Integrating Requirements Specification and Model-Based Testing in Agile Development

Dalton N. Jorge, Patrícia D. L. Machado, Everton L. G. Alves, Wilkerson L. Andrade
Federal University of Campina Grande (UFCG), Campina Grande, PB, Brazil
Software Practices Laboratory (SPLab)
daltonjorge@copin.ufcg.edu.br, {patricia, everton, wilkerson}@computacao.ufcg.edu.br

***Abstract*—In agile development, Requirements Engineering (RE) and testing have to cope with a number of challenges such as continuous requirement changes and the need for minimal and manageable documentation. In this sense, extensive research has been conducted to automatically generate test cases from (structured) natural language documents using Model-Based Testing (MBT). However, the imposed structure may impair agile practices or test case generation. In this paper, inspired by cooperation with industry partners, we propose CLARET, a notation that allows the creation of use case specifications using natural language to be used as central artifacts for both RE and MBT practices. A tool set supports CLARET specification by checking syntax of use cases structure as well as providing visualization of flows for use case revisions. We also present exploratory studies on the use of CLARET to create RE documents as well as on their use as part of a system testing process based on MBT. Results show that, with CLARET, we can document use cases in a cost-effective way. Moreover, a survey with professional developers shows that CLARET use cases are easy to read and write. Furthermore, CLARET has been successfully applied during specification, development and testing of industrial applications.**

***Index Terms*—Agile Development, Requirements Engineering, Model-Based Testing, Use case specification.**

I. INTRODUCTION

Requirements Engineering (RE) in agile development is still not a well-defined and understood practice [7]. While the central artifact in agile development is the code, requirements documents are mandatory in industry particularly due to the need for contract specification. However, due to the nature of agile development, RE has to be performed quickly along with a number of small development cycles. Moreover, it is difficult to anticipate the exact time when requirements should be documented since they may not be sufficiently elicited during a number of cycles. In this sense, agile RE has been performed informally and Natural Language (NL) seems to be the prevalent requirements notation [7], [28].

Likewise, while test automation is a key practice in agile development, system level testing has been performed manually based on test scripts written in NL [2], [4], [28]. Such test cases may be the only documentation of requirements. To handle this problem, Model-Based Testing (MBT) [27] practices can be applied to automatically generate test cases from requirements documents in other to improve the cost-effectiveness of system level testing and target the effort from writing test cases to documenting and updating requirements.

A synergy between RE and MBT practices can provide a proper alignment of the activities and bring several benefits to software development and evolution [2], [3], [27]. While creation of test models from scratch is prohibitive, RE practices can provide key information that allow for the automatic generation of models. On the other hand, benefits of MBT may motivate the need for constructing more complete and well-formed requirement documents and for keeping them up to date. In this sense, this synergy demands that requirement documents become a central artifact to both practices. However, while a number of MBT approaches and tools has already been presented in literature, as well as successful reports confirming that expectations on MBT can be met in practice [10], [11], [21], [22], there are few reports on its use in industry, particularly in agile development, and there is still a lack of effective experimentation of techniques [8]. A number of challenges still need to be faced such as the costs of creating test models and/or reusing development artifacts [4], [19], [27].

We propose *CentraL Artifact for RE and mbT* (CLARET), a notation devoted to create use case specifications using NL as a central artifact for: i) generation of formatted use case documents as part of RE practices and ii) test models for system level MBT [17]. Basically, CLARET defines the structure and syntax of the main fields required in a use case specification that is supported by a standard semantic interpretation of how flows are connected. A tool set [1] checks on well-formedness and generates test models as labelled transition systems, from which test cases can be generated, for instance, by using the LTS-BT tool [5].

CLARET has been motivated by the needs of our industry partners. On one hand, NL use case specifications with enriched document format and layout need to be developed as organizational contracts. On the other hand, due to the expected continuous changes to requirements, agile development is required with system testing as a frequent validation activity. With CLARET, use cases can be written and updated as plain text making requirements evolution more feasible and yet preserving the necessary structure required for system level test case generation. In this paper, we present empirical studies based on the experience achieved in its use. These studies show that the development team find CLARET specifications easy to read and write. Moreover, the CLARET approach is effective for system specification and testing of industrial applications. We also discuss current challenges and lessons learned.

