to check if the use cases have been specified correctly and assist test models generation. This test model is the basis for the generation of procedures and test cases in a way that the test analyst must manually identify and insert the necessary data to generate the test cases. This paper also presents an evaluation of the tool that is carried out with two software engineers and a group of students. The authors discuss the data related to the specification time and the model verification, but with low emphasis on the test generation.

Gutiérrez et al. (2015) present a model-based approach for test case generation which focuses on the use of metamodels to increase the generalization of the solution with different approaches. Their solution uses meta models to model use cases and test elements, thus making transformations in the models until the test cases can be obtained. This work presents three industrial use cases, one of them in an agile context, and it also summarizes the lessons learned. Even with the introduction of extra models and their respective transformations, the authors reported effort reductions with the use of the proposed tools. However, not much information is provided about the approach's effort in the agile environment.

Yue et al. (2015) present RTCM, an approach for the textual specification of test cases through similar elements from use cases. This approach provides some predefined patterns for test specification and a tool called aToucan4Test, whose primary goal is to assist the whole generation process of manual and automated test cases. To analyze the feasibility of the solution, the authors present the use of the tool in two industrial case studies in the domain of cyber-physical systems. The assessment is focused on automated test scripts generation, in which the authors report a significant reduction in the implementation effort. Finally, the authors present the lessons learned from the process.

Jorge et al. (2018) propose CLARET, a domain-specific language that allows use case creation using structured natural language and test cases. They also present a supporting tool that allows the specification and validation of use cases but also converts them to Labeled Transitions Systems to create the test cases. This work describes industrial study cases in an agile environment, where the software engineers write use cases using CLARET and generate test cases by using the developed tool. They also present the lessons learned and results on the effectiveness of the solution.

Sneed (2018) reports his experience in the industry with the semi-automatic generation of tests. The generation approach consists of extracting information through Natural Language Processing, using either requirement in plain text or use cases enriched by keywords. The expressions of the text are compared with grammars to identify actions, states, and business rules that serve as the basis to test conditions.

[1] https://www.uml.org/

Table 1. Comparison of related work.

Work	Generation Goal	Artifacts for Generation	Tool-supported	Industrial Evaluation
Some and Cheng (2008)	Test Scenarios	Textual Template and models	Yes	No
Massollar et al. (2012)	Test Procedures	Textual Template and models	Yes	Yes
Gutiérrez et al. (2015)	Test Cases	Models	Yes	Yes
Yue et al. (2015)	Test Cases	Textual Template and models	Yes	Yes
Jorge et al. (2018)	Test Cases	Textual Template	Yes	Yes
Sneed (2018)	Test Cases	Textual Template	Yes	Yes
Nogueira et al. (2019)	Test Cases	Textual Template and models	Yes	No
Current Work	**Test Procedures**	**Textual Template**	**Yes**	**Yes**

Finally, the tester must change the test conditions to insert inputs and outputs. The author reports experience in four industrial projects, summarizing data related to effort. This work is similar to ours, mainly in the use of keywords to ease the extraction of information, though little information is provided on how these tests can be changed through the tool. Additionally, the integration of the tool with other systems that supports the testing process is not presented.

Nogueira et al. (2019) propose an approach for the automatic generation of tests from use cases. For this purpose, the authors propose the use of a controlled natural language. The first step consists of modeling use cases through language, which allows the declaration of system interactions, entries, and conditional expressions. After that, the specification is converted into CSP models, so that the variables and data types are converted to formalism. In the third step, the analyst specifies the purposes of testing that will guide test generation. Finally, the generation is performed using an LTS model, where the traces represent test scenarios and the specified domain is used to create the tests. The authors reported the implementation of a tool that abstracts the formalism of the approach for testers. Among the similarities, it is possible to highlight the use of use cases of partners in the industry. However, the tool usage by test analysts is not presented.

The main goal of our paper is to report the experience of the automatic generation of test procedures from textual use cases. To accomplish this, we implemented a tool that fulfills the needs of a particular agile project. Table 1 summarizes a comparison of our work with the related work presented in this section. It is possible to verify that most studies have focused on the generation of test cases rather than test procedures. However, these approaches impose the additional cost of formal models to increase efficacy (Massollar et al., 2012; Yue et al., 2015; Gutiérrez et al., 2015). To analyze and extract the use cases, we used predefined structures in the use cases without restricting their specification with a syntax grammar (Nogueira et al., 2019; Sneed, 2018). This latter would require changes in all use cases of the project's documentation.

The work most related to ours is the one by Some and Cheng (2008) and Massollar et al. (2012). In our tool, we use a concept similar to the scenarios presented by the forementioned authors to create the procedures, linking the use case flows through references in steps. The main difference was to use a simpler representation of use cases that do not rely on models or formal languages. This impacts the efficacy during the test generation. On the other hand, there is the benefit of offering a more practical solution with less effort related to specification. Thus, the solution can be considered an initial approach for test automation, which integrates with other systems of testing projects. Moreover, our paper discusses the results of effort metrics collected in the context of a test factory and presents feasibility study results with users.

3 Research Method

To guide the execution of the original study (Santos et al., 2019), we used a methodology with steps that were based on the transfer technology model proposed by Gorschek et al. (2006). This model favors a cooperation between the academia and the industry and can be beneficial for both. It allows researchers to study relevant industry issues and validate their results in a real environment. The methodology used in this paper has five steps. In this paper, we improved the solution of Step 2 and the proof of concept of Step 3. We also added Step 5 to perform the evaluation with users. The steps are described thereupon.

Step 1 - Identifying potential improvement areas based on industry requirements: In this step, we performed observations on the test activities of a real test project (see Section 4). To assist information gathering, involved researchers asked the test team about needs regarding the testing process. We identified improvement issues related to the test specification process execution. After that, test analysts of the project were interviewed to gather more details about how the test activities can be executed to reduce the effort[2]. As a result, we identified the requirements of an automation tool for supporting the test analysis and specification.

The requirement for the solution is that it should not cause too many modifications in the artifacts (e.g., use case and test cases templates) of the process. This solution must also be as practical as possible to reduce the effort related to formal specifications. Most of the solutions presented in Section 2 requires the introduction of additional models or modification of the use case templates. We also could not find an automated solution that fulfill the projects needs, so a customized solution must be implemented.

[2]Effort calculated in man-hour

Table 2. Used metrics during the proof of concept.

Group	Metric	Formula	Interpretation
Test Effort	Efficiency of the Test Specification	Amount of Specified Test Cases / Total Time	The biggest is considered better
Test Coverage	Coverage Requirements	Amount of covered use case flows / Total amount of use case flows	The biggest is considered better
Test Coverage	Test cases per requirement	Total amount of test cases per requirement	N/A
Test Effort Economy	Effort Variance	[(Real Effort - Estimated Effort) / Estimated Effort] * 100	Considering the formula, it is possible to assume that: (i) When the Effort Variance is positive, it means that extra time (effort) will be necessary to complete the planned work. (ii) When the Effort Variance is negative, it means that it will be necessary less time (effort) to complete the planned work. (iii) Otherwise, if the Effort Variance is zero, it means that it takes the estimated effort.
N/A	Total Amount of Test Cases	Total Amount of Test Cases	N/A

Step 2 - Solution design: After the previous step, we started the elaboration of a solution. The goal was to use practices and concepts from the literature that would best fit the requirements of the industrial project. In order to do so, we made a review of some solutions presented in the literature that could help in the development of a customized solution. As described in Section 2, many approaches are supported by additional models to increase the effectiveness of the generated tests. However, we chose to avoid model-dependent approaches, since the objective was an easy-to-implement solution that does not require the manipulation of an additional formalism or that could somehow affect the Sprints of the project. As the use cases of the industrial project were specified in the Portuguese language, we also chose not to use NLP-based approaches, once the solutions found are designed for usage with the English language. Additionally, we did not intend to use a fixed syntax to avoid impacts on the specification of use cases. The latter was necessary because the project employs a use case *template* in the requirements elicitation that should not be changed.

Among the tools in the literature that propose test generation, Specmate (Freudenstein et al., 2018) is one of the current tools with more features. Although it still depends on additional models for test generation, its procedure specification and test data insertion process is straightforward and does not depend on additional models. We follow similar interactions to build the user interface of our solution. Our steps to generate the test scenarios draw similarities with the work of Some and Cheng (2008), but with simpler coverage criteria.

Given the considerations mentioned above, we decided to develop a tool that would partially automate the specification process of test procedures. Thus, test analysts could have more control over the test specification. This tool should also receive input data to create test cases.

Step 3 - Performing a proof of concept: To perform an initial evaluation of the tool, a proof of concept was made in one Sprint of the same project used as the context to build the solution, which was started in June 2019 and finished in July 2019. This proof of concept was conducted by one of the researchers and assisted by the test leader of the software project. Aiming to analyze the impact of the tool in the testing teamwork, metrics related to the number of test cases, requirements coverage, effort, and variance were collected. These metrics are part of the Test Factory process (de Castro Andrade et al., 2017) and are based on articles available in white literature (Seela and Yackel) and academic work (Lazic and Mastorakis, 2008). Table 2 summarizes the metrics used and their respective formulas. Based on the pilot results, which were presented in our previous paper (Santos et al., 2019), we obtained initial data about the feasibility of the tool and its benefits. We also identified some failures in the tool, which were fixed before the tool was deployed. The results of the proof of concept are presented in Section 6.

