Wiki support for automated definition of software test cases

Leandro Antonelli, Mariangeles Hozikian, Guy Camilleri, Alejandro Fernandez, Julian Grigera, Diego Torres, Pascale Zaraté

Wiki Support for Automated Definition of Software Test Cases

Leandro Antonelli, Mariángeles Hozikian, Guy Camilleri, Alejandro Fernandez, Julian Grigera, Diego Torres, Pascale Zarate

Abstract

The design of tests is a very important step in the software development process since the set of tests should match the users' expectations (captured through the requirements) with the finished product (the application) in order to show that the requirements are implemented. Being considered as a cumbersome activity, efforts have been made to automatize and alleviate the burden of test generation, but it is still a largely neglected step. We propose taking advantage of existing requirement artifacts, like Scenarios that describe the dynamic of the domain in a very early stage of software development, to obtain tests from them. In particular, the approach proposed complements the Scenarios that are textually described with a glossary, the Language Extended Lexicon. Thus, a set of rules to derive tests from Scenarios is also proposed. The tests are described using the Task/Method model. The main contributions of this work consist of an extension of a previously presented set of rules, and their implementation in a tool based on a media wiki platform that, through semantic support, makes it possible to record Scenarios and the Language Extended Lexicon to obtain the tests in an automatic way.

Introduction

Developing software still remains a very complex process involving several actors and consisting in different steps. We can notice that the testing stage is one of the most challenging ones, and it is frequently avoided because it demands a significant effort, and its importance is not always acknowledged. As a consequence, the delivered software may fail to meet the stakeholders' expectations. The V-model (Forsberg et al., 2005) suggests relating every step in software construction (that is: requirements, architecture, and design) with a step in software testing (that is: usability, integration and module tests). Acceptance tests are the final barrier to check whether the software application meets the user needs.

In particular, this proposal is focused in an early stage of requirements definition, because from them the ultimate test will arise. The objective is deriving tests in an automatic way from requirements artifacts. A set of rules to perform the derivation and tool support to implement them are provided. It is important to mention that although the approach proposed derives the tests in an automatic way, they should be reviewed by an expert to make eventual adjustments. This revision is necessary because the requirements are usually too abstract with a coarse grain of details, while test should be too specific and detailed. Thus, the tests obtained from the proposed approach should be reviewed and corrected. Also, the revision of the tests could in turn suggest the revision and improvement of the requirements that produced the tests.

This is another benefit of the proposed approach, to provide the design of tests to check the application, but also to trigger the revision of the requirements.

The approach combines three modeling techniques:

- Scenarios to describe behavior of the application domain. That is, the Scenarios that describe in some way the requirements of the software.
- the Language Extended Lexicon (LEL): a particular glossary coming from the requirement engineering field that complements the description of the Scenarios. In fact, the LEL describes the vocabulary used to describe the Scenarios.
- the Task/Method modeling approach, coming from the Artificial Intelligence field, particularly from the knowledge-based systems. It is used to describe the tests

The contributions of this paper are multiple. First, we propose a set of rules that uses the Scenarios and the LEL as input to obtain tests as output. Some preliminary versions of the rules were presented in previous work (self-reference removed). Then, we propose a tool able to support the capture and description of Scenarios and LEL, which is based on media Wiki, so it provides semantic support. This tool also implements the set of rules for translating the requirements described through Scenarios and LEL in tests described through the Task/Method model. Thus, the tool is able to generate the tests automatically.

Since Scenarios are described with narrative text, and the LEL is a good complement to describe and organize the used vocabulary, the benefit of using the LEL is double. On one hand it helps the requirements engineer to use a precise vocabulary reducing the ambiguity, but it also structures the description in order to simplify the application of the rules. The proposed approach uses the Scenarios as input to suggest the acceptance tests described through the Task/Method model. The tests generated cover all the alternatives of the different paths that the flow of the execution could follow inside the Scenario. The tool automatizes the application of the rules, but it also provides semantic support. This paper proposes an extension of a previous work in which the LEL was not used. Thus, according to several case studies and the received feedback from a previous publication the approach was enriched by using the LEL and some semantic support.

Thus, the originality of the paper arises from the combination of:

- early requirements specification (Scenarios and LEL)
- a knowledge paradigm from the artificial intelligence (Task/Method model)
- tool support that also provides semantic support. This semantic capability is very important considering that requirements are described textually.

This work is applied to the RUC-APS project. RUC-APS is a H2020 RISE-2015 project, aiming at Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems. In this context, the used examples are based on agricultural production.

The rest of the paper is organized as follows. First, related work is introduced. Then the background describes Scenarios, Language Extended Lexicon, and the Task/method paradigm. In the third section, the tool as an automatized support and a proof of concept of the proposed approach is described. In the fourth section, the rules to obtain tests from Scenarios are presented. Finally, some conclusions and future work are discussed.

Related work

Our proposal agrees with Bjarnason and Borg (Bjarnason and Borg, 2017) since we based our approach in obtaining test from requirements in order to perform the right tests to assure that the application satisfies user needs. We also agree with the importance of providing an automated tool to support the process as it is stated by Svensson and Regnell (Svensson and Regnell, 2015).

Hsieh et al. (Hsieh et al., 2013) propose an approach to derive and execute test cases from Use Cases, in particular. They perform an analysis of the steps of the Use Cases (also supported in the vocabulary) in order to determine all the alternatives to design the different tests to cover all the possibilities. The difference with our approach is that we rely on an earlier product, the Scenarios, and although they are not as specific as the Use Cases, we can reduce the effort as we deal with the definition earlier.

Chen and Li (Chen and Li, 2010) propose an approach to design tests using Use Cases without knowing the internal structure of the Use Cases. The tests are designed considering the relationships of the Use Cases. Although our approach analyses the internal structure, both approaches are similar because of the level of detail of the tests. Nevertheless, our approach is based on a description agreed between while Chen and Li based the test in the relationship inferred by their approach.

Entin et al. (Entin et al. 2009) propose an approach to derive tests from Scenarios. They focus their work in obtaining test independent from the platform. Thus, they analyze the Scenarios and derive general Scenarios. Although they use specific Scenarios as input, they generalize them and obtain the test from the general one, as our approach.