2332-6441/18/$31.00 ©2018 IEEE
DOI 10.1109/RE.2018.00041

RE 2018, Banff, Canada
Industry Innovation Paper

Targeting use case specifications for test case generation is not a novel approach to MBT [19], [21], [29]. However, most of the work is devoted to the generation of automated test cases and, therefore, they rely on the use of a controlled language to allow for semantic interpretation of stories. Additionally, proposed approaches may consider the need for additional modeling [29]. On the other hand, CLARET is devoted to automatic generation of manual test cases and allow for specification of use cases using NL, broadening its applicability and promoting the use of MBT in the context of manual and agile practices. Automated execution often demands high costs on test case creation and evolution. Therefore, a considerable amount of system level test cases is still manually executed [2]. Moreover, there are several ways to write a use case and a number of templates have been devised in industry. From CLARET, we can generate RE documents based on different templates, and, at the same time, provide a valuable input for MBT processes in the form of well-formed test models.

This paper is structured as follows. Section II quickly introduces MBT. Section III presents CLARET and an overview of a process under which it can be applied. Sections IV, V, and VI describe several studies on usage of CLARET in the industrial context. Section VII presents lessons learned. Section VIII discusses related work and Section IX presents concluding remarks and pointers to further work.

II. MODEL-BASED TESTING

Model-Based Testing relies on the existence of a suitable and well-formed model from which test cases can be generated [9], [27]. The main goal of MBT is to improve efficiency and effectiveness of testing processes by speeding up test case creation based on a systematic procedure to cover test requirements. This procedure can make testing more reliable, as it automatically generates, based on well-founded theories, important artifacts such as test data and oracle.

An MBT process starts with the creation/reuse of a model, named *test model*, from which test cases, and all necessary artifacts for test execution, can be generated. Although there are different notations for defining test models (e.g., [12], [18], [25]), at system testing level, a test model can be obtained from Requirements Engineering artifacts, particularly natural language specifications. Studies have shown that developers can produce accurate NL specifications [13]. However, as development and testing may have different goals, it may not always be feasible to generate test models from requirement documents such as use case specifications. For instance, they may lack structure and precise details on flows, actors, and so on, particularly NL documents. Moreover, RE artifacts may become obsolete if not updated continuously, particularly in agile development.

III. CLARET

CLARET is a domain specific language designed to be the central artifact for automatic generation of both test cases and formatted use case documents. The main goal of CLARET is to provide a practical way for using NL in the specification of use cases integrating RE and MBT processes.

```
1  systemName ::= 'systemName' StrLiteral usecase*
2  usecase ::= 'usecase' StrLiteral '{' elem '}'
3  elem ::= version? actor+ pre basic (exception |
       alternative)* post
4  version ::= 'version' StrLiteral 'type' StrLiteral
       'user' StrLiteral 'date' StrLiteral
5  actor ::= 'actor' ident StrLiteral
6  pre ::= 'preCondition' StrLiteral (',' StrLiteral)*
7  basic ::= 'basic' steps
8  post ::= 'postCondition' StrLiteral (',' StrLiteral)*
9  alternative ::= 'alternative' [0-9]+ StrLiteral steps
10 exception ::= 'exception' [0-9]+ StrLiteral steps
11 steps ::= '{' (action | response)* '}'
12 action ::= 'step' [0-9]+ ident StrLiteral (af | bs)*
13 response ::= 'step' [0-9]+ 'system' StrLiteral
       (ef | bs)*
14 af ::= 'af' '[' [0-9]+ (',' [0-9]+)* ']'
15 ef ::= 'ef' '[' [0-9]+ (',' [0-9]+)* ']'
16 bs ::= 'bs' [0-9]+
```

Listing 1. EBNF Production Rules.