Step 4 - Solution deploy: In this step, the elaborated solution is deployed in the project for use. In our case, we deployed the tool for use in our industrial project. Then, we used our tool during some releases of the referred project. The data collected during this usage is presented in Section 6.

Step 5 - Carrying out a feasibility study with other professionals: After the deployment of the solution, we performed a feasibility study with professionals from the testing area of other software projects. The main objective of this evaluation was to obtain data about the efficiency of the tool with professionals from different contexts who have had experience in the specification of tests based on use cases. The results of this feasibility study are presented in Section 7.

4 Proof of Concept Environment

In this section, we present the environment in which the solution was created and the proof of concept in Section 6 was conducted. We performed the proof of concept in a Research, Development, and Innovation project concerned with requirement elicitation and software testing of the Software A[3]. This software aims to manage a passive optical network.

This project can be considered distributed since the client, development and requirement/test teams belong to different institutions and work in different locations. The team responsible for the Software A's requirement/tests makes use of the SCRUM (Schwaber and Beedle, 2002) framework with Sprints that lasted a month.

This environment was the basis to build the tool presented in Section 5. It was also used to conduct the proof of concept to be presented in Section 6.

Subsection 4.1 presents the testing team's profile. Subsection 4.2 describes the tools and patterns adopted in the project. Subsections 4.3 and 4.4 detail, respectively, the requirement and the test process used. These processes were elaborated based on the previous experiences (Aragão et al., 2017; Vieira et al., 2018b) of the GREat's[4] test factory in test projects.

4.1 Team profile

The test factory team involved in Software A project is composed of a test manager, one requirement analyst, two test analysts, one trainee (tester), and one researcher. Among the members, only one of the analysts and the trainee executed test cases. The analyst has a fourteen-month experience in requirement elicitation and worked for one year and six months in test execution. The trainee has an experience of a year and four months in requirement elicitation and test execution. Both of them have a fourteen-month experience in requirement elicitation and test activities in the Software A project.

The requirement/test team performed both the requirement and test activities, in which the use cases are the basis for the test case specification. In addition to the tests based on use cases, the team also conducted exploratory tests during the execution of the Sprints. The high knowledge of Software A's requirements allowed the test analysts to generate more concise test documentation, thus providing more agility during the process. Therefore, the analysts executed the tests based on the documents and their own experience. However, concise test documentation can also be costly to create and maintain, especially in a project with a lot of requirement changes and fixed release date.

4.2 Tools and Patterns of the Internal Processes

To guide the activities of the requirement and test processes (see Sections 4.3 and 4.4), the testing team used the following tools: JIRA[5], for use case and task management;

[3]The system name was omitted due to confidentiality agreement.
[4]http://www.great.ufc.br
[5]https://www.atlassian.com/software/jira

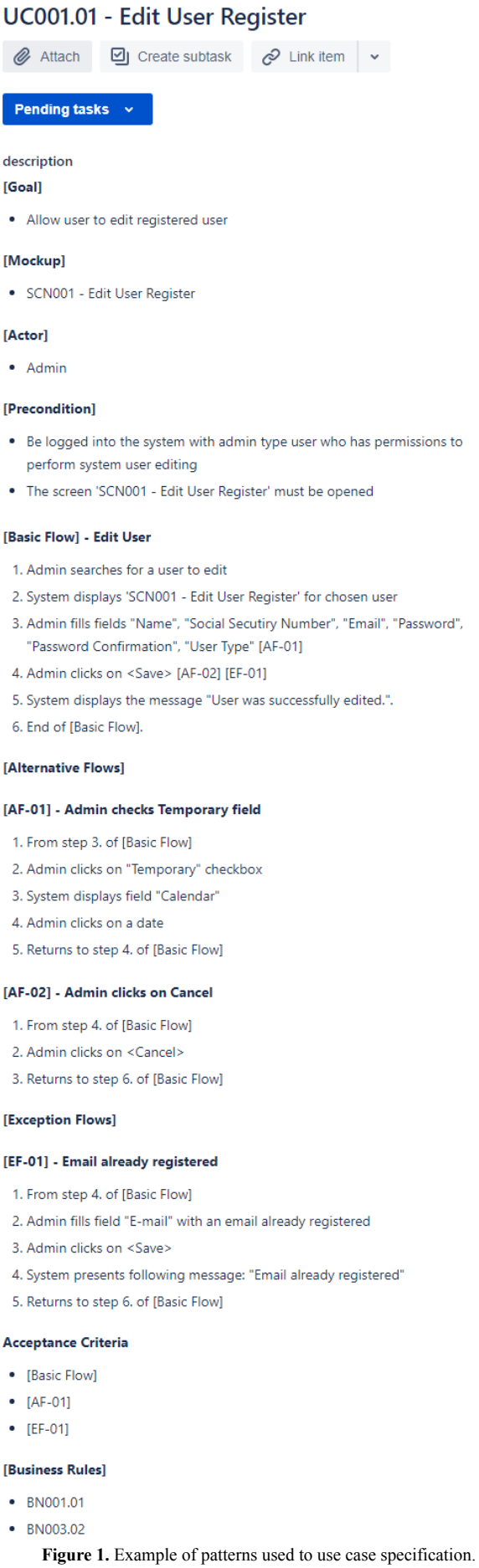

Figure 1. Example of patterns used to use case specification.

Confluence[6], for business rules and general documentation; TestLink[7], for test plan and test cases management; and the browsers Google Chrome[8] and Firefox[9], for the test case execution.

Since the beginning of Software A project, the stakeholders decided to perform the requirement specification using well-defined templates, aiming to improve the understandability for all stakeholders. Therefore, we used special symbols that ease the identification of elements in the use case steps. Figure 1 shows a fictitious example of a use case to edit a registered user, where the Basic Flow starts in the tag *[Basic Flow]*. Likewise, in step 3 of the Basic Flow, the input fields are identified by double quotations. In step 4, the clickable visual elements are written between <> symbols. The use case also has information about the use case's goal, related mockups, preconditions, and the acceptance criteria. The latter refers to the flows that should not have any critical bug so the use case implementation can be considered "done".

4.3 Requirements Process

In order to organize the requirement engineering tasks, we followed a requirement process with the following activities: Elicitation, Analysis, Specification, and Validation. According to Wiegers and Beatty (2013), these steps are essential to requirement engineering in a software project. During the implementation of the project, the analysts performed the requirements activities in a way its outputs could be used as input to the Sprint's backlog.

Although the project had agile characteristics, the client requested the detailing of the documentation for Software A because of its complex features. Thus, we specified the requirements through textual use cases. Each step task is presented as follows.

1. **Elicitation:** This step aims to identify the system requirements by consulting the stakeholders. In this process, the team performs an interview with stakeholders and the elaboration of usage scenarios with interface prototyping using the Balsamiq[10] tool.
2. **Analysis:** This step is responsible for verifying the consistency, completeness, and viability of previous elicited requirements. Hence, the stakeholders prioritize the requirements aiming to identify which ones have a faster and higher return of investment to the client and final customer.
3. **Documentation:** In this step, the analysts specify the requirements as use cases and communicate them to the team. The system business rules are also documented.
4. **Validation:** In this step, the analysts assure that the requirements have acceptable description and can be sent to the development. In this paper, this step also involves the creation and validation of high fidelity prototypes.

[6]https://www.atlassian.com/software/confluence
[7]http://www.testlink.org
[8]https://www.google.com/chrome
[9]https://www.mozilla.org
[10]https://balsamiq.com/wireframes/

4.4 Testing Process

The testing process provides real feedback from the behavior of the software (Bertolino, 2007), and the organization of activities allows its communication, monitoring, and improving (Mette and Hass, 2008).

Processes can vary depending on the institution. Still, there are generic processes (Mette and Hass, 2008; ISO/IEC29119-2, 2013) that can be adapted for the organization's purposes. GREat's Test Factory project (de Castro Andrade et al., 2017) is based on MPS.BR (Montoni et al., 2009) and has three steps: (i) Planning; (ii) Elaboration; and (iii) Execution. In the context of this project, the requirement analysts send the documents, specified as described in Section 4.3, to the test activities so that the specification and execution of the tests are performed before the system is released. We present a brief description of the main activities of this process as follows:

1. **Planning:** This step consists in verifying the test goals and perform the required actions to transform the test strategy in an operational plan.
2. **Specification:** This step aims to elaborate tests to meet the demands of the test plan. This also includes automated test scripts specification, when necessary.
3. **Execution:** At last, the final step relates to executing the tests and store results. In this step, the test analysts must verify the test incidents. Therefore, the analysts also generate a test report and send it to the client with lessons learned.