There are many others Inter-scenario dependency approaches (Husain et al., 2015) that provide a high-level organization of the artifact to cover different dependencies between them. Boucher et al. (Boucher and Mussbacher, 2015) transform workflow models (Use Case Maps) into Acceptance Test Cases that can be automated with the JUnit framework. Huaikou et al. (Huaikou and Ling, 2000) analyze Z specifications, in particular, the prior and posterior state of every operation to generate test cases. Philip et al. (Philip et al., 2017) analyze a model with safety requirements to generate fault trees representing functional hazards. Then, test cases for validation of mitigation of hazards are generated automatically from the model. Nomura et al. (Nomura et al. 2014) model the business context in a matrix representing the dependency between business process, from which they design test scenarios from the perspective of Personas to cover the different situations. Sarmiento et al. (Sarmiento and Leite, 2016) propose a similar approach using scenarios. The work of Nogueira et al. (Nogueira et al., 2012) propose generating test cases from specifically formatted use cases templates that capture key requirements' features as control flow, input and output. Using a similar approach, but using annotated UML Activities Diagrams, Vieira et al. also are able to automatically generate tests with the aim of maximizing coverage (Vieira et al., 2006).

Budha et al. (Budha et al., 2011) propose an approach based on Use Case diagrams that describe the behavior of the system. Their approach generates test cases to detect use case dependency faults using multiway trees. They transform the use case diagram into a tree, and they traverse the tree to generate the test cases so as to detect any existing intra and inter use case dependency faults among the various use cases being invoked by the

different actors interacting with the software system. Our approach is similar because we test the dependency of every step in the Scenario related to the previous. We test whether the previous steps are satisfied or not. Khamaiseh and Xu (Khamaiseh and Xu, 2017) proposed a technique based on Use Cases, in which they determine misuse case to test vulnerabilities. Our approach is similar because every step we test could fail, and for us, this is considered a misuse.

Intra-scenario approaches (Husain et al., 2015) focus on the detail of some product (a piece of requirement or a design artifact) and analyze the steps or elements of each artifact to design tests. Pandit et al. (Pandit et al., 2016) automatically design UATs based on acceptance criteria written in the form of Given-When-Then template. These criteria are divided into steps, and dependencies amongst steps are arranged in a dependency graph. Lei et al. (Lei and Wang., 2016) propose a framework to analyze testing constraints in requirements as the basis of test scenarios. Lipka et al. (Lipka et al., 2015) derive test scenarios from use cases stated in natural language, enriched with annotations to connect the specification with the source code of the application. Although our approach does not consider annotations, we consider a glossary (the LEL) and this glossary could be used in a future extension of our approach to link requirements with test and source code.

Background

This section gives a brief introduction to Scenarios and the Language extended Lexicon, which are the input to the proposed approach to derive Tests. It also describes the Task Method / Model, i.e. the technique used to describe the tests.

Scenarios and Language Extended Lexicon

Scenarios describe interactions between users and a future system (Potts, 1995). They are also used to understand the context of the application because they promote communication when there is a great variety of experts (Carroll, 2000). Leite (Leite et al., 1997) defines a Scenario with the following attributes: (i) a title that identifies the Scenario; (ii) a goal or aim to be reached through the execution of the episodes; (iii) a context that sets the starting point to reach the goal; (iv) the resources, relevant physical objects or information that must be available, (v) the actors, agents that perform the actions, and (vi) the set of episodes.

The following Scenario describes the activity of sorting out a set of products according to their quality before sending them to different customers. It is important to mention that each Scenario describes a task (summarized in the goal) that is divided into smaller tasks (described in the episodes). Every one of these smaller tasks (i.e. every episode) can be performed successfully or not, these are the kind of Scenarios that our approach works with. Furthermore, it is also important to remark that Scenarios describe some dynamic aspects of the domain but not specific requirements of a software application. Thus, this particular example describes tasks performed by a farmer, and can be automatized with information and communication technology.

The Language Extended Lexicon (LEL) is a glossary used to capture and describe the domain´s language (Leite and Franco, 1993). The LEL is composed of terms (also called symbols) that are classified into four types: Subject, Object, Verb, and State. Subjects represent active elements that perform actions. Objects are passive elements on which subjects perform actions. A verb is used to represent the actions. Finally, States represent situations in which subjects and objects can be located.

Scenario: Sorting out the product
Goal: Determining the destination of a product according to its quality level
Context:
The farmer gets orders from different clients.
The clients have different requirements about quality levels of the products.
The products have been recently harvested and they are dirty.
Resources: Product, orders, quality levels.
Actors: Farmer
Episodes:
The farmer washes the product
The farmer brushes the product
The farmer ranks the quality level
The farmer chooses the destination
The farmer packs the product

Listing 1. Scenario about Sorting out product

A symbol is described by two attributes: (i) the notion and (ii) the behavioral responses. Notion describes the symbol denotation and explains its literal meaning. Behavioral responses describe its connotation, that is, the effects and consequences of the relationship between the defined symbol and others symbols defined in the LEL (Sarmiento et al., 2014). These two attributes are very important to describe the symbols, because they make a complete description, while the notion describes the symbol from an internal point of view, the behavioral responses make a description from the exterior of the symbol. Nevertheless, for the description of this approach, we will only use the categorization of the symbols. That is why we will not provide examples of fully described symbols. Moreover, for the proposed approach, we only deal with subject, object, and verbs, but we do not use states. Listing 2 enumerates the symbols used in the Scenario and their category.

Subject: Farmer
Object: Product
Object: Orders
Object: Destination
Object: Quality level
Verb: Wash
Verb: Brush
Verb: Rank
Verb: Choose
Verb: Pack

Listing 2. Terms used in the Scenario about sorting out the products

There is a relationship between the attributes of the Scenario and the LEL. In general, symbols of the subject category of the LEL are used as actors in Scenarios, and objects in the LEL are resources in Scenarios. Then, verbs in the LEL are used to describe the name of the Scenario. For these names in particular, it is common to use expressions that contain verbs followed by an object. Then, the actions described in each episode also have some similar structure. The only difference is that each episode begins with a subject that performs the action: <subject of the LEL> + <verb of the LEL> + <object of the LEL>. Thus, the mapping between LEL glossary and Scenario should be the following:

Scenario: <Verb of the LEL> + <Object of the LEL>
Goal: ...
Context: ...
Resources: <Object of the LEL>
Actors: <Subject of the LEL>
Episodes:
<Subject of the LEL> + <Verb of the LEL> + <Object of the LEL>
<Subject of the LEL> + <Verb of the LEL> + <Object of the LEL>

Listing 3. Mapping between Scenario and LEL

The definition of Scenarios and the glossary LEL should be done iteratively, and both elements (LEL and Scenarios) should be described at the same time. That is, the experts can describe some terms in the LEL and then use these terms to describe Scenarios. And the experts can also describe some Scenario and while describing it they realize that some terms used in the Scenario are very important and should be described in the glossary, thus the terms are also defined in the LEL.