A. CLARET *Grammar*

CLARET has structure and syntax containing the main fields required by a use case document (**systemName**, **usecase**, **actor**, **preCondition**, **postCondition**, **basic**, **alternative**, and **exception**). All fields are aligned with the UML use case definition, minimizing the learning curve of CLARET. For specifying the CLARET grammar, we used the Extended Backus-Naur Form (EBNF) [16], a family of metasyntax notation which is often used for formal description of programming languages. Listing 1 presents CLARET's grammar.

Due to its similarity with the JSON's definition[1], we believe developers are likely to have little trouble either to understand or to use CLARET, particularly due to the fact that the intended use of CLARET is in agile development, where most team members are skilled programmers.

Line 1 of Listing 1 shows the **systemName** production rule. It is the starting point of a use case specified in CLARET and includes all use cases of the system. The **usecase** rule (Line 2) represents a use case defined by a version (Line 4), at least one actor (Line 5), pre-conditions (Line 6), a basic flow (Line 7), optional alternative flows (Line 9), optional exception flows (Line 10), and post-conditions (Line 8).

The **version** element (Line 4) is an optional information regarding the history of the use case such as version number, description, who has edited the use case along with the edition date. The version information is important for controlling the evolution of the system and its artifacts.

All external actors related to the use case are specified through the **actor** element (Line 5). This rule allows the definition of aliases (*ident*) for each actor to reference them in use case steps in an easy and consistent manner.

The **pre** rule (Line 6) is used to define expressions that must be satisfied before the execution of the basic flow of the use case. This keyword is mandatory even if no conditions exists,

[1] http://json.org/

in this case with an empty quotes. Same obligation is applied to **post** rule (Line 8).

The basic flow is the main scenario of the use case and it is specified using the **basic** rule (Line 7). In CLARET, each scenario is composed of a list of steps (Line 11). In turn, each step is composed of a sequence number, either the system keyword or an actor identifier, an NL string literal describing the step and skips, if exist, to other flows (*af*–alternative flow; *ef*–exception flow; *bs*–basic step) (Lines 12 and 13). Each step means a single interaction between an actor and the system (Line 12), or a response from the system (Line 13). After the execution of the basic flow, post-conditions can be specified through the **post** rule (Line 8).

Finally, Lines from 14 to 16 can be used to specify all deviations from the basic flow to alternative, or exception flows, as well as vice-versa.

For the sake of well-formedness of test cases, the static semantics of CLARET requires each actor step to be followed by a system step. The reason is that test cases for manual execution are usually organized as sequences of steps actions and corresponding expected results. Consequently, the first step of an alternate flow must be an actor step that replaces the one it originates from the current flow, whereas the first step of an exception flow must be a system step. Furthermore, the skip return must be to a different type of step from the one it originates and also exception flows must end with a system step if there is not return to the main flow.

CLARET enables MBT because, different from plain natural language, CLARET use cases are structured in a way that flows can be captured in the form of an LTS model which is the input for the test case generation tool that can explore different combinations of flows of execution. Moreover, use cases contain all fields that are necessary for test case description, particularly, test cases for manual execution such as actors (who performs each step), preconditions (setup conditions) and postconditions, and the explicit distinction between action and response (making it clear what are the test results to observe).

B. A CLARET *Example*

Listing 2 exemplifies the use of CLARET in the specification of the following user requirement of an email system: "*In order to access her email inbox the user must be previously registered in the system and must provide correct username and password. The system may suggest previously attempted user names. In case of incorrect combination of username and password, the system must display an error message and ask for new data. If the user is not registered the system must inform that the user is not registered.*"

Firstly, we define the system name (Line 1) and the name of the first use case (Line 2). Then, we provide some optional information about the use case such as its version, the kind of modification, the name of the use case writer, and the modification date. Next, we define the actor (Line 4) followed by the use case precondition (Line 5).

