FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Generating Test Cases from Use Cases and Structured Scenarios with the RSL Language

Ana Catarina Leite Gomes

Mestrado em Engenharia de Software

Supervisor: Professora Ana Cristina Ramada Paiva

Co-supervisor: Professor Alberto Manuel Rodrigues da Silva

October 24, 2021

Generating Test Cases from Use Cases and Structured Scenarios with the RSL Language

Ana Catarina Leite Gomes

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

Chair: Professor Nuno Honório Rodrigues Flores
External Examiner: Professor María José Escalona Cuaresma
Supervisor: Professor Ana Cristina Ramada Paiva
Co-supervisor: Professor Alberto Manuel Rodrigues da Silva

October 24, 2021

Abstract

Requirements written in natural languages may lead to ambiguities and misunderstandings. One way to minimize these problems is by using controlled natural languages, such as Requirements Specification Language (RSL) that is used in this research.

This dissertation presents and discusses an approach to specify use cases and structured scenarios using the RSL language and generate test suites and test cases from those use cases and structured scenarios. Users are also able to manually refine and customize the generated test suites, consequently promoting the automation, re-usability and flexibility of this approach.

This investigation extends a previous end-to-end approach proposed by Maciel, Paiva and Silva. The proposed improvements concern the automatic generation of test suites from RSL specified requirements, using a full edges coverage algorithm, the manual refinement of the generated test suites and the implementation of reuse and sequencing mechanisms.

This approach extends the RSL design to be able to specify more details, namely use case scenarios and test cases with a sequence of concrete test steps. It also presents advanced mechanisms, such as data variables and setup and teardown blocks of actions, which allow the manual refinement of test cases in a flexible and reusable way.

All these improvements automated the test creation process, which reduces the effort and cost of repetitive tasks, also ensuring a higher quality of both requirements and tests.

The feasibility of the presented approach is illustrated and discussed based on an e-commerce web site properly designed for testing purposes.

Keywords: Requirements Specification, Test Case Specification, Use Cases, Use Case Scenarios, Test Cases Generation, RSL

Resumo

Requisitos escritos em linguagens naturais podem levar a ambiguidades e falhas de comunicação. Uma forma de minimizar tais problemas é com recurso a linguagens naturais controladas, como o Requirements Specification Language (RSL), empregue nesta investigação.

Esta dissertação apresenta e discute uma abordagem para especificar casos de uso e cenários estruturados usando a linguagem RSL e gerar suítes de teste e casos de teste a partir desses casos de uso e cenários estruturados. Os utilizadores também podem manualmente refinar e adaptar as suítes de teste geradas, contribuindo para a automação, reutilização e flexibilidade desta abordagem.

Esta investigação estende uma anterior abordagem *end-to-end* proposta por Maciel, Paiva e Silva. As melhorias propostas compreendem a geração automática de suítes de teste a partir de requisitos especificados em RSL, usando um algoritmo de *full edges coverage*, o refinamento manual das suítes de teste geradas e a implementação de mecanismos de reutilização e sequenciamento.

A abordagem estende o design da linguagem RSL para possibilitar a especificação de mais detalhes, nomeadamente cenários de caso de uso e casos de teste com uma sequência de etapas de teste concretas. Também apresenta mecanismos avançados, como variáveis de dados e blocos de ações de *setup* e *teardown*, que permitem o refinamento manual de casos de teste de forma flexível e reutilizável.

Estas melhorias contribuem para a automatização do processo de criação de testes, reduzindo o esforço e o custo de tarefas repetitivas, garantindo também uma maior qualidade tanto dos requisitos como dos testes.

A viabilidade da abordagem apresentada é ilustrada e discutida com base num site de *e-commerce* projetado devidamente para fins de teste.

Keywords: Especificação de Requisitos, Especificação de Casos de Teste, Casos de Uso, Cenários de Casos de Uso, Geração de Casos de Teste, RSL

Acknowledgements

First of all, I would like to thank my supervisor Professor Ana Cristina Ramada Paiva and co-supervisor Professor Alberto Manuel Rodrigues da Silva for their guidance, support and total availability during the last months.

I would like to express my appreciation to Critical Software for encouraging me to enroll in this course and providing me with conditions and support to successfully accomplish my goals in this master's degree.

To my friends for their support in overcoming the challenges during these two years.

A special thank to my colleagues with whom I have had the pleasure to work and were always available to advice and share ideas.

I would like to show my greatest appreciation to Diogo and Fiona who were also involved in this journey and were always present to share thoughts and motivate me in the good and bad times.

Last but not least, I deeply thank my family for their support and understanding throughout this journey. Without them it would have been even more difficult to successfully reach the end.

Ana Catarina Leite Gomes

"Life is too short for manual testing."

Harry Robinson
(test architect for Microsoft's Engineering Excellence Group)

Contents

List of Figures

List of Tables

Listings

Abbreviations

ATCGT	Automated Test Case Generation Tool
BDD	Behavior-Driven Development
BPST	Basic Path Sensitization Technique
CEG	Cause-Effect-Graph
CNL	Controlled Natural Language
CNLs	Controlled Natural Languages
CTT	ConcurTaskTrees
DAGs	Directed Activity Graphs
DFS	Depth First Search
DSL	Domain-specific Language
ER	Entity Reconciliation
FEC	Full Edges Coverage
FRs	FunctionalRequirements
GMF	Graphical Modeling Framework
GUI	Graphical User Interface
IFA	Interaction Finite Automaton
ITR	Integration Testing Rules
ID	Unique Identifier
LG	Link Grammar
MBT	Model-Based Testing
MBGT	Model-Based GUI Testing
MDE	Model-Driven Engineering
MDT	Model-Driven Testing
ME	Modeling Environment
NL	Natural Language
NLP	Natural Language Processing
OCL	Object Constraint Language
QRs	Quality Requirements
RE	Requirements Engineering
RSL	Requirements Specification Language
RUCM	Restricted Use Case Modeling
SaaS	Software as a Service
SSD	System Sequence Diagrams
SuT	System Under Test
SysML	Systems Modeling Language
TCSs	Test Case Specifications
TM	Text Mining
TSL	Test Specification Language
UCSs	Use Case Specifications
UCTM	Use Case Test Models
UML	Unified Modeling Language

Chapter 1

Introduction

The domain of this research fits in Requirements Engineering and Software Testing. This chapter presents the main problem of the project through the description of the context, motivation and the objectives of this dissertation.

1.1 Context

The successful delivery of a software product depends on how well the requirements are understood and transformed into relevant product features [67]. A requirement can be determined as a condition or capacity that a system must have to satisfy the system's stakeholders' needs [23].

A software requirements specification is a technical document that establishes an agreement between software developers and stakeholders, and shall exhibit a clear understanding of the requirements in an unambiguous way [55] considering two aspects [37]: the need to make the requirements document readable and liable to processing. These concerns will help in avoiding miscommunication, misinterpretation, and contribute to higher productivity and time and cost savings [37].

The most common way to write requirements is with natural languages; however, this may lead to ambiguities and inconsistencies. The literature shows that collecting unambiguous requirements is one of the most critical challenges in software engineering [8]. To overcome this problem, some researchers have used formal languages complemented with models and semi-formal methods; however, the use of formal languages adds complexity and requires mathematical knowledge that can increase the costs [21]. Even when formal (like B or Z) and semi-formal languages (like UML or SysML) are applied, there is always a need to use natural language sentences [42]. To get the best out of both solutions (i.e., of natural languages and semi-formal languages), some authors have proposed the use of controlled natural languages (CNLs) [43]. Its usage decreases the possibility of errors and misunderstandings during both the requirements and tests specification and may contribute to better validate these specifications. One example of a well-defined CNL for

requirements engineering (RE) is the RSL (Requirements Specification Language) [16], which allows specifying requirements and tests in a rigorous way, but still keeping these specifications human-readable. The support for the specification of both requirements and tests in an integrated way was the main reason for the adoption of RSL in this research.

1.2 Motivation and Objectives

RSL is a CNL that allows requirements engineers and testers to specify requirements and tests in a systematic and consistent way [17]. RSL includes a set of constructs logically organized in views that represent different RE-specific concerns (e.g., active structure, behavior, passive structure, requirements or tests) that can be applied at different levels of abstraction (e.g., at business, application, software, or even hardware level) [17]. Furthermore, RSL supports the establishment of alignments between requirements and tests, allowing the generation of test cases from use case scenarios. This automation process brings benefits by reducing the effort and cost of repetitive tasks, and ensuring higher quality of both requirements and tests.

In a previous work, Maciel et al. [46] [58] discussed an "end-to-end approach" that intends to generate test scripts from requirements and test specifications written manually in RSL. That approach is illustrated in Figure 1.1, and is a process with several tasks: *Task-1* consists in the manual specification of requirements. *Task-2* and *-3* consists in the manual specification and refinement of tests. *Task-4* involves the automatic generation of tests scripts from the tests specifications with RSL; in the reported experience, these test scripts were written in the Robot Framework's [1] language. *Task-5* is a manual and time-consuming activity that requires the mapping of the concrete GUI elements (of the system under test, SuT) with the elements defined in the test scripts. Finally, *Task-6* consists in the tests' execution against the SuT and produces a test report with the results.

Despite it being an ambitious and complex process, we identify some limitations with that approach. First, the *Task-3* could be automated to some extent, and, hence, could show a higher productivity level. Second, to achieve that purpose, we shall also extend the design of the RSL to be able to specify more details, namely use case scenarios, and test cases with a sequence of concrete test steps. Consequently, and as suggested in Figure 1.1, the focus and the contributions of this work are to research how to extend and improve these new tasks of that end-to-end process [46] [58], specifically: (*Task-2a*) Generate test cases from use case specifications; and (*Task-3a*) Refine the tests and data generated.

Chapter 5 illustrates the specification of requirements and tests, as supported by the RSL language. To better support the explanation, a running example is used in this dissertation, which is a fake e-commerce website named "MyStore" particularly designed to support test automation practices[2]. This "MyStore" application includes features that allow its users to register, authenticate, create and update products, create orders of products and checkout, etc.

[1] http://robotframework.org
[2] http://www.automationpractice.com

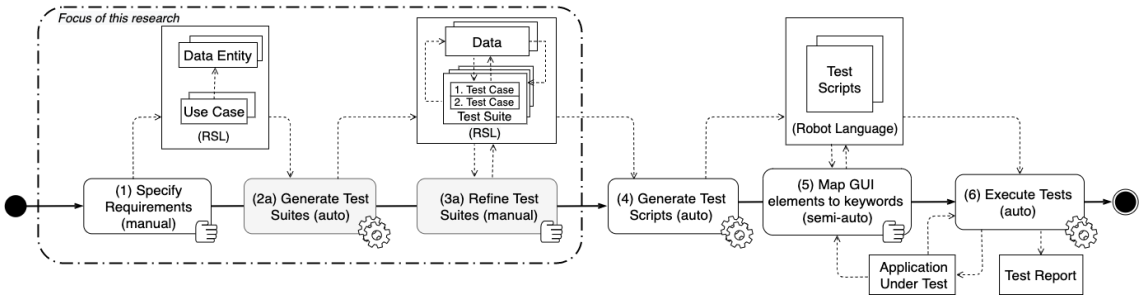

Figure 1.1: Activities diagram of the proposed RSL approach.

1.3 Structure of the Dissertation

This dissertation, besides this introduction, is structured as follows: Chapter 2 introduces the requirements engineering in the software development process and the advantages and challenges of RSL. Chapter 3 presents the State of the Art where the fundamental concepts of the scope of the dissertation are analyzed and studied. Chapter 4 discusses the proposed approach in what regards the tasks *Task-2a* and *Task-3a*. Chapter 5 presents the RSL elements for the specification of requirements (use cases and use case scenarios) and tests (test suites, test cases, test steps and variables). Chapter 6 describes the process of generating test suites with test cases, concrete test data and variables from use case specifications. Chapter 7 evaluates the proposal's test case generation process with a checkout use case example of an e-commerce application. Chapter 8 presents the conclusions and some open issues.

Chapter 2

Background

This chapter presents the importance of requirements engineering in the software development process. The challenges in requirements and testing specification are overviewed and the advantages of using RSL are presented.

2.1 Requirements and Tests Engineering

The initial activity in the software development process is the requirements engineering (RE), which consists in a set of tasks where the business stakeholders translate ideas or views into a requirements document [22].

A requirement can be defined as a condition or capacity that a system must have to satisfy the needs of its users [73]. There are several suggestions of standards requirements classifications, such as CMMI [76] or SPICE [39]. Pohl [60] classifies 3 types of requirements: First, the functional requirements that describe how a product must behave, what its features and functions are. Second, the quality requirements, also classified as non-functional requirements, that describe the desired characteristics of a system to be developed, such as performance, availability, dependability, scalability, or portability of a system. And third, the constraints requirements that describe the limits or boundaries of the system and that cannot be implemented.

RE is an iterative co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained [60]. During this process, the system requirements are determined, although it must be perceived that the requirements engineering process is iterative, since the requirements change frequently during the development process. Therefore, requirements often have to be refined or added during the development process.

The RE process includes four core activities, such as elicitation, analysis, documentation (specification), and verification and validation. During elicitation, the requirements are elicited from stakeholders and other sources, to refine the requirements in as much detail as possible.

These requirements are analyzed in the second phase for consistency, feasibility, incompleteness and ambiguity. Through the documentation phase, the requirements are described accurately, and different techniques are used to document the requirements by using natural language or conceptual models [60]. Next, the requirement's document is validated against the correctness and integrity of the requirements for the customer needs. To guarantee the quality criteria is met, documented requirements must be validated and negotiated at an early phase. The validated requirements document is the input for development system and the writing or the generation of system test cases. The last phase is requirements management, which ensures the consistency of requirements after changes in order to guarantee their implementation [60]. The process does not need to strictly follow these phases, since the activities in the requirements engineering process depend highly on the design, product and characteristics of the company [60].

Requirements and testing play an important role at different stages of the software development life cycle. There is a traditional separation between these two activities, consequently, it is not common to start the testing activity at the beginning of the software development process. However, most of the errors identified in the testing phase have already been introduced in the requirement or design phase. Therefore, the defects identified later in the software development life cycle are more expensive to fix than those identified in an earlier stage. The testing activity should start early to avoid the introduction of defects by supporting the specification of tests and also by generating such tests from requirements in the early phase. Consequently, starting testing in the requirement's phase can be cost-effective and useful to avoid bugs afterwards, and it helps to identify and quantify the problem's scope.

2.2 Requirements Specification

There are various techniques for specifying requirements such as natural language, formal specification languages, data flow diagrams and use cases.

The Software Requirement Specification document is the basis for an agreement between software developers and stakeholders and has to show a clear understanding of the requirements in an unambiguous way [55].

In this dissertation, the focus is the user requirements, natural languages and use cases, which are more suitable than formal specification languages, since all the stakeholders can easily understand the requirements.

The requirements are treated as input in the design, implementation and validation phase of software product development, for this reason, it is necessary to balance two important aspects during the writing of the requirements document. Firstly, the need to make the requirements document readable and, secondly, the need to make requirements documents liable to processing [37]. This concern will help to avoid miscommunication between stakeholders and misinterpretation of requirements. In addition, it will make the agreement easier and quickly among stakeholders and software developers regarding the implementation of what is written in the Software Requirements Specification. Making the agreement process faster, it will lead to higher productivity and

contribute to time and cost savings. According to Hull et al. [37], the specification language must meet the two criteria mentioned above to increase productivity and produce accurate queries to software developers. Some literature claims that collecting ambiguous requirements is one of the critical challenges in software engineering [8].

2.3 Requirements Specification Language

The Requirements Specification Language (RSL) is a CNL that proposes to specify software requirements in a systematic and consistent way [17]. RSL can be used and adapted to different contexts of the software engineering process [17]. RSL proposes that some tasks that are part of RE related activities can be automated, because it is not coupled to a specific process or tool, namely [21]:

1. Domain analysis: The relevant concepts and their relationships can be defined using the specification language and respect the defined constraints, while the state of the element's changes by the functionalities of the software.

2. Verification: The consistency checking on extracting the domain knowledge from the business context based on glossaries and general lexical resources can allow finding errors and performing rework to fix them due to the ambiguity or lack of clear definition.

3. Transformations: The artifacts of the requirements' representation such as diagrams, tables, reports and template-based requirements statements in CNL related to the specific perspectives. Silva et al. [21] mentioned that an initial draft of the domain and behavior models such as UML class diagrams and UML use case diagrams can be generated based on the defined specification in RSL and be used as input to the software development processes according to Model-Driven Engineering (MDE) paradigm.

RSL is logically organized in two perspectives, as illustrated in Figure 2.1 [17]: abstraction levels and the specific RE concerns. The views are defined according to the abstraction levels: business and system levels; and to the concerns: context, active structure, behavior, passive structure and requirements [17].

For the business level, RSL can specify the following business concerns: (1) the people and organizations that is related to the system; (2) business processes, events, and flows; (3) definition of the common terms used in the business domain; and (4) the general business goals of stakeholders. This level contains, respectively, the following views: Stakeholders (active structure concern), BusinessProcesses (behavior concern), Glossary (passive structure concern), and BusinessGoals (requirements concern). And the references to the systems used by the business and their relationships can also be defined at this level (context concern) [62].

For the system level, RSL supports the specification of following RE concerns: (1) describe the actors that interact with the system; (2) describe the behavior of some system's data entities (state machines) (3) describe the structure of the system, namely based on data entities and data

System (isFinal vs isReusable)	Concerns / Abstract Levels	Active Structure (Subjects)	Behavior (Actions)	Passive Structure (Objects)	Require-ments	Tests	Other	Relations & Sets
	Business	Stakeholder	ActiveElement (Task, Event)	DataEntity	Goal QR Constraint FR UseCase UserStory	Acceptance CriteriaTest UseCaseTest DataEntityTest StateMachine Test	GlossaryTerm Risk Vulnerability Stereotype IncludeAll IncludeElement	SystemsRelation Requirements Relation TestsRelation SystemElements Relation ActiveFlow SystemView SystemTheme
	Application	Actor	StateMachine	DataEntityCluster				
	Software			DataEnumeration				
	Hardware							
	Other							

Figure 2.1: Classification of RSL constructs: abstraction levels versus RE specific concerns.

entity views; and (4) specify the requirements of the system according to different styles. This level contains respectively the following views: Actors (active structure concern); StateMachines (behavior concern); DataEntities and DataEntityViews (passive structure concern); and multiple types of Requirements such as SystemGoals, QualityRequirements (QRs), Constraints, FunctionalRequirements (FRs), UseCases, and UserStories (requirements concern). All these elements and views should be defined in the context of a defined System (context concern) [62].

RSL can increase the consistency and reduce the ambiguity in a software development process. It also provides a model that allows the validation among the generated elements and the possibility to automatically generate others RE artifacts. Regarding the software testing activities, some artifacts can be generated based on the elements of the specific domain model defined using RSL.

RSL has been developed using the Xtext framework, promoting a rigorous specification and the possibility to validate and automatically transform it in multiple representations and formats [62]. Xtext is a framework that allows the creation of specific programming languages and domain languages with the usage of grammar, that helps to create a structured language. The Xtext-based format can be written using the Eclipse IDE, using a specific plugin, thus having the modern benefits of an IDE such as immediate feedback, incremental syntax checking, suggested corrections, and auto-completion [1].

The RSL specification allows testers with no software development background to specify tests in an easier way, but maintaining rigor and formality [17].

2.4 Tests Specification

The test plan has a set of documents appended, where are defined the test cases needed to execute the test items designated in the test design specification. This document has several components that IEEE standards demand to appear in a specific order. Also, if some contents of the test case specification appears in other documents, they should be referenced [38].

[1] https://www.eclipse.org/Xtext/index.html

When a test case is specified correctly, there is no time wasted analyzing the results of an incorrect test result. Since, the development of test software and test documentation represent a large investment of an organization's resources. These documents should be kept in a test repository, because they can be used again by other teams for testing or can be reused in regression testing to test further releases of a product or related products.

The test design specification must be carried out in a thoughtful and organized way, because it can support testing in the actual project and for reuse of other projects.

2.5 Test Specification Language

Test Specification Language (TSL) is a controlled specification language that applies the black-box functional testing design technique for writing human-readable and computer-executable test cases. TSL is inspired by RSL, which is a rigorous requirements' specification language that allows specifying requirements based on defined grammar, nomenclature and writing style [17].

TSL allows performing the following types of testing [17]: (a) domain analysis testing that uses equivalence partitioning and boundary value analysis for the definition of data structure that represents the domain elements of the software. (b) use case testing that allows the creation of test cases based on the use case scenarios. (c) state machine testing that allows to test the possible states of the state machine based on a coverage criterion. (d) acceptance criteria that allows to perform a testing strategy based on GWT (Give, When, Then) structure which is used in Behavior-Driven Development (BDD).

There are some previous works that applied the TSL approach to generate tests from requirements written in RSL. Silva et al. [18] work propose the use of TSL, which is a tool that implements the Model-based testing technique that organizes the tasks from the software testing activities and can also automate some of them. The authors [18] propose the TSL approach to support the specification and generation of software tests defined using TSL. These specifications can be used as an input to generate other TSL specs according to predefined strategies. These specifications can be used for documentation purposes as well.

The use of RSL and TSL seems to be a sustainable approach to improve the test cases' generation process, because it provides a structured language to specify the elements that are related to each other. Furthermore, TSL supports techniques such as equivalence class partitioning and boundary value analysis. These techniques are more effective to find bugs in the software.

2.6 Software Testing

The software testing activity is critical to ensure the quality of software systems [78].

Testing belongs to the validation activity and is applied through the entire development life cycle. The test cases are generated based on the requirements documents, and therefore, it is important that the requirements are clearly written so that the test cases can be generated efficiently and effectively.

The goal is to create test cases based on the requirement documents and run them against the product in order to check if the generated results are the expected or not. To perform this phase, it is necessary an executable product and set of test cases. The test cases are an aggregation of inputs, execution conditions, and expected results developed for example to execute a path program or to verify a specific requirement. Jalote [41] defines testing as a quality assurance technique applied through software development to find requirements with inconsistencies, design and coding errors in the programs. In his turn, Binder [9] states that software testing is the execution of code using a combination of input and state to detect bugs. Software testing is not limited to discovering design and coding errors, but it can detect other kinds of defects.

Furthermore, the testability attribute is related to the effort needed to test a software system to ensure it performs its intended functions. A quantification of testability could be the number of test cases required to adequately test a system, or the cyclomatic complexity of an individual module.

The disadvantage of having lower results of testability is the increased testing effort and, consequently, less tests performed in a certain period of time and, therefore, the probability of finding defects is smaller [78]. According to Bolton [2], anything that makes testing harder or slower gives bugs more time or more opportunities to hide.

The advantages for software engineer of being able to increase the testability of the software are: cost reduction, increased quality of testing activities and, consequently, to produce higher quality software.

2.7 Conclusion

Despite being uncommon to start the testing activity at the beginning of the software development process, starting testing in the requirement's phase is a cost-effective strategy that can reduce defects and enhance software quality.

Requirement documents are a crucial input to the design, implementation, and validation stages of software development, being essential to ensure they are unambiguous, readable and liable to processing [37].

Generating test cases from clearly written requirement documents is a time and cost-effective solution. Using RSL improves the test cases' generation process, contributing to find more bugs in a software.

[2] https://www.developsense.com/blog/2009/07/testability/

Chapter 3

Related Work

This chapter introduces the state of the art about test cases generation from functional requirements written in natural languages and analyzes the existing approaches. It addresses two main topics related to the work to be carried out: the methodologies and techniques that have been used to generate test cases from functional requirements written in NL, and identifying their main advantages and disadvantages. The analysis of each approach will be presented following the methodology proposed by the author and also relevant definitions that are important to understand the context and the methodology.

Several sets of keywords were used by combining relevant terms in test case specification, test case generation and requirements to tests. Some examples are: “requirements specification”, “requirements specification language”, “requirements-based testing”, “requirements to tests”, “model-based testing”, “test case specification”, “test cases generation”, “acceptance tests”, “use cases”, “use case scenarios”, etc. These terms were used in specific and general search engines such as Google Scholar, Scopus, IEEE, ACM, B-On and Engineering Village.

3.1 Test Cases Generation Techniques

An important aspect to achieve a successful development system is to have high-quality requirements [3]. There are several published approaches of how to ensure that conditions on high quality requirements are fulfilled, like templates and style guidelines for requirements documents, inspection of requirements, system and acceptance testing.

Requirements engineering and software testing are different tasks in the software development process. Despite being considered conceptually distinct tasks, there are relationships among requirements and tests. The requirements document, along with code documents, is the most important input for test engineers, because the requirements define how the product to be tested should behave. Therefore, the test engineers generate the product’s test cases from all the relevant documents that include information about the system requirements.

This investigation is an overview of the current work carried out by academic researchers and the industry on methodologies and techniques that allow to derive test cases from the system requirements, and also to provide requirements that allow to generate test cases automatically or in an easy and simple way.

The focus of this work is to identify different approaches and techniques to generate test cases from functional requirements written in CNLs.

There are some strategies that generate test cases based on natural language [11] [25], others from models, such as use cases [34] [71], execution traces [59] [69], task models [68] [5] and patterns [49] [50, 51, 52].

3.1.1 Test Cases Based on Natural Language Requirements

The requirement phase includes a set of activities, which help to specify the impact of the software on the organization, customer needs, and how users will interact with the developed product [1]. Requirements are the basis of the system design and if they are not correct the end product will also contain errors [38]. Most of the costly errors are usually created in the requirements phase as a result of confusion between stakeholders or poor requirements [12] [45]. To detect as many errors as possible during the requirements phase, iteration between stakeholders through clear communication in natural language must be supported.

Requirements are written in Natural Language in most of the projects. Research indicates that 79% [44] of the requirements are described in natural language and only 7% [54] apply formal specifications. While the use of NL to describe requirements is known to be ambiguous, it is used because it simultaneously facilitates understanding and sharing [54].

Some literature addresses the problems of specification of requirements in NL due to the inconsistency, incompleteness and ambiguity characteristic of the language itself. First, it can increase misunderstandings between stakeholder needs and, consequently, generate unnecessary conflicts. Second, it is the lack of structure from NL and testability of requirements that makes automation of testing tasks difficult.

The first overview is based on generating test cases from functional requirements in natural language [25]. Dwarakanath et al. [25] propose a parser tool that generates test cases written in unconstrained natural language. The tool, called Litmus, is implemented as a plug-in into Microsoft Word and Excel using .NET, and allows users to generate test cases from functional requirements written in English. The Litmus tool analyzes a sentence, marks it with a label for traceability purpose, and then generates one or more test cases from it. The tool uses a "Link Grammar parser" (LG) to analyze and convert the requirements descriptions. LG analyses the text requirement using the English grammar syntax to extract the connection between the words and convert them to labeled *links*. The output information created through this process allows users to extract entities and then generates test cases. Throughout this process, the entities extracted can be validated by the user, increasing the accuracy of the results. To be able to generate a test,

[1] https://ecomputernotes.com/software-engineering/softwarerequirement

Dwarakanath et al. [25] define the test intents template with the necessary information from a sentence requirement.

The tool was evaluated through the test cases generated. Dwarakanath et al. [25] state that the tool has a positive impact as it brings value in the Test Design phase of the software development life cycle. However, the tool presents some problems generating accurate test cases when the text requirements were incomplete or not understandable. Also, requirements with unintelligible values failed (e.g., "Verify the system is able to send one-second block.") and there are problems caused by the lack of context and by being domain agnostic (e.g., "Verify lot number combinations are unique.").

There are other similar works that generate test cases from requirements written in NL.

According to Boddu et al. [11], the major causes of failures of software projects are the poorly organized and rapidly changing requirements weakly related to the users. Another important issue is the creation of requirements documents and document templates written in NL, which is a highly labor-intensive and skill-intensive task prone to errors. Nevertheless, the authors [11] highlight the important role of the use of NL in requirements documentation and reinforce that it is unlikely to be replaced since it is the best medium of communication understood by users and engineers. However, as Dwarakanath et al.'s [25] approach has claimed, Boddu et al. [11] are also aware that the requirements suffer from several problems [75]: ambiguity and impreciseness, inaccuracy, inconsistency and incompleteness.

The authors' [11] approach tries to solve the requirements engineering problem arising out of imprecise and inaccurate NL requirements using natural language processing (NLP) and text mining (TM) technology for analyzing requirements written in NL.

Boddu et al. [11] developed a requirements' analysis tool, called RETNA, which receives the requirements in NL and classifies them according to complexity measures. Then, the tester refines the requirements to check their consistency and generates the test cases.

The authors [11] state that it is difficult to generate test cases from requirements in a raw natural language form. Therefore, it is necessary to extract from the NL requirements a logical description that allows an automatic classification of the requirements based on its type and complexity (e.g., number of nouns and verbs). Thereafter, it may be necessary for the tester to refine the information by confirming the consistency of the requirements. At the end of the process, it is possible to construct a model from which test cases can be extracted automatically. The tool has been evaluated through several case studies that reveal good results. RETNA can be used as an automatic testing tool, and the input and the output specifications can both be written in English. The authors [11] state that the tool performance depends on the user interactions and the quality of the NL requirements. Although RETNA is not a complete tool, Boddu et al. [11] claim that NLP tools can become part of the requirements of an engineer's life.

3.1.2 Model-Based Test Case generation

Model-Based Testing (MBT) is a black box technique which allows models to be built, so predictions about the system can be made, enhancing the abstract comprehension of a system's behavior.

In MBT, the test cases are derived from the model that represents the system under test. The goal of MBT is to develop a model with all the necessary information from the elements of the software under testing and to be able to generate test cases in order to detect bugs. To develop the model, it is important to include the requirements that are usually written in NL. This model represents a more structured way to specify the requirements, avoiding the ambiguousness of NL. These models can be defined using modeling languages, such as Unified Modeling Language and Business Process Model Notation. These languages can have either a textual or a graphical representation.

Therefore, MBT is a relevant technique, since it allows generating test artifacts based on requirements with the advantage of decreasing the ambiguity and helping to connect the elements and its information to create and execute the test cases.

3.1.2.1 Test Cases Derived From Use Cases

System testing allows us to verify the behavior of the system under test and to guarantee the satisfaction of its requirements. The majority of the analyzed approaches propose that use cases should be used for deriving test cases. A use case is composed of events, which describe the behaviors of a system without exposing the system's structure. Consequently, the test cases are generally derived from flows of events, including the basic and alternate flows.

Use cases were first proposed as a method to document the functionality of a planned or existing system based on simple models [40]. The use case approach is based on two concepts that are used simultaneously: use cases diagrams and use cases specification. UML case diagrams are simple models for documenting schematically the interconnection of a system's functions and the bond between those functions and their environment from a user's perspective [57]. UML also describes a relationship among use cases or actors [64]. The use case specification advantage over the use case diagrams is that it allows the documentation of any information about use cases, such as the systematic interaction between a use case and an actor. This information is documented textually using templates along with use case diagrams.

Cockburn [14] proposes the application of different templates for textual specification of use cases. The requirement's text description is written in structured natural language using a template. The application of the structure will minimize the disadvantages of writing requirements in NL.