Tests description using the Task/Method Model paradigm

The task/method paradigm is a knowledge modeling paradigm that sees the reasoning as a task. This paradigm mainly comes from the Artificial Intelligence research domain, more precisely from knowledge acquisition and modeling, expert systems, and knowledge-based systems fields (Trichet and Tchounikine, 1999). The essential advantage of this paradigm is to have a declarative form to express knowledge which can be easily processed by tools such as execution engines (for performing tasks) or planners (for defining new ways to achieve tasks) (Camilleri et al., 2003).

A task/method model is composed of two submodels: the domain model and the reasoning model. The domain model describes the objects of the world being used (directly or indirectly) by the reasoning model. This model can be viewed as an application ontology, and is often described with the UML language and implemented with an object-oriented language. The reasoning model describes how a task can be performed. It uses two modeling primitives:

1. Task: a task is a transition between two world state families (an action) and is defined as follow:
    - *Name*: Task name
    - *Par*: Typified list of parameters handled by the task
    - *Objective*: Goal state of the task

- *Methods*: Method list achieving the task

2. Method: a method describes one way of performing a task. A method is characterized by the following fields:
   - *Heading*: Task achieved by the method
   - *Prec*: Preconditions which must be satisfied to be able to apply the method
   - *Effects*: Effects generated by the successful application of the method
   - *Control*: achievement order of the subtasks
   - *Subtask*: subtask set

In this work we will focus only on fields/concepts used in this work, but a complete presentation of this modeling paradigm can be found in (Camilleri et al., 2003). The task's field *Name* specifies the name of the task. The field *Par* contains the list of parameters, that is all objects handled by the task. A terminal task is a directly executable task. Its execution does not require decomposition. Terminal tasks correspond to the last level of decomposition. Roughly, the reasoning part of a Task/Method model describes the achievement of tasks thanks to methods which decompose tasks into subtasks. This task decomposition ends by terminal tasks for which no method is provided. These terminal tasks are thus directly executable.

For example, the Task / Method Model for the same Scenario should be the one described in Listing 4.

Method: M1
Task : Sorting_Out (product)
Control:
    wash (farmer, product);
    brush (farmer, product);
    rank (farmer, quality_levels);
    choose (farmer, destination);
    pack (farmer, products);
Method: M11
Task: bush (farmer, product)
precondition:
    not Product_correctly_washed
Control:
    message("not correctly washed, stop");
    stop;
Method: M12
Task: brush (farmer, product)
Precondition:
    not Product_correctly_brushed
Control:
    message("not correctly brushed, stop");
    stop;
and so on…

Listing 4. Task / Method model for the Scenario of sorting out products

As it was stated in the introduction, our approach deals with Scenarios that consider situation true/false, where an action can be either performed correctly or not. Thus, after every episode, the outcome of the episode (represented by a subtask) should be analyzed. If the outcome is successful the execution of the Scenario continues, but if the outcome fails, the Scenario ends. Thus, the situation considered for the Task / Model method of listing 4 is described in listing 5.

- Product correctly washed /not correctly washed
    - If Product correctly washed then brush / if not correctly washed then stop
- Product correctly / not correctly brushed
    - If Product correctly brushed then rank the quality level / if not correctly brushed then stop
- Product with a rank of the quality level known / unknown
    - If Product has a known rank of the quality level then choose the destination / If has not a known rank of the quality level then stop
- Product with a destination known / unknown
    - If Product has a destination known then pack the product / If Product has not a destination known then stop
- Product packed / not packed
    - If Product was packed then finished / If Product was not packed then stop

Listing 5. Tests flow for the Scenario about sort out products

Prototype

We developed an application to support the proposed approach of deriving tests from Scenarios. The application allows the collaborative description of Scenarios and the LEL, and it also implements the rules to obtain tests described in Task/Method model. In this section we describe the technology and the architecture of the tool, and we also describe the use of the tool, which exemplifies the whole approach.

Technology and architecture

A Media Wiki (MediaWiki, 2018) platform is used as a repository of the Scenarios and the LEL, as well as a tool to derive the Tests. Many extensions have been added to Media Wiki. An ad-hoc collaborative catalog and editor was added to manage Scenarios and LEL. This tool is enriched with a semantic Media Wiki extension that allows the semantic support and the creation of forms to make the CRUD operations (create, retrieve, update and deleted) in a more user-friendly way. Finally, the semantic extension also provides a query language for semantic searches in order to implement the derivation of tests.

Wikis are tools that allow users with no knowledge in programming to easily edit information on the web. This feature fosters the participation and collaboration of non-technical users. In traditional wikis there are three different elements to build each page: (i) the code that can be edited by every user and it is stored in the server; (ii) a template that defines the elements and their position; and (iii) the HTML code provided in real time by the server with the page requested by the user, it is a combination of (i) and (ii).

There are many software systems to implement wikis. Media Wiki is an open source implementation written in PHP that uses the MySql database engine. Wikipedia and other projects of Wikimedia use Media Wiki. Since it relies on the wikitext format, users with no knowledge of HTML or CSS can easily edit the pages. Another advantage of Media Wiki is the management of the links between pages. Although the destination of a link does not exist, the link can also be written and Media Wiki shows it anyway, when the user clicks the link, Media Wiki allow to create the page. Another important feature is the configuration management. Media Wiki stores in a database all the different versions of each page, so, all previous versions can be accessed is necessary.