It is worth noting that, during the whole process, the test team controlled and kept track of the activities, allowing them to make some improvements in the next process execution.

5 Tool for Semi-Automatic Generation of Tests Procedures

In our previous work (Santos et al., 2019), we introduced the tool used to generate tests from use cases in the context of the Software A project. In this paper, we will refer to this tool as UC2Proc. This tool was mainly developed by one researcher and one test analyst. Regarding the improvements of the tool (see Section 5.1), another test analyst was responsible for implementing additional features. The features of the UC2Proc tool comprise processing structured textual use cases from JIRA, the generation of test procedures from the use case flows, the edition of the extracted test procedures, the addition of input data to create test cases, and, finally, sending generated test cases to the TestLink.

For the current study, our tool was improved based on the pilot results. All the tool features and its improvements are detailed in Subsection 5.1. In Subsection 5.2, we present its package diagram, and in Subsection 5.3 we present some interfaces and how the tool works.

5.1 Tool Features

The features of the UC2Proc are described as follows:

(1) Integration with JIRA. In our tool, the test analyst can search for the identifier of a use case issue from the JIRA system. Next, regular expressions are used to extract the information from the textual use cases, which processes the following elements: objective, preconditions, flows, steps, references to other flows, and data entries in the steps. This operation only works correctly if the use cases are strictly specified according to the patterns configured in the tool.

(2) Testing Procedures Generation. The information extracted from the textual use cases is used to generate test procedures. To achieve so, the UC2Proc first creates test scenarios that are composed of flow sequences to be visited in the use cases. To accomplish this task, we used an approach similar to the one presented by Some and Cheng (2008). Then, we implemented an algorithm that generates state sequences starting in the 'Basic Flow' and visits all alternative/exception flows that depart from it. Thus, starting from each of the alternative/exception flows, all their respective steps are analyzed and the paths to other flows are visited. This scenario of flow sequences are used to compile the input steps and variables that will compose the test procedures.

In the current version of our tool, we also added a new function that identifies the business rules referenced in the use case. The tool then creates two tests for each rule: one with the purpose of validating it and the other aiming at verifying the violation of the rule. The user then visualizes the reference and manually fills the steps of the test procedure. The coverage criteria, although simple, allows generating scenarios that go through all flows and some transitions. However, a deep search for all state machine paths is not performed, which could lead to the generation of some scenarios with several flows. The intention of this functionality is to generate test procedures similar to those that the test analysts manually generate in the project.

Algorithm 1 explains the process to generate the scenarios for each use case. Lines 1 to 5 declare the necessary variables, where **testScenerarios** is the list of scenarios with the use case flows, **currentPath** is an auxiliary variable and **testProcedures** is the final list of test procedures. The first step is to create a scenario for the basic flow. Thus, the algorithm iterate over each step of the basic flow. From line 8 to line 11, a new scenario is created from basic flow to each flow that is called in the basic flow steps. Next, the lines of each alternative/exception flow are analyzed, and one scenario is created starting from then to each new flow reference. In summary, scenarios are generated by exploring a maximum of one level from each flow. After creating the test scenarios, the algorithm iterates over each scenario, creating a procedure containing title, goal, and preconditions from the scenario. Thus, the algorithm iterates over each business rules from use case and creates a test procedure containing the title of the rule and blank space in steps and output, which the user must manually fill. At last, the algorithm returns the list of test procedures generated.

(3) Testing Procedures Management. The developed tool allows the test analyst to add, edit, and delete a test procedure, as well as to manage the steps within a procedure. Our tool also extracts and displays to the user the inputs listed in the steps of each procedure. These inputs can be added, edited, or removed when editing the steps, but it is not possible to automatically extract references of the screens *mock-ups* and actors of the use case.

Algorithm 1: Generation of Test Procedures

```
   Result: List of test procedures
1  useCase ← use case to be processed;
2  businessRules ← list of business rules in useCase;
3  testScenarios ← ∅;
4  currentPath ← ∅;
5  testProcedures ← ∅;
6  Comment: creation of scenarios
7  testScenarios ← testScenarios∪ basic flow in
     useCases;
8  for each flow in basic flow from useCase do
9  |  currentPath ← basicflow ∪ flow;
10 |  testScenarios ←
   |    testScenarios ∪ currentPath;
11 end
12 for each remaining flow in useCase do
13 |  currentPath ← reconstruct path from basic
   |    flow to current flow;
14 |  for each flow reference in current flow do
15 |  |  testScenarios ← currentPath∪ flow
   |  |    reference;
16 |  end
17 end
18 Comment: creation of test procedures
19 for each scenario in testScenarios do
20 |  procedure ← create test procedure with title,
   |    goal, and preconditions from scenario;
21 |  for each flow reference in scenario do
22 |  |  Add user steps in procedure;
23 |  |  Add system steps in procedure;
24 |  end
25 |  testProcedures ←
   |    testProcedures ∪ procedure;
26 end
27 Comment: Creation of procedures to
     bussiness rules
28 for rule in businessRule do
29 |  procedure ← create new procedure with title of
   |    businessRule Comment: the steps must be
   |    created by the test analyst
30 |  testProcedures ←
   |    testProcedures ∪ procedure;
31 end
```

(4) Testing Data Insertion. After generating the test procedures, the test analyst can insert the test data and generate instances of test cases with different input data.

(5) Integration with TestLink. After the generation of the test procedures and addition of the data to generate test cases, the tool sends the test suite to TestLink. For this, the test analyst must configure the test project name in which the generated test cases must be uploaded.

(6) Template Customization. In our previous work (Santos et al., 2019), the tool was limited to a fixed use case template. In the present work, we added new functionality that allows the customization of the patterns to detect elements of textual use cases. It is worth noting that the general struc-

ture of the use cases is fixed and must be followed. However, the user can create its own regular expressions using a form in the tool. The major advantage of this functionality is that it makes the tool more customizable, allowing it to adapt to patterns of different projects or organizations.

5.2 Package Diagram

The UC2Proc tool was developed as a Web App using the *framework* Ionic[11] and the JavaScript programming language [12]. We also used the JIRA and TestLink APIs to allow communication with these services. Figure 2 presents a UML

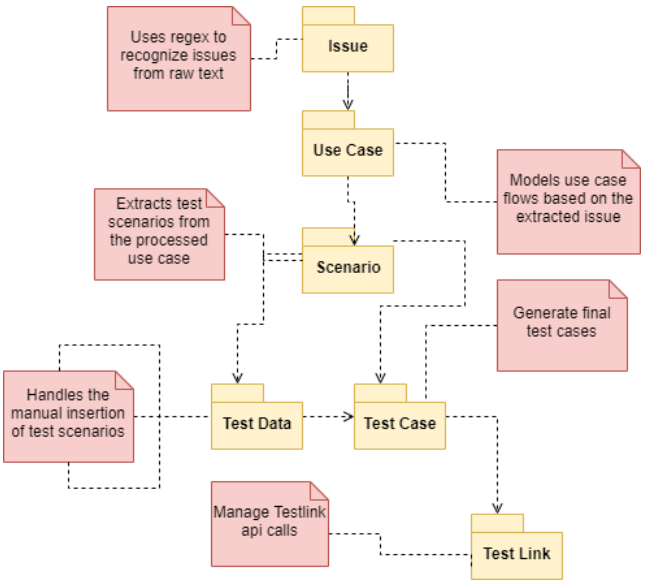

Figure 2. Package Diagram of developed UC2Proc.

package diagram representing an overview of the tool's architecture. This figure highlights the main modules of the tool: *Issue*, *Use Case*, *Scenario*, *Test Data*, *Test Case* and *TestLink*.

The *Issue* module manages the issues received from the JIRA API, which uses the defined regular expressions to extract the necessary elements from the issue. These elements are used to instantiate the objects (e.g., Basic Flow, Business Rules) used in the test generation process.

The *Use Case* module receives the issue extracted from the previous module. Next, it instantiates a use case object based on the information received in such a way that each flow contains steps, the flow/event that triggers it, and the flows possible to reach it.

After that, the *Scenario* module generates the usage scenarios from the flows labeled in the Use Case module. The generation process follows the algorithm presented in Subsection 5.1. The *Test Data* module handles the manual input of test data in the test scenarios.

Finally, the *Test Case* module generates the test cases that will compose the test suite and uses the *Testlink* module to send them to the test management tool.

[11] https://ionicframework.com/
[12] https://www.javascript.com/

5.3 Tool's Usage

This section presents an example of the use of our tool, given the use case presented in Figure 1. At first, to use the tool, the user must configure the integration with external management systems, detailed as follows:

- **JIRA:** The test analyst must provide the URL of the JIRA API, username, and API key of an account. This information can be obtained from the security page of the JIRA user account; and
- **Testlink:** The test analyst must provide the URL of the TestLink API, which can be obtained from the analyst responsible for the maintenance of the TestLink in the organization, and the API key, which can be found on the user page in TestLink.