We specify the main scenario using the "basic" element (Lines 6-11). Note that the main scenario is composed of 4

```
1  systemName "Email"
2  usecase "Log in User" {
3    version "1.0" type "Creation" user "Manager" date  "
       01/01/2018"
4    actor emailUser "Email User"
5    preCondition "There is an active network connection"
6    basic {
7      step 1 emailUser "launches the login screen"
8      step 2 system "presents a form with username and
        password fields and a submit button"
9      step 3 emailUser "fills out the fields and click on
         the submit button" af[1]
10     step 4 system "displays a successful message"
        ef[1,2]
11   }
12   alternative 1 "Username is predicted" {
13     step 1 emailUser "selects a suggested user name,
        types password and click on the submit button"
        bs 4
14   }
15   exception 1 "User does not exist in database" {
16     step 1 system "alerts that user does not exist"
        bs 3
17   }
18   exception 2 "Incorrect username/password combination"
        {
19     step 1 system "alerts that username and/or password
         are incorrect" bs 3
20   }
21   postCondition "User successfully logged"
22 }
```

Listing 2. Use Case in CLARET.

steps (Lines from 7 to 10). Each step has an identification, an agent (actor or system), and an action described in NL. Note that in Line 9 there is a possible deviation specified by the "*af*" mark (alternative flow) meaning that the first step of alternative flow 1 can be executed instead of the step 3 of the main flow. After the execution of step 1 of alternative flow 1, there is a return point to the step 4 of the basic flow, meaning that basic flow may resume after the alternate step. Similarly, there is another possible deviation in Line 10 indicating, through "*ef*" (exception flow), the possibility of execution of exception flows 1 or 2 instead of step 4 of the main flow. Finally, we specify the post-condition in Line 21 indicating that, if step 4 of the basic flow is successfully executed, the user must be logged in the system.

From this specification, we can generate RE documents in different formats. Currently, the CLARET toolset exports formatted documents according to a template defined by our industry partners. Furthermore, from the use case, we can identify the following linearly independent[2] flows of execution as test cases: [bs:1 → bs:2 → bs:3 → bs:4], [bs:1 → bs:2 → af[1]:1 → bs:4], [bs:1 → bs:2 → bs:3 → ef[1]:1 → bs:3 → bs:4], [bs:1 → bs:2 → bs:3 → ef[2]:1 → bs:3 → bs:4].

C. CLARET *Inside Agile Development*

An overview of CLARET regarding its tool support and related artifacts is shown in Figure 1. For the sake of simplicity, it focus on activities concerned with test generation, use case creation, and changes. Note that CLARET is a central artifact for both RE and MBT. Moreover, we follow the iterative philosophy of Agile RE assuming that the RE process includes

[2]The ones that have a least one different step when compared to the others.

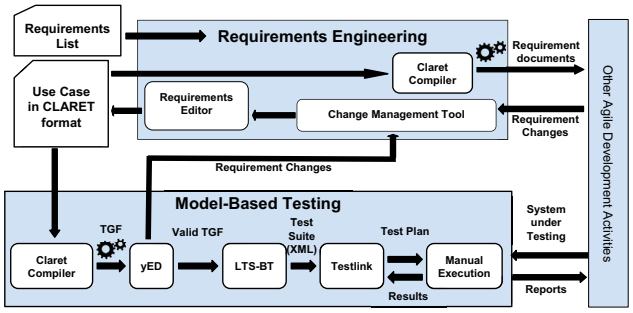

Fig. 1. CLARET tools and artifacts.

activities such as Change Management and Requirements Management to support continuous changes to requirements, by creating/updating CLARET documents – a key feature for effectiveness of both Agile RE and MBT [6], [15].

The RE flow of artifacts starts with the user requirements list (for example, the product backlog) that is the base for creating CLARET use cases. From it, use cases can be created and updated continuously. As needed during the agile cycles, the CLARET tool compiles the use case specifications for generating: i) TGF[3] test models for MBT and ii) RE Docs for communication with stakeholders in the RE process. In the scope of MBT, the TGF artifacts can be visually analyzed using the yEd tool[4] in order to seek for possible inconsistencies and unspecified scenarios that leads to change requests.

In the MBT flow, from (validated) TGF models, test cases can be generated using the LTS-BT tool [5]. LTS-BT can generate test cases extracting paths from TGF models using a Depth-First Search (DFS) algorithm based on coverage criteria and also test suite reduction strategies. Generated test cases can be exported in an XML format accepted as input by Testlink[5], a test management tool that supports manual test case execution. For execution, a test plan can be defined and manually executed on the system under testing. Testlink provides reports on test results that may lead to change requests.