Media Wiki has been extended with the Semantic Media Wiki framework that allows retrieving information from the Wiki in a similar way that data is retrieved from a database. In a Semantic Media Wiki, each piece of data is represented by a triplet consisting of a subject, a predicate, and object. Every element of this triplet corresponds to a category of the LEL glossary, and it is also related to the template used in the description of the Scenarios. Some attributes as Title, actors, and resources uses some the categories, but the episodes are described with the triplet. Thus, the Semantic Media Wiki framework provides the infrastructure to organize and store the information and a query language to implement the derivation rules in a straightforward way. Another technology added to Media Wiki is Page Forms, also called Semantic Forms. This extension allows to add, edit and retrieve data using customized forms. It is based on templates that are used in Pages called Special Pages.

Another extension used is *Input types*, which allows defining the basic elements and their attributes. Nevertheless, the episodes of the Scenarios are complex structures (they are a set of triplets), thus another extension was needed to deal with them: the Semantic Internal Object extension.

Figure 1 shows a conceptual model that describes the types of Scenarios and LEL. Every LEL symbol has two attributes: notion and impact (behavioral responses). There are 4 categories of LEL Symbols but only 3 were used in this approach: Subject, Object, and Verbs. The Scenarios is the core of the approach. It has a title, composed by a verb and an object. It also has an objective and a context, both are regular text. Then, actors and resources are important, because actors are mapped to subjects (subject category of the LEL and subjects in the semantic triplets), and resources are mapped to objects (object in the LEL and the triplets). The episodes are a set of expressions, in fact, they are full sentences with a subject and a predicate that is represented by a verb and an object. This full sentence is composed by symbols of the three categories, but they also are the semantic triplets.

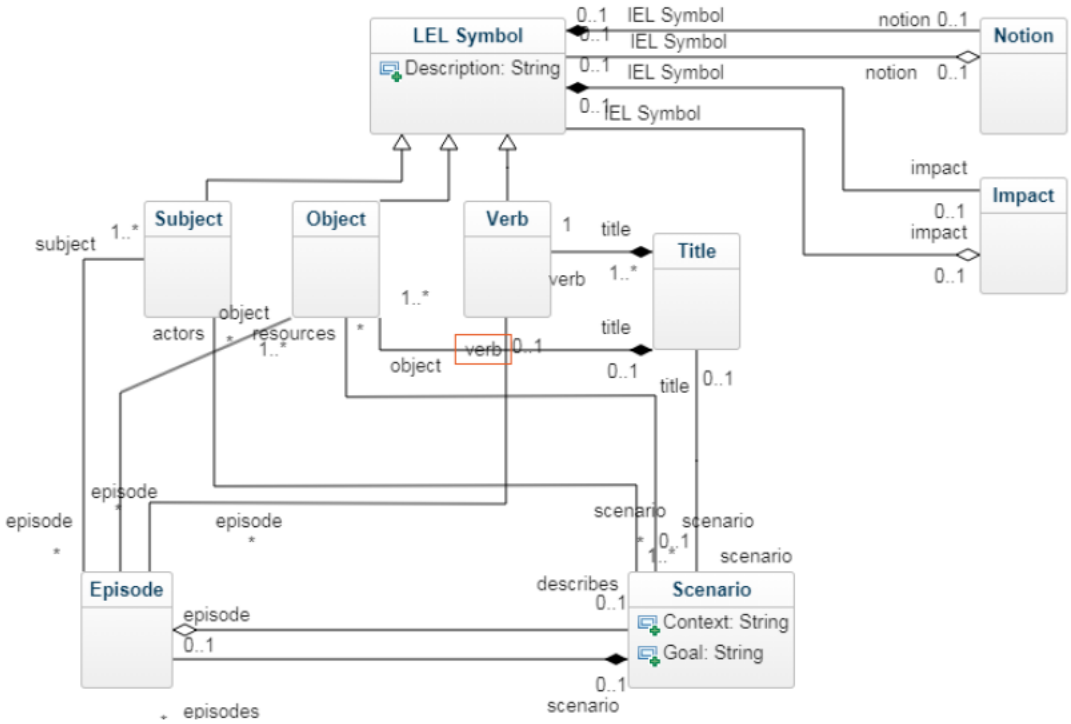

Figure 1. Scenario and LEL model

Table 1. Configuration summary

Categories		Forms	Templates		Properties
Scenarios		Scenario	Scenarios		Its context is
					The objective is
			Episode	EpisodesData	EpisodeObject
					Who participates
					What does the participant do
					What does the participant uses
			ResourcesScenario		The resources are
			ActorsScenario		The actors are
LELSymbol	Subjects	Subject	Subject		ImpactProperty
					NotionProperty
	Objects	Object	Object		ImpactProperty
					NotionProperty
	Verbs	Verb	Verb		ImpactProperty
					NotionProperty

In order to define the entities and provide the functionality to edit them, it is necessary to define semantic properties, the templates, the forms, and the categories. The semantic properties are the essential elements in a semantic website because they determine the links between the information. The templates define how the data is shown on a page. The template translates the semantic information in editable data. Forms are used to add and edit data through one or more templates. Finally, categories determine the elements that the Media Wiki manages. Table 1 summarizes the configuration of categories, forms, templates, and properties made to manage Scenarios and LEL.

Use of the application

The application provides support for the whole proposed approach. It provides functionality to describe Scenarios and symbols of the LEL, since they are the input necessary to derive the tests. It also implements some derivation rules to automatically obtain the tests.

Figure 2 shows the form to visualize and navigate the information of a Scenario. The screenshot shows the same Scenario described in Listing 1. The title of the Scenario is the title of the page. Then, the Scenario has information about its context and its objective that is textual information (String freely typed by the user). The actors and resources are LEL symbols, which means that the user can type them freely or select them from a suggestion list. In both cases, the elements of these attributes are linked to the LEL symbols. As a result, the user can click the words to navigate to that information. But more important, the tool recognizes Farmer, Product, Orders and Quality levels as symbols of the LEL, and they are used in the derivation of the tests. Then, the episodes are represented by triplets, where each element of the triplets is a reference to a LEL. So, it can also be navigated and the tool uses it with derivation rules. The page also shows a link to edit the information, and to view the history of the different modifications performed. This both links are not shown in figure 2, but they should appear in the top right corner.