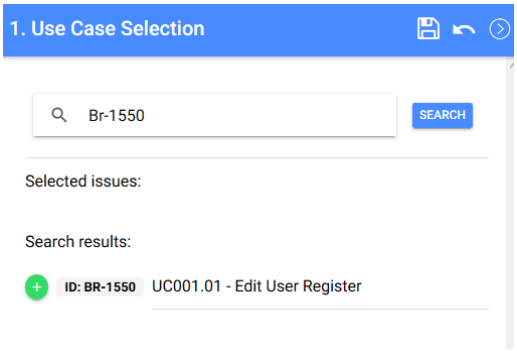

Figure 3. Tool's issue searching.

Once the authentication information is configured, the user must start by searching the use cases. In order to do so, it should type the use case issued ID from JIRA in the field "search". The system then displays the search results and allows the found issues to be added by the "+" button, as shown in Figure 3.

Thus, the user must click on the right arrow button and the system runs Algorithm 1. Hence, it displays the generated scenarios for each flow and business rule from the use case. Assuming the structure of the use case presented in Figure 1, the expected result must generate one test procedure for each flow (Basic Flow, AF-01, AF-02, EF-01) and two additional procedures for the business rules.

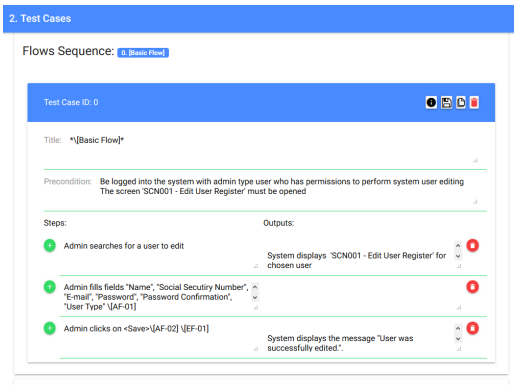

Figure 4. Example of scenario generation.

Table 3 presents the test procedures generated with the following fields: (i) **Scenario**, the test scenario generated; (ii) **Title**, which is the test procedure title; and (iii) **Actions**

Table 3. Example of test procedures generated by tool.

Id	Scenario	Title	Actions	Outputs
1	[Basic Flow]	[Basic Flow] - Edit User	[Basic Flow] 1,3,4,6	[Basic Flow] 2,5
2	[Basic Flow],[AF-01],[Basic Flow]	[AF-01] - Admin checks Temporary field	[Basic Flow] 1 [AF-01] 2,4 [Basic Flow] 4,6	[Basic Flow] 2 [AF-01] 3 [Basic Flow] 5
3	[Basic Flow],[AF-02],[Basic Flow]	[AF-02] - Admin clicks on Cancel	[Basic Flow] 1,3 [AF-02] 2 [Basic Flow] 6	[Basic Flow] 2,5
4	[Basic Flow],[EF-02]	[EF-01] - Email already registered	[Basic Flow] 1,3 [EF-01] 2 [Basic Flow] 6	[Basic Flow] 2 [EF-01] 4
5	BN001.01	Validates BN001.01	Blank	Blank
5	BN003.02	Validates BN003.02	Blank	Blank

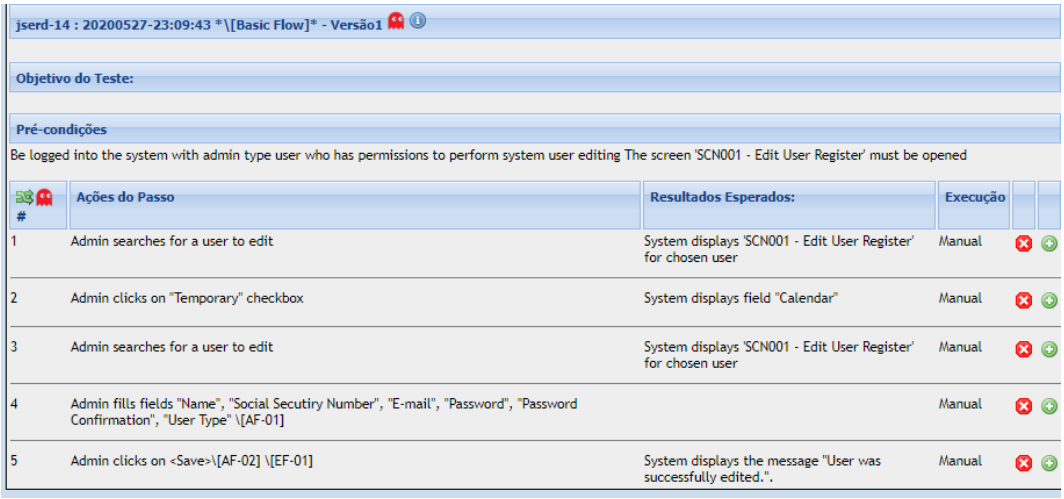

Figure 5. Test cases exported to Testlink.

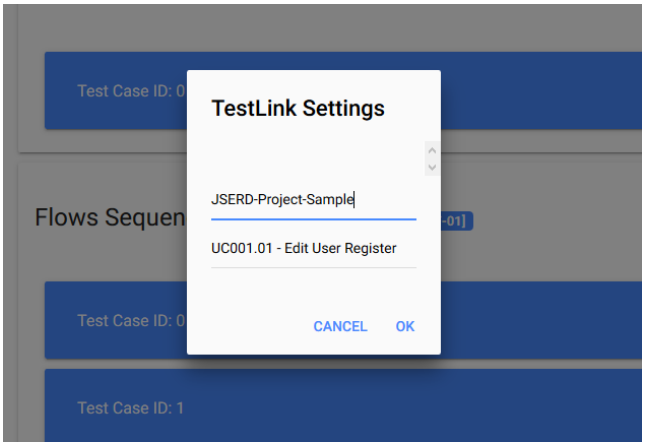

Figure 6. Example of Testlink integration pop-up.

and **Outputs**, which represent the steps and expected results through the use case flow (between "[]") and the step numbers. For instance, the first row of 3 represents basic flow scenario, which assumes steps 1, 3, 4 and 6, steps that contain "The Admin", from use case as test actions, and steps 2 and 5, steps that contain "System", as the expected result. Thus, the user can add, remove, or modify test cases or test data using the fields shown in Figure 4.

After the edition of the test procedures and generation of the test cases, the user must click on the button "Send to Testlink", and choose the name of the project and test suite in TestLink, as shown in Figure 6. The tool then creates a new test suite into the chosen project and export all current test procedures as test cases into Testlink.

An example of test cases generated from use case shown in Figure 1 is depicted in Figure 5.

6 Proof of Concept

In this section, we detail the process of the study execution after the deployment of the solution described in Section 3. After that, we performed a proof of concept conducted in the environment presented in Section 4.

Subsection 6.1 describes the steps to perform the evaluation. Subsection 6.2 summarizes and discusses the results related to the test effort.

6.1 Proof of Concept Steps

We evaluated the tool in three steps: (1) selection of use cases, which produces a list of requirements without previous test cases; (2) effort estimation, where the analyst evaluates the time to complete the task based on its experience; and, (3) automatic generation of tests, comprising the actual use of the generated solution. These steps were performed by one of the researchers and the test analyst of the project.

These steps are described next.

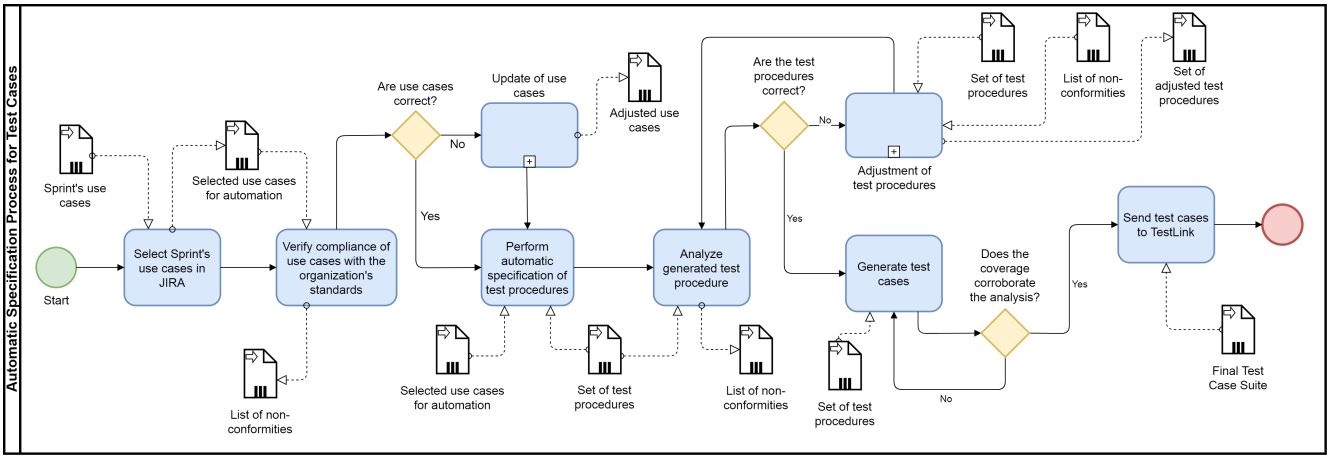

Figure 7. Process to Semi-Automatic Specification of Test Procedures.