In the studies presented in Sections V and VI, CLARET was applied in a SCRUM framework for effective team collaboration in agile development. As expected, use case documents are not required to be complete at any stage in or after a Sprint, especially the ones that do not lead to release deployment. This agile practice does not impair test case generation since, in our context, test cases are manually executed. Thus, they can be informal and do not need to be completely elaborated, particularly at early Sprints. For instance, test cases can be generated out of only main flows of functionalities with the best understood steps so far. As understanding and knowledge are acquired throughout the Sprints, CLARET documents can become more elaborate so that more complete test suites can be obtained either prior to release deployment or if needed by the team, to validate the increment of a given Sprint. CLARET documents are open to change and test suites can be generated when needed by the team to achieve a Sprint Goal without any mandatory process.

[3]https://en.wikipedia.org/wiki/Trivial_Graph_Format
[4]https://www.yworks.com/products/yed
[5]http://www.testlink.org/

IV. EXPLORATORY STUDIES ON USE CASE SPECIFICATION

To evaluate CLARET's effectiveness, we performed two studies: (1) a study in which we asked developers to specify requirements using either CLARET or free-writing and we measured quality aspects of the created artifacts, such as time, requirements coverage, completeness and consistency; and (2) a survey in which we asked developers to inspect use cases written with both CLARET and free-writing and evaluate them regarding legibility and expressiveness.

Despite the existence of a number of other NL use case specification notations, in our studies, we opted to focus on free-writing only. The reason is that free-writing suits well our context of investigation, as it is often used in Agile RE and even MBT processes have been defined based on them [2]. Also, free-writing was used as a baseline comparison to investigate whether CLARET fits in a NL use case writing context. CLARET is solely devoted for high level system testing and test cases are sequences of user-system level steps.

A. User Study on Quality

In this study, we recruited 8 professional developers that have been working on projects for different software companies, including roles as software developers, quality engineers, and project managers. All participants have experience on requirements specification (average of 33.5 use cases per participant). Most participants have professionally created/ maintained use cases in different projects.

1) Procedure: The participants were divided into two groups: Group I - participants that would model requirements using CLARET (4 participants); and Group 2 - participants that would model requirements using free-writing (4 participants). Prior to the study sessions with participants from Group 1, a seminar was held to set the required background on the CLARET syntax and semantics of its elements. Next, we asked the participants to specify a use case from a giving system using the respective specification strategy. For this, we provided a user requirement definition which contained a high-level description of a system used in a school environment (Table I). This system keeps track of students' absent days from school and sends warning messages to parents when a student exceeds the threshold of acceptable absent days.

TABLE I
USER REQUIREMENT DEFINITION.

The system must verify daily whether a student has exceeded the acceptable limit of absent days (limit is set by the school department). In case this limit is exceeded, an SMS message is sent to the student's parents informing her number of absent days. The system will send a single message every new exceeded absent day. The parent must have previously registered a valid cellphone number and enabled the service of SMS warnings. The system must have a default warning message to be used in SMS messages.

2) Research Questions: [*RQ 1*] Do developers create artifacts with higher requirement coverage when using CLARET? [*RQ 2*] Are CLARET artifacts consistent and complete? [*RQ 3*] Can developers save time when specifying requirements using CLARET? To investigate them, we selected three metrics:

- RC (Requirement Coverage): Prior to the study, a requirement list was created based on the user requirement definition (Table I). This list has nine basic requirements that should appear in the use case in order to assure it covers the user requirements definition. This coverage analysis simulates an inspection performed by a requirement engineer in which she reviews a recently created requirement artifact aiming at identifying incorrectness or completeness/consistency issues. The list of basic requirements is available in our website[1]. RC calculates the rate of covered requirements over the expected number.

- UCCC (Use Case Completeness and Consistency): The Open UP project[6] provides a checklist [26] with 24 questions for verifying whether a use case is specified in a consistent and complete manner. After the study, we asked an independent experienced requirement engineer to inspect all requirement artifacts (use cases) created by the participants and to go through the Open UP checklist. UCCC measures the rate of found positive answers.