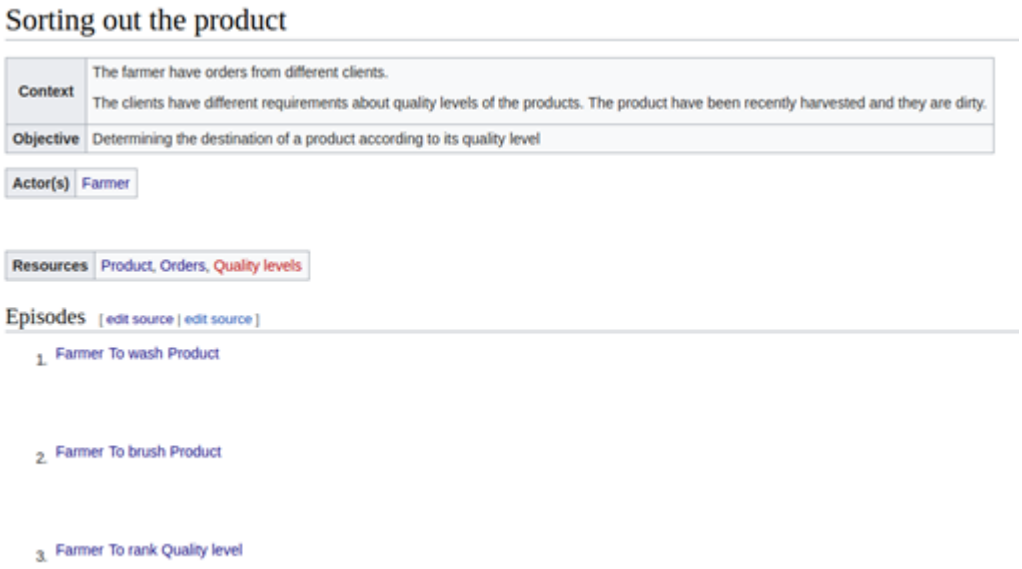

Figure 2. Scenario navigation form

The visualization and edition forms are different because edition forms allow to enter information freely and using suggestions, while the visualization forms allow navigating the information. Nevertheless, there is a more important reason: after the information is entered, the application transforms from text to semantic information, i.e. the application makes the links between Scenarios and LEL symbols. Figures 3 and 4 show the forms to edit the same Scenario. Figure 3 shows the attribute context where the information is plain text as well as in objective attribute. Then, actors and resources, which are LEL elements (that is why they are entered as text), are converted to tokens. Figure 4 shows the forms to edit the episodes. Since episodes have a specific structure (triplets of subject, predicate, and object), the form displays a template to enter an actor (the subject), a verb (the predicate) and the resource (the object). The Scenario may have many episodes.

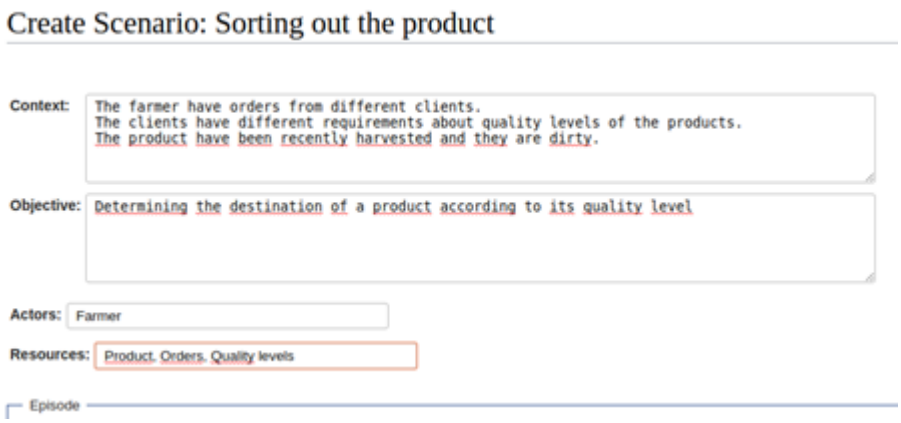

Figure 3. Scenario edition form

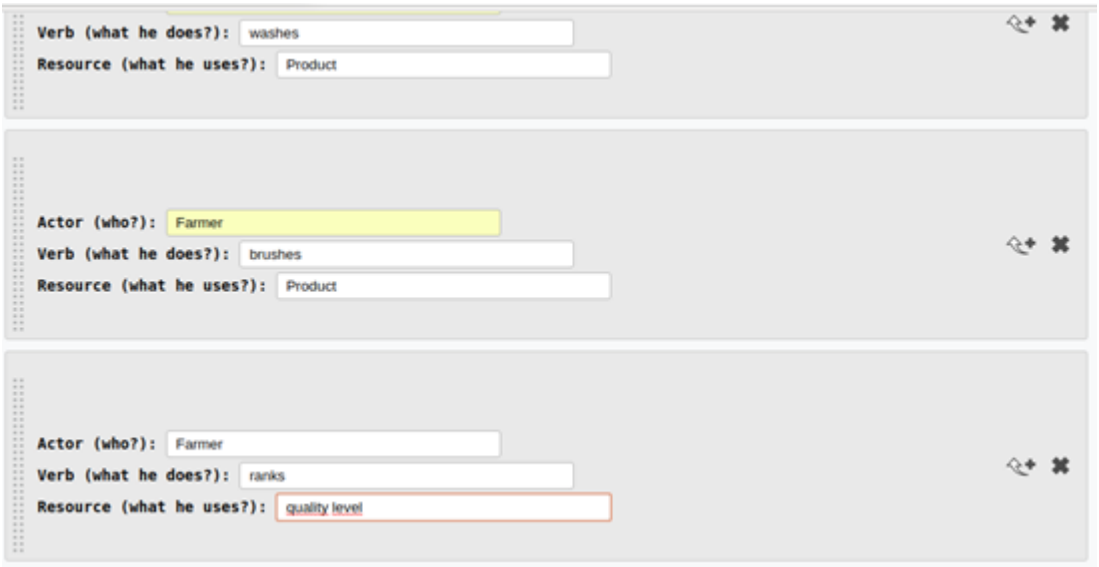

Figure 4. Scenario episodes edition form

The tool categorizes every element (Scenario or type of the symbols: subject, object or verb) and this category is used to structure the descriptions in order to derive the tests. But this

category is also used to group the information and show it to the user, so he can check and enrich. Figure 5 shows a form with all the LEL symbols defined in the catalog. This page is an index automatically calculated. The list displayed is a query similar to the needed to apply the rules to derive de tests. The page is configured to retrieve all the elements that belong to LEL Symbol category, in particular objects.