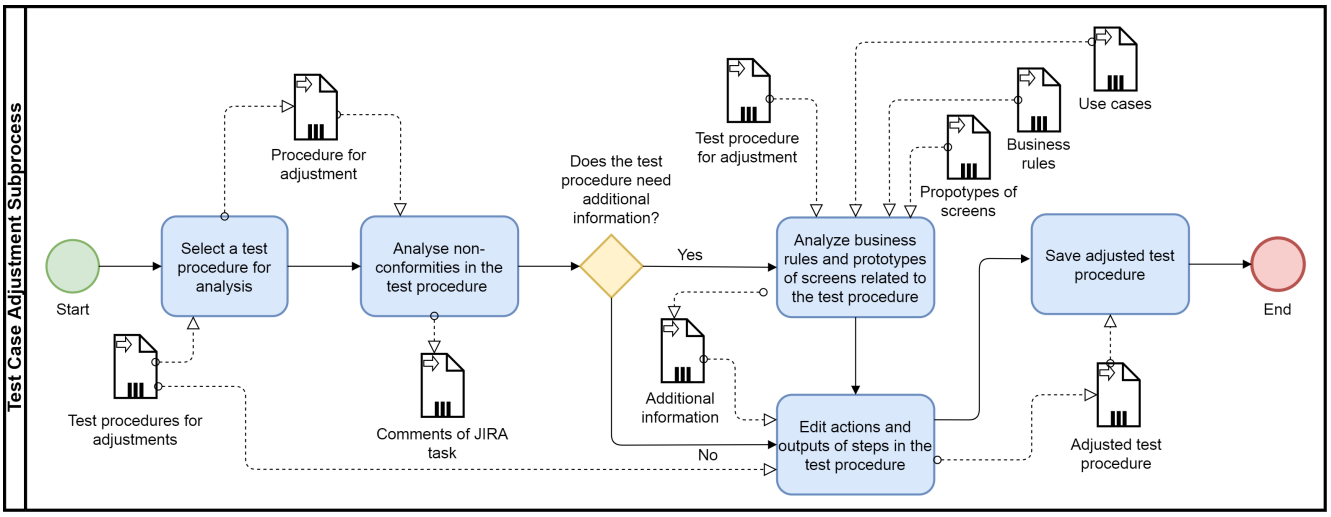

Figure 8. Subprocess to Adjust Test Procedure.

1. Use Case Selection. The first step of the proof of concept was the selection of the use cases. To prevent the bias associated with the analyst's knowledge, we selected use cases that were not analyzed and specified before. Taking into account that textual patterns of the project were already applied to documents, the analysts did not perform editions in the use cases.

To present results as close as possible in the original context of Software A project, we selected use cases from a real release of the software under development. Considering the aforementioned conditions, the team used all the artifacts produced during the proof of concept in the real release.

2. Effort Estimation. The test analyst estimated the manual effort to specify the tests in minutes. Then, the analyst calculated the effort with a metric defined by the following equation: $3 * (NofFlows + NofBusinessRules) + W$. The number three is a multiplier factor representing the analyst's effort in minutes to specify the tests for a use case with N flows and N business rules. Additionally, the metric includes a weight W that adds the extra time based on the analyst's perception. The cases with W equals to -1 refers to flows with many repeated steps, which required lower test specification effort. The greater the inexperience of the analyst with the functionality under test, the greater the value of the W weight.

The team of Software A project created the metric to fill its needs of manual effort estimation, considering margins of error based on the analyst's opinion. The whole equation is based on recent experience gained in the project. Despite the metric is ad hoc and not validated in controlled experiments or case studies, the results were accurate enough in our previous evaluations (Santos et al., 2019).

3. Generation of Test Procedures. The test analyst must perform the test cases generation based on use cases. The test analyst carries this process using the proposed tool and the TestLink, aiming to compare the required time to complete the task and the estimation effort results. It is worth noting that the test analyst did not specify the test cases manually during the proof of concept.

To use the tool and generate the test cases, the test analyst performed the activities following the process illustrated in Figures 7 and 8. The details of each activity are presented as follows.

1. **Select Sprint's use cases from JIRA:** This activity receives as input the list of selected use cases from *backlog*, which were selected at the beginning of the proof of concept. At this point, the test analyst must select the use cases for the automatic generation of test procedures.
2. **Verify compliance of use cases with the organization's patterns:** In this activity, the test analyst checks

Table 4. Tool results for selected use cases.

Id	UC flows	Business rules	UC steps	Template correction time	Adjust time	Testlink correction time	Weight	Estimated time	Actual time	Test cases generated
1	1	0	4	00:01:14	00:00:32	00:00:10	2	00:05:00	00:01:56	1
2	3	0	11	00:01:11	00:00:44	00:00:43	3	00:12:00	00:02:38	3
3	1	0	4	00:01:02	00:00:18	00:00:25	2	00:05:00	00:01:45	1
4	3	1	32	00:02:41	00:02:28	00:00:56	3	00:15:00	00:06:05	5
5	3	0	16	00:01:27	00:00:41	00:00:41	3	00:12:00	00:02:49	3
6	6	2	34	00:01:45	00:07:21	00:00:33	2	00:26:00	00:09:39	6
7	3	0	17	00:03:12	00:00:54	00:00:25	3	00:12:00	00:04:31	3
8	3	0	18	00:03:45	00:01:01	00:00:45	3	00:12:00	00:05:31	3
9	3	2	17	00:02:12	00:07:29	00:00:50	0	00:15:00	00:10:31	7
10	3	2	16	00:01:26	00:02:35	00:00:50	0	00:15:00	00:04:51	7
11	3	1	12	00:02:53	00:02:33	00:00:39	2	00:14:00	00:06:05	5
12	4	3	25	00:01:31	00:05:30	00:01:42	-1	00:20:00	00:08:43	13
13	2	1	13	00:04:47	00:01:58	00:00:26	0	00:09:00	00:07:11	5
14	2	1	10	00:02:08	00:00:40	00:00:26	0	00:09:00	00:03:14	2
15	9	1	40	00:02:03	00:01:53	00:00:41	10	00:40:00	00:04:37	13
16	8	3	54	00:03:58	00:07:00	00:01:06	2	00:35:00	00:12:04	11
17	4	2	24	00:02:46	00:01:37	00:01:00	0	00:18:00	00:05:23	6
18	6	1	27	00:02:00	00:03:06	00:00:44	4	00:25:00	00:05:50	9
19	1	1	7	00:00:57	00:01:16	00:00:22	-1	00:05:00	00:02:35	2
20	5	5	30	00:00:43	00:04:52	00:01:11	0	00:30:00	00:06:46	10
21	4	2	24	00:02:00	00:09:55	00:01:16	0	00:18:00	00:13:11	7
22	9	3	47	00:05:02	00:05:07	00:00:30	4	00:40:00	00:10:39	12
23	6	1	26	00:01:31	00:04:49	00:00:30	4	00:25:00	00:06:50	8
24	8	3	47	00:06:00	00:04:00	00:01:21	2	00:35:00	00:11:21	12
Total	**100**	**35**	**555**	**00:58:14**	**01:18:19**	**00:18:12**	**47**	**07:27:00**	**02:34:45**	**154**

the compliance of use cases from the previous activity regarding the organization's patterns. The focus of this activity is also to analyze whether the use cases follow the patterns configured in the automated tool. The test analyst must record any non-conformity in the corresponding JIRA task.

3. **Use case update:** In this sub-process, use cases that do not follow the pattern configured in the tool must be updated by the requirements analyst. This activity receives as input the list of incorrect use cases. In the end, the requirements analyst must produce updated use cases according to the established patterns.
4. **Perform automatic specification of test procedures:** This activity contemplates the use of the tool to perform the automatic generation of test procedures. In order to do so, the test analyst must have access to verified use cases. At the end of the process, a set of test procedures must be generated.
5. **Analyze the generated test procedures:** After generating the procedures, the test analyst should perform an analysis of the generated steps. In this one, the test analyst assures the correct extraction of steps and outputs from the use cases. If errors are found in the procedures, the team records the occurrences in the JIRA.
6. **Adjust the set of test procedures:** This sub-process consists of adjusting the set of test procedures when there are non-conformities, and the test analyst is responsible for performing them. The sub-process must receive the list of test procedures for adjustments and produces as output the adjusted set.
7. **Generation of test cases:** After analyzing the procedures and performing the necessary adjustments, the test analyst should provide the test data to generate the test cases. The test analyst repeats this action until the desired coverage is obtained. The activity receives as input a set of test procedures and must produce as output a suite of test cases.
8. **Send test cases to the TestLink:** The last activity of the process is to send the test cases to TestLink. The activity receives the test case suite as input and automatically sends them to the TestLink tool. After sending the test cases to TestLink, the test analyst must perform any additional update directly in the TestLink. If more tests cases are generated for the the test suite already present in the TestLink, the test analyst must also add them manually.