Gutiérrez et al.'s [33] approach suggested creating a process based on the Model-Driven paradigm to generate test cases from functional requirements. The authors [33] propose to accomplish the objective by applying Model-Driven Engineering (MDE) and Model-Driven Testing (MDT). First, the software paradigm is founded on the design and use of models to create software artifacts, and second, it assures that the requirements specified are properly supported by the final system [7]. According to Gutiérrez et al. [33], creating an approach based on MDT allows defining models, transformations and processes which reduce time and allow traceability between requirements and tests.

As a difficulty in generating test cases, the authors [33] identify a lack of formalism in functional requirements. However, when the tester tries to automate the process from them, the task becomes more onerous due to the formalism of functional requirements, and therefore, it takes

comparatively more time and knowledge, hence increasing the costs. Nonetheless, when the formalism in functional requirements' problem is overcome, a favorable return on investment is reached.

Gutiérrez et al.'s [33] approach presents the test case generation process using metamodels and transformations following the MDE perspective. During the process, the metamodel for testing artifacts is generated as textual templates, UML diagrams or other syntax. The goal of the transformation process is to obtain a model from functional requirements to test scenarios. The authors [33] developed a tool to perform this process automatically. Gutiérrez et al. [33] suggest that applying MDE without a software tool to automate the process is a tedious and expensive task to be executed by the tester.

Gutiérrez et al. [33] claim that this approach has advantages over other approaches, since it does not need specific metamodel notation. On the other hand, this metamodel can be adapted and applied to another approach to generate the same process. The authors [33] suggest devoting a greater effort to the quality of requirements to maintain the quality of the products and also seek greater involvement with the system as users to detect mistakes and flaws in the implementation.

Chen & Li [13] suggest that it is necessary to build a model to automate test case generation from specifications. In this approach [13], the SSD editor is used to specify use cases with System Sequence Diagrams (SSD). The authors [13] developed a prototype software called Automated Test Case Generation Tool (ATCGT) to generate test cases from use cases. The model semantics of Interaction Finite Automaton (IFA) are derived from the classification of interactions between actors and a system. The editor in textual mode prepares the input to interact with IFA modeling. When the SSD is submitted, relevant IFA is modeled automatically. Then, this system implements the transformation and a textual representation of the IFA is generated for users to verify. At the end of the process, the test case generator executes the algorithm and generates a set of test paths. With the semantics supplied, each test path is mapped to an actual test case.

Chen & Li [13] claim that this approach is efficient and effective to generate a near optimal test case and can obtain a high test coverage in the early development software stage.

Wang et al. [79] discuss how to generate executable acceptance tests by exploiting behavioral information in use case specifications. In safety critical systems, the acceptance testing is performed to ensure that the software system meets what was defined in the functional requirements expressed in NL. According to the standards defined by this industry, each requirement should be validated by this testing in a clearly traceable way. These requirements specifications are usually large and described in NL as use case specifications, therefore the process of acceptance test case generation is very expensive and error-prone. The authors [79] propose a method, called Restricted Use Case Modeling (RUCM), that introduces a template with keywords and restriction rules to reduce ambiguity in requirements. This method includes an advanced NLP solution to generate Object Constraint Language (OCL) constraints that capture pre and postconditions of use case steps. The Use Case Test Models (UCTM) captures the control flow implicitly described in a RUCM specification and enables the model-based identification of use case scenarios, which significantly reduces time and manual human effort.

Zhang et al. [80] proposed an approach that performs an automated execution of test cases from use cases. The use cases and the test cases are specified in restricted natural language using a defined template. The authors [80] suggest an automated approach based on the Restricted Use Case Modeling approach (RUCM), to systematically and automatically derive Test Case Specifications (TCSs) and test cases from requirements. RUCM is based on a template and a set of restriction rules for textual Use Case Specifications (UCSs). Zhang et al. [80] use the template and restriction rules to reduce imprecision and incompleteness in UCSs. A RUCM Use Case has a basic flow and one or more alternative flows, depending on whether a condition occurs. The conditional logic sentences are specified through a group of keywords, such as "IFTHEN-ELSE-ELSEIF-ENDIF". They also facilitate the automated generation of the test cases.

The transformation from RUCM specifications to TCSs and test cases are implemented in aToucan4Test tool. This tool generates a TCS for each use case.

Zhang et al. [80] list some advantages of this approach as the UCSs being described more systematically and test cases being generated automatically. Using RUCM/ RTCM, the use cases/ TCSs can be specified precisely and are understandable by stakeholders. Therefore, specifying use cases with RUCM provides an easier way for modeling Behavioral Models for Testing, reducing dependence on domain experts. Consequently, test case maintenance is easier when compared to the common practice.

Hamza & Hammad [35] discuss how to convert UML use case diagrams into customized activity diagrams. Then, these activity diagrams are converted into directed activity graphs (DAGs), from which test cases are extracted. However, this technique does not deal properly with the order in which test cases should be executed because of existing dependencies among them.

Alrawashed et al. [2] and Somé & Cheng [72] apply Cockburn's template [14] to specify use cases using a structured natural language. The use case steps are represented by nodes, and edges represent the sequence among them. Then, the steps are converted to create a control flow. After creating the control flow, the two approaches diverge. Alrawashed et al. [2] generate test cases from the control flow diagram with a genetic algorithm to optimize and evaluate the adequacy of generated test cases. On the other hand, Somé & Cheng [72] generates test cases using depth-first traversal of the control flow but in a not completely automated manner.

Fischbach et al. [27] translate acceptance criteria into a Cause-Effect-Graph (CEG) to automatically derive the test cases. First, each acceptance criterion is converted into a dependency tree. Then, the algorithm traverses such trees to extract the causes and effects. Finally, the algorithm derives the minimum number of test cases from the CEG by applying the Basic Path Sensitization Technique (BPST) [27]. However, the full generation of test cases, from acceptance criteria, can hardly be achieved and, so, according to some practitioners, this approach should be seen as a supplement to the existing manual process [27].

Silva et al. [58] present an approach to improve the definition and the validation of the requirements' specification in RSL. According to Silva et al. [58], RSL is logically organized in two views: the abstraction level and the specific RE concerns/ behavior. These views are defined by levels of abstraction: business and system levels; and concerns: context, active structure,

behavior, passive structure and requirements. The business level supports the specification of the business-related concerns and includes the views of Stakeholders (active structure concern), BusinessProcesses (behavior concern), Glossary (passive structure concern), and BusinessGoals (requirements concern). The system level supports the specification of RE concerns/ behavior and contains different views, such as Actors (active structure concern), StateMachines (behavior concern), DataEntities and DataEntityViews (passive structure concern), and multiple types of Requirements. The requirements types can be split by: SystemGoals, QualityRequirements (QRs), Constraints, FunctionalRequirements (FRs), UseCases, and UserStories (requirements concern). These elements and views are defined in the context of a specific System (context concern).

In RSL, the StateMachines view determines the behavior of DataEntities in their relationships with use cases. A StateMachine contains various states depending on DataEntity conditions which may occur during its life cycle. Some actions are defined through DataEntity status. Furthermore, use case actions can occur on a given DataEntity state, and trigger a transition to the next state.

In summary, the goal of this approach is to support the specification and generation of software tests defined in TSL through the requirements' specification defined in RSL. RSLingo supports a set of strategies to generate test cases from RSL constructs and allows the automation of the process. This process is supported in Xtext format with the integration of the Eclipse IDE tool.

The UseCaseTestCase results are a consequence of the sequences of steps defined in RSL use cases' scenarios, and also by adding input values to the data entities. A use case is a description of a specific use of the system by an actor. They help to define the functional system requirements by describing the process flow of the system. For each use case, one scenario and zero or more alternative/ exceptional flows are created. Use Case tests are derived from the various process flows expressed by a RSL use case.

The authors [58] state that it is relevant to automate processes for TSL test case generation, such as generating TSL test cases from other RSL requirements specifications. Silva et al. [58] conclude by exploring the automation of other processes, such as the extraction of test scenarios based on the flows expressed by use cases.

3.1.2.2 Test Cases Derived From Execution Traces

From different input requirements, there are several ways to obtain models to generate test cases. One way is the usage information of the services, that is, the tool captures the user behavior going through the website and saves it as data. Besides providing information, websites are also crucial to enterprises, since they support applications for business development through business tools. Therefore, these websites need frequent maintenance and improvement in order to adjust the client demands [63].

The generation of test cases is usually based on models of the software under test. The tools to generate test cases usually use models. However, if the models do not exist, and if they are outdated or if it is challenging to update them, there is a need to resort to other artifacts [69]. Beyond that, during software maintenance, it is important to continuously adapt the test cases to the changes or improvements in order to validate new versions of the software.

Silva et al. [69] and Paiva et al. [59] propose a generation of test cases based on registering the user interactions, but with different approaches. Paiva et al. [59] suggest generating test cases from user execution traces, which is an alternative to deal with the absence of models, and Silva et al. [69] generate test cases from the usage information of the services.

Paiva et al. [59] present a similar process to the web analytics tool to capture the user execution traces, which are later analyzed to extract a subset that is representative according to some criteria, and consequently, generate the test cases from the user interactions. The goal is to apply a set of mutation operators over the initial test cases to generate new ones that exercise different features of the application under test.

Whenever a test case passes the defined criteria, it is selected and added automatically to the initial set of test cases, enriching the test suite. Each test case contains a sequence of steps performed by the user, which are stored in a JSON file. These steps represent the user actions on a web element and contain the necessary information to reproduce it. The test cases are generated through the following process. First, the user execution traces are saved in a Neo4j database, and after that, the abstract test cases are extracted as JSON files and transformed into concrete test cases using random data generators. The mutation operators are applied to generate the candidate test cases, and subsequently each test case is converted into Java and executed using the Selenium tool. Thereafter, the result is compared to the baseline and if the test case is similar to any of the existing one, according to a certain metric, it is discarded and not added to the test suite.

Paiva et al. [59] validate the approach through two case studies: First, the authors [59] assess if the augmented test cases were able to detect the problems they aimed to address, and second, they measure the effect on the mutation test cases' number when varying the number of real test cases and the effect on the relevant test cases when varying the Levenshtein distance. The authors [59] conclude that it is possible to see that the number of candidate test cases varies linearly with the increasing number of initial test cases. Furthermore, Paiva et al. [59] verify that Levenshtein distance has an impact over the subset of candidate test cases that are considered relevant.

Silva et al. [69] present an approach to automatically generate test cases from the usage of Software as a Service (SaaS) through the REQAnalytics tool [29] [32]. This tool captures the usage information of the services, analyses it and produces a set of reports in a language more close to the business and recommendations for service improvement [69]. The research extends this tool to generate and maintain test cases for regression testing with less effort and maintain traceability of the generated tests with requirements [69].

The REQAnalytics tool [30] [28] suggests recommendations to the software requirements specification from the usage data and the information of mapping the functional requirements with the web pages and UI elements.

The process to generate different test case scenarios from the usage information of the SaaS follows four phases. The tool collects logs using a web analytics tool (OWA) to calculate the most frequent paths. These most frequent paths represent the main usage of the SaaS under analysis and enable it to generate test scripts useful for regression tests. In the next phase, the tester provides input data to test and validates the success and failure cases. The test scripts generated by this

process are used for regression testing and are executed every time there is a change in the SaaS to verify if the functionalities are still working.

Silva et al.'s [69] approach validates that it's possible to generate recommendations to improve the SaaS using REQAnalytics [31] and also check if those improvements do not affect the quality of the SaaS.

3.1.2.3 Test Cases Derived From Task Models

In the early phases of development, as an alternative to user testing, some authors proposed analytic methods to identify potential usability problems using models of the devices. These techniques analyze the system that implements the models. However, they may ignore the problems introduced during the implementation of the system. Silva et al. [68] suggest applying MBT, since it allows the test of a software artifact against a model of the expected product, also called the oracle.

Silva et al.'s [68] approach describes an effort to develop tool support enabling the use of task models as oracles for MBT of user interfaces. The user testing provides higher fidelity results when evaluating the user response to the system. However, there are challenges of user interface testing, namely covering all potential problems, which can be expensive, and having access to final users.

MBT increases the software testing systematization and automation, comparing the state and behavior of a software product against its model [68]. The test cases can be generated automatically from the model, and are executed over the model and the software implementation. The results obtained are compared, and if there is an inconsistency between the expected and actual results, a potential problem has been identified. Silva et al.'s [68] concern is to avoid introducing problems in the proposed design during the implementation, as well as in the maintenance and redesign contexts.

In user interface testing, it is recommended that the models used as oracles are expressed at the same level of abstraction as required during the user interface design process. Silva et al.'s [68] approach suggests using task models as oracles for MBT of user interfaces to address the abstraction level problem. This approach applies ConcurTaskTrees (CTT) as the task modelling notation. However, the focus of Silva et al.'s [68] approach is not the test case generation for user interfaces, but the test oracle which will be used as a measure of the implementation's quality.

The Task models are usually used to model GUIs and to describe usage scenarios in a higher level of abstraction. The authors [68] suggest creating GUI models at a more abstract level to decrease the effort in their creation, adopting task models as oracles. The main advantage of task modelling is to identify useful user-device interaction abstractions highlighting the issues that should be considered when designing interactive applications [68].

Consequently, Silva et al. [68] defend that the best way to test the behavior of systems' user interfaces is through a task model, which will allow them to generate a model against which to test. After generating the oracle, the tester can make some adjustments due to the need to add more semantics to the model. Finalizing these changes to the oracle, the test cases are generated

automatically and executed over the GUI under test using the framework. Nonetheless, Silva et al. [68] present some limitations in generating the test oracles directly from task models, regarding testing specific implementation details, and testing behaviors outside the task model. In the first statement, the authors [68] report the difficulty to test specific details when they are not present in the task model. Silva et al. [68] advocate a deeper analysis on the subject, however, as a solution, they propose to add more information about the specific implementation patterns used in the task model. In the second statement, the authors [68] defend that it is not possible to test behaviors outside the task model, since the oracle being used will not be able to cope with them.

Nevertheless, Silva et al. [68] sustain that it is simple to envisage other scenarios with different variations of the task model and use them to test the system against, since the testing set up process is mostly automated. The authors [68] defend using task models for generating test oracles despite the limitations, since the created oracles will test the expected users "normal" behaviors to which the application was designed.

Barbosa et al.'s [5] concern is automating the testing process to reduce analysis costs by proposing a systematic approach to test graphical user interfaces. The MBT methods can automate the generation of test cases from a model of a SuT [5]. However, these methods do not present good results when applied to interactive systems, due to the need to build detailed models of the GUIs.

One solution to get around the problem is to increase the level of abstraction of the models.

Barbosa et al.'s [5] work follows an existing approach described in [68], using ConcurTaskTrees as the task modelling notation. Silva et al.'s [68] approach presents how the task models can be applied to incorporate more abstract models in the MBT of GUIs. The task models do not capture the frequent mistakes that users make or the alternative scenarios that differ from the expected behavior. However, Barbosa et al.'s [5] approach analyses the feasibility of using task models to test GUIs against incorrect user behavior in a MBT environment.

Based on this previous work, [68], Barbosa et al. [5] develop an automated approach to generate mutations of the task models. They propose an algorithm to execute the changes to the original models by introducing common user mistakes. Firstly, Barbosa et al. [5] selected different applications which were analyzed to detect patterns in the construction of their task models. Then, the algorithm detects those patterns in the task models and creates a strategy to generate the mutants through the capture of the different types of mistakes on them.

Barbosa et al. [5] developed CMTTool to validate this approach, and used the TOM tool to support the process of MBT of GUIs from task models. The approach was applied to various use cases and the results showed that this approach allows to detect the faults from the erroneous user behavior. Therefore, these two tools increase the coverage of the capture of the common user behavior of the MBT approach.

In conclusion, Barbosa et al. [5] suggest the mutations being used allow capturing important user interface problems. Nonetheless, they are aware that they need to obtain more data in the future to support their statement. The authors [5] state that the automation of the process with

more data will allow them to experiment with different mutations to determine the best set of mutations.

3.1.2.4 Test Cases Derived From Patterns

As in the previous proposals [5] [68], Monteiro & Paiva's [49] objective is to increase the level of abstraction of the GUI models, promote reuse and reduce the effort in building models to MBGT. Monteiro & Paiva [49] present an approach to Model-Based GUI Testing (MBGT) using a modeling environment (ME) for a domain-specific language (DSL). The difference between this approach and the previous ones [5] [68] is that it proposes to use the PARADIGM language.

This approach [49] took the concept of UI patterns into the context of GUI testing. A pattern describes a problem which occurs several times, and also describes the solution to the problem that can be applied repetitively. The test patterns are the Behavioral elements within PARADIGM. Monteiro & Paiva [49] suggest defining test strategies for testing UI patterns. These test strategies allow the tester to test the user behavior through a set of configurations, which are mapped to interface controls present in the GUI, and then by executing the generated test cases.

Monteiro & Paiva [49] built a PARADIGM modeling environment which provides support to construct well-formed GUI models using Eclipse Graphical Modeling Framework (GMF). The GUI models constructed with PARADIGM follow a set of rules to guarantee that the models are well-formed. According to the authors [49], constructing a PARADIGM modeling environment allows the configuration of models with test input data, generating test cases from those models and executing them on an interface GUI. The test case generation process is performed by a recursive algorithm which traverses the model to generate all the possible test paths, and then, the test cases are generated from those paths. Finally, the test case execution is supported by ME, which provides an execution module for web applications.

Monteiro & Paiva [49] claim this work contributes to improving ME functionalities and the PARADIGM language, creating new behavioral elements which allow to increase the level of abstraction of the constructed models.

Moreira & Paiva's [50, 51, 52] approach suggests applying Pattern-Based GUI Testing (PBGT) methodology that aims to systematize and automate the GUI testing process, applying UI Test Patterns and reusing GUI testing strategies. GUIs represent an important role in the success of the software, and therefore it is important to test GUIs for their functional correctness. As already mentioned in Monteiro & Paiva's [49] approach, UI Patterns are used recurrently to solve common GUI design problems and can be implemented differently depending on the different purposes.

PBGT is a model-based GUI testing approach that provides possible test strategies with different configurations, which allow the testing of various implementations of a UI Pattern. The UI Test Pattern is defined within a domain-specific language (PARADIGM), and it enables testing the GUI that was developed by applying a set of UI Patterns. The elements from DSL's UI Test Patterns allow building models describing the GUI testing goals in a higher level of abstraction.

Moreira & Paiva's [50, 51, 52] approach is supported by the PBGT tool, which provides a complete testing environment and functionalities for modeling, configuration, automated test case

generation and execution, and test coverage analysis. The PBGT requires the GUI model written in PARADIGM, which facilitates structuring the models in different levels of abstraction in order to deal with complexity and to facilitate organization. The PBGT process includes six steps: modeling; configuration; test case generation; test case execution; results analysis and, when required, model update [50, 51, 52]. The modeling step can be performed manually by the tester or automatically from an existing software application by the PARADIGM-RE component, which explores the application under test and captures existing UI Patterns. In the next step, the tester configures each UI Test Pattern with the required data (input data, preconditions, and checks to perform during test execution). After that, the PARADIGM-TG component generates the test cases, calculating all the paths of the model. The test cases are generated from those paths and considering the configurations provided previously by the tester. Finally, to execute the tests, it is required to map the model UI Test Patterns and the UI Patterns of the web application under test. Then, the PARADIGM-TE component executes the generated test cases over the application under test and produces reports with the results of the tests.

Moreira & Paiva [50, 51, 52] proposed seven UI Test Patterns (Input, Login, Master/Detail, Find, Sort, Call and Option Set) that encompass a satisfactory range to model web and mobile applications. However, one of the limitations of the PBGT tool is that it cannot test desktop and iOS applications. According to Moreira & Paiva [50, 51, 52], PARADIGM has advantages, since it allows reusing UI Test Patterns to test different UI Patterns implementations and reusing existing elements or extending them to be applied by others.

Nabuco & Paiva [53] follow the same PBGT methodology as Moreira & Paiva [50] and Monteiro & Paiva [49]. There is a lack of standards and conventions in web apps which makes automatic testing for the Web difficult and also inhibits the reuse of test code. Furthermore, it is important for the user to have a sense of comfort and easiness when navigating through Web applications. Therefore, the developers use User Interface (UI) Patterns, which consist of solutions to common design problems that contribute to a better implementation. Nabuco & Paiva' [53] suggest defining generic and reusable test strategies to verify when we have different implementations in the web applications. These test strategies are adapted to the different applications during the configuration process. The authors [53] propose the PBGT approach, since it facilitates reusing and reduces the effort of the modeling activity within a MBT process.

Nabuco & Paiva's [53] goal is to provide different test case generation techniques from models to test those Web applications. The modeling and configuration process is similar to what Moreira & Paiva [50] and Monteiro & Paiva [49] applied, however, Nabuco & Paiva [53] improve the generation test case process by implementing filtering techniques.

The authors [53] were motivated by the available computational resources and reducing the time to create filters that reduce the number of test cases generated, therefore adding more flexibility. During the process of test cases' generation, the filters are applied to the Test Paths and, consequently, to each Test Path element configurations, therefore reducing the number of test cases generated. The authors [53] claim that the PARADIGM is effective and, by applying the filtering techniques, provides better coverage, finds more bugs and kills more mutants.

Nabuco & Paiva [53] identify two advantages from this approach. First, the test process allows us to generate different test suites and execute them through different workstations. Second, when an error occurs, it is easy to identify the element or configuration that caused the error and stopped the execution.

3.1.2.5 Test Cases Derived From Big Data

The recent emergence of big data and cloud computing originated a need for extracting knowledge from the information demanded by each user's needs [26]. Considering the large amount of data sources that store similar information, there is a need for heterogeneity and cross-domain reconciliation.

The Entity Reconciliation (ER) problem is a major research topic in data quality management. This process combines data from distinct sources and tries to group them by a common unifying key. The management of big data volumes introduces some challenges and has increased the need to extract high-quality reconciliation of entities. Therefore, the ER problem creates the need to test the operations and applications designed to extract the reconciliations, thus ensuring the accuracy of the reconciliation and the quality of the reconciled data.

Blanco et al.'s [10] approach suggests the MDE paradigm for testing the applications that implement ER problems. MDE decreases the level of abstraction through the models, consequently allowing the automation of the testing process in the early phases of development. The authors [10] propose creating data models from the sources, which are grouped by business rules that allow generating test cases from them. The focus of Blanco et al.'s [10] approach is the automation of test coverage to conduct the generation of the test cases. This work starts by defining the test models, called Integration Testing Rules (ITR) models, which are composed of a set of rules that describe the test conditions. These rules allow the creation of test coverage items that will conduct the generation of the test data sources and the test reconciled solution.

For the authors [10], the key contribution of this approach was the decrease in the effort of designing the expected output and checking it against the actual output. Nevertheless, Blanco et al. [10] identified some limitations in their approach. Firstly, the ITR metamodel may not have the necessary elements to represent the ITR models. To resolve this issue, all the ER domains should be analyzed to detect if the ITR metamodel needs to be extended in order to be possible to create the ITR models. Secondly, the rules from ITR models may not express clearly all the features that must be tested in ER application. To solve this, the rules should be extended to add more complex conditions. The authors [10] claim that adding integration tests in the early stages of the ER application helps to detect flaws and to deliver a better product in the end.

In conclusion, Blanco et al. [10] claim there are two main advantages in testing through a test model which enables the use of the test coverage to guide the test case design process. These advantages are the decrease in the amount of data that needs to be stored in the data sources used for testing and the increase in test coverage.

3.1.3 Behavior-Driven Development (BDD)

The agile software engineering movement defends that testing should occur at an earlier phase in the development process. Specifically, the Test-Driven Development (TDD) approach claims that acceptance tests should be written at an earlier in the development process [6]. Consequently, Behavior-Driven Development (BDD) has been identified as a result of problems that occur with TDD when agile software practices are applied in the process [56]. BDD uses natural language to describe the acceptance tests, which represent all the scenarios that should be accomplished by the final system. During the BDD process, the product owners write the acceptance test to detail the system behavior. Then, they examine and approve the acceptance tests before the programmer starts to code the program. The language used to write the tests is Gherkin, which allows specifying the test in an understandable way. Typically, this process is a manual process that is error-prone and time-consuming.

The acceptance tests provide the necessary information to automatize the extraction of the structure of the implementation and the test cases. Soeken et al.'s [70] approach proposes a methodology to extract semi-automatically the design information from the sentences using natural language processing techniques. The authors [70] suggest a design flow where users can choose the suggested pieces of code extracted from the sentences. Thereafter, this suggestion can be refined after confirming the idea was the intended to convey in the sentence. Soeken et al. [70] claim that this approach is a meaningful step towards an automatization of BDD and this approach can be the basis for further work.

Alferez et al.'s [1] approach discusses the gap between requirements models and AC in the context of BDD.

The goal of Acceptance testing is to guarantee that a complete system meets its specified requirements. The main step is to define the system behavior through Acceptance Criteria (AC) written in natural language, which facilitates the communication between the stakeholders. The authors [1] suggest a UML-based modeling methodology. This modeling methodology tries to generate AC represented by a sequence of scenarios in Gherkin language, and then, the AC results are used to generate test scenarios. Alferez et al. [1] explain that they can extract the test paths from the models defined according to the methodology and transform them into AC. The authors [1] implemented an algorithm to extract the intents of the cases. The scenarios described by the acceptance tests provide enough information in order to automatize the extraction of both the structure of the implementation and the test cases. After evaluating an applied case study, Alferez et al. [1] conclude that the approach adds value to the process and is feasible.

Silva et al.'s [18] approach supports the specification and generation of software tests from requirements specifications originally defined in RSL. The process follows two phases: First, RSL requirements specifications are the input for the RSL-to-TSL transformation that generates TSL specifications. Second, TSL specs are the input for the TSL-to-Gherkin transformation that generates Gherkin specifications. These specifications can be used as additional documentation or for test execution.

3.2 Discussion

Requirements written in NL are easy to understand because no special technical knowledge is required. However, requirements written in NL may be ambiguous, which raises additional challenges to interpret them correctly [20].

Natural Language Processing (NLP) may be used to solve ambiguity problems and provide concrete and correct information to developers. Although there is extensive research on the use of NLP, there are still limitations. Researchers focused more on identifying ambiguity, than trying to avoid or resolve it [4]. The ambiguity factor must be addressed at an early stage by representing requirements in a machine-readable format [77].

In a MBT approach, test cases may be generated automatically from models. These models may have a textual or graphical representation [66] and are a more structured way to specify the requirements, avoiding the ambiguousness of natural language [66]. MBT approach can generate test cases based on use cases, execution traces, task models, patterns and big data.

Most of the analyzed use case approaches use a template for textual specification written in structured natural language. In the previous subsections, different methods to obtain models from requirements to generate test cases have been scrutinized.

Gutiérrez et al.'s [33] approach presents models, transformations and processes which reduce time and allow traceability between requirements and tests. The use cases model captures the control flow described in the specification to generate the use cases, which significantly reduces time and manual human effort. Wang et al. [79] approach supports the generation of acceptance test cases from requirements specifications in NL to reduce the manual effort and ensure requirements coverage. The approach [79] is supported on advanced NLP solution to generate Object Constraint Language (OCL) constraints that capture pre and postconditions of use case steps. However, the advantage of this approach is that it does not obligate restrictions on the use case specifications.

Alrawashed et al. [2] and Somé & Cheng [72] generate test cases from the control flow to reduce the test complexity and to enhance the percentage of test coverage. Our approach goes beyond these two approaches by generating test cases and data using information defined in the specifications.

Hamza & Hammad [35] approach converts UML use case diagrams into custom activity diagrams that are converted to DAGs, from which test cases are extracted. Nevertheless, this technique presents a sequence problem, as it requires that the test cases should be executed in a specific order, due to existing dependencies among them.

Other researchers try to solve the absence of models or increase the level of abstraction of the generated models. Silva et al. [69] and Paiva et al. [59] propose different approaches to generate test cases based on registering the user interactions. Paiva et al. [59] suggest generating test cases from user execution traces, which is an alternative to deal with the absence of models, and Silva et al. [69] generate test cases from the usage information of the services with less effort and maintaining traceability of the generated test.

Barbosa et al.'s [5] work follows an existing approach described by Silva et al. [68], using task modelling notation. Silva et al.'s [68] approach presents how the task models can be applied to incorporate more abstract models in the MBT of GUIs.

Monteiro & Paiva's [49] work contributes to improving ME functionalities and the PARADIGM language, creating new behavioral elements which allow to increase the level of abstraction of the constructed models.

Moreira & Paiva's [50, 51, 52] approach suggests applying PBGT methodology to systematize and automate the GUI testing process, applying UI Test Patterns and reusing GUI testing strategies. Nabuco & Paiva [53] follow the same PBGT methodology as Moreira & Paiva [50, 51, 52] and Monteiro & Paiva [49], however, Nabuco & Paiva [53] improve the generation test case process by implementing filtering techniques.

Some researchers suggest the use of formal languages complemented with models and semi-formal methods. Mandrioli et al. [47] use formal languages to define requirements providing a solution for test generation and requirements validation. However, the use of formal languages enhances complexity and requires some mathematical knowledge that can increase cost [21].

A modeling language such as UML could be a solution, however it is incomplete for modeling requirements because it lacks models that tie requirements to business value and models that present the system from an end user's perspective [16].

Some researchers propose CNLs [43]. These languages benefit from being close to natural language, but providing some structure that allows automatic manipulation, decreasing the possibility of errors and misunderstandings during both the requirements and tests specification.

RSL is an example of such a CNL, which allows specifying requirements and tests in a rigorous way, maintaining the human-readability of specifications. The main reason for the adoption of RSL in this research was its support for the specification of both requirements and tests in an integrated way. RSL contains a set of constructs that represent different RE-specific concerns (e.g., active structure, behavior, passive structure, requirements or tests) and can be used at different levels of abstraction (e.g., at business, application, software, or even hardware level) [17]. In addition, RSL supports the alignments between requirements and tests, enabling the generation of test cases from use case scenarios. This automation process brings benefits by reducing the effort and cost of repetitive tasks, and ensuring higher quality of both requirements and tests.

The current dissertation extends Maciel's [46] [58] end-to-end work by improving the process of generating and refining test suites from the specification of use cases defined in RSL. To achieve that purpose, we extended the RSL design to be able to specify more details, namely use case scenarios and test cases with a sequence of concrete test steps.

Chapter 4

Proposed Approach

Due to the traditional separation between requirements and testing activities, it is not common to start the testing activity at the beginning of the software development process. This approach tries to avoid this situation by supporting the specification of tests at an early stage of the project and also by generating such tests from requirements and ensuring their alignment.

The objective of this research is to improve the process of generating and refining test suites from the specification of use cases defined in RSL. This research also takes into consideration, and extends, the end-to-end approach previously proposed by Maciel et al. [46] [58], although not sufficiently explored and detailed in that previous work.

The proposed approach (shown in the dashed rectangle in Figure 1.1) is focused on the following tasks: (1) requirements specification that serves as a basis (2) to generate the specification of the test suite with test cases; and be further (3) manually refined by the tester.

To better support the explanation, the "MyStore" running example is used, considering the following requirements: (1) the user must create an account on "MyStore" (this account includes basic user information, such as username (e-mail) and password); (2) Before using the application, the user must authenticate to access the system; and (3) The user shall create and update the products (that will then become available for sale).