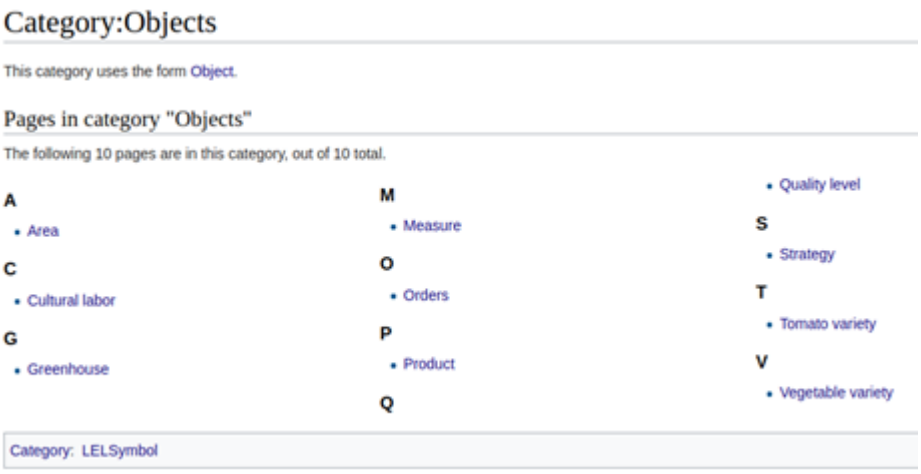

Figure 5. LEL symbols list of elements

Finally, the resulting test from the previous Scenario is displayed in Figure 6, the rules to derive the test are described in the following section. The strategy consists basically in obtaining one test suite from each Scenario. For example, the Scenario about Sorting out the products described in Listing 1 has the objective of preparing (washing and brushing) and packing the tomatoes (ranking and choosing). They should be washed because they could have dust or some chemicals, which is why they should be even brushed. After the tomatoes are adequately cleaned, they should be analyzed in terms of size, damages, and imperfections. Different clients have different demands according to quality, so packing the tomatoes implies ranking them according to quality levels, and choosing them in order to satisfy clients requirements.

During the whole process, many things could go wrong, for example, if tomatoes are not correctly washed, and when this happens, the selection of the adequate tomato could also go wrong. Thus, the situation where the tomatoes are not correctly washed must be tested. Another possible situation is that the selection fails; the tomato could be correctly washed and brushed, but there is a criteria mistake during selection and it is not correctly chosen. In this case, a client could receive a tomato that does not satisfy his requirements. Thus, every step in the Scenario could have two different results: the operation can be either done correctly or not. The different combinations (for example fail step one, or succeed step one but fail step two, and go on) are the whole set of test to perform for a Scenario. In this proposal, we do not analyze the details for performing each step (washing, brushing, etc…) and succeeding or failing them. We consider that every step could only succeed or fail. In further work, we plan to analyze every step (every action described with a LEL symbol of verb category) and give details to test different situation that could fail or succeed each step.

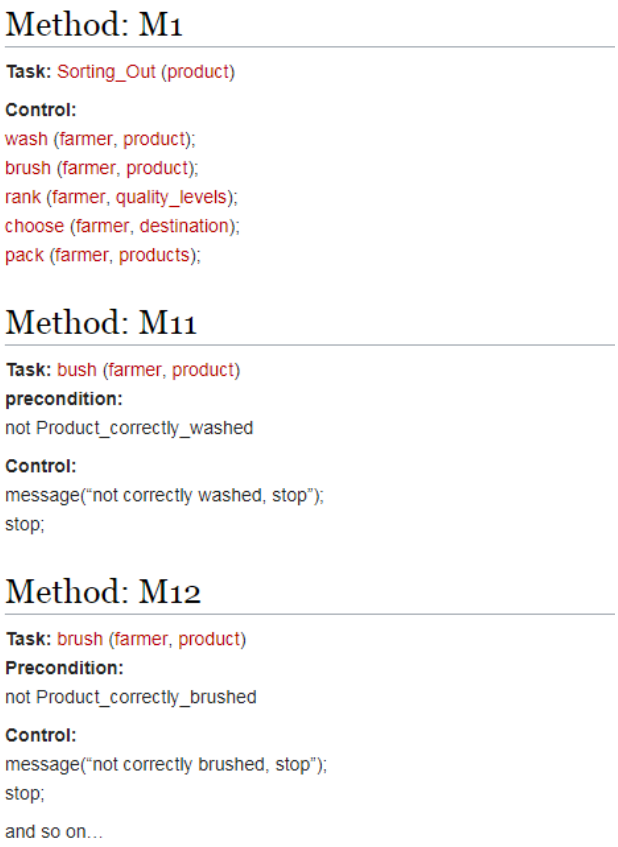

Figure 6. Task Method / Model test visualization form

```
= Method:  M11 =
<b>Task:</b>
       [[ {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhatDoesTheParticipantDo
       }} ]]
       ([[
       {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhoParticipate
       }}]],
       [[
       {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhatDoesTheParticipantUses
       }} ]]) <br>
<b>precondition:</b> <br> not Product_correctly_washed <br>
<b>Control:</b> <br>  message("not correctly washed, stop");<br>
stop;<br>
```

Listing 6. Source code for Method definition

An example of source code to make queries is listed in Listing 6. The source code shows wiki language as well are semantic queries. The queries are the blocks that begin with #ask. In the blocks, the categories described in table 1 are used to find specific elements of the Scenario.

Test Derivation Rules

This section enumerates the rules proposed to derive tests from Scenarios. Some preliminary versions of the rules were presented in previous work (self-reference removed). Although the rules could look similar to the previous ones, the Scenarios they rely on a more structured template to describe them.