6.2 Metrics Results and Discussion

During the processing of use cases, the test analyst manually collected the times obtained at the end of each activity. Hence, the only instrument used in this step was a spreadsheet with the same fields of Table 4.

Table 4 shows the results of the tool usage. In this table, we

detail the main use case information, such as the number of flows, steps, business rules, and the estimated modeling time of the use case. The table also presents the time spent to correct the non-conformity of patterns in the use case, the time required to edit the test suit (e.g., the addition of the business rules details), and, finally, the time necessary to adjust the test cases to be exported to TestLink. In total, 24 use cases were used, distributed among five Sprints, totaling 100 flows and 555 steps.

As shown in Table 4, the use of the tool in the Sprint yielded 154 test cases, taking, in total, 2 hours and 34 minutes to be carried out. This represents a reduction of approximately 65,38% in effort compared to the 7 hours and 27 minutes of estimated manual effort.

From the results of the columns "Business rules", "UC steps number" and "Test Adjust Time" it is possible to verify that, most of the times, the use cases with a higher number of business rules and steps demanded a greater effort for correction. This complements the results of our previous work, which shows that the complexity of the use cases impacts the generation of tests since the manual intervention of the analysts is necessary.

In the previous paper (Santos et al., 2019), the result of the pilot study was presented using nine use cases belonging to one System A project Sprint, which accounts for 41 flows and 186 steps. In this proof of concept, the test analyst estimated the time needed for the modeling of each use case and compared it with the time spent during automation. The reduction of effort in the pilot was approximately 65%. Thus, analyzing the results obtained, it is possible to identify an effort below the estimated for the generation of test cases, especially in use cases with a larger quantity of flows and steps. Regarding the question raised in the title of the paper, it can be said that this generation was worthwhile in the particular context of the project. However, more research must be carried out to generalize the findings.

6.3 Threats to Validity

After the proof of concept execution, we have identified some limitations which must be discussed. So, we have carried out the presentation of these limitations as threats to validity, as showed in (Wohlin et al., 2012). We discussed the following threats.

Regarding the external validity, which determines the generalization of results, we used a metric to estimate the manual effort that could be a threat to the proof of concept. The metric estimates the time required to complete the specification of the test cases, and it was created based on the recent experience of the test analysts. However, the resultant value takes into account specific characteristics related to the context of the Software A, e.g., extra time to adjust the test cases after the specification. Nevertheless, we believe that the metric is simple enough to be adapted for other projects.

Also, regarding the generalization of the study, the usage of patterns in the tool for data extraction can hinder the tool usage by other teams. To mitigate this, we tried to develop the tool to read patterns as general as possible, but that still fit the needs of the current project. Additionally, we also improved the tool to allow users to create custom patterns with regular expressions.

At last, the proof of concept was conducted by one of the authors, who is also the test analyst in the project. To reduce the possible bias of the evaluation, the test analyst used a tool as similar as possible to the real context. Additionally, the proof of concept was a pilot evaluation to analyze the feasibility of the tool.

7 Feasibility Study

To analyze the viability of using the developed solution in different contexts, we perform a proof of concept in a real Sprint with users trained and used to project context of Software A. Aiming to obtain more data about the use of the UC2Proc tool, we performed a feasibility study with test analysts from another industry project in partnership with the GREat[13] laboratory. This feasibility study focuses on measuring the tool's efficiency from the user's perspective, without previous experience with the developed solution in this case. Additionally, we also collected users' opinions about the positive and negative points of the solution.

Subsection 7.1 details the methodology adopted to conduct the feasibility study. Subsection 7.2 discusses the results obtained after the feasibility study with the users. Subsection 7.3 explains some of the limitations in the conduction of the work.

7.1 Methodology

We decided to plan the feasibility study using the DECIDE framework proposed by Preece et al. (2004). This framework aims to guide evaluations with users through a checklist with well-defined activities ranging from defining objectives to evaluating data. We selected DECIDE because it is easy to apply in practice, allowing the assessment to be conducted by inexperienced assessors (Preece et al., 2004). The following paragraphs will describe the six activities related to planning and execution.

(1) Define objectives. The first item on the DECIDE checklist concerns the definition of the objectives that should guide the feasibility study. Since our focus was to analyze whether the solution generated was capable of reducing the testing specification time, we defined the following objectives: (i) to evaluate the efficiency of using the tool to specify tests by the test analyst and (ii) discover the positive and negative perceptions of the test analyst about the use of the tool.

(2) Explore questions. The second item of DECIDE is the definition of the questions that must be answered at the end of the study. Considering as our goal the analysis of the user performance with the automated tool, we elaborated on the following question: was the tool able to increase the testing specification speed compared to the manual specification? Regarding the objective of identifying positive and negative aspects, we elaborated on the following question: what are the positive and negative perceptions of the analyst during the tool usage?

[13] http://www.great.ufc.br

(3) Choose the evaluation paradigm. The third item corresponds to choosing the evaluation paradigm and techniques to answer the questions of item two. In order to identify the efficiency of the generated solution, the participants of the feasibility study performed a manual and an automatic task. The details of the activities are described as follows:

i *Questionnaire for profile identification*: the evaluators applied this questionnaire to identify some general characteristics of the test analysts, such as their experience with use cases and automated test specification tools. The form filled by the users is presented in Table 6 of the Appendix A with the name *Professional Profile*.

ii *Manual specification of tests*: the first task performed by the test analyst concerns the manual specification of tests related to a use case. In order to achieve it, the participant received a document describing the use case, being instructed to make the specification in the TestLink. The use case for this activity is a fictional system for creating movie schedules in theaters, with a total of five flows and 24 steps. During the performance of the activity, two evaluators made notes about the comments and doubts of the participants and other considerations about the execution of the task.

iii *Tool presentation*: after that, the researcher presented the tool and showed an example of how to use it. The use case employed in the example is the same as the one in the manual task.

iv *Semi-automatic specification*: in the second task, the test analyst received a use case document of music software that allows the creation of playlists. Even though it is a different system, the use case has the same complexity (number of flows, steps, and references between flows) as used in the manual task. Then, they were instructed to process it with the tool and send it to TestLink. The two evaluators also observed and made notes just as in the manual task.

v *Open questionnaire*: finally, analysts needed to answer an open questionnaire with questions about the positive and negative aspects of the tool. The final form is presented in Table 6 of the Appendix A with the name *User - Tool Evaluation*.

(4) Identify practical questions. The fourth item of DECIDE corresponds to identifying issues related to the selection of users and materials to be used. The study population was professionals working on software testing projects at the Test Factory of GREat Lab. We selected two subjects for the pilot study and four for the final evaluation. Besides that, all tasks were performed in a controlled environment with the aid of a computer.

(5) Decide how to deal with ethical issues. The purpose of the fifth item concerns how to protect the privacy and other issues related to the participants of the feasibility study. At this point, test analysts were asked to sign a consent form to participate in the research and were informed about the purpose of the research, the data anonymization, and how it would be conducted.

(6) Data evaluation. The last item of DECIDE is about evaluating, interpreting, and presenting the data obtained during the evaluation. The performance of the users was evaluated by comparing the execution times of the tasks manually and with the tool's support. To answer the question regarding the positive and negative aspects of using the tool, the two evaluators analyzed and discussed the notes collected during the execution of the tasks and the answers from the form with open questions. Both results were combined to provide the final topics related to positive/negative points of the solution.

Regarding the context of this project, it was impossible to carry out evaluations with many users, so we did not use statistical tests. Instead, we presented the data and discussed the times obtained and the possible reasons that led to the results. Given our focus on assessing efficiency, we did not assess the correctness of the test cases produced, as coverage requirements and specification type may change for different projects.

7.2 Results

This subsection presents the results obtained after conducting the feasibility study. Before the final evaluation, we conducted two pilot tests based on the planning of Subsection 7.1 to make possible improvements. The tests were performed with a test analyst and a trainee in test analysis. After performing the tests, the evaluators detected inconsistencies in the use cases of the tasks that were promptly corrected. We also decided to reduce the size of the test cases to take less time from the professionals' work. Finally, we made a few adjustments using forms and other evaluation materials.

Table 5. Experience of participants.