4.1 Specify Requirements (Task-1)

The first task of this approach is the specification of requirements, which are defined through an agreement among stakeholders, developers and testers. Requirements are specified by using RSL constructs, such as data entity, data, actor, use case and respective relationships. In the running example we defined two data entities (User and Product), two actors (Administrator and Anonymous) and six use cases (sign-up, log in, create product, manage products, delete product and delete user). Some of these specifications are shown above in Listings 5.1 and 5.2.

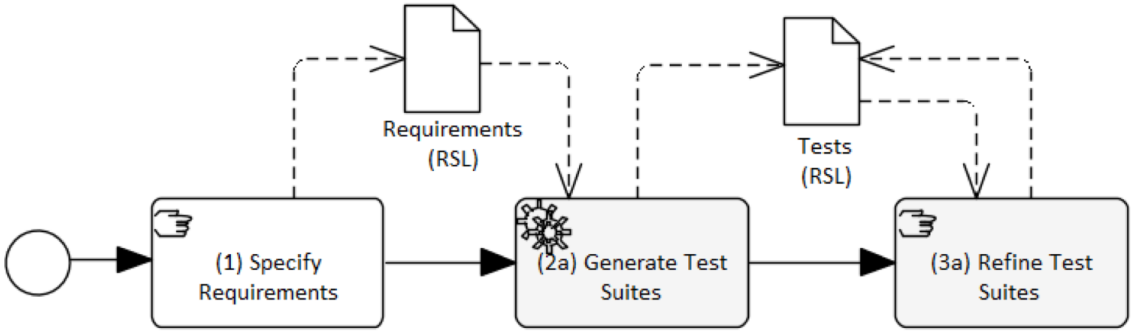

Figure 4.1: Testing generation and transformation processes.

4.2 Generate Test Suites (Task-2)

RSL supports specification and generation of test cases directly from the requirements specifications. The goal is to generate test suites automatically after finalizing a use case specification.

The generation of a test suite from a use case is done by directly creating a test suite for each use case, and establishing in the test case a reference to the involved use case. Then, the use case scenarios (main, alternative and exception branches) are transformed into test cases. One test case is generated for each scenario, with respective steps and variables if relevant.

In the "MyStore" example the "User successfully creates the product", the test case steps are obtained from the use case main scenario specification. The test case presents the necessary steps to perform in order to achieve the result and compare it with the expected value. The user inserts valid product information, following the specification steps, and at the end the System verifies if the success message appeared.

The data is also generated in parallel to the test suite transformation process. Automating the data generation process reduces the time-spent and costs usually associated with this kind of tasks, since manual data entry is a tedious process, and it is also frustrating and repetitive, for many of the testers.

```
1  *** Data ***
2  Data d_Products : e_Product [
3      withValues
4    | e_Product.ID | e_Product.SKU    | e_Product.Name +|
5    | 0001         | "9001"          | "Product 1"    +|
6    | 0002         | "9002"          | ""             +|
7  ]
```

Listing 4.1: Example of data specification (in RSL).

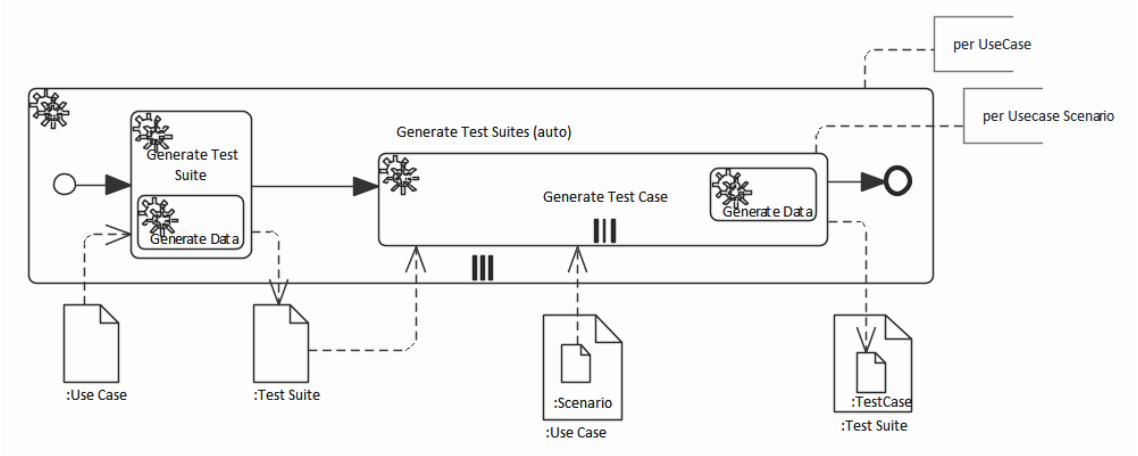

Figure 4.2: Tests and data generation process.

4.3 Refine Test Suites (Task-3)

The generated tests might be subject to manual refinements. After the transformation of the tests, the following activities can be performed: refinement, reuse and sequencing of the generated tests.

4.3.1 Refinement

One of the first steps in refinement is to assign values to the entities and create temporary variables, which are essential for data transmission between the test suites and the test cases. Listing 4.2 shows the creation of a product use case with the necessary refined information. The test suite has 4 global variables. The first two variables keep success and error validation messages, and the last two variables keep the product and user data values.

The test case contains the necessary information to perform the steps in order to achieve the results and compare them with the expected values.

In Listing 4.3, the test case gets data from data entities defined.

The tester chooses the actor/action types to apply and defines the operation extension values. Listing 4.3 shows the operation extension assigned values and respective data entity/variable

```
1  *** Test Suite ***
2  TestUseCase t_uc_3_CreateProduct "Create a Product" [
3    useCase uc_3_CreateProduct
4
5    variable vSuccess [ "success" ]
6    variable vError [ "error" ]
7    variable vMessageSystemNotification [ "The product ID already exists", "The product ID is empty", "
         The product name is empty", "The product was successfully created" ]
8    variable v_Products [ withData (d_Products)]
9    variable v_Users [ withData (d_Users)]
10   ...
11 ]
```

Listing 4.2: Example of a test suite specification (in RSL).

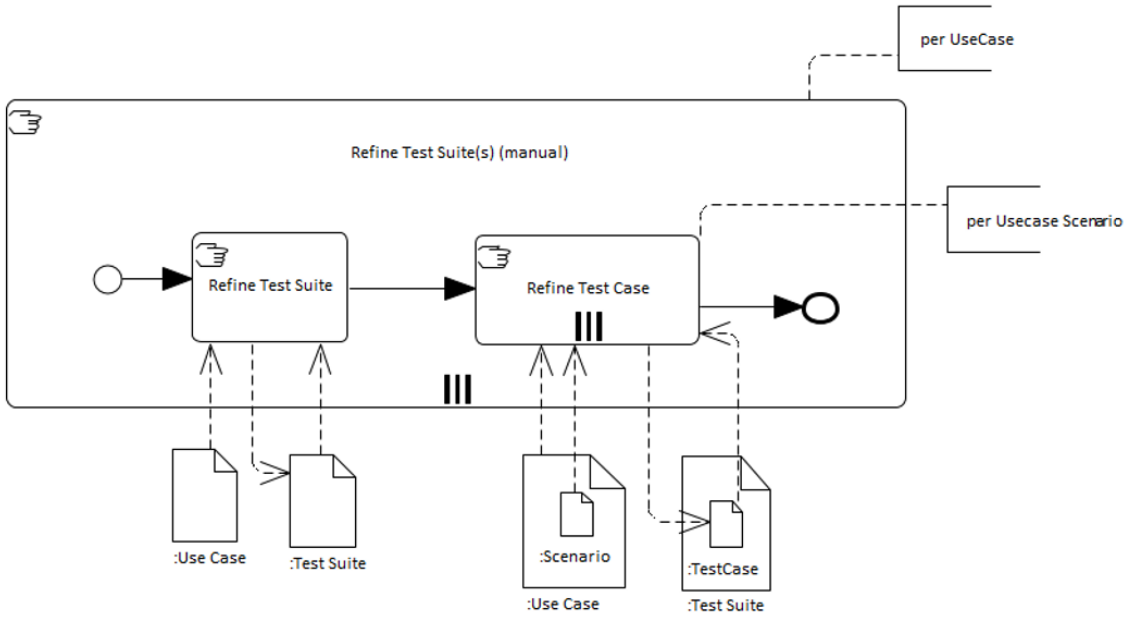

Figure 4.3: Refine process.

attributes. To execute step 9, the actor prepares the data which reads from a respective data entity/variable attribute: v_Products.Name.

The test case ends with a check in order to evaluate, at step 11, if the test was successful or "has failed". When required by the tester, it is possible to refine the data by editing or adding new records.

4.3.2 Reusing

During manual refinements, the tester might apply reuse mechanisms whenever necessary. Setup and teardown are reuse mechanisms which allow saving time during implementation and also reduce test execution time.

There are several advantages to use the reuse mechanism: First, less confusion when reading the tests since all of the code is visible from within each test. Secondly, less chance of setting up too much or too little for the given test. And finally, less chance of sharing state between tests, which creates unwanted dependencies between them.

Listing 4.4 shows the test suite of the product creation use case with the test case of the main scenario. In the "MyStore" application, the user must sign-up before creating the product. Since the sign-up action is required for all test cases of the test suite, the tester manually assigns a setup action with the test case that allows the user to sign-up (i.e., the "createUsers" include action). Then, the sign-up test case receives values by variable parameters which were defined in the test suite creation product (Listing 4.5).

```
*** Test Case ***
testCase ts_3_2_CreateProduct "User creates a product" [
  variable v_Products withData ( d_Products[2] )

  step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the browser"
  step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open product
       page"
  step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in the
       product ID"
  step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_Product. into
       element ('product_ID') ) "Validates the product ID already exists"
  step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product ID already
       exists"
  step s6 (Actor:PrepareData:Fill from v_Product. into element ('product_ID') ) "Fills in the product
        ID"
  step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from v_Product.ID into
       element ('product_ID') ) "Validates the product ID is empty"
  step s8 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the product ID"
  step s9 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills in the
       product name"
  step s10 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to create
       the product"
  step s11 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
       successfully created"
]
```

Listing 4.3: Example of a test case specification (in RSL).

```
*** Test Suite ***
TestUseCase t_uc_3_CreateProduct "Create a Product" [
...
 variable v_Products [ withData (d_Products)]
 variable v_Users [ withData (d_Users)]

 setup [ step setup_1 execute t_uc_1_SignUp.ts_1_1_Signup withVariable ( v_Users ) ]
 teardown [
  step teardown_1 execute t_uc_6_DeleteUser.ts_6_1_DeleteUser withVariable ( v_Users )
  step teardown_2 execute t_uc_5_DeleteProduct.ts_5_1_DeleteProduct  withVariable ( v_Products )]

 testCase ts_3_1_CreateProduct "User creates a product" [
   step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the browser"
   step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open product
        page"
   step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in the
       product ID"
   step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills in the
       product name"
   step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to create
       the product"
   step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
       successfully created"
  ]
]
```

Listing 4.4: Example of a test suite specification (in RSL).

```
*** Test Suite ***
TestUseCase t_uc_1_SignUp "SignUp" [
 useCase uc_1_SignUp
 variable v_Users [ withData (d_Users)]

 testCase ts_1_1_Signup "Anonymous User Signs Up With Valid Information" [
  parameter p_User

  variable v_Users is p_User otherwise withData ( d_Users )
  ...
]
```

Listing 4.5: Example of a test suite specification (in RSL).

```
*** Test Suite ***
TestUseCase t_uc_1_SignUp "SignUp" [
  useCase uc_1_SignUp

  setup [ step setup_1 execute t_uc_6_DeleteUser.ts_6_1_DeleteUser withVariable ( v_Users ) ]
  ...
]
```

Listing 4.6: Example of a test case precondition specification (in RSL).

The parameter variable is defined and assigned values from a data entity in the test case. When the setup of the product creation test suite is executed, the sign-up test case is called and the values are sent as a parameter.

All the steps in the test flow are performed successfully. The user is able to log in into the system and successfully create the product. After running all the test cases, the teardown is executed and deletes the records which were created or changed during the test suite. The teardown mechanism cleans up the data, even if the test case does not pass, and all the data returns to its original state.

4.3.3 Sequencing

After the test cases have been generated automatically, the tester can assign manually the precondition and postcondition values. The precondition and postcondition allow the creation of sequencing mechanisms for the test cases. The preconditions specify the setup needed for a test case to be successfully executed following a sequence.

In the "MyStore" example, the sign-up test case starts when an actor registers into the e-commerce system with valid information, having the following precondition: "The user does not exist in the system". Consequently, the precondition must be true before the test case is allowed to proceed. The postcondition for a test case should also be true regardless of the flow being executed. The listing 4.7 shows the creation product test case with the following postcondition: a respective product ID (v_Products[1].ID) must be created in the system.

```
*** Test Suite ***
TestUseCase t_uc_3_CreateProduct "Create a Product" [
  useCase uc_3_CreateProduct
  postcondition v_Products[1].ID
  ...
]
```

Listing 4.7: Example of a test case postcondition specification (in RSL).

4.4 Conclusion

The proposed approach can contribute to push the start of testing activities to the beginning of the software development process, to the requirements engineering phase. It supports the specification of tests at an early stage of the project, generating test suites from RSL specified use cases and ensuring they are aligned.

Requirements are specified using the various RSL constructs and the relationships between them.

After a use case is specified, test suites are automatically generated, as well as their data. This reduces the time, cost and tediousness usually associated with these kinds of tasks.

The generated tests can be refined, with the assignment of values to the entities and creation of temporary variables to transmit data between the test suites and the test cases.

Furthermore, the tester might apply reuse mechanisms whenever necessary, such as setup and teardown. Besides providing time savings during implementation and test execution, these mechanisms also make the code more legible and easier to extend and the tests less prone to share state between them, avoiding unwanted dependencies.

Moreover, the tester can assign precondition and postcondition values, hence allowing the creation of sequencing mechanisms for the test cases.

Chapter 5

RSL Language Extensions

This chapter describes the data entities extended to be able to generate test suites and test data from the specification of use case scenarios defined in RSL. All the data entities were described through examples to illustrate the concepts more clearly.

5.1 Requirements Specification

RSL allows the specification with different levels of abstraction and with different types, such as functional requirements, constraints, use cases, user stories or quality requirements. This work focuses on the RSL constructs particularly related with the specification of use cases, namely considering the following constructs: actor, use case, scenario, data entity and respective relationships [46], [58], that we briefly introduce below.

5.1.1 Data Entity

A data entity represents a structural entity that exists in information systems commonly associated with data concepts identified from the domain analysis. These entities can be represented by data structures, such as tables or collections that may include the specification of attributes, foreign keys and other check data constraints [16, 17]. Listing 5.1 shows the RSL specification of two data entities identified in the "MyStore" application: User and Product with their respective attributes, as well as data element (d_Users) with concrete data.

```
1  *** DataEntities ***
2  DataEntity e_User "User" : Master [
3    attribute ID "ID" : Integer [primaryKey]
4    attribute firstName "Name" : Text [isNotNull]
5    attribute email "Email" : Text [isNotNull isUnique]
6    attribute password "Password" : Text [isNotNull isEncrypted]
7    attribute statusActive "StatusActive" : Boolean]
8  DataEntity e_Product "Product" : Master [
9    attribute ID "ID" : Integer [primaryKey]
10   attribute SKU "SKU" : Integer [isUnique]
11   attribute Name "Name" : Text [isNotNull]]
```

```
*** Data ****
Data d_Users : e_User [
  withValues
  | e_User.ID | e_User.email       | e_User.password +|
  | 1001      | "john@email.com"   | "#password1234" +|
  | 1002      | "peter@email.com"  | "#password1236" +| ]
```

Listing 5.1: Specification of data entities and data elements (in RSL).

5.1.2 Use Case Scenario

A use case defines a set of behaviors of the system that gives an observable result of value for the related actors [14]. The behavior of a use case is generally described by one or more scenarios. A use case may have preconditions, which need to be met before its start and post-conditions, which define the final state of the system after the use case is completed. A use case can be classified by a specific type such as 'EntityCreate', 'EntityRead', 'EntityUpdate', 'EntityDelete' [16]. These use case types are commonly found in business information systems that support data management tasks [16], [19], [62].

```
Actor a_Admin "Administrator" : User
Actor a_Anonymous "Anonymous" : User
*** Use Case ***
UseCase uc_2_Login "Login" : EntityRead [ primaryActor a_Anonymous
                dataEntity e_User...]
```

Listing 5.2: Specification of actors and a use case (in RSL).

5.1.3 Scenarios

A use case can be detailed by one or more scenarios, each one comprising a sequence of steps. A step is executed by an actor or by the system to fulfill the use case's goal. A use case has one main scenario and may have other alternative and exception scenarios.

Listing 5.3 shows the specification of the login use case's main scenario. In this scenario, the system checks if the "username (e-mail)" and "password" provided by the user are recognized by the system. This specification includes two exception scenarios: (1) when the user is not registered, and (2) when the user fills in a wrong password. In both scenarios, the system will report an error message.

```
*** Use Case (scenarios) ***

mainScenario uc_2_Login_Main (Main) [
 step s1 (System:Execute:Open) "Opens the login page in the browser"
 step s2 (Actor:PrepareData:Fill) "Fills in the email field" [
   scenario s2a_NotYetRegistered (Exception) "Not yet registered" [
     step s2a.1 (Actor:PrepareData:Fill) "Fills in an email field that is not registered"
     step s2a.2 (System:ReturnResult:Display) "Displays an error message" nextStep s2 ] ]
 step s3 (Actor:PrepareData:Fill) "Fills in the password field"
 step s4 (Actor:CallSystem:SelectAction) "Clicks on the login button"
 step s5 (System:Execute:Validate) "Validates the pair <email, password>" [
   scenario s5a_WrongPasswordScenario (Exception) "Login with a different password" [
     step s5a.1 (System:ReturnResult:Display) "Displays an error message" nextStep s2 ] ]
```

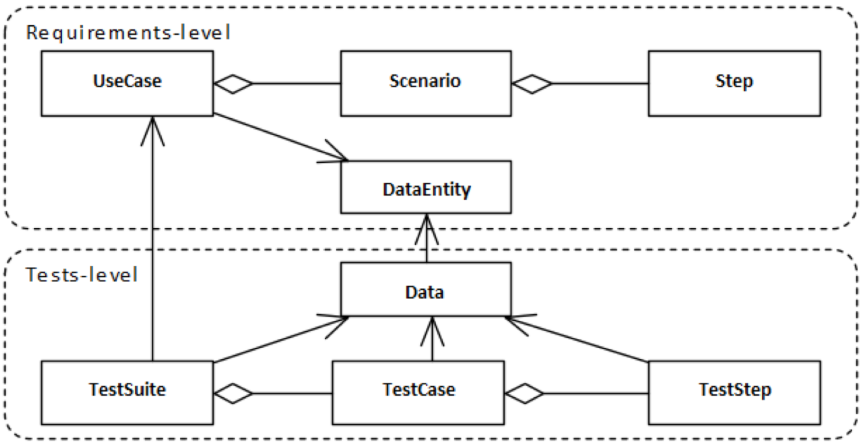

Figure 5.1: Use case and Test Suite RSL elements (partial metamodel).

```
14  step s6 (System:ReturnResult:ShowData) "MyAccount button appears with the user name" ]
```

Listing 5.3: Specification of a use case scenario (in RSL).

5.2 Test Specification

Figure 5.1 shows a partial view of the RSL metamodel with the key elements discussed in this dissertation. The top of that figure shows that a UseCase has several Scenarios and each Scenario has several Steps. On the other hand, at the bottom, a TestSuite has several TestCases, and a TestCase has several TestSteps. In addition, a TestSuite (of type TestUseCase) shall refer to a specific UseCase, and may use several concrete data elements (Data), which shall be conformed with a DataEntity. This section presents the main elements used to test use case related specifications, and some advanced features (e.g., variables, setup and teardown actions) that support reusability and flexibility.

5.2.1 Test Suites

In summary, a test suite shall include a test case corresponding to the main scenario and other test cases to the alternative and exception paths. As suggested in Listing 5.4, a Test Suite RSL construct includes a unique identifier (ID), a name, a type, a reference to a given requirement (e.g., use case, user story, goal) and a set of test cases 5.5. The type of the Test Suite constraints the type of the requirement assigned. In this dissertation, we are only concerned with the use tests, i.e., test suites of type “TestUseCase”.

```
*** Test Suite ***
TestUseCase t_uc_3_CreateProduct "CreateProduct TestSuite" [
  useCase uc_3_CreateProduct
  ...
]
```

Listing 5.4: Specification of a test suite (in RSL).

5.2.2 Test Cases

Listing 5.5 shows a test suite refined with test cases, test steps, variables and references to data entities. Each test case has a unique identifier (ID), a name that describes its purpose, and a set of test steps.

5.2.3 Test Steps

A test step has a unique identifier, the identification of the entity which performs the action (e.g., System or Actor), an action, and a brief description. A step action has the following possible subTypes: PrepareData (describing actor's action to insert data, such as text, checkbox selection), CallSystem (describing actor's actions, such as, button submit, button reset), Execute (describing actions executed by the system, e.g., 'Open Browser' and 'Open App') and ReturnResult (to collect application data and store it in temporary variables).

The Table 5.1 shows the test steps' classification. A step is classified with an operation type and, eventually, an operation extension type. These operation types describe the action that will be performed in the respective step. The operation extension types are subtypes of the aforementioned types that specify the action. Each step must have an operation extension that will allow it to specify the target element. Finally, the test check defines the validation that will be performed in the step where it was specified.

```
*** Test Suite + Test Case ***
TestUseCase t_uc_3_CreateProduct "CreateProduct TestSuite" [
 useCase uc_3_CreateProduct

 testCase ts_3_1_CreateProduct "User creates a product" [
  step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the browser"
  step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open product
      page"
  step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in the
      product ID"
  step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills in the
      product name"
  step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to create the
       product"
  step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
      successfully created"
]
```

Listing 5.5: Specification of a test suite and test case (in RSL).

Table 5.1: Test Step classification (in RSL).

Step		Operation Extension	
Type	**SubType**	**Type**	**Optional Parameter**
Actor	**PrepareData**	**Fill**	from [<DataName[Line]> | <DataName[Line][Column]> |<Variable> | <String>] into element (<UIPartName>)
		Select	from [<DataName>] into element (<UIComponentName>)
	CallSystem	**SelectAction**	element(UIPartName)
System	**Execute**	**Open**	browser(url <Variable> | <String>)
			app(path V<Variable> | <String>)
		Close	browser(url <Variable> | <String>)
			app(path <Variable> | <String>)
		SaveData	from element(UIForm)
	ReturnResult	**ShowData**	element (<UIPartName>) withData [<DataName[Line]> | <DataName[Line][Column]> |<Variable> | <String>]
		Check	textOnScreen(<Variable> | <String>) from [<DataName[Line]> into element (<UIPartName>)
			elementOnScreen [<DataName>] into element (<UIPartName>)
			textOnElement(<Variable> | <String>) from [<DataName[Line] | <DataName[Line][Column]> into element (<UIPartName>)
			variableValue <String>
		DisplayMessage	message(<Variable> | <String>)

5.2.4 Setup and Teardown

A test suite execution runs all the inner test cases of a test suite. Setup can be used to set the initial state before executing the test suite, whereas teardown performs cleanup after the test suite execution.

When a setup is defined, it is usually coupled with a teardown definition, in order to roll back the performed updates so that the following test suite executions start from the same initial desired state.

5.2.5 Variables and Parameters

Concrete data elements and variables are useful to specify concrete data that can be used by test suites and/or test cases. These elements can be defined at a test suite (i.e., they can be referred to by its test cases) or at a test case (i.e., they only get visible inside the test case).

Listing 5.8 shows the specification of a test suite with the test suite level variables (i.e., vSuccess, vError, vURL). This group of variables can be later referred to in test cases. Listing 5.8 also shows in the specification of a test case, the values assigned to the internal variables and inserted into the steps, which contain the necessary information to execute the flow of events.

Listing 5.9 shows the specification of a concrete data element (d_Products) defined in a tabular format with several rows.

Parameters Variables can be passed as parameters. Listing 5.10 shows two different possible ways to do that: (1) withData; (2) withValues.

```
1  *** Test Suite setup ***
2  TestUseCase t_uc_3_CreateProduct "CreateProduct TestSuite" [
3  ...
4   setup [ step s1 execute t_uc_1_SignUp.ts_1_1_Signup  withVariable ( v_Users ) ]
5
6  *** Test Suite teardown ***
7   teardown [
8      step s1 execute t_uc_6_DeleteUser.ts_6_1_DeleteUser withVariable ( v_Users )
9      step s2 execute t_uc_5_DeleteProduct.ts_5_1_DeleteProduct withVariable ( v_Products ) ]
10    ...
11 ]
```

Listing 5.6: Specification of a test suite (in RSL).

```
*** Test Case setup ***
testCase ts_3_1_CreateProduct "User creates a product" [
  ...
 setup [ step s1 execute t_uc_2_Login.ts_2_1_Login withVariable ( v_Users[1] ) ]
  ...
]

*** Test Case teardown ***
...
testCase ts_5_1_DeleteProduct "User deletes a product" [
  ...
 teardown [ step s1 execute t_uc_5_DeleteProduct.ts_5_1_DeleteProduct withVariable ( v_Products ) ]
  ...
]
```

Listing 5.7: Specification of a test case (in RSL).

```
*** Test Suite ***
TestUseCase t_uc_2_Login "Login TestSuite" [
  useCase uc_2_Login

  variable vSuccess [ "success" ]
  variable vError [ "error" ]
  ...
]

*** Test Case Level ***
 testCase ts_3_1_CreateProduct "User creates a product" [
  variable v_Products withData ( d_Products )

  step s1 (System:Execute:Open browser(url vURL)) "Opens the create product page in the browser"
  step s2 (Actor:PrepareData:Fill from v_Products[1].ID into element ('product_ID')) "Fills in the
      product ID field"
  ...
  ]
]
```

Listing 5.8: Specification at test suite and test case levels (in RSL).

```
*** Data ***
Data d_Products : e_Product [
    withValues
  | e_Product.ID | e_Product.SKU    | e_Product.Name +|
  | 0001         | "9001"           | "Product 1"    +|
  | ""           | "9002"           | "Product 2"    +|
]
```

Listing 5.9: Specification of a data element (in RSL).

```
*** Data ***
variable v_User is p_User otherwise withData ( d_Users )

*** Values ***
variable v_User is p_User otherwise withValues (
        | e_User.ID       | e_User.firstName  | e_User.email  | e_User.password +|
        | d_Users[1][0]   | d_Users[1][1]     | d_Users[1][2] | d_Users[1][3]   +|)
```

Listing 5.10: Specification of parameters (in RSL).

```
*** Login Test Suite ***
TestUseCase t_uc_2_Login "Login TestSuite" [
  useCase uc_2_Login
  variable v_Users [ withData (d_Users)]

  testCase ts_2_1_Login "User Logs In With Valid Information" [
  parameter p_User
  precondition v_Users.statusActive

  variable v_Users is p_User otherwise withData ( d_Users )
  ....

*** Create Product Test Suite ***
TestUseCase t_uc_3_CreateProduct "CreateProduct TestSuite" [
  useCase uc_3_CreateProduct

  setup [ step setup_1 execute t_uc_2_Login.ts_2_1_Login withVariable ( v_Users ) ]
  ...
]
```

Listing 5.11: Specification of variables (in RSL).

Listing 5.11 shows the ts_2_1_Login test case that when included by other tests can receive variables by parameter values. The parameter variable is defined and assigned values from data or from local values in the test case. When the setup t_uc_3_CreateProduct test suite is executed, it calls the ts_2_1_Login test case and sends the value as a parameter. The user is able to log in into the system and successfully create the product. The values assigned are defined in a tabular format with several rows.

5.2.6 Preconditions and Postconditions

It is possible to define preconditions (which must be true before the execution of the test case) and post-conditions (which must be true after the execution of the test case). Listing 5.12 shows the specification of the ts_2_1_Login user test case with the following precondition: “The user must be in the active state” — meaning that only registered and active users may successfully log in.

5.3 Conclusion

This chapter scrutinized the RSL constructs related to use case specification, as well as the main elements used to test them, also focusing on some advanced features to support reusability and flexibility.

A use case has several scenarios, comprising always one main scenario and may also contain alternative and exception scenarios. Each of these Scenarios has several Steps.

```
*** Test Case (precondition) ***
testCase ts_2_1_Login "User Logs in With Valid Information" [
    precondition d_User[1].statusActive
    ...
]
```

Listing 5.12: Specification of a test case precondition (in RSL).

A test suite has several test cases, including a test case corresponding to the main scenario and other test cases to the alternative and exception paths. It refers to a specific use case and uses several concrete data, conforming with a data entity. Each test case comprises an ID, a description of its purpose and a set of test steps. A test step has its unique identifier, an entity, an action and a brief description. The available step actions are PrepareData, CallSystem, Execute and ReturnResult. The test check defines the validation that will be performed in the step where it was specified.

Moreover, setup can be used to set the initial state before executing the test suite, whereas teardown performs cleanup after the test suite execution. Both are usually coupled, to perform and then roll back updates during test suite execution, so the next execution has the same initial desired state.

Variables and parameters can also be defined, in a test suite as a whole or in a single test case, so they can be later referred to in a test suite's test cases or in a test case's steps, respectively. Furthermore, it is possible to define preconditions and postconditions, which must be true, respectively, before or after the execution of the test case.

Chapter 6

Test Case Generation

This chapter describes the process of generating test suites with test cases, concrete test data and variables from use case specifications. The tests' generation process follows a sequence of tasks. First, the code generator extracts information from a use case specification. Second, the full edges' coverage algorithm generates a test suite and test cases from a given use case. Third, the generation of concrete test data and variables that can be referred in the test cases.

6.1 Tools Supported

The Eclipse IDE is the platform chosen to develop the test generation tool. One of the advantages of this IDE is that it supports the Xtext framework, which helps the user with incremental syntax checking, suggested corrections and auto-completion. Xtext is a framework that allows the creation of specific programming and domain-specific languages [48]. Xtend, [1] which originated from Xtext, is a statically-typed programming language which is translated to comprehensible Java source code. Xtend brings some benefits and advantages, such as concise syntax and some additional functionality namely the type inference, extension methods, and operator overloading [48].

RSL has been defined with the Xtext framework, which promotes a rigorous specification and the possibility of being automatically validated and transformed in multiple representations and formats [15].

6.2 Use Case Specification

A RSL use case specification comprises a main and several alternative and exception scenarios, which describe different courses of actions and events between an actor(s) and the system under consideration [14] [62].