Rule 1. Tasks Identification

As previously mentioned, a task in the Task/Method paradigm represents an action. It is thus natural to translate actions of Scenarios by tasks. In Scenarios, actions appear in the name of the Scenario and in the description of the episode. In the LEL mapping, these two elements correspond to the verb category. Therefore, the following rule can be stated:

Rule 1 (Tasks Identification):

- *from Scenario model: Each verb in the Scenario's episodes is translated into a task in Task/Method model. Each Scenario title is also a task in Task/Method model.*
- *from LEL model: Each element of the verb category is translated by a task.*

Examples:

Scenario: Sorting out the product → LEL mapping: <*sort out* verb> → Task: Sort out

Episode: The farmer washes the product → LEL mapping: <*wash* verb> → Task: wash

Episode: The farmer brushes the product → LEL mapping: <*brush* verb> → Task: brush

Episode: The farmer ranks the quality level → LEL mapping: <*rank* verb> → Task: rank

Episode: The farmer chooses the destination → LEL mapping: <*choose* verb> → Task: choose

Episode: The farmer packs the product → LEL mapping: <*pack* verb> → Task: pack

Listing 6. Task identification

Rule 2. Task's Parameters Identification

In Task/method models, task's parameters describe world objects required to the task execution. In Scenarios, these elements correspond to the "resource" field, and in the LEL model to the categories: "subject" and "object". Therefore, task's parameters can be identified thanks to the following rule:

Rule 2 (Task's Parameters Identification):

- *from Scenario model: Each resource linked to an action of a Scenario is translated into a parameter of the task representing the linked action of the Task / Method model.*
- *from LEL model: Each item of subject and object categories linked to an action is translated into a parameter of the task representing the linked action of the Task / Method model.*

Examples:

Episode: The farmer washes the product → LEL mapping: <*farmer* subject>, <*product* object> →Task: wash (farmer, product)
Episode: The farmer brushes the product → LEL mapping: <*farmer* subject>, <*product* object> → Task:brush (farmer, product)
Episode: The farmer ranks the quality level → LEL mapping: <*farmer* subject>, <*quality_level* object> → Task:rank (farmer, quality_level)
Episode: The farmer chooses the destination → LEL mapping: <*farmer* subject>, <*destination* object> → Task: choose (farmer, destination)
Episode: The farmer packs the product → LEL mapping: <*farmer* subject>, <*product* object> → Task: pack (farmer, product)

Listing 7. Parameters identification

Rule 3. Scenario's and Episode's method.

In Scenario model, the Scenario achievement is described in the "episodes" field, I.e. the scenario action (present in the "scenario" field) is executed by the achievement of its episodes ("episodes" field). In addition, in our approach, the execution of an episode can succeed or fail. In the Task/Method paradigm, executions of tasks are described by methods. We can thus state the following rule:

Rule 3 (Scenario's and Episode's method): The achievements of Scenarios and their episodes are associated to methods in Task / Method model.

Examples:

Achievement of Scenario Sorting out the product → Method: M1
Achievement of episode: The farmer washes the product → Method: M11
Achievement of episode: The farmer brushes the product → Method: M12
Achievement of episode: The farmer ranks the quality level → Method: M13
Achievement of episode: The farmer chooses the destination → Method: M14
Achievement of episode: The farmer packs the product → Method: M15

Listing 8. Method identification

Rule 4. Sequence of tasks

The sequence of different lines in the episodes part of a Scenario means that episodes must be executed in sequence. Each episode e_i of a Scenario s (task t) is translated by a task t_i; and the execution of the Scenario s by a method m of the task t, therefore the sequence $\{e_i\}$ of episodes (lines) determines the sequence $\{t_i\}$ of tasks in the control part of the method m of the task t in the Task / Method model. The use of expressions like "then", "after", etc... in the episodes of a Scenario determines also a sequence of tasks in the method's control part.

Rule 4 (Sequence of tasks): Different lines in the episodes part of a Scenario is translated by a sequence of tasks in the method achieving the task of this Scenario.

Examples:

Episodes: The farmer washes the product The farmer brushes the product The farmer ranks the quality level The farmer chooses the destination The farmer packs the product	→	Method: M1 Control Task: wash (farmer, product) Task:brush (farmer, product) Task:rank (farmer, quality_level) Task: choose (farmer, destination) Task: pack (farmer, product):

Listing 9. Sequence of tasks

Rule 5. Test Case Method

In this work it is assumed that each test case (Test cases part of scenario) corresponds to a binary achievement status of episodes: success or failure. If the achievement of an episode is succeeded, the execution of the Scenario continues with the following episode. In a failure situation, the scenario will stop. This stop case will be represented by a method for the next task (episode) in which the precondition field corresponds to the test case failure.

Rule 5 (Test Case Method): for each episode e_i (task t_i), a method me_{i+1} is added to the task t_{i+1} in which the precondition field corresponds to the test case failure.

For example:

```
Test case: Product correctly wahed / not correctly washed →
Method:  M12
Task:  brush(farmer,product) # next task
precondition: not Product_correctly_washed
Control:     message("not correctly washed, stop");
             Stop;
```

Listing 10. Test case method

Conclusions

In this paper, we have presented a tool that allows describing some early knowledge and requirements from a domain described as Scenarios, and obtain tests from them. The tests consist in providing the variation of all the combination of succeeding and fail alternatives for every step in the Scenarios. This element is able to capture very early and abstract information from the domain, thus, our proposal has two different goals: first, the tests provided can be used to check the Scenarios and improve them if necessary. Second, the tests can also be used to test the application implemented to satisfy the Scenarios. The Scenarios are complemented with a particular glossary, the Language Extended Lexicon. The LEL has two goals: since Scenarios are described textually, it is very important to have the description of the vocabulary used to describe them. But another important use of the LEL is to provide semantic support.

In this work, we show a first and preliminary semantic support to the Scenarios, but we plan to provide more information to the LEL in order to have more information to describe the Scenarios and to have more information to analyze and providing richer tests where different situations for succeeding and fail are considered. Since approaches like the one presented are very time demanding and error-prone to apply in software engineering, tools are extremely important to support them. So, this paper describes a wiki-based tool that allows to describe Scenarios, LEL symbols, and implement rules to derive tests according to the previous elements and their semantic relationships. We are now working in completing the set of rules, by adding rules required to translate more complex decision (switch or case situations instead of if then) as well as iterative behavior (loops or iterations like for each). We also plan to take benefit of the semantic support in order to improve the tests and consider more situation than a simple succeed or fail situation. We believe that this improvement will give benefit about testing the final application that satisfies the Scenario and also this improvement will make it possible to test the Scenario themselves and obtaining more complete and detailed Scenarios, which lately allow us to obtain better tests. This is a virtuous circle that is also reinforced by the semantic support. We also plan to develop some study cases in order to prove our beliefs.