Participant ID	Profession	Experience in software testing (years)	Experience with use cases (years)
User 1	Test Analyst Intern	2	2
User 2	Test Analyst	4	0
User 3	Test Analyst	4	1
User 4	Test Analyst	5	2

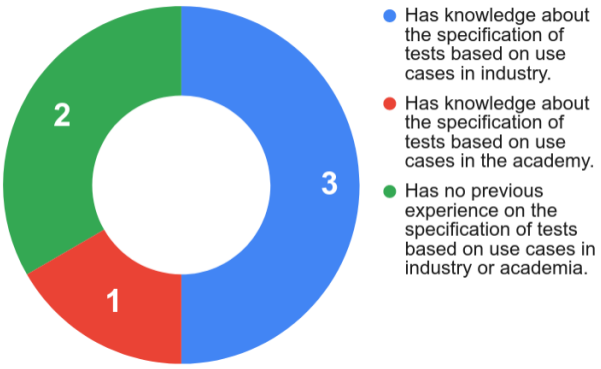

Figure 9. Participants experience in test case specification with use case.

After the pilot tests were executed, we selected four participants from a different project. All of those participants work as test analysts, but three of them were more experienced, and one of them was an intern. Table 5 summarizes the profile of the four professionals in the way that it details their experience with software testing and specification tests

based on use cases. Thus, as illustrated in the graph in Figure 9, most of the participants had some experience with specifying test cases in the industry, but none of them works with use cases in their current projects.

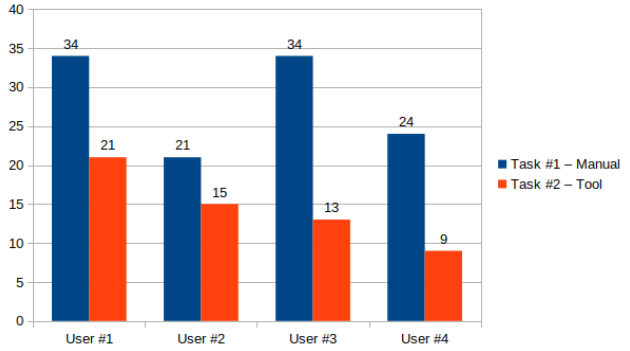

Figure 10. Time required in minutes to execute the manual and tool supported tasks.

During the performance of the manual tasks with and without the UC2Proc tool, we collected the total times per execution. Figure 10 presents a chart comparing the total times in minutes obtained by each of the four participants, in which all of them achieved lower specification time using the tool. While the average execution of the manual task was equal to 28.25 minutes, the average obtained with the use of the tool was equal to 14.50 minutes. Given the reduced size of the sample of participants that we obtained, we chose not to analyze statistical significance in the differences. Therefore, the answer to the first question of the feasibility study (*Was the tool able to increase the speed of tests specification when compared to the manual activity?*) gives more indications that the developed solution can increase the specification speed. However, more evaluations are needed to gather more data about its actual effectiveness.

After the activities were carried out, the participants were instructed to fill out an open questionnaire to point out the UC2Proc tool's positive and negative points. It is worth mentioning that the evaluators wrote down the participants' comments during the execution of the tasks. The form used by the evaluators is available in 6 of the Appendix A with the name *Evaluator - Tool Evaluation*.

The two researches that conducted the feasibility study performed a qualitative analysis of the questionnaire answers and users' comments during the evaluation. The evaluators used the notes to complement answers of the questionnaire about positive/negative points, so this results are presented together. To accomplish this, we grouped the most repeated and contrasting topics into the following categories about the tool: efficiency, utility, understanding, and visual acceptance. The content of these topics was then used to compose the list of positive and negative points presented below.

As positive points, all participants mentioned that they felt a reduction in the time by using the UC2Proc tool compared to manual activity. It means that the analyst's work could be streamlined. Half of the participants also found positive the integration of the tool with systems like TestLink and JIRA. This last comment may be related to the participants' working context so that they are used to working with this suite of systems to manage the testing activities.

The main negative point indicated by three of the four participants was about the titles generated by the tool. They were not very intuitive and could be difficult to understand during execution. This may happen because the UC2Proc tool creates the test case's title based on the titles of the last flow in the generated sequence. For instance, a test procedure with a flow sequence composed by Basic Flow, Alternative Flow and Basic Flow will have the title of the Basic Flow. This leads to test procedures with repeated titles.

Another negative point was the repetition of steps when the tool moves between different flows more than once. We observed it mainly from the Basic Flow to Alternative/Exception Flows. Still regarding the steps, two participants reported that it was necessary to correct some of them because there were system responses without user actions. It occurs because each system response has its own step in the generation process, even though they are in sequence. Finally, most of the users seemed confused when using the tool, once the buttons were not very intuitive and it offered little feedback on the actions. They also reported that some of the test procedures contained outcomes without steps, but this is how it is supposed to work, taking into account that the tool produces steps with only one outcome.

7.3 Limitations

Subsections 7.1 and 7.2 presented the methodology used to conduct the feasibility study and the results obtained, respectively. However, we identified some limitations in the evaluation that deserve to be discussed. The first limitation concerns the small number of participants, making it difficult to apply statistical tests to state the real difference between manual and automatic task times. Nevertheless, participants had varied experience with software testing in the industry, having already worked with different types of systems, tools for test process support, and types of requirements documents. Therefore, the selected group can be considered suitable for an initial evaluation outside the Software A project context.

Moreover, the participants used only fictitious systems documentation because of the confidentiality of the Software A. However, the use cases used are similar to the original documentation of the software. We also tried to create new use cases where they have complexities and interactions similar to the use cases from Software A.

Regarding the execution of tasks, all participants executed the manual task before the automated one. To mitigate this threat and reduce bias in this approach, the analysts used different use cases in both tasks.

Finally, during the feasibility study, some participants pointed out problems in the tool procedures. The main concern was about the repetition of some steps, mainly during the transition from Basic Flow to Alternative and Exception Flows. In its current version, the tool can incorrectly repeat some steps from the Basic Flow steps. Even so, we believe that it was not of high impact for the execution of the evaluation, considering that the repeated steps could be easily excluded.

8 Lessons Learned

As explained in Section 3, the performed study had activities related to the specification of the procedure. While using the tool during the Sprints, it was possible to obtain lessons regarding the solution and its use circumstances, so they are based on the researcher's observations, opinion of the requirements/tests team's members, and analysis of the collected metrics. These lessons represent some of the challenges obtained with the use of the solution in a way that actions taken during the Sprint were integrated into the process of using the tool. The main lessons learned during the process are listed as follows:

LL-1: The efficiency of a test case generator tool using use cases is strongly related to the following of the writing pattern. Use case modeling is a task that demands a high degree of instruction, communication, and knowledge about the software product. Being a manual activity, it is common to create certain textual documents susceptible to attention errors in the writing pattern. These errors could vary from problems in the spelling, plural, blanks in the markup characters (such as writing "[FA - 01] instead of [FA-01]") or even hidden logical loops. Such errors generated flaws in the tool and needed to be corrected, implying additional time in the process of semi-automated generation. Therefore, one action taken was to perform a detailed inspection to determine if the use cases follow the *template* of the project, thus making any necessary corrections. Having a use case with the correct pattern as an example helped analysts to identify inconsistencies more quickly and, consequently, enable the use of the tool. For teams working with poorly detailed documentation, the tool may not generate good results. However, it is possible to address in future work, more specific scripts for different types of projects and documentation.

LL-2: The deployment of an automation tool may not be worth the effort reduction. Throughout the design of Software A, the number of flows and steps was used to estimate the time needed to generate the tests. In some cases, this calculation inaccuracy is observed in use cases with a large number of flows, but that could be easily modeled manually or in an automated way. In this case, the tool has allowed a negligible reduction in efficiency gains, so the effort to adapt the use cases of a project to the presented *template* may not compensate, especially if the use case is straightforward. In these scenarios, the implementation of a semi-automated tool may require a great deal of manual work to adjust the use cases to a *template* and the generated procedures, which may impact the project activities. Nonetheless, for most of the use cases of Software A, there were indications that the time reduction for the test specification compensated the implementation of the tool in the Test Factory's context.

LL-3: Textual use cases do not express all the information necessary for good test coverage. In some use cases of Software A, it was not possible to automatically obtain all the necessary information to generate more test cases from the use case documentation. The reason for this is that business rules were expressed in unstructured natural language and screen prototypes were images. It prevented the extraction of some input variables for the procedures; the used use case patterns gave analysts freedom to specify. During the use of the tool, the test analyst needed to continue consulting the other documents during the analysis process and by that ensure the desired coverage.

LL-4: The integration of a solution with specific process tools is an important factor for efficiency gain. During the execution of a requirements/test process, the analysts may need to interact with different support tools to facilitate the activities' performance. Therefore, to facilitate the practical application of an automation tool, it becomes essential that the developed solution integrates with the other systems. For example, in this report, the analysts originally cloned the tests on TestLink, but the task generated many errors due to the lack of options for the test data and interface problems. In this sense, using the tool for specification activities and then submitting the tests to TestLink helped to decrease the errors.

LL-5: Generating additional tests for business rules were not advantageous in all cases. According to user reports, the implementation of the new functionality was advantageous since it signaled the business rules referenced in the case of use. On the other hand, three negative points were reported about the functionality. The first one is that a good part of the business rules could be covered with only one test case, generating less useful tests. The second point is related to use cases that were too specific and had detailed flows to the business rules; this way, it was necessary to remove the duplicate tests. Finally, the users needed to apply some effort to complement the test cases based on business rules, since only the title was generated.