[1] https://www.eclipse.org/xtend/

```
1  UseCase uc_3_CreateProduct "CreateProduct" : EntityCreate [
2      primaryActor a_Manager
3      dataEntity e_Product
4      precondition "The user is authenticated as a manager"
5      postcondition "The product is created in the system"
6      description "As a Manager I want to create a product"
7
8      mainScenario uc_3_CreateProduct_MainScenario (Main) "User creates a product" [
9      step s1 (System:Execute:Open) "Opens the products list page in the browser"
10     step s2 (Actor:CallSystem:SelectAction) "Selects the action open product page"
11     step s3 (Actor:PrepareData:Fill) "Fills in the product ID" [
12      scenario s3a_ProductIdAlreadyExists (Exception) "User fills in a product ID that already exists"
            [
13        step s3a1 (System:Execute:Check) "Validates the product ID already exists"
14        step s3a2 (System:ReturnResult:DisplayMessage) "The product ID already exists" nextStep s3 ]
15      scenario s3b_NotFillProductIdAndTheSystemAutoGeneratedIt (Alternative) "Does not fill in the
            product ID and the System auto-generated it" [
16        step s3b1 (System:Execute:Check) "Validates the product ID is empty"
17        step s3b2 (System:Execute:SaveData) "Auto-generates the product ID" nextStep s4 ] ]
18     step s4 (Actor:PrepareData:Fill) "Fills in the product name" [
19      scenario s4a_DoesNotFillInTheProductName (Exception) "User removes the product name" [
20        step s4a1 (System:Execute:Check) "Validates the product name is empty"
21        step s4a2 (System:ReturnResult:DisplayMessage) "The product name is empty" nextStep s4 ] ]
22     step s5 (Actor:CallSystem:SelectAction) "Selects the action to create the product"
23     step s6 (System:ReturnResult:DisplayMessage) "The product was successfully created"
24   ]
25 ]
```

Listing 6.1: Specification of a use case (in RSL).

The RSL syntax considers a use case with a name that uniquely identifies it, also describing a sequence of steps, a precondition and post-condition. Listing 6.1 shows a use case specification example in RSL with a main scenario, an alternative scenario and two exception scenarios. The use case structure is important as it will allow us to generate a test case for each scenario.

6.3 Implementing a Code Generator in Xtend

In the implementation of this end-to-end process (Figure 6.1), namely generate test cases from use case specifications (*Task-2a*) and test data (*Task-3a*), we developed the *"RSLUseCase2TestSuiteGenerator"* generator using the Xtend language.

The process comprises the following sequence of tasks; First, extract information from each use case elements. Second, through the sequence of steps of each use case scenario, generate the paths with the full edges' coverage algorithm that allows to generate the test cases. Third, generate concrete test data for each data entity (that are assigned in use cases) and create temporary variables. Fourth, generate the test suites with the test cases. Figure 6.1 describes the Code Generator process, which includes the generation of test suites with test cases and test data.

6.3.1 Extract Use Case Information

To extract information from the use case specifications, the use case shall include main scenarios and alternative and exception scenarios. Listing 6.1 shows the use case of "create product" with a main scenario, one alternative and two exceptions scenarios. Each scenario has its respective steps.

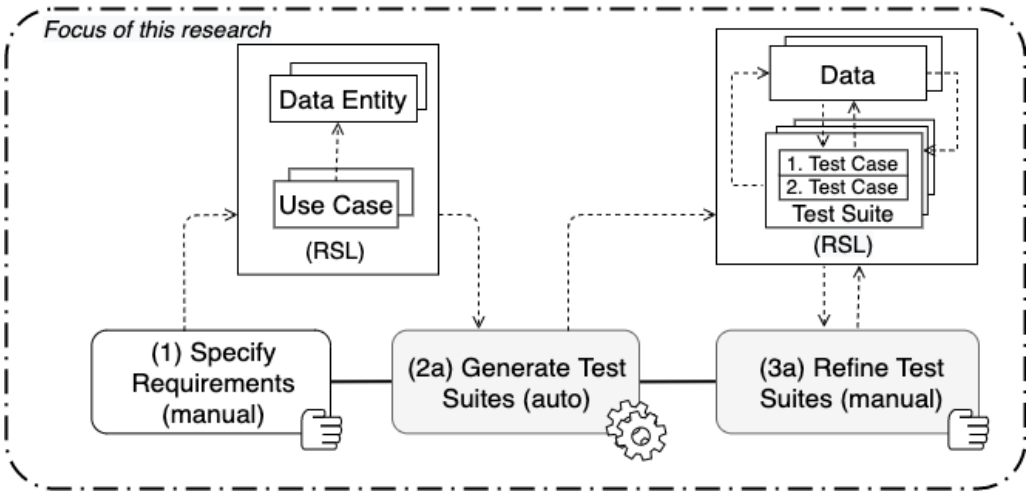

Figure 6.1: Test Suite, Test Case and Test Data Generation Process.

The code generator extracts the information using an iterator that searches over the specification file, in order to capture all the information from the RSL use case constructs.

Algorithm 1 shows how to extract all the use cases information from the specification by iterating through each of the use case's scenarios.

Algorithm 1 Extract Use Case Information

```
1: for each UseCase do
2:     for each Scenarios of the Use case do
3:         for each step scenarios do
4:             Add step to a List
5:         end for
6:     end for
7: end for
```

The same process is performed for the other constructs, like data entity and data. After capturing the structure of use case scenarios, it is necessary to generate all the paths that cover all the possible scenarios.

6.3.2 Full Edges Coverage Algorithm

The purpose of the full edges coverage algorithm is to generate a set of test cases that cover all use case scenarios at least once. Figure 6.2 represents the extracted steps from all the scenarios of the use case "create product". Each step represents a node, and the connections between these steps are the edges. The connections between two nodes, represented by a sequence of edges, is called a path.

To generate test cases from use case scenarios, it is necessary to generate all the paths that cover all the possible scenarios. In the Figure 6.2, the happy path is represented by the blue colored sequence flow, the alternative scenario is colored in green and the exception scenarios

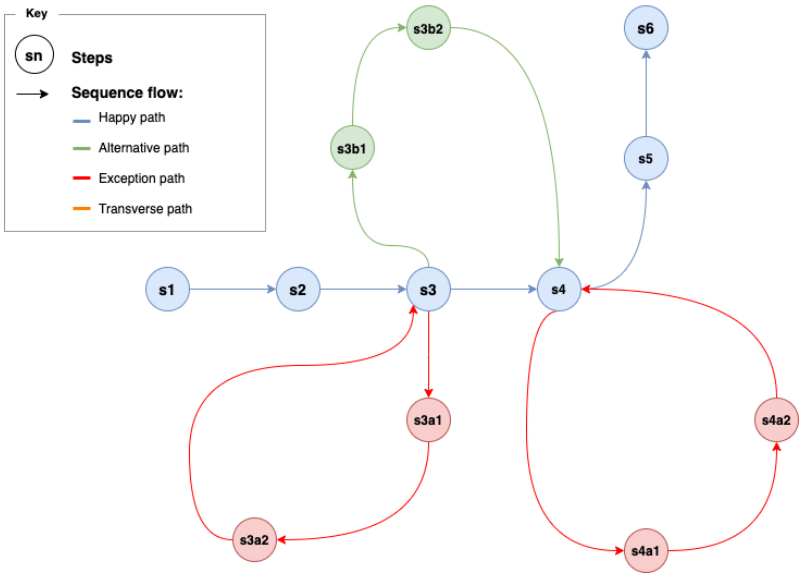

Figure 6.2: Graph representing the sequences of steps of the uc_3_CreateProduct.

in red. Happy path is the sequence of test steps that involves no alternative nor exception steps, representing the default successful flow of the program.

The generation of paths of the steps scenarios is based on a graph that is represented by an ordered pair G = (V , E) comprising a set of nodes and a set of edges [36]. A UML activity diagram is an example of a graph with the tasks represented as nodes and transitions as edges [65]. A UML activity diagram describes the dynamic aspects of the system and represents the flow from one activity to another activity [65].

Figure 6.2 shows all the nodes and the edges that the algorithm can traverse, starting from the first node and ending in the last node, to calculate all paths of a graph. Therefore, the test cases are generated from a selection of path sequences according to the "all-edges coverage" [74], which means a set of complete path sequences such that each node transition appears at least once.

This approach implemented the Full Edges Coverage (FEC) algorithm, which is a graph traversal algorithm that starts at the root node or any arbitrary node and explores as far as possible along each branch before backtracking [61]. The FEC algorithm generates the paths by going through each edge of the control flow. Starting from the first node, a decision algorithm is applied to the outgoing edges to decide the next step. If there is only one edge, a checking process is performed to verify if the next level is the end or another step. If there is more than one edge, the process continues and the steps of the next level are connected to the previous path. This process is repeated until the next level reaches the end.

The pseudocode related to the FEC algorithm is presented in Algorithm 2. This FEC algorithm starts from the root node and marks the visited edges and moves to the adjacent unmarked node. This loop continues until there is no unmarked adjacent node. Following each node, the algorithm backtracks and checks for other unmarked edges, traversing them as well. In the end, the algorithm returns the sequence of nodes along the path.

Algorithm 2 Full Edges Coverage Algorithm

```
1:  for each i  G.Adj[u] do
2:      if isVisited[u][i] == false then
3:          Add i to PathList
4:          isVisited[u][i] = true;
5:          printAllPathsUtil (s, d, isVisited, localPath List)
6:          if i == d then
7:              for each path in localpathList do
8:                  Add path to the PathList
9:              end for
10:         end if
11:     end if
12:     for each k  G.Adj[v] do
13:         isVisited[u][k] = false
14:     end for
15: end for
```

Figure 6.3 shows all the sequence of steps for the eight paths generated by the algorithm for the use case *uc_3_CreateProduct*.

6.3.3 Xtend Template to Generate the Test Suite and Test Cases

This approach defined a Xtend template to generate the test suite and test cases. After the FEC algorithm returns 8 paths from the use case *uc_3_CreateProduct*, we iterate over the results to create the test suite *ts_uc_3_CreateProduct* and generate test cases for each path. Figure 6.4 shows the Xtend template, which iterates through the algorithm results to generate the test cases. The Xtend template proceeds to generate all the test cases and also to identify the test case from the happy flow (or the main scenario) of the respective use case.

Listing 6.2 shows part of the test cases generated from the product creation use case (the full version is available in appendix B).

6.3.4 Generate Test Data

To generate data elements and useful variables to specify concrete data that can be used by test suites and test cases. The Data is generated in parallel to the test suite generation. The data generation process is important since it reduces the time-spent and costs usually associated with this kind of task. Furthermore, manual data entry is a tedious process, being considered frustrating and repetitive by many testers [24].

The code generator extracts the information using an iterator that searches over the specification file, in order to capture all the Data in the RSL specification.

Then, the Data is generated with a set of possible values to each property of the match Data Entity defined in the use case specification document. The data values are generated randomly depending on each data entity property type. If it is a string, number or boolean, it generates a

```
1  TestUseCase ts_uc_3_CreateProduct  "CreateProduct TestSuite" [
2      useCase uc_3_CreateProduct
3      postcondition v_Products[1].ID
4      variable v_Product [ withData (d_Product)]
5      variable vMessageSystemNotification [ "The product ID already exists", "The product ID is empty",
           "The product name is empty", "The product was successfully created" ]
6      setup [ step setup_1 execute ts_uc_1_Signup.t_uc_1_Signup_1 withVariable (v_User) ]
7      teardown [
8        step teardown_1 execute ts_uc_6_DeleteUser.t_uc_6_DeleteUser_0 withVariable (v_User)
9      ]
10
11     testCase t_uc_3_CreateProduct_0_happpypath "User creates a product" [
12       step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the browser"
13       step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
             product page"
14       step s3 (Actor:PrepareData:Fill from v_Products[1].ID into element ('product_id') ) "Fills in
             the product ID"
15       step s4 (Actor:PrepareData:Fill from v_Products[1].Name into element ('product_name') ) "Fills
             in the product name"
16       step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to create
              the product"
17       step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
             successfully created"
18     ]
19     testCase t_uc_3_CreateProduct_1 "User creates a product" [
20       step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the browser"
21       step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
             product page"
22       step s3 (Actor:PrepareData:Fill from v_Products[1].ID into element ('product_id') ) "Fills in
             the product ID"
23       step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_Products[1].
             Name into element ('product_id') ) "Validates the product ID already exists"
24       step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product ID
             already exists"
25       step s3 (Actor:PrepareData:Fill from v_Products[1].ID into element ('product_id') ) "Fills in
             the product ID"
26       step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from v_Products[1].
             ID into element ('product_id') ) "Validates the product ID is empty"
27       step s3b2 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the product ID
             "
28       step s4 (Actor:PrepareData:Fill from v_Products[1].Name into element ('product_name') ) "Fills
             in the product name"
29       step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[2]) from v_Products[1].
             Name into element ('product_name') ) "Validates the product name is empty"
30       step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product name
             is empty"
31       step s4 (Actor:PrepareData:Fill from v_Products[1].Name into element ('product_name') ) "Fills
             in the product name"
32       step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to create
              the product"
33       step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
             successfully created"
34     ]
35 ]
```

Listing 6.2: Example of the test suite and test case specification generated by the algorithm (in RSL).

random string, number, or boolean value. Also, valid and invalid values are taken into account in each property of the data entity. Depending on the entity's value type, a string, integer or boolean value will be generated.

Furthermore, the Data Entity defined in the use case will allow generating a Data element to be used in the test suite and test case. Listing 6.1 shows the product creation use case, which has defined data entity e_Product (Product). The Code Generator will create a variable v_Product in the test suite with the matched product data entity element defined in the use case, as described in line 3 of the Listing 6.1.

The Code Generator also creates a variable of type string array, called vMessageSystemNotification, that can be applied whenever the system returns a success message, an error message or a specific message. Listing 6.2 shows the specification of a test suite with vMessageSystemNotification variable with three string messages, namely "The product ID is empty", "The product name is empty" and "The product was successfully created". The v_Product and vMessageSystemNotification variables can be later referred to in the test cases.

Listing 6.2 shows the specification of two test cases, in which the values assigned to the internal variables are inserted into the steps, and contain the information needed to execute the flow of events.

Besides generating the test data, it is crucial to ensure the logical sequence of the system is respected when applying such data — for instance, to ensure the success of a log in test case, a user should have been previously persisted in the database.

To ensure the correct logical sequence and avoid having test cases depending on running before or after other specific test cases, setup and teardown methods can be implemented, as well as the possibility of enforcing preconditions and postconditions in each test case. Preconditions and postconditions define conditions that must be true, respectably, before a test case is allowed to be executed and after a test case finishes the execution of any of its flows — for example, to ensure the success of the sign-up test case, the precondition "The user does not exist in the system" must be true before registering the user or, as a post condition to ensure the success of the creation product test case, the product ID (v_Products[1].ID) must be created after the test is executed (Listing 6.2).

This approach provides setup and teardown mechanisms to perform the actions needed to verify that preconditions are true before executing the test. Setup defines the initial state before execution while teardown performs cleanup after execution, deleting all the data inserted or updated, thus returning all data to its original state.

These additional features in our approach ensure test case independence and execution in any given order. Hence, the success of tests does not depend on state set by a previous test neither in the order in which they are executed, therefore the failure of a test is not able to force the test suite to be stopped — as happened when tests depend on the state set by previous tests.

Ideally, the setup method performs actions that change the test case's initial state, causing its preconditions to be true. The test steps execute the actions that, in the end, should be verified as true by the postconditions. After the test case finishes its execution, the teardown method can

```
1  TestUseCase t_uc_1_SignUp "SignUp TestSuite" [
2   useCase uc_1_SignUp
3   variable v_Users [ withData (d_Users)]
4
5   testCase ts_1_1_Signup "Anonymous user signs up With Valid Information" [
6    parameter p_User
7    variable v_Users is p_User otherwise withData ( d_Users ) ... ]
```

Listing 6.3: Example of a test suite specification (in RSL).

revert all the changes in the system state, so the next test case is executed with no repercussions from the previous one.

Listing 6.2 shows the test suite create product, which requires the sign-up action for all the test cases of the test suite. The tester manually defines a setup test case that will allow the user to sign-up, receiving as parameters the required values defined in the test suite sign-up (Listing 6.3). The tester can then call the defined setup test case in other test suites. During the execution of the product creation test suite, the setup calls the sign-up test case and sends the values as parameters, as Listing 6.2 shows. At the end of the execution, all the steps are performed and the user is able to log in into the "MyStore" system and successfully create the product.

6.4 Conclusion

This chapter describes the process of generating test suites, test cases and data from use cases and scenarios specifications. The objective is to generate such test cases, directly from the use case specifications, as originally discussed in Maciel et al.'s [46] [58] end-to-end approach.

RSL supports the establishment of alignments between requirements and tests, allowing the generation of test cases from use case scenarios. The process starts with the extraction of the use case, data and data entity information specified in a RSL specification file. Based on the use case specification, all the test cases are generated from the defined test scenarios of a use case in RSL. The goal was to implement a full edge coverage algorithm to ensure that every alternative and exception branch on each use case is exercised.

Some examples were presented to comprehend a practical application of the generation of the test cases. The first example demonstrated the process of product creation. In this case, all the paths were generated and all the branches were covered, therefore generating all the test cases specifications for the use case. This automation improves the process of creating tests by reducing the effort and cost of repetitive tasks, and ensuring higher quality of both requirements and tests. Generating test cases from the initial specifications may increase confidence in the verification and validation phase of the software development process. Those tests can also be executed at the end of the generation process. The tester will then be able to execute these acceptance tests on the generated software product to verify that the process has, in fact, led to the intended result.

The second example shows the test data generation and variables that can be used by test suites and test cases. Since the manual specification of data is a repetitive and tedious process, the data

elements can be generated with the test suites' generation process, which reduces the time and costs of producing such tests.

It also addresses the problem of dependency between test execution by proposing setup and teardown mechanisms to avoid sharing of state between tests.

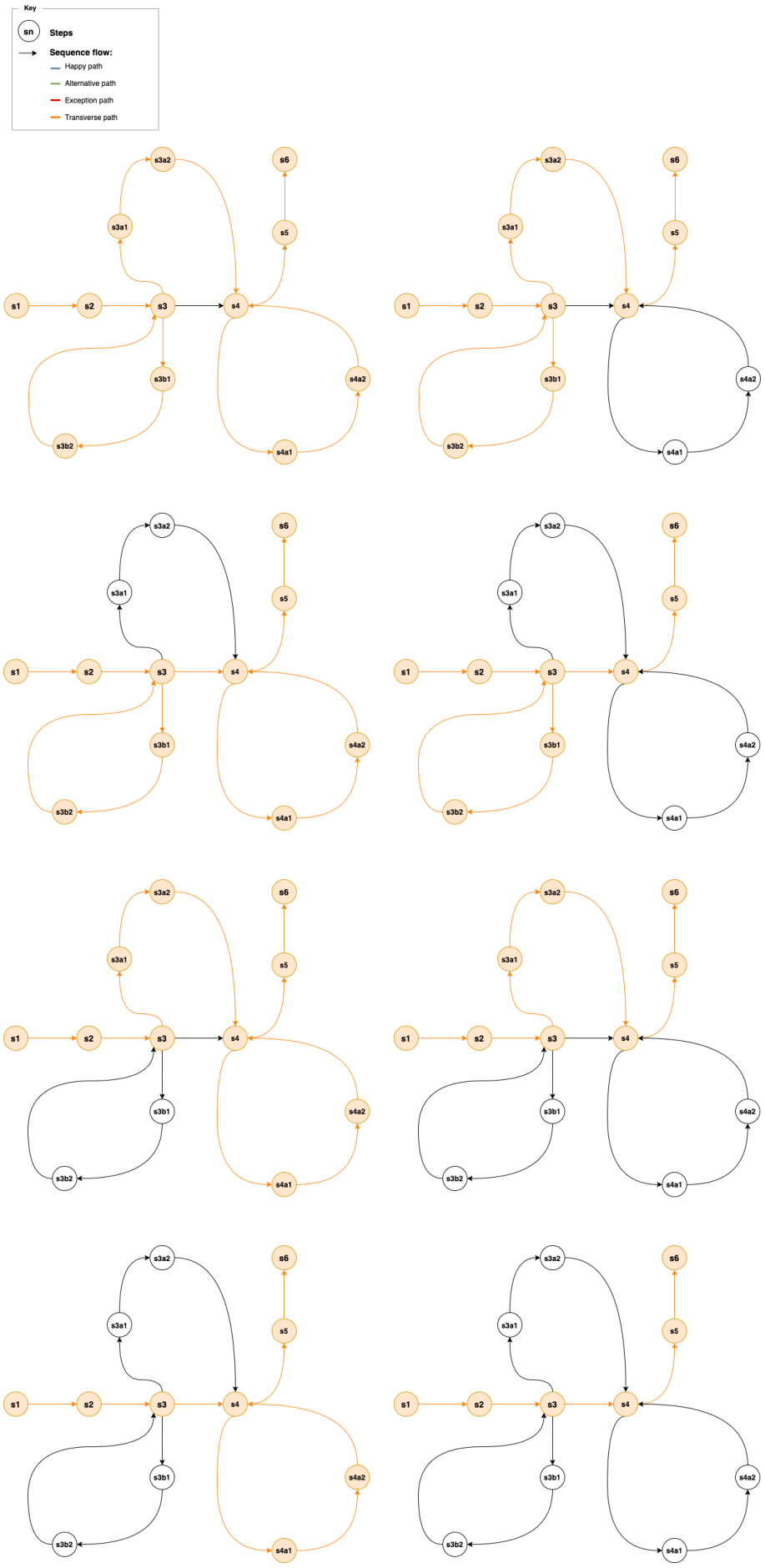

Figure 6.3: Paths generated by the FEC algorithm.

```
«FOR mainScenario : usecase.getMainScenarios()»
    «converterRSLToSteps(mainScenario, usecase)»
    «var allPaths = Graph.generateAllPaths(paths, usecase.name)»
    «FOR useCasePathsKey : allPaths.keySet()»
        «var pathsKey = allPaths.get(useCasePathsKey)»
        «FOR stepsKey : pathsKey.keySet»
            testCase t_«useCasePathsKey»_«stepsKey» "«mainScenario.nameAlias»" [
            «var paths = pathsKey.get(stepsKey)»
                «FOR path : paths»
                    «allSteps.get(path)»
                «ENDFOR»
            ]

        «ENDFOR»
    «ENDFOR»
«ENDFOR»
```

Figure 6.4: Xtend template to generate the tests (Xtend code).

Chapter 7

Evaluation of Test Case Generation

The objective of this approach is to improve and automate the generation of test cases, directly from the requirements specifications, as proposed by Maciel et al.'s [46] [58] end-to-end approach. This chapter evaluates the proposed generation of test cases from use case specification through an e-commerce application (named "MyStore") that includes features to create orders of products and checkout, and other store related operations. The examples presented in this work's demonstration are from the product purchase process.

7.1 Case Study

The proposed solution focuses on the generation of test suites and test cases. Therefore, to confirm the proposed improvements, this chapter analyzes the accuracy, quality and coverage of the generated test cases using an example from product purchase process of the "MyStore" e-commerce application (Figure 7.1).

The goal is to generate all the test cases from the buy a product example and also to generate the test data from each test suite. The process starts by identifying all the data entities, the actors and the dependent use cases from the product purchase process. We identify six use cases that cover most of the application's functionalities to buy products, which will be tested through the generated test cases using RSL specification. Figure 7.2 shows the use case diagram that summarizes the relationships between the use cases, the actors and the system. This use case diagram represents the product purchase use case and also includes the functionality of other use cases needed to successfully perform a purchase, such as sign-up, log in, shopping cart, delivery address to enable shipping options, calculate taxes and shipping and payment.

As a precondition, customers can only proceed to checkout in order to purchase all the selected items if they are registered and logged in the system. The customer starts by signing up to create an account and follows by log in the system with their information filled correctly.

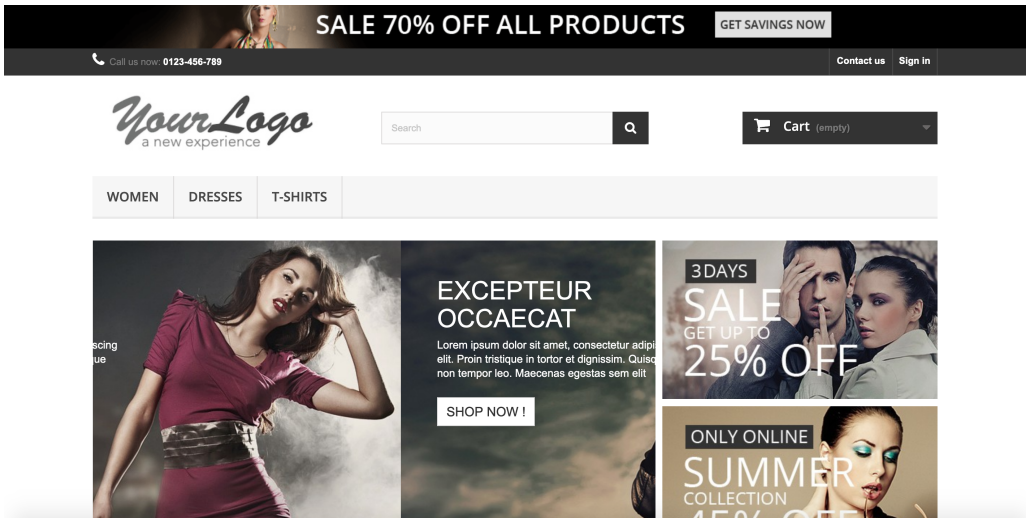

Figure 7.1: “MyStore” e-commerce application.

The “sign-up” use case describes the user registering for the first time within the application. The “sign-up” use case specification, as shown in Listing B.2, starts when the user accesses the system feature which enables him to create an account by entering his information. Figure 7.4 shows the graph representing the steps from the “sign-up” use case, with the happy path, and also illustrates two exceptions: the user signs up with different passwords and the user signs up with an email that already exists in the system.

The “log in” use case defines the process of logging in to the system to access its functionalities. Listing B.3 presents the “log in” use case specification with three scenarios: It starts with the main scenario: the user correctly types his/her email and password. As an exception scenario, the user fills in an email not registered in the system and the system displays an error message. Another exception scenario is specified with the following example: the user fills in an invalid password, and the system displays an error message. Figure 7.5 shows the graph representing the steps from the “log in” use case.

To make a purchase, the customer adds two products to the “MyStore” shopping cart. The cart persists the selected products until the customer proceeds to the checkout. Listing B.5 presents the “add item to cart” use case specification with the happy path: the user successfully adds two products to the cart. The use case has two exception scenarios. First, the user selects an out-of-stock product and second, the user selects an unavailable product size. Figure 7.6 shows the graph representing the steps from the “add item to cart” use case.

Thereafter, the customer proceeds to checkout and the system checks if the delivery address is correctly filled in. The customer can choose to fill a new address to his profile or can use the delivery address as the billing address. Listing B.6 presents the “add delivery address” use case specification with three scenarios. Figure 7.7 shows the graph representing the steps from the “add delivery address” use case. First, the user successfully adds the delivery address to the profile. As an alternative scenario, the user uses the delivery address as the billing address. In the exception

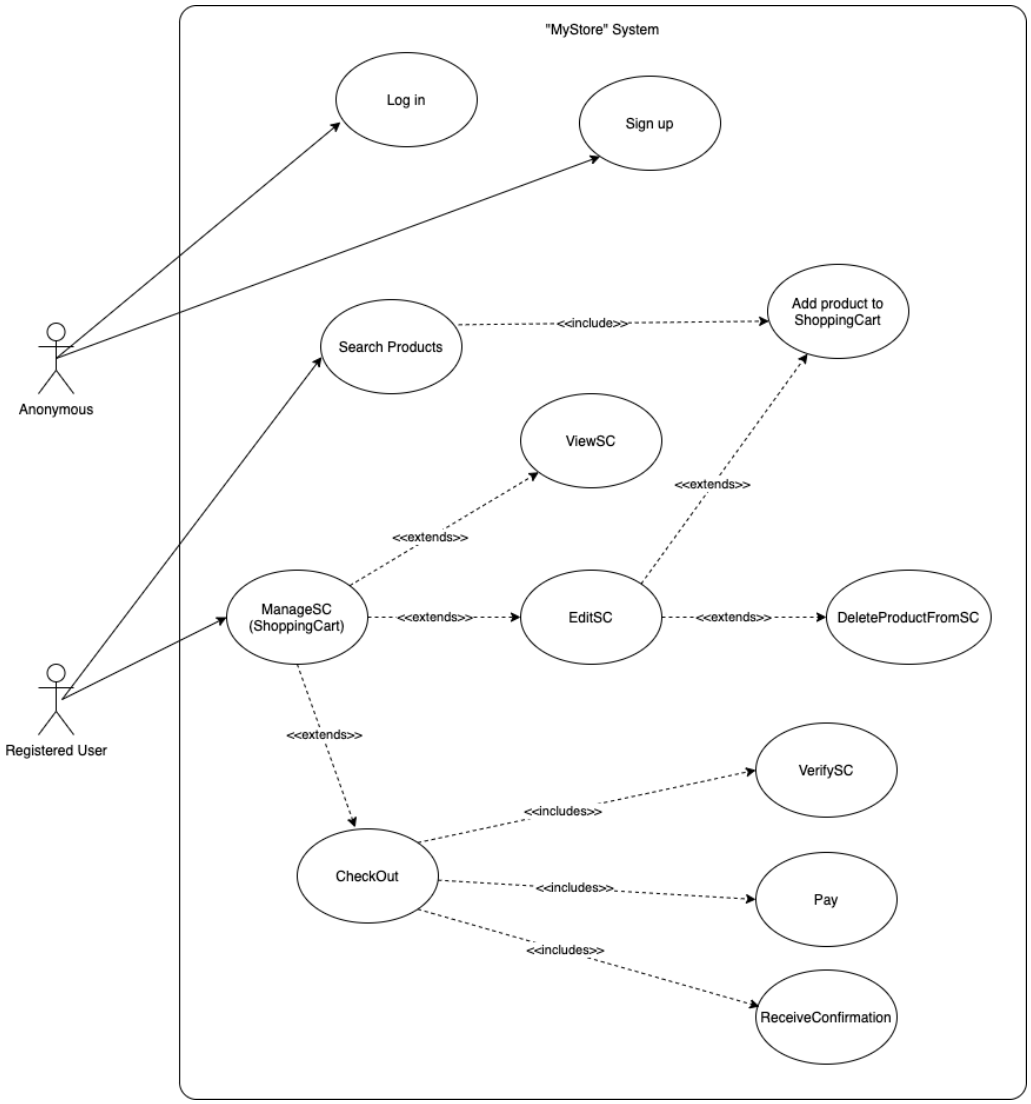

Figure 7.2: Product purchase use case diagram.

scenario, the user fills in an incorrect zip code, and the system rejects it.

In the next step, the customer has to select one of the shipping options available for his region, as well as the respective shipping fee. The system verifies if the selected shipping option is correct and available. Listing B.7 presents the "choose shipping option" use case specification with a happy path the user successfully chooses the shipping option. And two exception scenarios: first, the user chooses a shipping option not available in their country. Second, the user does not agree with the terms of service and the system displays an error message. Figure 7.8 shows the graph representing the steps from the "choose shipping option" use case.

Lastly, the customer chooses one of the payment methods available — card, bank wire and check. The customer enters the payment information, which the system validates and records. Listing B.8 presents the "choose payment method" use case specification with three scenarios. There are two common situations, the scenario of successfully purchasing items with a valid credit

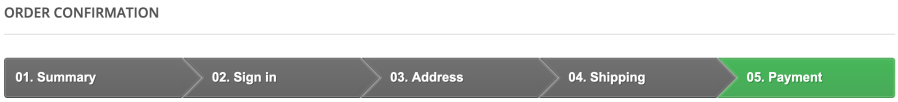

Figure 7.3: Checkout steps.

card, or the scenario of failing to purchase items because of a credit payment denial (e.g., expired card). As an alternative scenario, another payment method is allowed, such as a voucher. At the end of the process, the customer receives a receipt from the system. Figure 7.9 shows the graph representing the steps from the "choose payment method" use case. After choosing one of the previous options, the system verifies if it is valid. When all the steps are completed, the customer will be able to finish his order successfully.