Acknowledgements

Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015

References

(Bjarnason and Borg, 2017) Bjarnason E. and Borg M. (2017) "Aligning Requirements and Testing: Working Together toward the Same Goal," in IEEE Software, vol. 34, no. 1, pp. 20-23.

(Boucher and Mussbacher, 2015) Boucher M., Mussbacher G. (2017) “Transforming Workflow Models into Automated End-to-End Acceptance Test Cases,” in *Proceedings - 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering, MiSE 2017*, 2017, pp. 68–74.

(Budha et al., 2011) Budha G., Panda N. and Acharya A. A. (2011) "Test case generation for use case dependency fault detection," 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, 2011, pp. 178-182.

(Camilleri et al., 2003) Camilleri G., Soubie J.L., Zalaket J. (2003) “TMMT: Tool Supporting Knowledge Modelling,” in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 2773, pp. 45–52.

(Carroll, 2000) Carroll, J.M. (2000) Five reasons for scenario-based design. Interacting with computers 13.1. doi: 10.1016/S0953-5438(00)00023-0. pp 43-60.

(Chen and Qiang, 2010) Chen L. and Li Q. (2010) "Automated test case generation from use case: A model based approach," 2010 3rd International Conference on Computer Science and Information Technology, Chengdu, 2010, pp. 372-377.

(Entin et al., 2009) Entin V., Siegl S., Kern A., Reichel M. and Meyer-Wegener K. (2009) "A Scenario-Centric Approach for the Definition of the Formal Test Specifications of Reactive Systems," 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, Windsor, 2009, pp. 179-183.

(Forsberg et al., 2005) Forsberg K., Mooz H., Cotterman H. (2005) Visualizing Project Management, 3rd edition, John Wiley and Sons, New York, NY.

(Hsieh et al., 1013) Hsieh C., Tsai C. and Cheng Y. C."Test-Duo: A framework for generating and executing automated acceptance tests from use cases," 2013 8th International Workshop on Automation of Software Test (AST), San Francisco, CA, 2013, pp. 89-92.

(Huaikou and Ling, 2000) Huaikou M., Ling L. (2000) “A test class framework for generating test cases from Z specifications,” in *Sixth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2000*, pp. 164–171.

(Husain et al., 2015) Hussain A. *et al.*, (2015) “Review on formalizing use cases and scenarios: Scenario based testing,” *2015 Int. Conf. Emerg. Technol.*, vol. 3, no. 3, p. 1.

(Khamaiseh and Xu, 2017) Khamaiseh S. and Xu D. (2017) "Software Security Testing via Misuse Case Modeling," 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Orlando, FL, pp. 534-541.

(Lei and Wang, 2016) Lei H., Wang Y. (2016) “A model-driven testing framework based on requirement for embedded software,” in *11th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pp. 1–6.

(Leite and Franco, 1993) Leite J.C.S.dP., Franco A.P.M. (1993) “A strategy for conceptual model acquisition”, In Requirements Engineering conference. IEEE. doi:10.1109/ISRE.1993.324851, pp 243–246.

(Leite et al., 1997) Leite J.C.S.dP., Rossi G., Balaguer F., Maiorana V., Kaplan G., Hadad G., Oliveros A. (1997) “Enhancing a requirements baseline with scenarios”, In requirements Engineering. Vol 2.4. doi: 10.1109/ISRE.1997.566841. pp 184-198.

(Lipka et al., 2015) Lipka R., Potuak T., Brada P., Hnetynka P., Vinarek J., (2015) “A Method for Semi-Automated Generation of Test Scenarios Based on Use Cases,” in *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pp. 241–244.

(MediaWiki, 2018) Media Wiki (2018) available at: https://www.mediawiki.org (accessed September 1st, 2018)

(Nomura et al.m 2014) Nomura N., Kikushima Y., Aoyama M. (2014) “A Test Scenario Design Methodology Based on Business Context Modeling and Its Evaluation,” *21st Asia-Pacific Softw. Eng. Conf.*, vol. 1, pp. 3–10.

(Nogueira et al. 2012) Nogueira, S., Sampaio, A., & Mota, A. (2014). Test generation from state based use case models. Formal Aspects of Computing, 26(3), 441-490.

(Pandit et al., 2016) Pandit P., Tahiliani S., Sharma M. (2016) “Distributed agile: Component-based user acceptance testing,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–9.

(Philip et al., 2017) Philip G., Dsouza M., Abidha V. P. (2017) “Model based safety analysis: Automatic generation of safety validation test cases,” in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pp. 1–10.

(Potts, 1995) Potts C. (1995) “Using schematic scenarios to understand user needs”. In proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques. ACM. doi: 10.1145/225434.225462.

(Sarmiento et al., 2014) Sarmiento E., Leite J.C.S.dP., Almentero E. (2014) “C&L: Generating model based test cases from natural language requirements descriptions”. In Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on. IEEE, 2014. doi: 10.1109/RET.2014.6908677

(Sarmiento and Leite, 2016) Sarmiento E., Leite J.C.S.dP., Almentero E., Sotomayor G. (2016) “Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets,” *Electron. Notes Theor. Comput. Sci.*, vol. 329, pp. 123–148.

(Svensson and Regnell, 2015) Svensson, R. B. and Regnell B. (2015) "Aligning Quality Requirements and Test Results with QUPER's Roadmap View for Improved High-Level Decision-Making," 2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing, Florence, 2015, pp. 1-4.

(Trichet and Tchounikine, 1999) Trichet F., Tchounikine P. (1999) “DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods,” *Expert Syst. Appl.*, vol. 16, no. 2, pp. 105–120.

(Vieira et al,. 2006) Vieira, M., Leduc, J., Hasling, B., Subramanyan, R., & Kazmeier, J. (2006, May). Automation of GUI testing using a model-driven approach. In Proceedings of the 2006 international workshop on Automation of software test (pp. 9-14). ACM.