- Time: The time spent by each participant for creating the use case.

3) Results and Discussion: Based on the artifacts created by the participants, we measured the requirement coverage values (RC). Considering the participants from Group 1, we observed in average 94% of requirements coverage, while this rate dropped to 91% for the Group 2. Even though there is a numerical improvement of 3% when using CLARET, no significant statistical difference can be found. Therefore, by using CLARET, developers can create artifacts with similar requirements coverage when compared to free-writing (*RQ 1*).

Regarding consistency and completeness, the null hypothesis is that the groups would achieve similar UCCC in average. However, an analysis performed by the experienced requirement engineer found that while participants from Group I created use cases that covered an average of 88.5% elements of the Open Up checklist, the artifacts created by participants from Group II covered only 51%. Table II presents the rates obtained by each participant. A Chi-Squared proportion test with 95% of confidence ($p - value = 3.774e^{-08}$) found that the difference between the rates is statistically significant, which leads us to answer our second research question (*RQ 2*) by stating that CLARET may enable the creation of more consistent and complete specification artifacts.

TABLE II
UCCC RATES BY PARTICIPANT.

	Free-writing				CLARET			
Participant	F1	F2	F3	F4	C1	C2	C3	C4
UCCC	12	12	14	11	22	23	22	18

[6]https://www.eclipse.org/epf/openup_component/openup_vision.php

We investigated which elements from the Open Up checklist were most missed by participants that used free-writing. We observed that, when creating use cases freely, participants were often inconsistent with actors specification and step labels. Moreover, several steps were not clearly defined or associated with any actor. Those are common mistakes that often make free written use case ambiguous and/or hard to read, and may prevent their use for automatic test generation. Most of these issues were not found in the use cases written using CLARET. The controlled and simple syntax of CLARET helped participants to visualize these kind of issues and to fix them right away which ended up creating more readable use cases.

Finally, we measured the time spent by each participant when specifying the use cases. On average, participants in Group I spent 34.39 minutes modeling their use cases, while participants from Group II spent 34.11 minutes. Although slight bigger, the time spent using CLARET showed to be more productive, since the rates of requirement coverage (RC), and consistency and completeness (UCCC) were higher. Moreover, it is important to mention that, for most participants from Group I, the study sessions were their first contact with CLARET. We believe this time tend to decrease as more experience a developer gets with CLARET syntax and its way of modeling. However, an investigation in this sense is yet to be done. Thus, in the context of our user study, we cannot say that developers would save time by using CLARET instead of free-writing (*RQ 3*), however, the time spent was almost identical for both strategies.

B. Survey on Legibility and Expressiveness

To assess whether CLARET improves use case legibility and expressiveness when compared to free-writing specification, we performed a survey with professional developers.

1) Procedure: We selected 6 developers that have experience on specifying use cases with both strategies (free-writing and CLARET) to participate in our survey. Since we wished to explore retrospective knowledge, we focused on participants with experience in CLARET acquired from different projects of our industrial partners. We asked participants to answer an online survey in which two use case specifications of a generic system were provided, one using CLARET and the other using free-writing. Each participant had to inspect the artifacts and rate them (from 1 to 5) regarding their legibility (very legible - 1; not legible - 5) and expressiveness (very expressive - 1; not expressive - 5). Specifications and questionnaire are available in our website [1].

2) Research Questions: [*RQ 4*] Are CLARET artifacts easier to read? [*RQ 5*] Are CLARET artifacts more expressive?

3) Results and Discussion: Figure 2 presents the results of our survey. While 100% of all participants granted to CLARET the best ratings regarding legibility (50% - 2; 50% - 1), for the free-writing artifacts this rate dropped to 67%. Regarding expressiveness, 83% of the participants rated the CLARET artifacts as very expressive (rates 1 or 2), while the free-writing artifacts were rated as expressive only by 17%. These results enable us to answer *RQ 4* and *RQ 5*. In the context of our survey, developers find CLARET artifacts easier to read and

more expressive than free-written use cases. This conclusion evidences the contribution that CLARET brings: more legible and expressive requirements artifacts. Moreover, these artifacts tend to be more useful during software development and to be easier to update and maintain in practice.