7.1.1 Results

This section presents the results, describing the generated test cases and test data, according to the following evaluation criteria: generate all possible paths and automate the test data process as much as possible to reduce the tester's data refinement effort.

Table 7.1 presents the number of scenarios (main, alternative and exception) and the number of test cases generated after the algorithm execution for each use case.

In the "sign-up" use case, the FEC algorithm generates a test suite with 4 test cases. The first test case generated is the short path from the happy path. Then, it generates two different paths, each covering an exception path. The last generated test case is the longest path, which traverses the two exception scenarios. Therefore, we can verify that the test suite generated covered all the scenarios.

The FEC algorithm generates a "log in" test suite with 4 test cases. The first test case covers the happy flow, the second generated test case traverses both exception scenarios. The other two tests covered an exception scenario each. In the "log in" use case, all the scenarios are covered by a test case.

As shown in Table 7.1, the algorithm generates a "add item to cart" test suite with 4 test cases. The first test case covers the main scenario, and the second test case covers the two exception

Table 7.1: Analysis of the generation process.

	Scenarios - Inputs			**Outputs**
Use Cases	**Main**	**Alternative**	**Exception**	**Test Cases**
1 - Sign-up	1	0	2	4
2 - Log in	1	0	2	4
3 - Add item to cart	1	0	2	4
4 - Add delivery address	1	1	1	4
5 - Choose shipping option	1	0	2	4
6 - Choose payment method	1	2	0	3

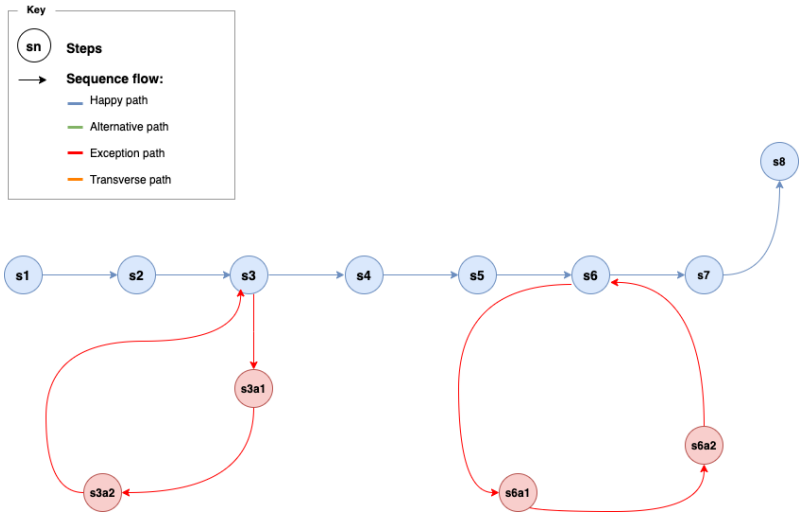

Figure 7.4: Graph representing the steps from the “Sign-up” use case.

scenarios in one path. The third only covers the first exception path, and the fourth traverses the second exception scenario. In the “add item to cart” use case, all the scenarios are covered by a test case.

In the “add delivery address” use case, the algorithm generates a test suite with 4 test cases. The first test case covers the happy path and it is the short path. The second generated test case traverses the alternative and exception scenarios. The third generates a path that only covers the alternative path. The fourth test case covers the exception path. In this use case, all the scenarios are covered with a test case.

The FEC algorithm generates a “choose shipping option” test suite with 4 test cases. The first test case covers the main scenario. The second generated test case traverses the exception scenarios. The third generates a path that only covers the first exception path. The last test case covers the second exception path. In the “choose shipping option” use case, all the scenarios are covered with a test case.

The algorithm generates a “choose payment method” test suite with 3 test cases. The first test case covers the short path (happy flow). The second generated test case traverses the first alternative scenario. The last generated test case covers the second alternative path. In the “choose payment method” use case, all the scenarios are covered with a test case.

All the test data were generated with values to each property of the data entities. To each test suite, a couple of variables was generated: one comprising the element from the data entity (v_Product or v_User), a variable with the link to “MyStore” (vURL), a variable for the database name (vDataBase) and another with the system messages to be used in each test case (vMessageSystemNotification).

When it is not possible to assign the data entity attribute to the test step, a “#TBD” (To Be Done) term is generated, so the tester can easily understand when his/her refinement is needed. For example, in the “sign-up” test suite is generated the following step in each test case: ’step

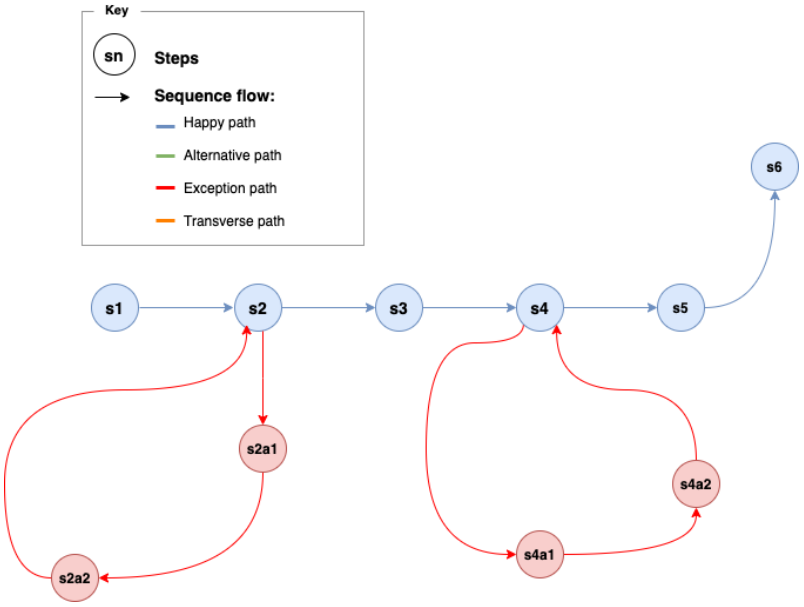

Figure 7.5: Graph representing the steps from the “Log in” use case.

s2 ("Actor:PrepareData:Fill from ’#TBD’ into element (’User_Field1’) "Fills in the name")’. The tester then should replace “#TBD” by the attribute name from the user data entity.

Another refinement occurs when the HTML element has to be filled in the test step. In Figure 7.10, the tester has to replace the element (’User_Field1’) by the “customer_firstname” HTML tag identifier (ID), with the following result: ’step s2 (Actor:PrepareData:Fill from **v_User[0].Name** into element (’**customer_firstname**’)) "Fills in the name"’.

There are some improvements that can be implemented in our approach in the future. The data entity attributes could not be automatically assigned to the respective step of the test case, requiring a manual refinement by the tester. A natural improvement would be the automatic assignment of the data entity attributes to the steps of the test case, reducing the manual input needed before executing the test suite. The mapping from HTML elements to test steps could also be automated in the future, as now it also requires manual input by the tester.

All the test cases generated are refined and available in appendix B.21.

7.2 Comparison with Related Work

Many of the analyzed approaches generate test cases from models following the MBT approach. These approaches build a model from the requirements and compare the state and behavior of a software product against this model. A comparison of our approach to the state of the art approaches for generating test cases is presented below.

Gutiérrez et al.’s [33] approach presents a process based on the Model-Driven paradigm to generate test cases from functional requirements. This approach generates test cases using metamodels and transformations following the MDE perspective. The metamodel for testing artifacts

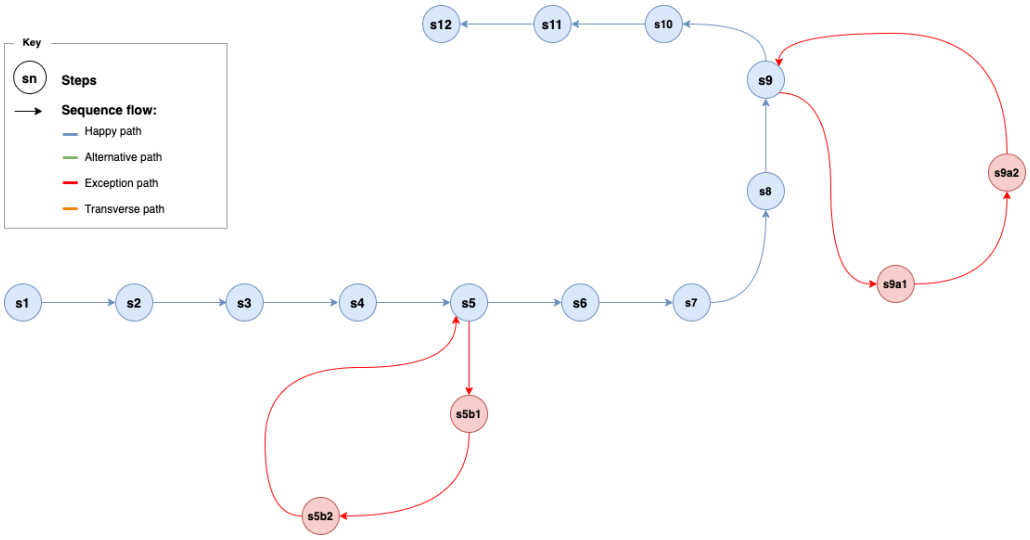

Figure 7.6: Graph representing the steps from the “Add item to cart” use case.

is generated as textual templates, UML diagrams or other syntax. The key point of this approach is the fact that metamodels, which describe the relevant information to manage in the process, do not require a specific notation. Compared to our approach, Gutiérrez et al.’s [33] lack the generation of test data to be used in the test cases, setup and teardown mechanisms and preconditions and postconditions.

Chen & Li [13] present a tool for automated test case generation from use cases. The algorithm transforms use cases to IFA and generates test cases from IFA. However, contrarily to our approach, it does not implement any way to generate test data.

Wang et al. [79] introduce UMTG, an automated approach for system test case generation, including test data and executable test cases. This approach combines NLP with constraint solving to allow the automatic identification of test scenarios and test inputs. Similarly to our approach, UMTG expresses use case specifications in a structured form, in its case called RUCM, which allows it to extract behavioral information. Furthermore, UMTG allows the specification with OCL of the preconditions, and postconditions of the use case steps that represent internal operations. However, it does not implement any mechanism to prevent shared state between tests in order to avoid unwanted dependencies between test cases, as our approach does.

Zhang et al. [80] propose an automated approach based on the RUCM approach to automatically derive test cases from requirements, based on structural test coverage criteria. The authors assess the applicability of the proposed approach to modelling four systems in the industry, using the test generation tool called aToucan4Test. However, this approach does not generate test data nor does it employ reuse mechanisms, leading to more repetition and less clear test specifications.

Hamza & Hammad [35] approach generates the test sequences from the software UML use case diagrams. This approach is advantageous since, as the UML diagrams are generated in the analysis phase to identify the software specifications, developers can also benefit from them to

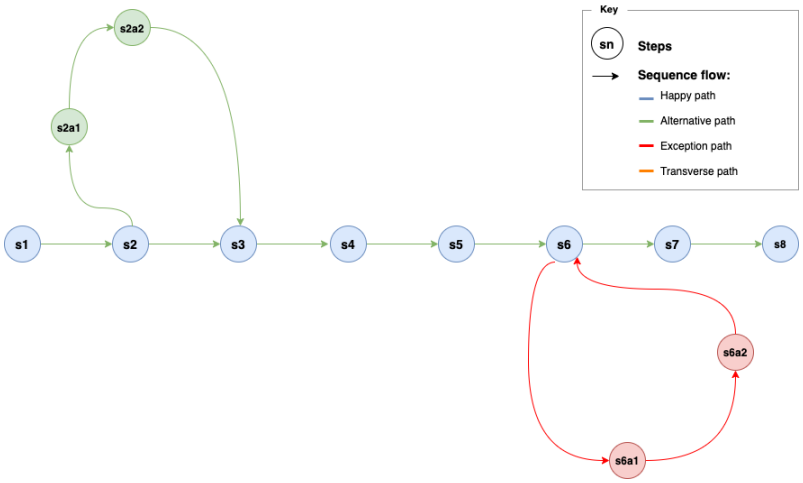

Figure 7.7: Graph representing the steps from the "Add delivery address" use case.

easily generate the test sequences. The approach makes use of the UML use case diagrams beyond the translation of software requirements to software specifications. However, this approach does not consider the order of the use cases which can affect the test sequences, since some use cases should be tested before or after a certain event.

Alrawashed et al. [2] and Somé & Cheng [72] use Cockburn's template [14] to specify use cases using a structured natural language. Similarly to what is proposed in this work, both these approaches represent the use case steps with nodes, with edges representing node sequences. Then, the steps are converted to create a control flow. After creating the control flow, the two approaches take different paths. Alrawashed et al. [2] proposed a fully automatic test approach to generate time-efficient test cases from the use case description model. The test cases are generated from the control flow diagram with a genetic algorithm to optimize and evaluate the adequacy of generated test cases. Their contribution is to generate the test cases with the highest test coverage with a near-optimal number of test cases. On the other hand, Somé & Cheng [72] generate test cases using depth-first traversal of the control flow, but in a not fully automated way. Our approach goes beyond these two approaches by generating test cases and data using information defined in the specifications.

Fischbach et al.'s [27] approach translates acceptance criteria into a CEG to automatically derive the test cases. Each acceptance criterion is converted into a dependency tree, which is traversed by the algorithm to extract the causes and effects. The algorithm generates the minimum number of test cases from the CEG by applying the BPST. However, the full generation of test cases, from acceptance criteria, can hardly be achieved and, so, according to some researchers, this approach should be seen as a supplement to the existing manual process.

The comparison of test case generation approaches results can be consulted in Table 7.2.

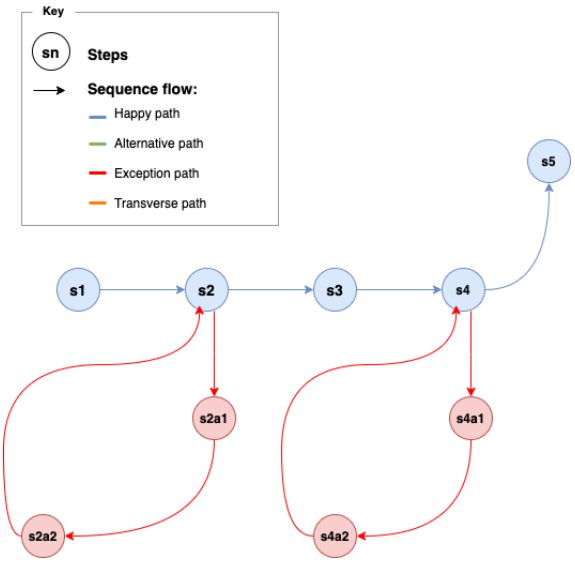

Figure 7.8: Graph representing the steps from the "Choose shipping option" use case.

7.3 Conclusion

Our approach uses RSL, a CNL, to specify software requirements in a more systematic and consistent way, decreasing the possibility of errors and misunderstandings during both requirements and test specification. The expression of use case specifications in a structured form is similar to what has been done by Wang et al. [79].

This approach generates the test cases with a full edges coverage algorithm to ensure that all the use case scenarios are covered, after constructing a control flow from nodes and edges in a similar way to Alrawashed et al. [2] and Somé & Cheng [72]. Furthermore, it generates test data with the test suites' generation process from the RSL specifications, which reduces the time and costs needed to specify tests. The test case generation improves what has been done previously by Gutiérrez et al. [33], Chen & Li [13], Zhang et al. [80], Alrawashed et al. [2], Somé & Cheng [72] and Fischbach et al. [27].

Table 7.2: Comparison of test case generation approaches.

Approaches	Generation of test data	Reuse mechanisms	Refinement
Gutiérrez et al. (2015) [33]	no	no	no
Chen & Li (2010) [13]	no	no	no
Wang et al. (2019) [79]	yes	yes	yes
Zhang et al. (2014) [80]	no	no	no
Hamza & Hammad (2020) [35]	no	no	no
Alrawashed et al. (2019) [2]	no	no	no
Somé & Cheng (2008) [72]	no	no	no
Fischbach et al. (2020) [27]	no	no	no
Current Approach (2021)	**yes**	**yes**	**yes**

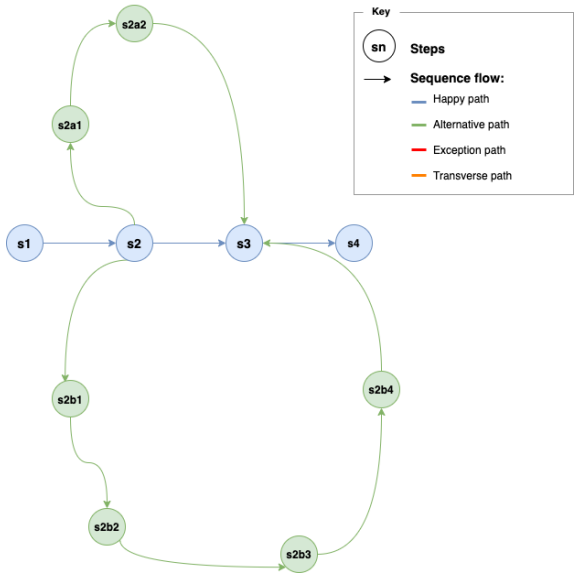

Figure 7.9: Graph representing the steps from the “Choose payment method” use case.

Moreover, this approach implemented refinement, reuse and sequencing manual mechanisms that can be applied by the tester to the generated test specifications. These mechanisms avoid the downsides and improve on contributions from Gutiérrez et al. [33], Zhang et al. [80] and Hamza & Hammad [35].

This approach allows testers to define preconditions and postconditions, providing sequencing mechanisms that specify the state in which a test suite can be executed. The setup and teardown mechanisms are a very important contribution, reducing the shared state between tests, as verified in Wang et al.’s [79] approach, and avoiding dependencies among them during test suite execution.

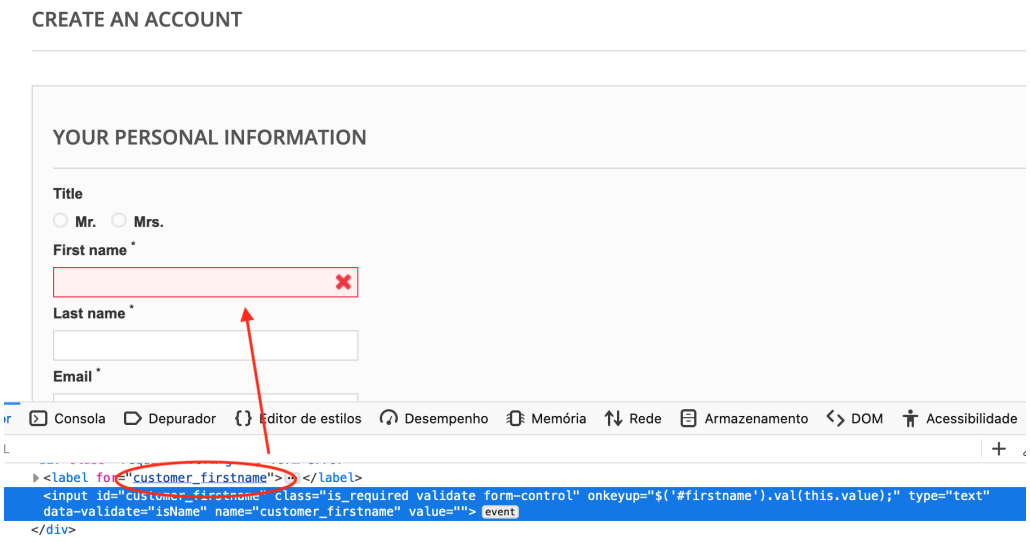

Figure 7.10: HTML element mapping.

Chapter 8

Conclusion

The present investigation arose from the inconsistency, incompleteness and ambiguity problems that could result from describing requirements in NL and the difficulty of using formal complemented with models and semi-formal methods.

In this dissertation, the focus is on the level of user requirements, NL and use cases which are more suitable than formal specification languages, since all the stakeholders need to easily understand the requirements and the system requirements document serves as a contract between the concerned parts. Although NL has several disadvantages, such as ambiguities, inaccuracies and redundancies, it's the most used technique to describe requirements from the system. To get the best out of both solutions (i.e., of natural languages and semi-formal languages), some authors have proposed the use of CNLs. The usage of a CNL decreases the possibility of errors and misunderstandings during both the requirements and tests specification and may contribute to better validate these specifications. RSL is one example of a well-defined CNL for RE, which allows specifying requirements and tests in a rigorous way but still keeping the specifications human-readable. Using a more structured requirements' specification language at an early stage of the software development process contributes to a reduction in the effort of test design, by automatically generating test cases, since the manual effort required to produce requirements specifications is high.

Once RSL supports both the specification of both requirements and tests in an integrated way, it proved to be the most suitable language to generate test suites and test cases from use cases specified in RSL.

RSL approach proposes to use structured and CNL processing based on former languages to help to specify software requirements in a more systematic and consistent way. Silva et al. [17] classifies RSL as a formal language, because "it consists of a set of constructs with well-defined semantics, and is computer-processable through a notation that can be represented through a context-free grammar". However, the authors do not consider RSL to be a formal method language, since

"the design principles of RSL are distinct from those of formal method languages, which are more oriented toward the mathematical/logical representation" [17].

This investigation extended the end-to-end approach previously proposed by Maciel et al. [46] [58]. In their research, the authors discussed an "end-to-end approach" that intends to generate test scripts from requirements and test specifications written manually in RSL. Nonetheless, some limitations were identified in this previous approach, which were attended to in the current investigation. First, the *Tasks-2a* and *-3a* could be automated to some extent, and hence, could show a higher productivity. Second, the design of the RSL could be extended with more details, namely use case scenarios, and test cases with a sequence of concrete test steps. Consequently, the key contribution of this work is to discuss how to extend and improve these new tasks of that end-to-end process: Generate Test Suites (*Task-2a*) and Refine Test Suites (*Task-3a*). Since the manual specification of data is a repetitive and tedious process, the data elements were also generated with the test suites' generation process, which reduces the time and costs of producing such tests.

The focus of our proposal involved the tasks: specify requirements, generate test suites and manually refine test suites. RSL supports the specification of test suites and test cases directly from the requirements specifications. Our proposal also has implemented refinement, reuse and sequencing manual mechanisms that can be applied to the generated test specifications. The tester can refine and define concrete values to the data entities and create temporary variables, in order to perform the steps to achieve the results and compare it with the expected values. Furthermore, the tester can still manually define sequencing mechanisms, which allow executing a test case successfully following a specific sequence. Finally, the tester can use reuse mechanisms that have some advantages: first, they are easy to write and clear test specifications. Second, less chance of setting up too much or too little for the given test. And third, less chance of sharing state between tests, which creates unwanted dependencies between them.

To carry out the test cases generation based on the use case specification, a full edges coverage algorithm was implemented to cover all the edges to ensure that every alternative and exception branch on each use case is exercised. The results were satisfactory and all the branches were covered, therefore, generating all the test cases specifications for the use cases examples. We verify that this automation improves the process of creating tests by reducing the effort and cost of repetitive tasks, and ensuring higher quality of both requirements and tests. All the test cases expected were generated from the purchase product example in the "MyStore" e-commerce application.

We achieve the proposed objectives from the generation of test cases from the requirements' specification, the extension of the language and also the data generation to be performed in the test. However, we identify some open issues and challenges to be addressed. The first open issue is to integrate the contributions of this research with the entire test automation process originally proposed by Maciel et al. [46] [58], namely considering its integration with test automation and Robot process automation tools.

In the future, we want to apply this research to more case studies to perform validation of this approach. We also pretend to extend this research to support other types of requirements and tests, for instance for cyber-security testing.

Finally, explore machine learning and artificial intelligence techniques to improve the generation of concrete data elements in order to be automatically assigned to test suites.

Appendix A

RSL Grammar

This appendix contains all the improved grammar elements used in this approach.

A.1 Use Case

```
1  UseCase:
2    'UseCase' name=ID  (nameAlias=STRING)?  ':' type=UseCaseType ('['
3      ((isNegative ?= 'isNegative') | (isPositive ?= 'isPositive'))?
4      ((isConcrete ?= 'isConcrete') | (isAbstract ?= 'isAbstract'))?
5      ((isSolution ?= 'isSolution') | (isProblem ?= 'isProblem'))?
6      ('stakeholder' stakeholder=[Stakeholder | QualifiedName] )?
7
8      ('primaryActor' primaryActor=[Actor | QualifiedName] )
9      ('supportingActors' supportingActors+=RefActor)?
10     ('triggeredBy' triggeredBy=[ActiveEvent | QualifiedName])?
11
12     ('dataEntity' dataEntity=[DataEntityGeneric | QualifiedName] )?
13
14     ('precondition' precondition=STRING)?
15     ('postcondition' postcondition=STRING)?
16
17     (actions= UCActions)?
18     (extensionPoints= UCExtensionPoints)?
19
20     (includes= UCIncludes)?
21     (extends+= UCExtends)*
22
23     (acceptanceCriteria+= AcceptanceCriteria)*
24
25     ('priority' priority=PriorityType)?
26     (tags+=Tag)*
27     ('description' description=STRING)?
28
29     (mainScenarios+=MainScenario)*
30   ']')?;
31
32 /************************************ */
33 MainScenario:
34   'mainScenario' name=ID '(' 'Main' ')' (nameAlias=STRING)?  ('['
35     ('description' description=STRING)?
36     (steps+=Step)*
37   ']')?;
38
39 /************************************ */
40 Step:
41 'step' name=ID
42
43   ( ( '(' (type=StepOperationType (':' subType=StepOperationSubSubType )?  ) ')'
         (nameAlias=STRING))
44     |
```

```
45      ( ('<' typeUC=StepOperationUCType) (usecase= [UseCase | QualifiedName]) '>'
            (nameAlias=STRING)? )
46    )
47
48    ('actor' actor=[Actor | QualifiedName] )?
49    ('repeat' repeat=STRING)?
50    ('description' description=STRING)?
51    ('nextStep' next=[Step | QualifiedName])?
52    ('[' (scenarios += Scenario)* ']' )?
53    ;
54
55  /************************************ */
56  Scenario:
57    'scenario' name=ID  '(' type=ScenarioType ')' (nameAlias=STRING)  ('repeat' repeat=STRING)?
          ('['
58
59      ('description' description=STRING)?
60      ( steps+=Step )*
61    ']')?;
```

Listing A.1: Use case grammar (in RSL).

A.2 Test Suite

```
1   TestUseCase:
2     'TestUseCase' name=ID  (nameAlias=STRING)?   ('['
3     'useCase' useCase=[UseCase | QualifiedName]
4
5     (tags+=Tag)*
6
7     ('description' description=STRING)?
8     (variables+=TestVariable)*
9
10    ('oneTimeSetup' '[' (oneTimeSetupSteps+=SetupStep)* ']')?
11
12    ('oneTimeTeardown' '[' (oneTimeTeardownSteps+=SetupStep)* ']')?
13
14    ('setup' '[' (setupSteps+=SetupStep)* ']')?
15
16    ('teardown' '[' (teardownSteps+=SetupStep)* ']')?
17
18    (scenarios+=TestScenario)*
19    ']')?;
20
21  /************************************ */
22  TestScenario:
23    'testCase' name=ID  (nameAlias=STRING)?  ('['
24      (testsParameter+= TestsParameter)*
25    ('precondition' (testScenarioPre=[Data | QualifiedName] | testScenarioPre=[Data |
          QualifiedName] array ?='['(INT)?']' | (testScenarioPre= [TestVariable | QualifiedName]
          array ?='['(INT)?']' nameAlias=STRING) | (testScenarioPre = [DataAttribute |
          QualifiedName] nameAlias=STRING)) precondition=STRING)?
26    ('postcondition' (testScenarioPre=[Data | QualifiedName] | testScenarioPre=[Data |
          QualifiedName] array ?='['(INT)?']' | (testScenarioPre= [TestVariable | QualifiedName]
          array ?='['(INT)?']' nameAlias=STRING) | (testScenarioPre = [DataAttribute |
          QualifiedName] nameAlias=STRING)) precondition=STRING)?
27
28    ((isConcrete ?= 'isConcrete') | (isAbstract ?= 'isAbstract'))?
29
30    (testsVariable+= TestsVariable)*
31
32    ('dataEntity' entity= [DataEntity | QualifiedName]  ('withValues' '(' entityTable=
          DataAttributeValues ')'))?
33
34    ('description' description=STRING)?
35
36    (testSteps+= TestStep)*
37    ']')?;
38
```

```
39 /************************************ */
40 TestStep:
41 'step' name=ID
42     ( '(' (type=StepOperationType (':' extension=OperationExtension)?  ) ')'
           (nameAlias=STRING))
43
44   ('actor' actor=[Actor | QualifiedName] )?
45   ('repeat' repeat=STRING)?
46   ('description' description=STRING)?
47   ('nextSteps' next=[Step | QualifiedName])?
48 ;
```

Listing A.2: Test suite grammar (in RSL).

A.3 Test Case

```
1  TestScenario:
2    'testCase' name=ID  (nameAlias=STRING)?  ('['
3      (testsParameter+= TestsParameter)*
4    ('precondition' (testScenarioPre=[Data | QualifiedName] | testScenarioPre=[Data |
         QualifiedName] array ?='['(INT)?']' | (testScenarioPre= [TestVariable | QualifiedName]
         array ?='['(INT)?']' nameAlias=STRING) | (testScenarioPre = [DataAttribute |
         QualifiedName] nameAlias=STRING)) precondition=STRING)?
5    ('postcondition' (testScenarioPre=[Data | QualifiedName] | testScenarioPre=[Data |
         QualifiedName] array ?='['(INT)?']' | (testScenarioPre= [TestVariable | QualifiedName]
         array ?='['(INT)?']' nameAlias=STRING) | (testScenarioPre = [DataAttribute |
         QualifiedName] nameAlias=STRING)) precondition=STRING)?
6
7    ((isConcrete ?= 'isConcrete') | (isAbstract ?= 'isAbstract'))?
8
9    (testsVariable+= TestsVariable)*
10
11   ('dataEntity' entity= [DataEntity | QualifiedName]  ('withValues' '(' entityTable=
         DataAttributeValues ')'))?
12
13   ('description' description=STRING)?
14
15   (testSteps+= TestStep)*
16   ']')?;
17
18
19 TestStep:
20 'step' name=ID
21     ( '(' (type=StepOperationType (':' extension=OperationExtension)?  ) ')'
           (nameAlias=STRING))
22
23   ('actor' actor=[Actor | QualifiedName] )?
24   ('repeat' repeat=STRING)?
25   ('description' description=STRING)?
26   ('nextSteps' next=[Step | QualifiedName])?
27 ;
```

Listing A.3: Test suite grammar (in RSL).

Appendix B

"MyStore" Case Study

To support the "MyStore" run example, the following requirements' specification is used for each use case: (1) Sign-up; (2) Log in; (3) Create product; (4) Add item to shopping cart; (5) Add delivery address; (6) Choose shipping option; (7) Choose payment method; (8) Delete product; and (9) Delete user.