9 Conclusion

This paper presented an experience report about the generation of test procedures in an industrial context. This paper is an extension of our previous work (Santos et al., 2019), whose the main goal was to analyze the feasibility of inserting a tool to automate the generation of tests based on use cases.

We implemented the solution in partnership with the industry, thus enabling the generation of a product that better suits the needs of the requirements/testing team, which leads to the question of this paper: "Is it feasible to use a tool to generate test cases from textual use cases in the test process at a test factory?".

Our previous results showed that the proposed solution positively contributed to the analysts' activities. Therefore, we have extended the current work through the following contributions: (i) improvements in the tool's generation of test procedures; (ii) data related to more testing cycles of Software A; and (iii) feasibility study with test analysts.

Regarding the tool's improvement, we realized that only indicating the procedures of business rules to be tested might not be sufficient. In such manner, we obtained low gain in the effort. Therefore, the results reinforced the need for specification of business rules in a structured manner.

When it comes to the tool usage in more releases, we concluded that the effort reduction in the test generation was maintained, as well as the relationship between the complexity of the use cases and the time spent in manual interven-

tion during the specification process. The reduction in effort equaled 65,38% in the context of the industry software project. Furthermore, the majority of the effort required was adjusting the test procedures generated by the tool.

In addition to the proof of concept, the feasibility study has provided further insight into the efficiency of the solution. Although all users completed the task more quickly using the tool, they pointed out interface issues that can make the software hard to use.

These evaluations also enabled the generation of one additional lesson learned regarding the generation of tests for business rules, which demanded additional effort to remove unnecessary tests. This set of lessons learned can give more information about the introduction of an automated tool in a testing process.

Considering the characteristics of Software A project, the team decided for the development of a simple custom solution. Nonetheless, finding the right degree to which the testing process had to adapt to the insertion of new tools was challenging. Regardless of the decision about the usage of custom solutions or other available solutions, we believe that more work is needed to provide practical insights in the context of test factories, which could benefit projects in distributed scenarios.

Concerning future work, we plan to research how to document business rules to increase the efficiency of generating test procedures. In the current work, this improvement has become even more evident, considering the perceived effort necessary to update procedures with partial descriptions of business rules. The test analysts of Software A also pointed out that several use cases needed corrections in its patterns. Since this hinders the usage of the tool, we also plan to apply techniques of static analysis in the requirements documentation. Finally, we intend to make improvements in the tool based on user comments and analyze how the test procedures can assist the generation of automated scripts for functional tests.

A Instruments of the Feasibility Study

Table 6 presents the following forms used in the evaluation: (1) Professional Profile, used to to collect the professional profile; (2) User - Tool Evaluation, filled by the participants to report the positive and negative points of the solution; and, (3) Evaluator - Tool Evaluation, used by the research to collect the time and general notes during the tasks.

References

R. M. d. C. Andrade, I. d. S. Santos, V. Lelli, K. M. de Oliveira, and A. R. Rocha. Software testing process in a test factory-from ad hoc activities to an organizational standard. In *ICEIS (2)*, pages 132–143, 2017.

B. Aragão, I. Santos, T. Nogueira, L. Mesquita, and R. Andrade. Modelagem interativa de um processo de desenvolvimento com base na percepção da equipe: Um relato de experiência. In *Anais do XIII Simpósio Brasileiro de Sistemas de Informação*, pages 428–435. SBC, 2017.

B. S. Aragão, R. M. C. Andrade, I. S. Santos, R. N. S. Castro, V. Lelli, and T. G. R. Darin. Testdcat: Catalog of test debt subtypes and management activities. In *Testing Software and Systems*, pages 279–295, Cham, 2019. Springer International Publishing. ISBN 978-3-030-31280-0.

A. Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2829-5. . URL `http://dx.doi.org/10.1109/FOSE.2007.25`.

R. M. de Castro Andrade, I. de Sousa Santos, V. Lelli, K. M. de Oliveira, and A. R. C. da Rocha. Software testing process in a test factory - from ad hoc activities to an organizational standard. In *ICEIS*, 2017.

D. Freudenstein, M. Junker, J. Radduenz, S. Eder, and B. Hauptmann. Automated test-design from requirements-the specmate tool. In *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, pages 5–8. IEEE, 2018.

V. Garousi and M. Felderer. Living in two different worlds: A comparison of industry and academic focus areas in software testing. *IEEE Software*, (1):1–1, 2017.

V. Garousi and M. V. Mäntylä. When and what to automate in software testing? a multi-vocal literature review. *Information and Software Technology*, 76:92–117, 2016.

T. Gorschek, P. Garre, S. Larsson, and C. Wohlin. A model for technology transfer in practice. *IEEE software*, 23(6): 88–95, 2006.

J. Gutiérrez, M. Escalona, and M. Mejías. A model-driven approach for functional test case generation. *Journal of Systems and Software*, 109:214–228, 2015.

ISO/IEC29119-2. Iso/iec/ieee international standard - software and systems engineering –software testing –part 2:test processes. *ISO/IEC/IEEE 29119-2:2013(E)*, pages 1–68, Sept 2013. .

D. N. Jorge, P. D. Machado, E. L. Alves, and W. L. Andrade. Integrating requirements specification and model-based testing in agile development. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 336–346. IEEE, 2018.

L. Lazic and N. Mastorakis. Cost effective software test metrics. *WSEAS Transactions on Computers*, 7(6):599–619, 2008.

J. L. Massollar, R. M. de Mello, and G. H. Travassos. Structuring and verifying requirement specifications through activity diagrams to support the semi-automated generation of functional test procedures. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*, pages 239–244. IEEE, 2012.

A. Mette and J. Hass. Testing processes. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*, pages 321–327. IEEE, 2008.

M. A. Montoni, A. R. Rocha, and K. C. Weber. Mps. br: a successful program for software process improvement in brazil. *Software Process: Improvement and Practice*, 14 (5):289–300, 2009.

Table 6. Forms of the feasibility study.

Form	Fields	Possible Answers
Professional Profile	1. What is your age?	Open
	2. What is your position?	Open
	3. How much time do you have in software testing?	Open
	4. Do you have previous experience on test cases based on use cases?	1. I have knowledge about the specification of tests based on use cases in the industry
		2. I have knowledge about the specification of tests based on use cases in the academia
		3. I have not previous knowledge about the specification of tests based on use cases
	5. How much experience do you have with use case based testing specification in the industry?	Open
User - Tool Evaluation	1. What are the positive points of the tool used in the evaluation?	Open
	2. What are the negative points of the tool used in the evaluation?	Open
Evaluator - Tool Evaluation	1. Time to complete task	Open
	2. General notes	Open

G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler. *The art of software testing*, volume 2. Wiley Online Library, 2004.

S. Nogueira, H. Araujo, R. Araujo, J. Iyoda, and A. Sampaio. Test case generation, selection and coverage from natural language. *Science of Computer Programming*, pages 84–110, 2019.

J. Preece, Y. Rogers, and H. Sharp. *Interaction design*. Apogeo Editore, 2004.

G. Samarthyam, M. Muralidharan, and R. K. Anna. Understanding test debt. In *Trends in Software Testing*, pages 1–17. Springer, 2017.

E. B. d. Santos, L. S. d. Costa, B. S. Aragão, I. d. S. Santos, and R. M. d. C. Andrade. Extraction of test cases procedures from textual use cases to reduce test effort: Test factory experience report. In *Proceedings of the XVIII Brazilian Symposium on Software Quality*, pages 266–275, 2019.

K. Schwaber and M. Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

S. Seela and R. Yackel. 64 essential testing metrics for measuring quality assurance success. URL `https://www.qasymphony.com/blog/64-test-metrics/`.

H. M. Sneed. Requirement-based testing-extracting logical test cases from requirement documents. In *International Conference on Software Quality*, pages 60–79. Springer, 2018.

S. S. Some and X. Cheng. An approach for supporting system-level test scenarios generation from textual use cases. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 724–729. ACM, 2008.

L. S. Vieira, C. G. L. Barreto, E. B. dos Santos, B. S. Aragão, I. de Sousa Santos, and R. M. C. Andrade. Automação de testes em uma fábrica de testes: Um relato de experiência. In *Anais do XIV Simpósio Brasileiro de Sistemas de Informação*, pages 80–73. SBC, 2018a.

L. S. Vieira, C. G. L. Barreto, E. B. dos Santos, B. S. Aragão, I. de Sousa Santos, and R. M. d. C. Andrade. Automação de testes em uma fábrica de testes: Um relato de experiência. In *Anais do XIV Simpósio Brasileiro de Sistemas de Informação*, pages 80–73. SBC, 2018b.

K. Wiegers and J. Beatty. *Software requirements*. Pearson Education, 2013.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

T. Yue, S. Ali, and M. Zhang. Rtcm: a natural language based, automated, and practical test case generation framework. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 397–408. ACM, 2015.