Notice that results are very specific to the context of the survey that includes developers with previous knowledge on CLARET. Moreover, the study has the purpose of observation with the aim of improving the practice in the specific context.

C. Threats to validity

In terms of *construct validity*, alternative metrics could also have been considered for evaluating CLARET. For instance, we could leverage the experience of professional requirement engineers for inputting a subjective analysis regarding artifacts' quality. This analysis would complement the evaluation of our metrics (RC, UCCC and Time). However, for our investigation, we believe that the used measures were appropriate. Future additional studies might consider different metrics.

In terms of *internal validity*, before the study sessions, CLARET was introduced through a presentation that included examples of how to use it and its main features. Moreover, the first author was available during the sessions to help participants with any usability issue.

In terms of *external validity*, some circumstances of our studies may hinder the generalization of their results. First, in both the study on quality and the survey, a limited number of user requirement definitions were used, which prevent us from saying that they represents all possible ones. However, although simple, the used definitions emulate scenarios that are present in most system specifications (basic flow, alternative and exception flows). Moreover, as dealing with professional developers, we tried not to overload them with a series of specifications, which could impact their motivation for participating in our studies, and consequently the found results. Furthermore, one may argue the participants do not represent the whole universe of developers. However, to minimize this threat, we recruited professional developers from different projects, with different roles and levels of experience.

V. CASE STUDY ON USE CASE SPECIFICATION

In this section, we present a case study on requirements specification, using CLARET, of an industrial application under operation, performed in the context of a cooperation between our research lab and Ingenico do Brasil Ltda[7]. The application considered in this study is a payment system with 78 use cases including communication, configuration, sale and payment transactions. The goal was to specify all functional requirements of a system under refactoring and also to generate a regression system test suite from them.

1) Procedure: Based on existent documentation, interaction with the client and exploratory testing, the team iteratively performed refactoring, wrote use cases in CLARET and generated test cases from them following the agile practice described in Section III-C. Both the textual and TGF files were reviewed, as needed, to check for validity, completeness related to knowledge acquired and well-formedness. For each release to be deployed, the CLARET compiler generated the requirements documents and test suites that refer to it.

The part of the team concerned with specification and testing was composed of two professional engineers and two undergraduate students in Computer Science. One of the professionals had previous experience as professional on writing requirements and acceptance test cases whereas the other professional was a beginner. As usual in the agile context, these participants had also performed other technical tasks within each Sprint. Before they started, we performed a basic training on writing use cases and the CLARET notation. It took the team eight months to complete the task that was split into 16 Sprints with a total of 3 releases to the client.

After delivery, we performed a questionnaire among the participants of the case study with the goal of understanding current benefits and drawbacks of the CLARET approach, including the four participants directly involved and also the project manager and two developers responsible for the refactoring task. The questionnaire had objective questions using a Likert scale along with open questions so that the participants could freely express their opinion and explain their answers to the objective questions.

2) Research Questions: [*RQ 1*] Do CLARET and the CLARET tool set enable requirements specification for RE and MBT? [*RQ 2*] Are CLARET use cases suitable for system-level test case generation? These questions were addressed through the questionnaire answers and also by observation of the team while performing the task.

3) Results and Discussion: A total of 78 use cases was specified using CLARET. From them, 236 test cases were generated, organized as a test suite per use case. Both use cases and test suites were constructed, augmented and inspected interactively as requirements elicitation occurs as part of Sprint planning and feedback from the client. CLARET documents were continuously subject to modification as expected.