B.1 Use Case Specification

```
Package WebApp
System myStore "MyStore - RSL Specification" : Application []

/*******************************************************
   DataEntities view
*******************************************************/

DataEntity e_User "User" : Master [
  attribute ID "ID" : Integer [isNotNull isUnique]
  attribute firstName "Name" : Text [isNotNull]
  attribute email "Email" : Text [isNotNull isUnique]
  attribute password "Password" : Text [isNotNull isEncrypted]
  attribute statusActive "StatusActive" : Boolean
  attribute role "Role" : Text [isNotNull]
  primaryKey (ID)
  description "User"]

DataEntity e_Product "Product" : Master [
  attribute ID "ID" : Integer [isUnique]
  attribute SKU "SKU" : Integer [isUnique]
  attribute Name "Name" : Text [isNotNull]
  attribute VATCode "VATCode" : Integer
  attribute VATValue "VATValue" : Integer
  attribute Value "Value" : Integer
  primaryKey (ID)
  description "Product"]

Actor a_Admin "Administrator" : User [description "Actor who manages products and orders, etc."]
Actor a_Manager "Manager" : User [description "Actor who manages products, etc."]
Actor a_User "User" : User [description "Actor who create orders of products"]
Actor a_Anonymous "Anonymous" : User [description "Anonymous uses the system"]


/*******************************************************
   Data
*******************************************************/

Data d_User : e_User [
  withValues
```

```
    | e_User.ID | e_User.firstName | e_User.email      | e_User.password   | e_User.role   +|
    | "1001"  | "John"       | "john@email.com"   | "#password1234"   | "Anonymous"   +|
    | "1002"  | "Frank"      | "frank@email.com"    | "#password1235"   | "User"      +|
    | "1003"  | "Peter"      | "peter@email.com"    | "#password1236"   | "Anonymous"   +|
]

Data d_Product : e_Product [
  withValues
    | e_Product.ID  | e_Product.SKU  | e_Product.Name | e_Product.VATValue  | e_Product.Value
        +|
    | "20001"     | "9001"   | "Product 1"    | 23%         | 100.99      +|
    | ""        | "9002"   | "Product 2"    | 11%         | 1000        +|
    | "20003"     | "9003"   | "Product 3"    | 11%         | 3000        +|
    | "20004"     | "9004"   | ""       | 11%         | 8000        +|
]
```

Listing B.1: Data entities & actors and data specification (in RSL).

```
/*******************************************************
   UseCase
*******************************************************/
UseCase uc_1_Signup "Signup" : EntityCreate [
  primaryActor a_Anonymous
  dataEntity e_User
  precondition "The User email can not be already registered in the system."
  postcondition "The new user is created in the system"
  description "As an Anonymous user I want to sign up to become a user"

  mainScenario uc_1_SignUp_MainScenario (Main) "Anonymous User Signs Up With his/her
       Information" [
  step s1 (System:Execute:Open) "Opens the sign up page in the browser"
  step s2 (Actor:PrepareData:Fill) "Fills in the name"
  step s3 (Actor:PrepareData:Fill) "Fills in the email" [
     scenario s3a_AnEmailAlreadyExistsInTheSystem (Exception) "Anonymous User Signs Up With An
         Email That Already Exists In The System" [
       step s3a1 (System:Execute:Check) "Validates if the submitted email does not exists"
       step s3a2 (System:ReturnResult:DisplayMessage) "The submitted email does not exists"
           nextStep s3 ] ]
  step s4 (Actor:PrepareData:Fill) "Fills in the password"
  step s5 (Actor:PrepareData:Fill) "Fills in the confirm password"
  step s6 (Actor:CallSystem:SelectAction) "Selects the action sign up" [
     scenario s6a_SignUpWithDifferentPasswords (Exception) "Anonymous User Signs Up With
         Different Passwords" [
       step s6a1 (System:Execute:Check) "Validates the passwords are different"
       step s6a2 (System:ReturnResult:DisplayMessage) "The passwords are different" nextStep s6
           ] ]
  step s7 (System:Execute:Check) "Validates the passwords are correct"
  step s8 (System:ReturnResult:DisplayMessage) "The user was successfully created in the system"
  ]
]
```

Listing B.2: Sign-up use case specification (in RSL).

```
UseCase uc_2_Login "Login" : EntityRead [
  primaryActor a_Anonymous
  dataEntity e_User
  postcondition "The user gets access to the system"
  description "As an Anonymous user I want to login in the system"

    mainScenario uc_2_Login_MainScenario (Main) "Login With his/her Information" [
   step s1 (System:Execute:Open) "Opens the login page in the browser"
   step s2 (Actor:PrepareData:Fill) "Fills in the email" [
     scenario s2a_NotYetRegisteredScenario (Exception) "Login with an email not yet registered"
         [
       step s2a1 (System:Execute:Check) "Validates the email is not registered"
       step s2a2 (System:ReturnResult:DisplayMessage) "The email is not registered" nextStep s2
           ] ]
   step s3 (Actor:PrepareData:Fill) "Fills in the password"
   step s4 (Actor:CallSystem:SelectAction) "Selects the action login" [
```

```
15      scenario s4a_WrongPasswordScenario (Exception) "Login with a wrong password" [
16        step s4a1 (System:Execute:Check) "Validates the password is invalid"
17        step s4a2 (System:ReturnResult:DisplayMessage) "The password is invalid" nextStep s4 ] ]
18     step s5 (System:Execute:Check) "Validates the passwords are correct"
19     step s6 (System:ReturnResult:ShowData) "Displays profile menu with user information" ]
20 ]
```

Listing B.3: Log in use case specification (in RSL).

```
1  UseCase uc_3_CreateProduct "CreateProduct" : EntityCreate [
2    primaryActor a_Manager
3    dataEntity e_Product
4    precondition "The user is authenticated as a manager"
5    postcondition "The product is created in the system"
6    description "As a manager I want to create a product"
7
8    mainScenario uc_3_CreateProduct_MainScenario (Main) "User creates a product" [
9    step s1 (System:Execute:Open) "Opens the products list page in the browser"
10   step s2 (Actor:CallSystem:SelectAction) "Selects the action open product page"
11   step s3 (Actor:PrepareData:Fill) "Fills in the product ID" [
12      scenario s3a_ProductIdAlreadyExists (Exception) "User fills in a product ID that already
            exists" [
13        step s3a1 (System:Execute:Check) "Validates the product ID already exists"
14        step s3a2 (System:ReturnResult:DisplayMessage) "The product ID already exists" nextStep
              s3 ]
15      scenario s3b_NotFillProductIdAndTheSystemAutoGeneratedIt (Alternative) "Does not fill in
            the product ID and the System auto-generated it" [
16        step s3b1 (System:Execute:Check) "Validates the product ID is empty"
17        step s3b2 (System:Execute:SaveData) "Auto-generates the product ID" nextStep s4 ] ]
18   step s4 (Actor:PrepareData:Fill) "Fills in the product name" [
19      scenario s4a_DoesNotFillInTheProductName (Exception) "User removes the product name" [
20        step s4a1 (System:Execute:Check) "Validates the product name is empty"
21        step s4a2 (System:ReturnResult:DisplayMessage) "The product name is empty" nextStep s4 ]
              ]
22   step s5 (Actor:CallSystem:SelectAction) "Selects the action to create the product"
23   step s6 (System:ReturnResult:DisplayMessage) "The product was successfully created"
24   ]
25 ]
```

Listing B.4: Create product use case specification (in RSL).

```
1  UseCase uc_4_AddItemToShoppingCart "Add Item to Shopping Cart" : EntityCreate [
2    primaryActor a_User
3    dataEntity e_Product
4    postcondition "The product is added to cart"
5    includes uc_2_Login
6    description "As an User I want to buy a product"
7
8    mainScenario uc_6_AddItemToShoppingCart_MainScenario (Main) "User adds a product to the cart"[
9    step s1 (System:Execute:Open) "Opens the MyStore site in the browser"
10   step s2 (Actor:CallSystem:Select) "Selects a T-shirt"
11   step s3 (Actor:CallSystem:Select) "Chooses the product color"
12   step s4 (Actor:CallSystem:Select) "Chooses the product size"
13   step s5 (Actor:CallSystem:Select) "Chooses two units" [
14     scenario s5a_ProductWithoutStock (Exception) "User Selects a Product out stock" [
15       step s5a1 (System:Execute:Check) "Validates the product is out of stock"
16       step s5a2 (System:ReturnResult:DisplayMessage) "The product is out of stock" nextStep s5
             ] ]
17   step s6 (Actor:CallSystem:SelectAction) "Selects the action continue shopping"
18   step s7 (Actor:CallSystem:Select) "Selects a Dress"
19   step s8 (Actor:CallSystem:Select) "Chooses the product color"
20   step s9 (Actor:CallSystem:Select) "Chooses the product size" [
21     scenario s9a_ProductSizeNotAvailable (Exception) "User Selects a product size not
           available" [
22       step s9a1 (System:Execute:Check) "Validates the product size is not available"
23       step s9a2 (System:ReturnResult:DisplayMessage) "The product size is not available"
             nextStep s9 ] ]
24   step s10 (Actor:CallSystem:Select) "Chooses one unit"
25   step s11 (Actor:CallSystem:SelectAction) "Selects the action checkout"
```

```
  step s12 (System:ReturnResult:DisplayMessage) "The product is successfully added to the cart"
  ]
]
```

Listing B.5: Add item to shopping cart use case specification (in RSL).

```
UseCase uc_5_AddDeliveryAddress "Add delivery address" : EntityCreate [
  primaryActor a_User
  dataEntity e_User
  postcondition "The delivery address is added to the profile"
  includes uc_2_Login
  description "As an User I want to add my address to the profile"

  mainScenario uc_7_AddDeliveryAddress_MainScenario (Main) "User adds the delivery address to
       the profile"[
  step s1 (System:Execute:Open) "Opens the MyStore site in the browser"
  step s2 (Actor:CallSystem:Select) "Selects the delivery address option"[
    scenario s2a_UsesBillingAddress (Alternative) "Uses the delivery address as the billing
         address" [
      step s2a1 (System:CallSystem:Check) "Validates that the billing address exists"
      step s2a2 (Actor:CallSystem:Select) "Selects the billing address" nextStep s3 ]]
  step s3 (Actor:CallSystem:SelectAction) "Selects the action add a new address"
  step s4 (Actor:PrepareData:Fill) "Fills in the street"
  step s5 (Actor:PrepareData:Fill) "Fills in the zip code"
  step s6 (Actor:CallSystem:SelectAction) "Selects the action save address" [
    scenario s6a_ZipCodeIsIncorrect (Exception) "User fills in an incorrect zip code" [
      step s6a1 (System:Execute:Check) "Validates that the zip code is incorrect"
      step s6a2 (System:ReturnResult:DisplayMessage) "The zip code is incorrect" nextStep s7 ] ]
  step s7 (Actor:CallSystem:SelectAction) "Selects the action confirms delivery address"
  step s8 (System:ReturnResult:DisplayMessage) "The delivery address was successfully added"
  ]
]
```

Listing B.6: Add delivery address use case specification (in RSL).

```
UseCase uc_6_ChooseShippingOption "Choose Shipping Option" : EntityCreate [
  primaryActor a_User
  dataEntity e_User
  postcondition "The shipping option is added to the profile"
  includes uc_2_Login
  description "As an User I want to add my address to the profile"

  mainScenario uc_8_ChooseShippingOption_MainScenario (Main) "User chooses the shipping option
       "[
  step s1 (System:Execute:Open) "Opens the MyStore site in the browser"
  step s2 (Actor:CallSystem:Select) "Selects the shipping option" [
    scenario s2a_ShippingOptionNotAvailable (Exception) "User chooses a shipping option not
         available for his/her country" [
      step s2a1 (System:Execute:Check) "Validates the delivery option is not available"
      step s2a2 (System:ReturnResult:DisplayMessage) "The shipping option is not available for
           his/her country" nextStep s2 ] ]
  step s3 (Actor:CallSystem:SelectAction) "Selects the action checkout"
  step s4 (System:Execute:Check) "Validates the selected option agree with terms of service" [
    scenario s4a_CheckAgreeTermsOfService (Exception) "User does not agree with terms of
         service" [
      step s4a1 (System:Execute:Check) "Validates the checkbox is not selected"
      step s4a2 (System:ReturnResult:DisplayMessage) "The agree with terms of service option
           must be selected" nextStep s4 ] ]
  step s5 (System:ReturnResult:DisplayMessage) "The agree with terms of service option was
       successfully selected"
  ]
]
```

Listing B.7: Choose shipping option use case specification (in RSL).

```
UseCase uc_7_ChoosePaymentMethod "Choose Payment Method" : EntityCreate [
  primaryActor a_User
  dataEntity e_User
  postcondition "The payment method is added to the profile"
  includes uc_2_Login
  description "As an User I want to add my payment method to the profile"

  mainScenario uc_9_ChoosePaymentMethod_MainScenario (Main) "User chooses the payment method "[
  step s1 (System:Execute:Open) "Opens the MyStore site in the browser"
  step s2 (Actor:CallSystem:Select) "Selects the payment method" [
    scenario s2a_ApplyDiscountVoucher (Alternative) "User applies discount voucher" [
        step s2a1 (Actor:PrepareData:Fill) "Fills in the voucher code"
        step s2a2 (System:Execute:SaveData) "Discount the voucher value from the total amount"
             nextStep s3 ]
      scenario s2b_AddNewCard (Alternative) "User adds a new card" [
        step s2b1 (Actor:CallSystem:SelectAction) "Selects the action add a new card"
      step s2b2 (Actor:PrepareData:Fill) "Fills in the card number"
      step s2b3 (Actor:PrepareData:Fill) "Fills in expiration date"
      step s2b4 (Actor:PrepareData:Fill) "Fills in CVV" nextStep s3 ] ]
  step s3 (Actor:CallSystem:SelectAction) "Selects the action confirm the payment method"
  step s4 (System:ReturnResult:DisplayMessage) "The payment method was successfully added" ]
]
```

Listing B.8: Choose payment method use case specification (in RSL).

```
UseCase uc_8_DeleteProduct "DeleteProduct" : EntityDelete [
  primaryActor a_Admin
  dataEntity e_Product
  postcondition "The product is deleted from the system"
  includes uc_2_Login
  description "As an Anonymous User I want to delete a product"

  mainScenario uc_5_DeleteProduct_MainScenario (Main) "User deletes the product" [
  step s1 (System:Execute:Open) "Opens the manage products page in the browser"
  step s2 (Actor:CallSystem:Select) "Selects a product ID"
  step s3 (Actor:CallSystem:SelectAction) "Selects the action delete product" [
    scenario s3a_ProductIdDoesNotExist (Exception) "User does not have administrator role and
         can not delete a product" [
      step s3a1 (System:Execute:Check) "Validates the user can not delete a product"
      step s3a2 (System:ReturnResult:DisplayMessage) "The user can not delete a product"
          nextStep s3 ] ]
  step s4 (System:Execute:Check) "Validates the user role is administrator"
  step s5 (System:ReturnResult:DisplayMessage) "The product was successfully deleted"
  ]
]
```

Listing B.9: Delete product use case specification (in RSL).

```
UseCase uc_9_DeleteUser "DeleteUser" : EntityDelete [
  primaryActor a_Admin
  dataEntity e_User
  postcondition "The user is deleted from the system"
  includes uc_2_Login
  description "As an Administrator User I want to delete a user"

  mainScenario uc_6_DeleteUser_MainScenario (Main) "User deletes the user" [
  step s1 (System:Execute:Open) "Opens the manage users page in the browser"
  step s2 (Actor:CallSystem:Select) "Selects an User"
  step s3 (Actor:CallSystem:SelectAction) "Selects the action delete user" [
     scenario s3a_UserDoesNotExist (Exception) "User Deletes The Administrator User" [
       step s3a1 (System:Execute:Check) "Validates the user is an administrator and can not be
            deleted"
       step s3a2 (System:ReturnResult:DisplayMessage) "The user is an administrator and can not
            be deleted" nextStep s3 ] ]
  step s4 (System:Execute:Check) "Validates the user role is not an administrator"
  step s5 (System:ReturnResult:DisplayMessage) "The user was successfully deleted"
  ]
]
```

Listing B.10: Delete user use case specification (in RSL).

B.2 Test Case Specification Generation

```
Package WebApp
Import WebApp.myStore.*
System myStore "MyStore - RSL Specification" : Application []

/*******************************************************
   Data
*******************************************************/

Data d_User : e_User [    withValues
  | e_User.ID | e_User.firstName  | e_User.email  | e_User.password | e_User.statusActive |
      e_User.role +|
  | "1001"  | "John"  | "john@email.com"  | "#password1234" | "Anonymous" +|
  | "1002"  | "Frank" | "frank@email.com" | "#password1235" | "User"  +|
  | "1003"  | "Peter" | "peter@email.com" | "#password1236" | "Anonymous" +|
]

Data d_Product : e_Product [    withValues
  | e_Product.ID  | e_Product.SKU | e_Product.Name  | e_Product.VATCode | e_Product.VATValue  |
      e_Product.Value +|
  | "20001" | "9001"  | "Product 1" | ""  | ""  +|
  | ""  | "9002"  | "Product 2" | ""  | ""  +|
  | "20003" | "9003"  | "Product 3" | ""  | ""  +|
  | "20004" | "9004"  | ""  | ""  | ""  +|
]
```

Listing B.11: Data entities & actors and data specification (in RSL).

```
/*******************************************************
   TestSuite
*******************************************************/

TestUseCase ts_uc_1_Signup  "Signup TestSuite" [
  useCase uc_1_Signup
  variable v_User [ withData (d_User)]
  variable vURL [ "http://www.automationpractice.com" ]
  variable vMessageSystemNotification [ "The submitted email does not exists", "The passwords
      are different", "The user was successfully created in the system" ]

  testCase t_uc_1_Signup_0_happpypath "Anonymous User Signs Up With his/her Information" [
    step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
    step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the name"
    step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
    step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
    step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the confirm
        password"
    step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
    step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
        element ('#TBD') ) "Validates the passwords are correct"
    step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
        successfully created in the system"
  ]

  testCase t_uc_1_Signup_1 "Anonymous User Signs Up With his/her Information" [
    step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
    step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the name"
    step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
    step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
        into element ('#TBD') ) "Validates if the submitted email does not exists"
    step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The submitted
        email does not exists"
```

```
28      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
29      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
30      step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the confirm
            password"
31      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
32      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the passwords are different"
33      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The passwords
            are different"
34      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
35      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
36      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully created in the system"
37    ]
38
39    testCase t_uc_1_Signup_2 "Anonymous User Signs Up With his/her Information" [
40      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
41      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the name"
42      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
43      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates if the submitted email does not exists"
44      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The submitted
            email does not exists"
45      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
46      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
47      step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the confirm
            password"
48      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
49      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
50      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully created in the system"
51    ]
52
53    testCase t_uc_1_Signup_3 "Anonymous User Signs Up With his/her Information" [
54      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
55      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the name"
56      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
57      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
58      step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the confirm
            password"
59      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
60      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the passwords are different"
61      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The passwords
            are different"
62      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action sign up"
63      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
64      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully created in the system"
65    ]
66  ]
```

Listing B.12: Sign-up test suite specification (in RSL).

```
1   TestUseCase ts_uc_2_Login  "Login TestSuite" [
2     useCase uc_2_Login
3     variable v_User [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The email is not registered", "The password is
          invalid", "Displays profile menu with user information" ]
6
7     testCase t_uc_2_Login_0_happpypath "Login With his/her Information" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
9       step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
10      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
11      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
12      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
13      step s6 (System:ReturnResult:ShowData element('User_Field') withData '#TBD') "Displays
            profile menu with user information"
```

```
14    ]
15
16    testCase t_uc_2_Login_1 "Login With his/her Information" [
17      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
18      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
19      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the email is not registered"
20      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The email is
            not registered"
21      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
22      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
23      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
24      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the password is invalid"
25      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The password
            is invalid"
26      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
27      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
28      step s6 (System:ReturnResult:ShowData element('User_Field') withData '#TBD') "Displays
            profile menu with user information"
29    ]
30
31    testCase t_uc_2_Login_2 "Login With his/her Information" [
32      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
33      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
34      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the email is not registered"
35      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The email is
            not registered"
36      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
37      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
38      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
39      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
40      step s6 (System:ReturnResult:ShowData element('User_Field') withData '#TBD') "Displays
            profile menu with user information"
41    ]
42
43    testCase t_uc_2_Login_3 "Login With his/her Information" [
44      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
45      step s2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the email"
46      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the password"
47      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
48      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the password is invalid"
49      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The password
            is invalid"
50      step s4 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action login"
51      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the passwords are correct"
52      step s6 (System:ReturnResult:ShowData element('User_Field') withData '#TBD') "Displays
            profile menu with user information"
53    ]
54  ]
```

Listing B.13: Log in test suite specification (in RSL).

```
1   TestUseCase ts_uc_3_CreateProduct  "CreateProduct TestSuite" [
2     useCase uc_3_CreateProduct
3     variable v_Product [ withData (d_Product)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The product ID already exists", "The product ID is
          empty", "The product name is empty", "The product was successfully created" ]
6
7     testCase t_uc_3_CreateProduct_0_happpypath "User creates a product" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
9       step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
10      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
```

```
11      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
12      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
13      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
14    ]
15
16    testCase t_uc_3_CreateProduct_1 "User creates a product" [
17      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
18      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
19      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
20      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID already exists"
21      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
22      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
23      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID is empty"
24      step s3b2 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the
            product ID"
25      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
26      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product name is empty"
27      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            name is empty"
28      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
29      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
30      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
31    ]
32
33    testCase t_uc_3_CreateProduct_2 "User creates a product" [
34      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
35      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
36      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
37      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID already exists"
38      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
39      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
40      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID is empty"
41      step s3b2 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the
            product ID"
42      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
43      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
44      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
45    ]
46
47    testCase t_uc_3_CreateProduct_3 "User creates a product" [
48      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
49      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
50      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
51      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID already exists"
52      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
53      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
```

```
54      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
55      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product name is empty"
56      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            name is empty"
57      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
58      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
59      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
60    ]
61
62    testCase t_uc_3_CreateProduct_4 "User creates a product" [
63      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
64      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
65      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
66      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID already exists"
67      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
68      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
69      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
70      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
71      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
72    ]
73
74    testCase t_uc_3_CreateProduct_5 "User creates a product" [
75      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
76      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
77      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
78      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID is empty"
79      step s3b2 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the
            product ID"
80      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
81      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product name is empty"
82      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            name is empty"
83      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
84      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
85      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
86    ]
87
88    testCase t_uc_3_CreateProduct_6 "User creates a product" [
89      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
90      step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
91      step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
92      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product ID is empty"
93      step s3b2 (System:Execute:SaveData from element('Product_Form')) "Auto-generates the
            product ID"
94      step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
95      step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
96      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
```

```
97    ]
98
99    testCase t_uc_3_CreateProduct_7 "User creates a product" [
100     step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
101     step s2 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action open
            product page"
102     step s3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            ID"
103     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
104     step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product name is empty"
105     step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            name is empty"
106     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the product
            name"
107     step s5 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action to
            create the product"
108     step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
            successfully created"
109   ]
110 ]
```

Listing B.14: Create product test suite specification (in RSL).

```
1   TestUseCase ts_uc_4_AddItemToShoppingCart  "Add Item to Shopping Cart TestSuite" [
2     useCase uc_4_AddItemToShoppingCart
3     variable v_Product [ withData (d_Product)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The product is out of stock", "The product size is not
          available", "The product is successfully added to the cart" ]
6
7     testCase t_uc_4_AddItemToShoppingCart_0_happpypath "User adds a product to the cart" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9       step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a T-shirt"
10      step s3 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
11      step s4 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
12      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
13      step s6 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            continue shopping"
14      step s7 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a Dress"
15      step s8 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
16      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
17      step s10 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses one unit"
18      step s11 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            checkout"
19      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product is
            successfully added to the cart"
20    ]
21
22    testCase t_uc_4_AddItemToShoppingCart_1 "User adds a product to the cart" [
23      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
24      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a T-shirt"
25      step s3 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
26      step s4 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
27      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
28      step s5a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product is out of stock"
29      step s5a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            is out of stock"
30      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
31      step s6 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            continue shopping"
32      step s7 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a Dress"
33      step s8 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
```

```
34      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
35      step s9a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product size is not available"
36      step s9a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            size is not available"
37      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
38      step s10 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses one unit"
39      step s11 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            checkout"
40      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product is
            successfully added to the cart"
41    ]
42
43    testCase t_uc_4_AddItemToShoppingCart_2 "User adds a product to the cart" [
44      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
45      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a T-shirt"
46      step s3 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
47      step s4 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
48      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
49      step s5a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product is out of stock"
50      step s5a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            is out of stock"
51      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
52      step s6 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            continue shopping"
53      step s7 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a Dress"
54      step s8 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
55      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
56      step s10 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses one unit"
57      step s11 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            checkout"
58      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product is
            successfully added to the cart"
59    ]
60
61    testCase t_uc_4_AddItemToShoppingCart_3 "User adds a product to the cart" [
62      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
63      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a T-shirt"
64      step s3 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
65      step s4 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
66      step s5 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses two units"
67      step s6 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            continue shopping"
68      step s7 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a Dress"
69      step s8 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            color"
70      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
71      step s9a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the product size is not available"
72      step s9a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            size is not available"
73      step s9 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses the product
            size"
74      step s10 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Chooses one unit"
75      step s11 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
            checkout"
76      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product is
            successfully added to the cart"
77    ]
78  ]
```

Listing B.15: Add item to shopping cart test suite specification (in RSL).

```
1  TestUseCase ts_uc_5_AddDeliveryAddress  "Add delivery address TestSuite" [
2    useCase uc_5_AddDeliveryAddress
3    variable v_User [ withData (d_User)]
4    variable vURL [ "http://www.automationpractice.com" ]
5    variable vMessageSystemNotification [ "The zip code is incorrect", "The delivery address was
         successfully added" ]
6
7    testCase t_uc_5_AddDeliveryAddress_0_happpypath "User adds the delivery address to the
         profile" [
8      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the delivery
           address option"
10     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action add a
           new address"
11     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the street"
12     step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the zip code"
13     step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action save
           address"
14     step s7 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirms
           delivery address"
15     step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The delivery
           address was successfully added"
16   ]
17
18   testCase t_uc_5_AddDeliveryAddress_1 "User adds the delivery address to the profile" [
19     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
20     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the delivery
           address option"
21     step s2a1 (System:CallSystem:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates that the billing address exists"
22     step s2a2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the billing
           address"
23     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action add a
           new address"
24     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the street"
25     step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the zip code"
26     step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action save
           address"
27     step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates that the zip code is incorrect"
28     step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The zip code
           is incorrect"
29     step s7 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirms
           delivery address"
30     step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The delivery
           address was successfully added"
31   ]
32
33   testCase t_uc_5_AddDeliveryAddress_2 "User adds the delivery address to the profile" [
34     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
35     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the delivery
           address option"
36     step s2a1 (System:CallSystem:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates that the billing address exists"
37     step s2a2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the billing
           address"
38     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action add a
           new address"
39     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the street"
40     step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the zip code"
41     step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action save
           address"
42     step s7 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirms
           delivery address"
43     step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The delivery
           address was successfully added"
44   ]
45
46   testCase t_uc_5_AddDeliveryAddress_3 "User adds the delivery address to the profile" [
47     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
48     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the delivery
           address option"
49     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action add a
           new address"
50     step s4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the street"
51     step s5 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the zip code"
```

```
52      step s6 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action save
            address"
53      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates that the zip code is incorrect"
54      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The zip code
            is incorrect"
55      step s7 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirms
            delivery address"
56      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The delivery
            address was successfully added"
57    ]
58  ]
```

Listing B.16: Add delivery address test suite specification (in RSL).

```
1   TestUseCase ts_uc_6_ChooseShippingOption  "Choose Shipping Option TestSuite" [
2     useCase uc_6_ChooseShippingOption
3     variable v_User [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The shipping option is not available for his/her
          country", "The agree with terms of service option must be selected", "The agree with
          terms of service option was successfully selected" ]
6
7     testCase t_uc_6_ChooseShippingOption_0_happpypath "User chooses the shipping option" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9       step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
            option"
10      step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action checkout"
11      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the selected option agree with terms of service"
12      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
            terms of service option was successfully selected"
13    ]
14
15    testCase t_uc_6_ChooseShippingOption_1 "User chooses the shipping option" [
16      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
17      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
            option"
18      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the delivery option is not available"
19      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The shipping
            option is not available for his/her country"
20      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
            option"
21      step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action checkout"
22      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the selected option agree with terms of service"
23      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the checkbox is not selected"
24      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree
            with terms of service option must be selected"
25      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the selected option agree with terms of service"
26      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
            terms of service option was successfully selected"
27    ]
28
29    testCase t_uc_6_ChooseShippingOption_2 "User chooses the shipping option" [
30      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
31      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
            option"
32      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
            into element ('#TBD') ) "Validates the delivery option is not available"
33      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The shipping
            option is not available for his/her country"
34      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
            option"
35      step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action checkout"
36      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the selected option agree with terms of service"
37      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
            terms of service option was successfully selected"
38    ]
```

```
39
40   testCase t_uc_6_ChooseShippingOption_3 "User chooses the shipping option" [
41     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
42     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the shipping
           option"
43     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action checkout"
44     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
           element ('#TBD') ) "Validates the selected option agree with terms of service"
45     step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates the checkbox is not selected"
46     step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree
           with terms of service option must be selected"
47     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
           element ('#TBD') ) "Validates the selected option agree with terms of service"
48     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
           terms of service option was successfully selected"
49   ]
50 ]
```

Listing B.17: Choose shipping option test suite specification (in RSL).

```
1  TestUseCase ts_uc_7_ChoosePaymentMethod  "Choose Payment Method TestSuite" [
2    useCase uc_7_ChoosePaymentMethod
3    variable v_User [ withData (d_User)]
4    variable vURL [ "http://www.automationpractice.com" ]
5    variable vMessageSystemNotification [ "The payment method was successfully added" ]
6
7    testCase t_uc_7_ChoosePaymentMethod_0_happpypath "User chooses the payment method" [
8      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the payment
           method"
10     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirm
           the payment method"
11     step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
           method was successfully added"
12   ]
13
14   testCase t_uc_7_ChoosePaymentMethod_1 "User chooses the payment method" [
15     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
16     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the payment
           method"
17     step s2a1 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the voucher
           code"
18     step s2a2 (System:Execute:SaveData from element('User_Form')) "Discount the voucher value
           from the total amount"
19     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirm
           the payment method"
20     step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
           method was successfully added"
21   ]
22
23   testCase t_uc_7_ChoosePaymentMethod_2 "User chooses the payment method" [
24     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
25     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects the payment
           method"
26     step s2b1 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action add a
           new card"
27     step s2b2 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in the card
           number"
28     step s2b3 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in expiration
           date"
29     step s2b4 (Actor:PrepareData:Fill from '#TBD' into element ('#TBD') ) "Fills in CVV"
30     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action confirm
           the payment method"
31     step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
           method was successfully added"
32   ]
33 ]
```

Listing B.18: Choose payment method test suite specification (in RSL).

```
1  TestUseCase ts_uc_8_DeleteProduct  "DeleteProduct TestSuite" [
2    useCase uc_8_DeleteProduct
3    variable v_Product [ withData (d_Product)]
4    variable vURL [ "http://www.automationpractice.com" ]
5    variable vMessageSystemNotification [ "The user can not delete a product", "The product was
         successfully deleted" ]
6
7    testCase t_uc_8_DeleteProduct_0_happpypath "User deletes the product" [
8      step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage products page in the
           browser"
9      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a product ID"
10     step s3 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
           delete product"
11     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
           element ('#TBD') ) "Validates the user role is administrator"
12     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
           successfully deleted"
13   ]
14
15   testCase t_uc_8_DeleteProduct_1 "User deletes the product" [
16     step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage products page in the
           browser"
17     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects a product ID"
18     step s3 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
           delete product"
19     step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates the user can not delete a product"
20     step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user can
           not delete a product"
21     step s3 (Actor:CallSystem:SelectAction element('Product_Submit')) "Selects the action
           delete product"
22     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
           element ('#TBD') ) "Validates the user role is administrator"
23     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product was
           successfully deleted"
24   ]
25
26 ]
```

Listing B.19: Delete product test suite specification (in RSL).

```
1  TestUseCase ts_uc_9_DeleteUser  "DeleteUser TestSuite" [
2    useCase uc_9_DeleteUser
3    variable v_User [ withData (d_User)]
4    variable vURL [ "http://www.automationpractice.com" ]
5    variable vMessageSystemNotification [ "The user is an administrator and can not be deleted",
         "The user was successfully deleted" ]
6
7    testCase t_uc_9_DeleteUser_0_happpypath "User deletes the user" [
8      step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage users page in the
           browser"
9      step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects an User"
10     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action delete
           user"
11     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
           element ('#TBD') ) "Validates the user role is not an administrator"
12     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
           successfully deleted"
13   ]
14
15   testCase t_uc_9_DeleteUser_1 "User deletes the user" [
16     step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage users page in the
           browser"
17     step s2 (Actor:CallSystem:Select from '#TBD' into element ('#TBD') ) "Selects an User"
18     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action delete
           user"
19     step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD'
           into element ('#TBD') ) "Validates the user is an administrator and can not be deleted"
20     step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user is
           an administrator and can not be deleted"
21     step s3 (Actor:CallSystem:SelectAction element('User_Submit')) "Selects the action delete
           user"
```

```
22      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from '#TBD' into
            element ('#TBD') ) "Validates the user role is not an administrator"
23      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully deleted"
24    ]
25
26  ]
```

Listing B.20: Delete user test suite specification (in RSL).

B.3 Refined Test Case Specification

```
1   /*******************************************************
2      TestSuite
3   *******************************************************/
4
5   TestUseCase ts_uc_1_Signup  "Signup TestSuite" [
6     useCase uc_1_Signup
7     variable v_User [ withData (d_User)]
8     variable vURL [ "http://www.automationpractice.com" ]
9     variable vMessageSystemNotification [ "The submitted email does not exists", "The passwords
          are different", "The user was successfully created in the system" ]
10
11    testCase t_uc_1_Signup_0_happpypath "Anonymous User Signs Up With his/her Information" [
12      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
13      step s2 (Actor:PrepareData:Fill from v_User.Name into element ('user_name') ) "Fills in the
            name"
14      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
15      step s4 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
16      step s5 (Actor:PrepareData:Fill from v_User.Password into element ('user_confirmpassword')
            ) "Fills in the confirm password"
17      step s6 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action sign up"
18      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
19      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The user was
            successfully created in the system"
20    ]
21
22    testCase t_uc_1_Signup_1 "Anonymous User Signs Up With his/her Information" [
23      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
24      step s2 (Actor:PrepareData:Fill from v_User.Name into element ('user_name') ) "Fills in the
            name"
25      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
26      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Email into element ('user_email') ) "Validates if the submitted email does not
            exists"
27      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The submitted
            email does not exists"
28      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
29      step s4 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
30      step s5 (Actor:PrepareData:Fill from v_User.ConfirmPassword into element
            ('user_confirmpassword') ) "Fills in the confirm password"
31      step s6 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action sign up"
32      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are
            different"
33      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The passwords
            are different"
34      step s6 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action sign up"
35      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
36      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The user was
            successfully created in the system"
37    ]
```

```
38
39    testCase t_uc_1_Signup_2 "Anonymous User Signs Up With his/her Information" [
40      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
41      step s2 (Actor:PrepareData:Fill from v_User.Name into element ('user_name') ) "Fills in the
            name"
42      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
43      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Email into element ('user_email') ) "Validates if the submitted email does not
            exists"
44      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The submitted
            email does not exists"
45      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
46      step s4 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
47      step s5 (Actor:PrepareData:Fill from v_User.Confirmpassword into element
            ('user_confirmpassword') ) "Fills in the confirm password"
48      step s6 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action sign up"
49      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
50      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully created in the system"
51    ]
52
53    testCase t_uc_1_Signup_3 "Anonymous User Signs Up With his/her Information" [
54      step s1 (System:Execute:Open browser(url vURL) ) "Opens the sign up page in the browser"
55      step s2 (Actor:PrepareData:Fill from v_User.Name into element ('user_name') ) "Fills in the
            name"
56      step s3 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
57      step s4 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
58      step s5 (Actor:PrepareData:Fill from v_User.Confirmpassword into element
            ('user_confirmpassword') ) "Fills in the confirm password"
59      step s6 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action sign up"
60      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are
            different"
61      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The passwords
            are different"
62      step s6 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action sign up"
63      step s7 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
64      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully created in the system"
65    ]
66  ]
```

Listing B.21: Sign-up test suite refinement (in RSL).

```
1   TestUseCase ts_uc_2_Login  "Login TestSuite" [
2     useCase uc_2_Login
3     variable v_User [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The email is not registered", "The password is
          invalid", "Displays profile menu with user information" ]
6
7     testCase t_uc_2_Login_0_happpypath "Login With his/her Information" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
9       step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
10      step s3 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
11      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
12      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
13      step s6 (System:ReturnResult:ShowData element('user') withData v_User) "Displays profile
            menu with user information"
14    ]
15
16    testCase t_uc_2_Login_1 "Login With his/her Information" [
17      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
```

```
18      step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
19      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Email into element ('user_email') ) "Validates the email is not registered"
20      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The email is
            not registered"
21      step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
22      step s3 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
23      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
24      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the password is invalid"
25      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The password
            is invalid"
26      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
27      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
28      step s6 (System:ReturnResult:ShowData element('user') withData v_User) "Displays profile
            menu with user information"
29    ]
30
31    testCase t_uc_2_Login_2 "Login With his/her Information" [
32      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
33      step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
34      step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Email into element ('user_email') ) "Validates the email is not registered"
35      step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The email is
            not registered"
36      step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
37      step s3 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
38      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
39      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
40      step s6 (System:ReturnResult:ShowData element('user') withData v_User) "Displays profile
            menu with user information"
41    ]
42
43    testCase t_uc_2_Login_3 "Login With his/her Information" [
44      step s1 (System:Execute:Open browser(url vURL) ) "Opens the login page in the browser"
45      step s2 (Actor:PrepareData:Fill from v_User.Email into element ('user_email') ) "Fills in
            the email"
46      step s3 (Actor:PrepareData:Fill from v_User.Password into element ('user_password') )
            "Fills in the password"
47      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
48      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the password is invalid"
49      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The password
            is invalid"
50      step s4 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action login"
51      step s5 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Password into element ('user_password') ) "Validates the passwords are correct"
52      step s6 (System:ReturnResult:ShowData element('user') withData v_User) "Displays profile
            menu with user information"
53    ]
54  ]
```

Listing B.22: Log in test suite refinement (in RSL).

```
1   TestUseCase ts_uc_3_CreateProduct  "CreateProduct TestSuite" [
2     useCase uc_3_CreateProduct
3     variable v_Product [ withData (d_Product)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The product ID already exists", "The product ID is
          empty", "The product name is empty", "The product was successfully created" ]
6
7     testCase t_uc_3_CreateProduct_0_happpypath "User creates a product" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
```

```
9     step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
          product page"
10    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
11    step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
          in the product name"
12    step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
          create the product"
13    step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
          successfully created"
14  ]
15
16  testCase t_uc_3_CreateProduct_1 "User creates a product" [
17    step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
          browser"
18    step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
          product page"
19    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
20    step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
          v_Product.ID into element ('product_ID') ) "Validates the product ID already exists"
21    step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
          ID already exists"
22    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
23    step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
          v_Product.ID into element ('product_ID') ) "Validates the product ID is empty"
24    step s3b2 (System:Execute:SaveData from element('product_form')) "Auto-generates the
          product ID"
25    step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
          in the product name"
26    step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[2]) from
          v_Product.Name into element ('product_Name') ) "Validates the product name is empty"
27    step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product
          name is empty"
28    step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
          in the product name"
29    step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
          create the product"
30    step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
          successfully created"
31  ]
32
33  testCase t_uc_3_CreateProduct_2 "User creates a product" [
34    step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
          browser"
35    step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
          product page"
36    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
37    step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
          v_Product.ID into element ('product_ID') ) "Validates the product ID already exists"
38    step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
          ID already exists"
39    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
40    step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
          v_Product.ID into element ('product_ID') ) "Validates the product ID is empty"
41    step s3b2 (System:Execute:SaveData from element('product_form')) "Auto-generates the
          product ID"
42    step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
          in the product name"
43    step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
          create the product"
44    step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
          successfully created"
45  ]
46
47  testCase t_uc_3_CreateProduct_3 "User creates a product" [
48    step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
          browser"
49    step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
          product page"
50    step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
          the product ID"
51    step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
          v_Product.ID into element ('product_ID') ) "Validates the product ID already exists"
```

```
52      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
53      step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
54      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
55      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[2]) from
            v_Product.Name into element ('product_Name') ) "Validates the product name is empty"
56      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product
            name is empty"
57      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
58      step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
            create the product"
59      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
            successfully created"
60    ]
61
62    testCase t_uc_3_CreateProduct_4 "User creates a product" [
63      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
64      step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
            product page"
65      step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
66      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_Product.ID into element ('product_ID') ) "Validates the product ID already exists"
67      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            ID already exists"
68      step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
69      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
70      step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
            create the product"
71      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
            successfully created"
72    ]
73
74    testCase t_uc_3_CreateProduct_5 "User creates a product" [
75      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
76      step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
            product page"
77      step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
78      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
            v_Product.ID into element ('product_ID') ) "Validates the product ID is empty"
79      step s3b2 (System:Execute:SaveData from element('product_form')) "Auto-generates the
            product ID"
80      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
81      step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[2]) from
            v_Product.Name into element ('product_Name') ) "Validates the product name is empty"
82      step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product
            name is empty"
83      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
84      step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
            create the product"
85      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
            successfully created"
86    ]
87
88    testCase t_uc_3_CreateProduct_6 "User creates a product" [
89      step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
90      step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
            product page"
91      step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
92      step s3b1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
            v_Product.ID into element ('product_ID') ) "Validates the product ID is empty"
93      step s3b2 (System:Execute:SaveData from element('product_form')) "Auto-generates the
            product ID"
94      step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
```

```
95      step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
            create the product"
96      step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
            successfully created"
97    ]
98
99    testCase t_uc_3_CreateProduct_7 "User creates a product" [
100     step s1 (System:Execute:Open browser(url vURL) ) "Opens the products list page in the
            browser"
101     step s2 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action open
            product page"
102     step s3 (Actor:PrepareData:Fill from v_Product.ID into element ('product_ID') ) "Fills in
            the product ID"
103     step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
104     step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[2]) from
            v_Product.Name into element ('product_Name') ) "Validates the product name is empty"
105     step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product
            name is empty"
106     step s4 (Actor:PrepareData:Fill from v_Product.Name into element ('product_Name') ) "Fills
            in the product name"
107     step s5 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action to
            create the product"
108     step s6 (System:ReturnResult:DisplayMessage vMessageSystemNotification[3]) "The product was
            successfully created"
109   ]
110 ]
```

Listing B.23: Create product test suite refinement (in RSL).

```
1   TestUseCase ts_uc_4_AddItemToShoppingCart  "Add Item to Shopping Cart TestSuite" [
2     useCase uc_4_AddItemToShoppingCart
3     variable v_Product [ withData (d_Product)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The product is out of stock", "The product size is not
          available", "The product is successfully added to the cart" ]
6
7     testCase t_uc_4_AddItemToShoppingCart_0_happpypath "User adds a product to the cart" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9       step s2 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a T-shirt"
10      step s3 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
11      step s4 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
12      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
13      step s6 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            continue shopping"
14      step s7 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a Dress"
15      step s8 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
16      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
17      step s10 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses one unit"
18      step s11 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            checkout"
19      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product is
            successfully added to the cart"
20    ]
21
22    testCase t_uc_4_AddItemToShoppingCart_1 "User adds a product to the cart" [
23      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
24      step s2 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a T-shirt"
25      step s3 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
26      step s4 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
27      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
```

```
28      step s5a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_Product.Quantity into element ('product_Quantity') ) "Validates the product is out
            of stock"
29      step s5a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            is out of stock"
30      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
31      step s6 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            continue shopping"
32      step s7 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a Dress"
33      step s8 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
34      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
35      step s9a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from v_Product
            into element ('product_Size') ) "Validates the product size is not available"
36      step s9a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The product
            size is not available"
37      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
38      step s10 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses one unit"
39      step s11 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            checkout"
40      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product is
            successfully added to the cart"
41    ]
42
43    testCase t_uc_4_AddItemToShoppingCart_2 "User adds a product to the cart" [
44      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
45      step s2 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a T-shirt"
46      step s3 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
47      step s4 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
48      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
49      step s5a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_Product.Quantity into element ('product_Quantity') ) "Validates the product is out
            of stock"
50      step s5a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The product
            is out of stock"
51      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
52      step s6 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            continue shopping"
53      step s7 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a Dress"
54      step s8 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
55      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
56      step s10 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses one unit"
57      step s11 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            checkout"
58      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product is
            successfully added to the cart"
59    ]
60
61    testCase t_uc_4_AddItemToShoppingCart_3 "User adds a product to the cart" [
62      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
63      step s2 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a T-shirt"
64      step s3 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
65      step s4 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
66      step s5 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses two units"
67      step s6 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            continue shopping"
68      step s7 (Actor:CallSystem:Select from v_Product.Name into element ('product_Name') )
            "Selects a Dress"
```

```
69      step s8 (Actor:CallSystem:Select from v_Product.Color into element ('product_Color') )
            "Chooses the product color"
70      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
71      step s9a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[1]) from
            v_Product.Size into element ('product_Size') ) "Validates the product size is not
            available"
72      step s9a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The product
            size is not available"
73      step s9 (Actor:CallSystem:Select from v_Product.Size into element ('product_Size') )
            "Chooses the product size"
74      step s10 (Actor:CallSystem:Select from v_Product.Quantity into element ('product_Quantity')
            ) "Chooses one unit"
75      step s11 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
            checkout"
76      step s12 (System:ReturnResult:DisplayMessage vMessageSystemNotification[2]) "The product is
            successfully added to the cart"
77    ]
78  ]
```

Listing B.24: Add item to shopping cart test suite refinement (in RSL).

```
1   TestUseCase ts_uc_5_AddDeliveryAddress  "Add delivery address TestSuite" [
2     useCase uc_5_AddDeliveryAddress
3     variable v_User. [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The zip code is incorrect", "The delivery address was
          successfully added" ]
6
7     testCase t_uc_5_AddDeliveryAddress_0_happpypath "User adds the delivery address to the
          profile" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9       step s2 (Actor:CallSystem:Select from v_User.Address into element (user_Address) ) "Selects
            the delivery address option"
10      step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action add a
            new address"
11      step s4 (Actor:PrepareData:Fill from v_User.Street into element ('user_Street') ) "Fills in
            the street"
12      step s5 (Actor:PrepareData:Fill from v_User.Zipcode into element ('user_Zipcode') ) "Fills
            in the zip code"
13      step s6 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action save
            address"
14      step s7 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
            confirms delivery address"
15      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The delivery
            address was successfully added"
16    ]
17
18    testCase t_uc_5_AddDeliveryAddress_1 "User adds the delivery address to the profile" [
19      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
20      step s2 (Actor:CallSystem:Select from v_User.Address into element ('user_address') )
            "Selects the delivery address option"
21      step s2a1 (System:CallSystem:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Address into element ('user_Address') ) "Validates that the billing address
            exists"
22      step s2a2 (Actor:CallSystem:Select from v_User.Address into element ('user_Address') )
            "Selects the billing address"
23      step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action add a
            new address"
24      step s4 (Actor:PrepareData:Fill from v_User.Street into element ('user_Street') ) "Fills in
            the street"
25      step s5 (Actor:PrepareData:Fill from v_User.Zipcode into element ('user_Zipcode') ) "Fills
            in the zip code"
26      step s6 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action save
            address"
27      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Address into element ('user_Address') ) "Validates that the zip code is
            incorrect"
28      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The zip code
            is incorrect"
29      step s7 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
            confirms delivery address"
```

```
30      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The delivery
            address was successfully added"
31    ]
32
33    testCase t_uc_5_AddDeliveryAddress_2 "User adds the delivery address to the profile" [
34      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
35      step s2 (Actor:CallSystem:Select from v_User.Address into element ('user_Address') )
            "Selects the delivery address option"
36      step s2a1 (System:CallSystem:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Address into element ('user_Address') ) "Validates that the billing address
            exists"
37      step s2a2 (Actor:CallSystem:Select from v_User.Address into element ('user_Address') )
            "Selects the billing address"
38      step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action add a
            new address"
39      step s4 (Actor:PrepareData:Fill from v_User.Street into element ('user_Street') ) "Fills in
            the street"
40      step s5 (Actor:PrepareData:Fill from v_User.Zipcode into element ('user_Zipcode') ) "Fills
            in the zip code"
41      step s6 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action save
            address"
42      step s7 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
            confirms delivery address"
43      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The delivery
            address was successfully added"
44    ]
45
46    testCase t_uc_5_AddDeliveryAddress_3 "User adds the delivery address to the profile" [
47      step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
48      step s2 (Actor:CallSystem:Select from v_User.Address into element ('user_Address') )
            "Selects the delivery address option"
49      step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action add a
            new address"
50      step s4 (Actor:PrepareData:Fill from v_User.Street into element ('user_Street') ) "Fills in
            the street"
51      step s5 (Actor:PrepareData:Fill from v_User.Zipcode into element ('user_Zipcode') ) "Fills
            in the zip code"
52      step s6 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action save
            address"
53      step s6a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            v_User.Zipcode into element ('user_Zipcode') ) "Validates that the zip code is
            incorrect"
54      step s6a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The zip code
            is incorrect"
55      step s7 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
            confirms delivery address"
56      step s8 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The delivery
            address was successfully added"
57    ]
58  ]
```

Listing B.25: Add delivery address test suite refinement (in RSL).

```
1   TestUseCase ts_uc_6_ChooseShippingOption  "Choose Shipping Option TestSuite" [
2     useCase uc_6_ChooseShippingOption
3     variable v_User [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The shipping option is not available for his/her
          country", "The agree with terms of service option must be selected", "The agree with
          terms of service option was successfully selected" ]
6
7     testCase t_uc_6_ChooseShippingOption_0_happpypath "User chooses the shipping option" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9       step s2 (Actor:CallSystem:Select from v_User.Option into element ('user_Option') ) "Selects
            the shipping option"
10      step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
            checkout"
11      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
            'v_User.Agree' into element ('user_Agree') ) "Validates the selected option agree with
            terms of service"
12      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
            terms of service option was successfully selected"
13    ]
```

```
14
15   testCase t_uc_6_ChooseShippingOption_1 "User chooses the shipping option" [
16     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
17     step s2 (Actor:CallSystem:Select from 'v_User.Option' into element ('user_option') )
           "Selects the shipping option"
18     step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
           v_User.Option into element ('user_Option') ) "Validates the delivery option is not
           available"
19     step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The shipping
           option is not available for his/her country"
20     step s2 (Actor:CallSystem:Select from v_User.Option into element ('user_Option') ) "Selects
           the shipping option"
21     step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
           checkout"
22     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.Agree
           into element ('user_Agree') ) "Validates the selected option agree with terms of
           service"
23     step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
           v_User.Agree into element ('user_Agree') ) "Validates the checkbox is not selected"
24     step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree
           with terms of service option must be selected"
25     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.Agree
           into element ('user_Agree') ) "Validates the selected option agree with terms of
           service"
26     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
           terms of service option was successfully selected"
27   ]
28
29   testCase t_uc_6_ChooseShippingOption_2 "User chooses the shipping option" [
30     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
31     step s2 (Actor:CallSystem:Select from v_User.Option into element ('user_Option') ) "Selects
           the shipping option"
32     step s2a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
           v_User.Option into element ('user_Option') ) "Validates the delivery option is not
           available"
33     step s2a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The shipping
           option is not available for his/her country"
34     step s2 (Actor:CallSystem:Select from v_User.Option into element ('user_Option') ) "Selects
           the shipping option"
35     step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
           checkout"
36     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.Agree
           into element ('user_Agree') ) "Validates the selected option agree with terms of
           service"
37     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
           terms of service option was successfully selected"
38   ]
39
40   testCase t_uc_6_ChooseShippingOption_3 "User chooses the shipping option" [
41     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
42     step s2 (Actor:CallSystem:Select from v_User.Option into element ('user_Option') ) "Selects
           the shipping option"
43     step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action
           checkout"
44     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.Agree
           into element ('user_Agree') ) "Validates the selected option agree with terms of
           service"
45     step s4a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from
           v_User.Agree into element ('user_Agree') ) "Validates the checkbox is not selected"
46     step s4a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree
           with terms of service option must be selected"
47     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.Agree
           into element ('user_Agree') ) "Validates the selected option agree with terms of
           service"
48     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The agree with
           terms of service option was successfully selected"
49   ]
50 ]
```

Listing B.26: Choose shipping option test suite refinement (in RSL).

```
1  TestUseCase ts_uc_7_ChoosePaymentMethod  "Choose Payment Method TestSuite" [
2    useCase uc_7_ChoosePaymentMethod
```

```
3   variable v_User [ withData (d_User)]
4   variable vURL [ "http://www.automationpractice.com" ]
5   variable vMessageSystemNotification [ "The payment method was successfully added" ]
6
7   testCase t_uc_7_ChoosePaymentMethod_0_happpypath "User chooses the payment method" [
8     step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
9     step s2 (Actor:CallSystem:Select from v_User.Method into element ('user_Method') ) "Selects
          the payment method"
10    step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action confirm
          the payment method"
11    step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
          method was successfully added"
12  ]
13
14  testCase t_uc_7_ChoosePaymentMethod_1 "User chooses the payment method" [
15    step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
16    step s2 (Actor:CallSystem:Select from v_User.Method into element ('user_Method') ) "Selects
          the payment method"
17    step s2a1 (Actor:PrepareData:Fill from v_User.VoucherCode into element ('user_VoucherCode')
          ) "Fills in the voucher code"
18    step s2a2 (System:Execute:SaveData from element('user_form') ) "Discount the voucher value
          from the total amount"
19    step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action confirm
          the payment method"
20    step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
          method was successfully added"
21  ]
22
23  testCase t_uc_7_ChoosePaymentMethod_2 "User chooses the payment method" [
24    step s1 (System:Execute:Open browser(url vURL) ) "Opens the MyStore site in the browser"
25    step s2 (Actor:CallSystem:Select from v_User.Method into element ('user_Method') ) "Selects
          the payment method"
26    step s2b1 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action add a
          new card"
27    step s2b2 (Actor:PrepareData:Fill from v_User.CardNumber into element ('user_CardNumber') )
          "Fills in the card number"
28    step s2b3 (Actor:PrepareData:Fill from v_User.CardDate into element ('user_CardDate') )
          "Fills in expiration date"
29    step s2b4 (Actor:PrepareData:Fill from v_User.CVV into element ('user_CVV') ) "Fills in CVV"
30    step s3 (Actor:CallSystem:SelectAction element('user_submit') ) "Selects the action confirm
          the payment method"
31    step s4 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The payment
          method was successfully added"
32  ]
33 ]
```

Listing B.27: Choose payment method test suite refinement (in RSL).

```
1  TestUseCase ts_uc_8_DeleteProduct  "DeleteProduct TestSuite" [
2    useCase uc_8_DeleteProduct
3    variable v_Product [ withData (d_Product)]
4    variable v_User [ withData (d_User)]
5    variable vURL [ "http://www.automationpractice.com" ]
6    variable vMessageSystemNotification [ "The user can not delete a product", "The product was
         successfully deleted" ]
7
8    testCase t_uc_8_DeleteProduct_0_happpypath "User deletes the product" [
9      step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage products page in the
           browser"
10     step s2 (Actor:CallSystem:Select from v_Product.ID into element ('product_ID') ) "Selects a
           product ID"
11     step s3 (Actor:CallSystem:SelectAction element('product_submit') ) "Selects the action
           delete product"
12     step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.ID
           into element ('user_ID') ) "Validates the user role is administrator"
13     step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The product was
           successfully deleted"
14   ]
15
16   testCase t_uc_8_DeleteProduct_1 "User deletes the product" [
17     step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage products page in the
           browser"
```

```
18      step s2 (Actor:CallSystem:Select from v_Product.ID into element ('product_ID') ) "Selects a
            product ID"
19      step s3 (Actor:CallSystem:SelectAction element('product_submit')) "Selects the action
            delete product"
20      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User
            into element ('user_ID') ) "Validates the user can not delete a product"
21      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user can
            not delete a product"
22      step s3 (Actor:CallSystem:SelectAction element('product_submit')) "Selects the action
            delete product"
23      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_Product
            into element ('user_ID') ) "Validates the user role is administrator"
24      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[1]) "The product was
            successfully deleted"
25    ]
26
27  ]
```

Listing B.28: Delete product test suite refinement (in RSL).

```
1   TestUseCase ts_uc_9_DeleteUser  "DeleteUser TestSuite" [
2     useCase uc_9_DeleteUser
3     variable v_User [ withData (d_User)]
4     variable vURL [ "http://www.automationpractice.com" ]
5     variable vMessageSystemNotification [ "The user is an administrator and can not be deleted",
          "The user was successfully deleted" ]
6
7     testCase t_uc_9_DeleteUser_0_happpypath "User deletes the user" [
8       step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage users page in the
            browser"
9       step s2 (Actor:CallSystem:Select from v_User.ID into element ('user_ID') ) "Selects an User"
10      step s3 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action delete
            user"
11      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.ID
            into element ('user_ID') ) "Validates the user role is not an administrator"
12      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully deleted"
13    ]
14
15    testCase t_uc_9_DeleteUser_1 "User deletes the user" [
16      step s1 (System:Execute:Open browser(url vURL) ) "Opens the manage users page in the
            browser"
17      step s2 (Actor:CallSystem:Select from v_User.ID into element ('user_ID') ) "Selects an User"
18      step s3 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action delete
            user"
19      step s3a1 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.ID
            into element ('user_ID') ) "Validates the user is an administrator and can not be
            deleted"
20      step s3a2 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user is
            an administrator and can not be deleted"
21      step s3 (Actor:CallSystem:SelectAction element('user_submit')) "Selects the action delete
            user"
22      step s4 (System:Execute:Check textOnScreen(vMessageSystemNotification[0]) from v_User.ID
            into element ('user_ID') ) "Validates the user role is not an administrator"
23      step s5 (System:ReturnResult:DisplayMessage vMessageSystemNotification[0]) "The user was
            successfully deleted"
24    ]
25
26  ]
```

Listing B.29: Delete user test suite refinement (in RSL).

Appendix C

Installation Instructions

C.1 Software

- Java 8 or higher
- Latest Eclipse IDE version
- Xtext plugin
- Gradle plugin

C.2 Instructions

C.2.1 Install JAVA

Ensure that Java 8 (or above) is installed.

https://java.com/en/download/help/download_options.html

C.2.1.1 On Windows

Ensure JAVA_HOME environment variable is set and points to JDK installation

Check environment variable value e.g.

echo "JAVA_HOME"

"C:/Program Files/Java/jdk1.7.0_51"

Adding to PATH: Add the unpacked distribution's bin directory to PATH environment variable by opening up the system properties (WinKey + Pause), selecting the "Advanced" tab, and the "Environment Variables" button, then adding or selecting the PATH variable in the user variables with the value C:/Program Files/apache-maven-3.6.2/bin. The same dialog can be used to set JAVA_HOME to the location of JDK, e.g. C:/Program Files/Java/jdk1.7.0_51

Open a new command prompt (Winkey + R then type cmd) and run mvn -v to verify the installation.

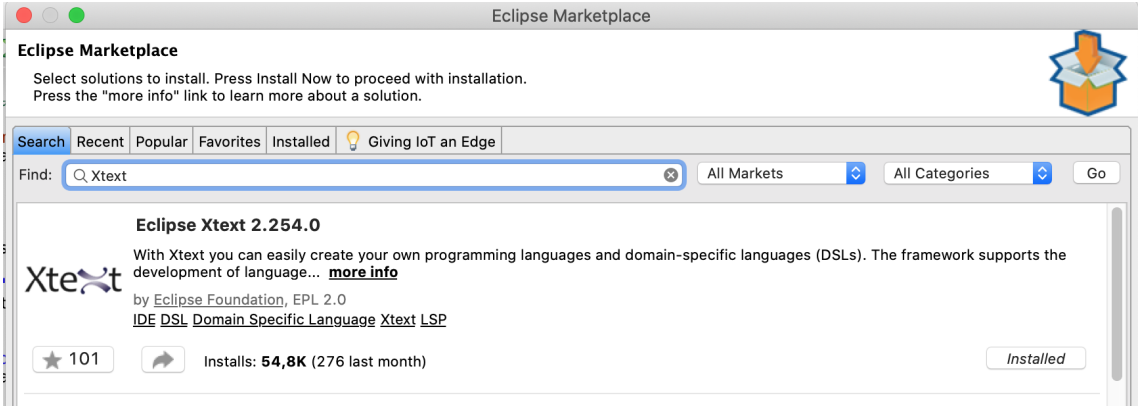

Figure C.1: Install Xtext plugin in Eclipse

C.2.1.2 On Mac

Although there are other ways, we suggest installing the package manager homebrew and then install maven through homebrew. To be able to install homebrew we need to have Command Line Tools installed and a Mac in OS X 10.10 or later.

C.2.2 Install Eclipse

Download and install the latest version of Eclipse IDE.

https://www.eclipse.org/downloads/

C.2.3 Install Xtext

- After opening Eclipse, go to the Help menu and then find Eclipse MarketPlace.
- Find Xtext (Figure C.1)
- Click in the button install

C.2.4 Install Gradle

- Go to the Help menu in Eclipse and then find Eclipse MarketPlace.
- Find Gradle (Figure C.2)
- Click in the button install

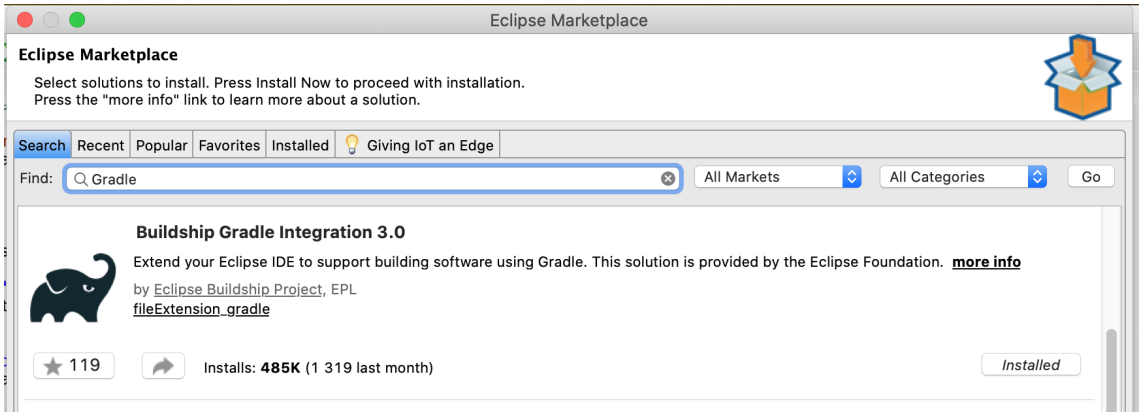

Figure C.2: Install Gradle plugin in Eclipse

Appendix D

User Guide

D.1 Software

- Java 8 or higher
- Latest Eclipse IDE version
- Xtext plugin
- Gradle plugin

D.2 Instructions

D.2.1 Create a XText project

Create a new Xtext project or **open** the RSL project. It creates several Eclipse projects, but the focus of our implementation is on two directories:

- **org.itlingo.rsl.Rsl/src/org/itlingo/rsl/Rsl.xtext** – This file contains RSL grammar.
- **org.itlingo.rsl.Rsl/src/org/itlingo/rsl/generator/RslGenerator.xtend** – The file responsible for generating the test suite and test cases specification in RSL.
- **org.itlingo.rsl.Rsl/src/org/itlingo/rsl/generator/Graph.java** – This file contains the algorithm for generating the test cases.

D.2.2 How to generate XText Artifacts

Every time the grammar file (Rsl.xtext) is updated, it is necessary to generate the Xtext Artifacts. To perform this operation, navigate to the file Rsl.xtext, right-click on it, and navigate to **Run As | Generate Xtext Artifacts**. Once the console displays the message/line "**Done**", the code generation phase is finished (Figure D.1).

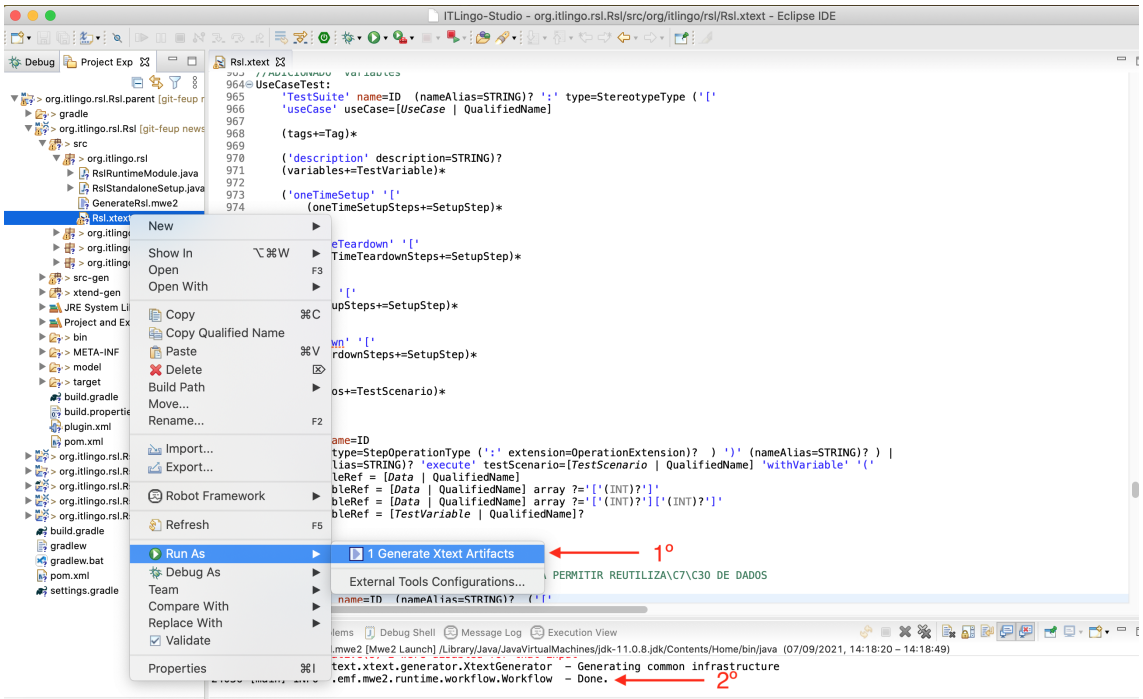

Figure D.1: Generate Xtext Artifacts.

D.2.3 Code Generator in Xtend

The file responsible for generating the test suite and test cases is RslGenerator.xtend (Figure D.2).

D.2.4 Full Edges Coverage Algorithm

The Graph.java file contains the algorithm for generating the test cases.

D.2.5 Configure a New Eclipse Instance

To define a new Eclipse instance to start writing the RSL Specification, right-click on the org.itlingo.rsl.RSL project and navigate to **Run As | Run Configurations** (Figure D.3). In the dialog, select **Eclipse Application**, and then the New button to create a new launch configuration. Select it and click on **Run** as shown in image above (Figure D.4).

After the first time, we just need to click in the **Launch Runtime Eclipse** (Figure D.5).

D.2.6 Create a New RSL Workbench

A new Eclipse instance will be run and a new **RSL workbench** will appear. In this instance, the RSL Specification will be available (Figure D.6).

- Create a new **General project (File | New | Project... | General | Project)**.

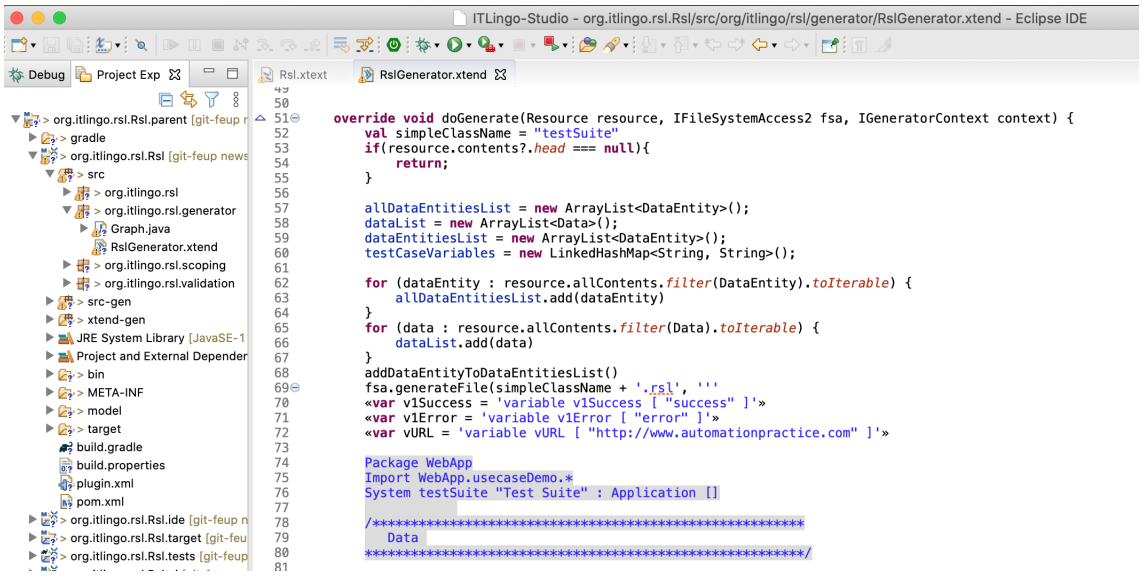

Figure D.2: RSLUseCase2TestSuiteGenerator generator.

- Inside this project, create a new file (File > New > Other > General > File). The name of the file is not relevant, but the file extension must be **.rsl**. As soon as the file is created, it will also be opened in a text editor, and you will be asked to convert the project to a Xtext project. It is important to accept the message to make the RSL editor work correctly in Eclipse. Automatically, it will be created in a new folder called src-gen.
- Start writing the use case specification in the RSL file.

D.2.7 Generate The Test Suite And Test Cases

Feel free to start specifying the requirements. Whenever the specification file is saved, the test suites and test cases will be generated in the **testSuite.rsl** file in the **src-gen** folder (Figure D.7).

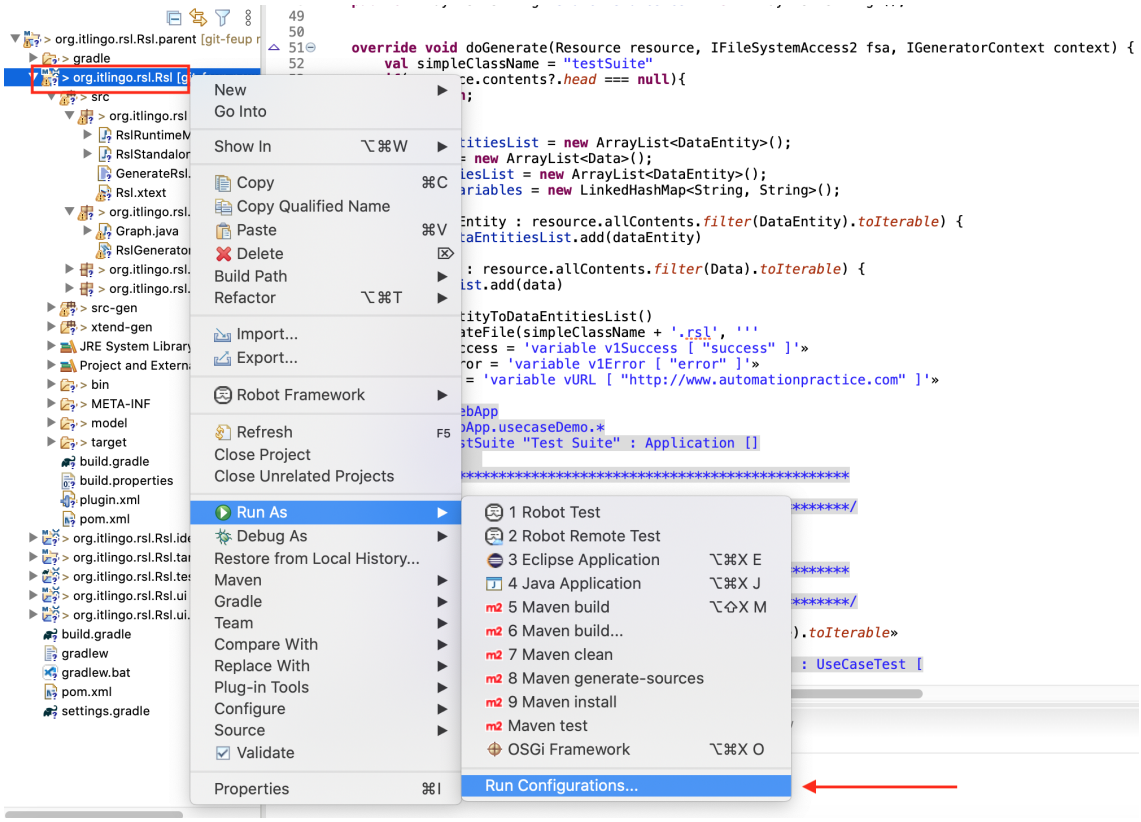

Figure D.3: Run Configuration.

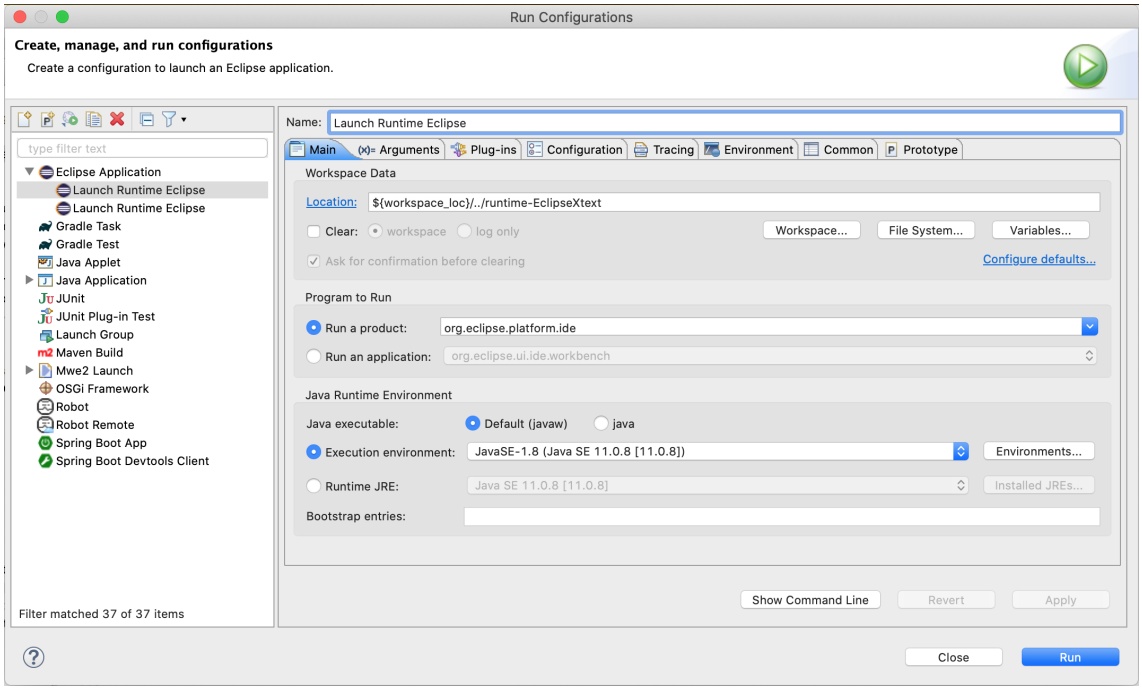

Figure D.4: Launch Configuration.

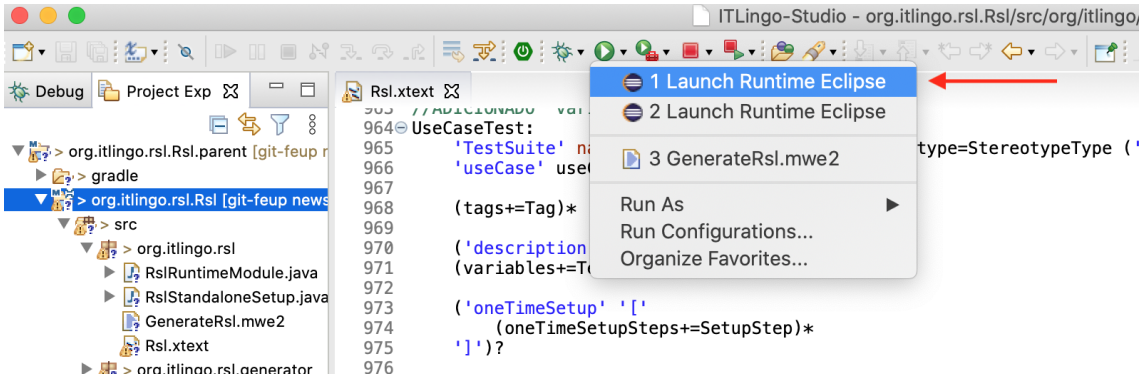

Figure D.5: Launch Runtime Eclipse

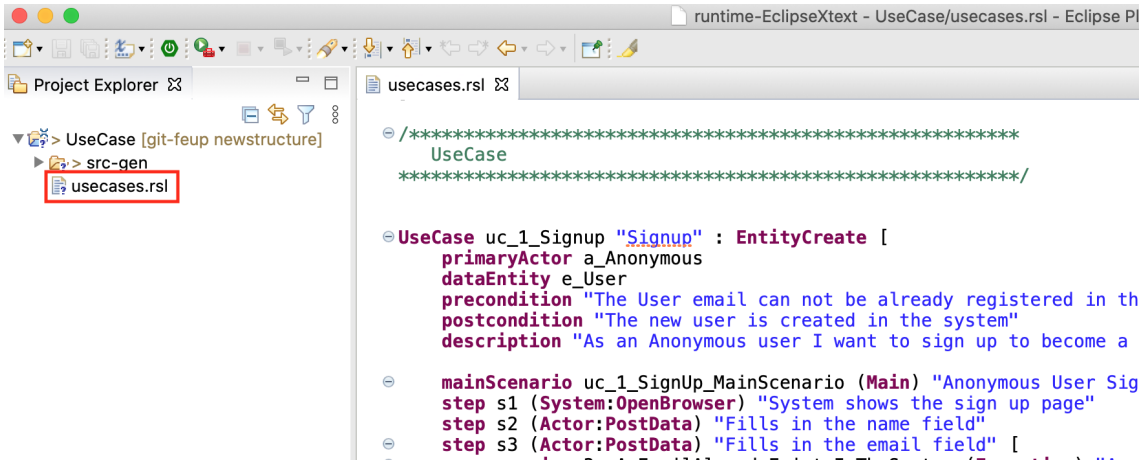

Figure D.6: Specification of a use case scenario (in RSL).

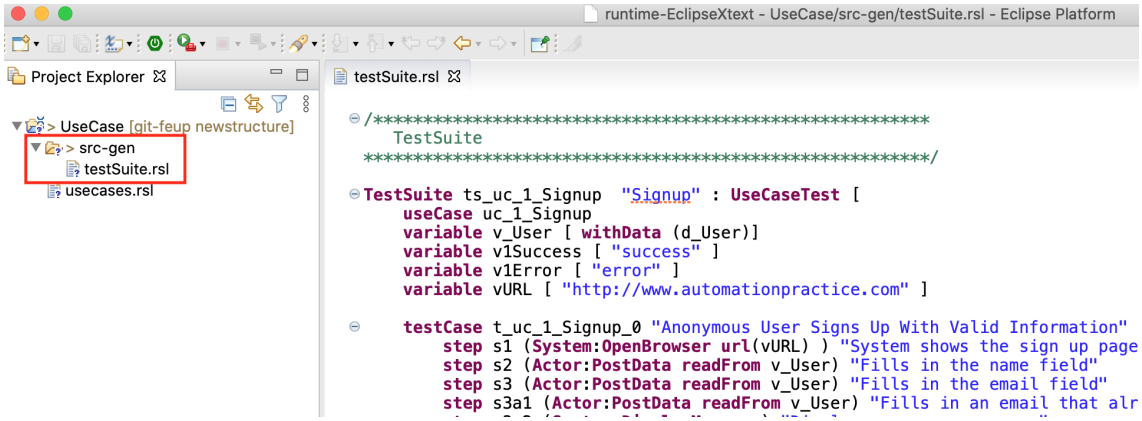

Figure D.7: Specification of a test suite (in RSL).

References

[1] Mauricio Alferez, Fabrizio Pastore, Mehrdad Sabetzadeh, Lionel C. Briand, and Jean Richard Riccardi. Bridging the Gap between Requirements Modeling and Behavior-Driven Development. *Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems, MODELS 2019*, pages 239–249, 2019.

[2] Thamer A. Alrawashed, Ammar Almomani, Ahmad Althunibat, and Abdelfatah Tamimi. An automated approach to generate test cases from use case description model. *CMES - Computer Modeling in Engineering and Sciences*, 119(3):409–425, 2019.

[3] Germán Arias, Diego Vilches, Claudia Banchoff, Ivana Harari, Viviana Harari, and Pablo Iuliano. The 7 key factors to get successful results in the IT Development projects. *Procedia Technology*, 5:199–207, 2012.

[4] Muneera Bano. Addressing the challenges of requirements ambiguity: A review of empirical literature. *5th International Workshop on Empirical Requirements Engineering, EmpiRE 2015 - Proceedings*, pages 21–24, 2016.

[5] Ana Barbosa, Ana C.R. Paiva, and José Creissac Campos. Test case generation from mutated task models. *Proceedings of the 2011 SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2011*, pages 175–184, 2011.

[6] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[7] A. Bertolino, E. Marchetti, and H. Muccini. Introducing a Reasonably Complete and Coherent Approach for Model-based Testing. *Electronic Notes in Theoretical Computer Science*, 116(SPEC.ISS.):85–97, 2005.

[8] Souhaib Besrour, Lukman Bin AB Rahim, and P.D.D. Dominic. A quantitative study to identify critical requirement engineering challenges in the context of small and medium software enterprise. *2016 3rd International Conference on Computer and Information Sciences, ICCOINS 2016 - Proceedings*, pages 606–610, 2016.

[9] Robert V. Binder. Design for Testability in Object-Oriented Systems. *Communications of the ACM*, 37(9):87–101, 1994.

[10] Raquel Blanco, José G. Enríquez, Francisco J. Domínguez-Mayo, M. J. Escalona, and Javier Tuya. Early Integration Testing for Entity Reconciliation in the Context of Heterogeneous Data Sources. *IEEE Transactions on Reliability*, 67(2):538–556, 2018.

[11] Ravishankar Boddu, Lan Guo, Supratik Mukhopadhyay, and Bojan Cukic. RETNA: From requirements to testing in a natural way. *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 262–271, 2004.

[12] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Trans. Softw. Eng.*, 14(10):1462–1477, October 1988.

[13] Lizhe Chen and Qiang Li. Automated test case generation from use case: A model based approach. *Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010*, 1:372–377, 2010.

[14] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 2000.

[15] Alberto Rodrigues Da Silva. Linguistic patterns and linguistic styles for requirements specification (I): An application case with the rigorous rsl/business-level language. *ACM International Conference Proceeding Series*, Part F132091, 2017.

[16] Alberto Rodrigues Da Silva. Rigorous specification of use cases with the RSL language. *Proceedings of the 28th International Conference on Information Systems Development: Information Systems Beyond 2020, ISD 2019*, 2019.

[17] Alberto Rodrigues Da Silva, Ana C. R. Paiva, and Valter E. R. Da Silva. A test specification language for information systems based on data entities, use cases and state machines. In Slimane Hammoudi, Luís Ferreira Pires, and Bran Selic, editors, *Model-Driven Engineering and Software Development*, pages 455–474, Cham, 2019. Springer International Publishing.

[18] Alberto Rodrigues da Silva, Ana C. R. Paiva, and Valter Emanuel R. da Silva. Towards a test specification language for information systems: Focus on data entity and state machine tests. *MODELSWARD 2018 - Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development*, 2018-January(iii):213–224, 2018.

[19] Alberto Rodrigues da Silva and Dušan Savić. Linguistic patterns and linguistic styles for requirements specification: Focus on data entities. *Applied Sciences (Switzerland)*, 11(9), 2021.

[20] Fabiano Dalpiaz, Alessio Ferrari, Xavier Franch, and Cristina Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, 35(5):115–119, 2018.

[21] David De Almeida Ferreira and Alberto Rodrigues Da Silva. RSL-IL: An interlingua for formally documenting requirements. In *2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE)*, pages 40–49. IEEE, 2013.

[22] Christian Denger and Maricel Medina Mora. Test case derived from Requirement Specifications. (033), 2003.

[23] Rajendra R. Dube and Shantanu K. Dixit. Process-oriented complete requirement engineering cycle for generic projects. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, pages 194–197, 2010.

[24] Elfriede Dustin, Thom Garrett, and Bernie Gauf. *Implementing automated software testing: How to save time and lower costs while raising quality*. Pearson Education, 2009.

[25] Anurag Dwarakanath and Shubhashis Sengupta. Litmus: Generation of test cases from functional requirements in natural language. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7337 LNCS:58–69, 2012.

[26] J.G. Enríquez, Francisco José Domínguez Mayo, M.J. Escalona, J.A. Garcia-Garcia, Vivian Lee, and Masatomo Goto. *Entity Identity Reconciliation based Big Data Federation-A MDE approach*. 01 2015.

[27] Jannik Fischbach, Andreas Vogelsang, Dominik Spies, Andreas Wehrle, Maximilian Junker, and Dietmar Freudenstein. SPECMATE: Automated Creation of Test Cases from Acceptance Criteria. *Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation, ICST 2020*, pages 321–331, 2020.

[28] Jorge Esparteiro Garcia and Ana C. R. Paiva. A Requirements-to-Implementation Mapping Tool for Requirements Traceability. *Journal of Software*, 11(2):193–200, 2016.

[29] Jorge Esparteiro Garcia and Ana C. R. Paiva. An automated approach for requirements specification maintenance. 03 2016.

[30] Jorge Esparteiro Garcia and Ana C. R. Paiva. REQAnalytics: A recommender system for requirements maintenance. *International Journal of Software Engineering and its Applications*, 10(1):129–140, 2016.

[31] Jorge Esparteiro Garcia and Ana C. R. Paiva. Manage software requirements specification using web analytics data. *Advances in Intelligent Systems and Computing*, 746:257–266, 2018.

[32] Jorge Esparteiro Garcia, Ana C. R. Paiva, and Anca Maria Bizoi. Test case generation from web usage information. *Procedia Computer Science*, 181:913–920, 2021.

[33] J. J. Gutiérrez, M. J. Escalona, and M. Mejías. A Model-Driven approach for functional test case generation. *Journal of Systems and Software*, 109:214–228, 2015.

[34] Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, and Jesús Torres. An approach to generate test cases from use cases. *ICWE'06: The Sixth International Conference on Web Engineering*, pages 113–114, 2006.

[35] Zahra Abdulkarim Hamza and Mustafa Hammad. Analyzing UML use cases to generate test sequences. *International Journal of Computing and Digital Systems*, 10(1):125–134, 2021.

[36] Alexander Hartmann and Martin Weigt. Phase transitions in combinatorial optimization problems: Basics, algorithms and statistical mechanics. pages 333–348, 03 2006.

[37] Elizabeth Hull, Ken Jackson, and Jeremy Dick, editors. *Requirements Engineering*. Springer, London, 2011.

[38] IEEE. Ieee standard glossary of software engineering terminology, 1990.

[39] International Organization for Standardization. ISO/IEC 12207:1995/Amd 2:2002 Information technology – Software life cycle processes, institution = International Organization for Standardization. Technical report, 2002.

[40] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, 1992.

[41] Pankaj Jalote. *An Integrated Approach to Software Engineering*. Springer Publishing Company, Incorporated, 3rd edition, 2010.

[42] Erik Kamsties. Understanding ambiguity in requirements engineering. In *Engineering and Managing Software Requirements*, pages 245–266. Springer, 2005.

[43] Tobias Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014.

[44] Mich Luisa, Franch Mariangela, and Novi Inverardi Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.

[45] Robyn R. Lutz. Analyzing software requirements errors in safety-critical, embedded systems. *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 126–133, 1993.

[46] Daniel Maciel, Ana Paiva, and Alberto Silva. From requirements to automated acceptance tests of interactive apps: An integrated model-based testing approach. pages 265–272, 01 2019.

[47] Dino Mandrioli, Sandro Morasca, and Angelo Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications. *ACM Transactions on Computer Systems (TOCS)*, 13(4):365–398, 1995.

[48] Marjan Mernik. Formal and practical aspects of domain-specific languages: Recent developments. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pages 1–651, 2012.

[49] Tiago Monteiro and Ana C. R. Paiva. Pattern based GUI testing modeling environment. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013*, pages 140–143, 2013.

[50] Rodrigo M.L.M. Moreira and Ana C. R. Paiva. PBGT tool: An integrated modeling and testing environment for pattern-based GUI testing. *ASE 2014 - Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 863–866, 2014.

[51] Rodrigo M.L.M. Moreira and Ana C. R. Paiva. Towards a pattern language for model-based GUI testing. *ACM International Conference Proceeding Series*, 09-13-July, 2014.

[52] Rodrigo M.L.M. Moreira, Ana Cristina Paiva, Miguel Nabuco, and Atif Memon. Pattern-based GUI testing: Bridging the gap between design and quality assurance. *Software Testing Verification and Reliability*, 27(3), 2017.

[53] Miguel Nabuco and Ana C. R. Paiva. Model-based test case generation for web applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8584 LNCS(PART 6):248–262, 2014.

[54] Colin J. Neill and Phillip A. Laplante. Requirements Engineering: The State of the Practice. *IEEE Software*, 20(6):40–45, 2003.

[55] Ayan Nigam, Neeraj Arya, Bhawna Nigam, and Deepika Jain. Tool for Automatic Discovery of Ambiguity in Requirements. *International Journal of Computer Science Issues (IJCSI)*, 9(5):350–356, 2012.

[56] Dan North. *Behavior Modification: The evolution of behavior-driven development*. Better Software, 2006.

[57] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, August 2011.

[58] Ana Paiva, Daniel Maciel, and Alberto Silva. *From Requirements to Automated Acceptance Tests with the RSL Language*, pages 39–57. 02 2020.

[59] Ana C. R. Paiva, André Restivo, and Sérgio Almeida. Test case generation based on mutations over user execution traces. *Software Quality Journal*, 28(3):1173–1186, 2020.

[60] Klaus Pohl and Chris Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB Compliant*. Rocky Nook, 1st edition, 2011.

[61] V. J. Rayward-Smith, Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. *The Journal of the Operational Research Society*, 42(9):pp. 540–549, 1991.

[62] Alberto Rodrigues Da Silva. Linguistic Patterns, Styles, and Guidelines for Writing Requirements Specifications: Focus on Use Cases and Scenarios. *IEEE Access (to be published)*, 2021.

[63] Gregg Rothermel, Roland H. Untcn, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

[64] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley, Boston, MA, 2 edition, 2004.

[65] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[66] Valdivino Alexandre De Santiago Júnior and Nandamudi Lankalapalli Vijaykumar. Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, 20(1):77–143, 2012.

[67] Unnati S. Shah and Devesh C. Jinwala. Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, 40(5):1–7, 2015.

[68] José L. Silva, José Creissac Campos, and Ana C. R. Paiva. Model-based User Interface Testing With Spec Explorer and ConcurTaskTrees. *Electronic Notes in Theoretical Computer Science*, 208(C):77–93, 2008.

[69] Pedro Silva, Ana C. R. Paiva, Andre Restivo, and Jorge Esparteiro Garcia. Automatic test case generation from usage information. *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, pages 268–271, 2018.

[70] Mathias Soeken, Robert Wille, and Rolf Drechsler. Assisted behavior driven development using natural language processing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7304 LNCS:269–287, 2012.

[71] Stéphane S. Somé. Use cases based requirements validation with scenarios. *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 465–466, 2005.

[72] Stéphane S. Some and Xu Cheng. An approach for supporting system-level test scenarios generation from textual use cases. *Proceedings of the ACM Symposium on Applied Computing*, pages 724–729, 2008.

[73] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[74] Y.N. Srikant and Priti Shankar. The compiler design handbook: Optimizations and machine code generation. *The Compiler Design Handbook: Optimizations and Machine Code Generation*, 01 2007.

[75] David Alan Stokes. Requirements analysis. *Software Engineer's Reference Book*, pages 16/1–16/21, 1991.

[76] CMMI Product Team. Cmmi for development, version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2010.

[77] Ashfa Umber and Imran Sarwar Bajwa. Minimizing ambiguity in natural language software requirements specification. *2011 Sixth International Conference on Digital Information Management*, pages 102–107, 2011.

[78] Jeffrey M. Voas and Keith W. Miller. Software Testability: The New Verification. *IEEE Software*, 12(3):17–28, 1995.

[79] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel Briand. Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[80] Man Zhang, Tao Yue, Shaukat Ali, Huihui Zhang, and Ji Wu. A systematic approach to automatically derive test cases from use cases specified in restricted natural languages. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8769:142–157, 2014.