Figure 3 summarizes the objective questions and answers gathered from the participants. Overall the participants rated the task of reading and writing use cases in CLARET as "Very Easy" and "Easy" with one "Neutral" opinion. Moreover, participants rated the quality of the artifacts generated by the CLARET compiler as "High Quality" and "Good Quality". The reason is that the CLARET approach was integrated easily in the flow of activities and tool set already in use. Use cases specification could be performed in any text editor making version control easy. Besides, the approach helped to accomplish the overall task of requirements specification and refactoring. One of the participants quoted as benefits of using CLARET: *"Facilitated and simplified the creation of use cases, and left such a task faster"*. Moreover, having system test cases automatically generated freed them from manually creating the tests. The test suites generated were used for validating the refactoring edits made to the system before each release, giving confidence to the team. One of the participants quoted as benefits of the

[7]www.ingenico.com.br

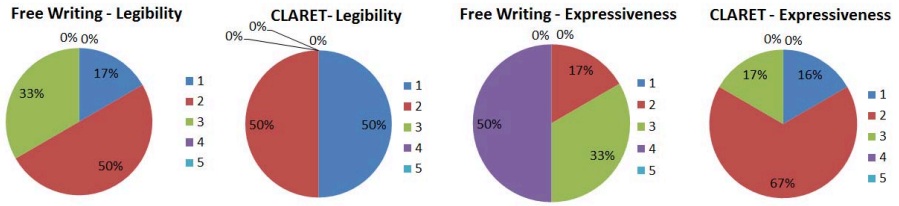

Fig. 2. Survey results regarding legibility and expressiveness.

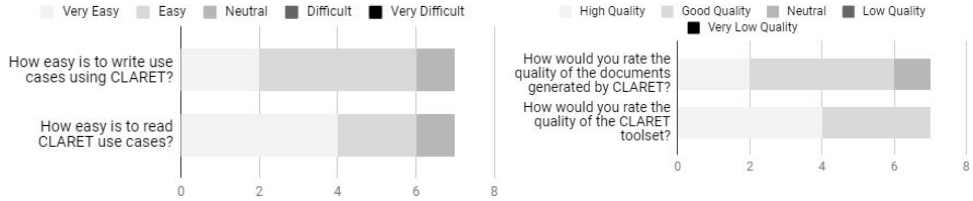

Fig. 3. Questionnaire results from case study.

approach: *"Ease of reading use cases, and this helped at the time of testing the application"*.

Regarding use cases as input for MBT, the regression test suite was completely generated from the same CLARET documents used in the RE practices. The resulting test cases are sufficiently described and self-contained providing all information necessary to manual execution such as preconditions, steps and expected results. Moreover, the team was able to address the goals of both practices when producing the documents without compromising the schedules of the sprints. Therefore, we can answer *RQ 1* by saying that CLARET and its tool set can enable requirements specification for both RE and MBT practices.

One challenge of this study was to produce a regression test suite to be used as a safety net during refactoring. This goal was achieved since the generated suite was continuously executed to validate the refactorings performed along with a number of sprints. There was no need to create additional test cases manually. Moreover, the test cases completely covered the main functionalities of the system, including all scenarios of interest as specified in the CLARET documents. Therefore, we can answer *RQ 2* by saying that CLARET use cases can be suitable for system-level test case generation.

However, as expected, challenges needed to be faced. Firstly, use case specification was not a practice in the team. Therefore, writing requirements was a challenge at first which was addressed by training. Secondly, due to poor structure of the flow of functionalities in the system, some requirements could not be written prior to refactoring, since they would lead to badly formed use cases. Finally, current version of the CLARET toolset still need improvements on usage such as more detailed error messages and more documentation on examples of use case specification. One of the participants quoted the following concern: *"There is a lack of documentation that explains better what can be written and what should not be written to generate the flows, this has generated some confusion when writing the use cases. Sometimes flows were generated that were not part of the application."* This happened when flows were mistakenly connected according to the CLARET grammar (Section III).

Even though use case statements can be completely described in natural language, CLARET has a grammar and documents compilation may issue syntax errors. Therefore, we must also be concerned about how incidence of syntax errors may affect CLARET application. For the sake of not interfering with the development environment, we did not collected the number of syntax errors related to CLARET grammar found by the participants. However, we could not observe any occurrence of a significant incidence to report. We believe that the use of a text editor with syntax highlighting and tabbing helped developers to cope with this issue.

4) Threats to validity: As being a case study with low control where the goal was to observe the practice, we mitigated undesired bias on the results by minimizing interference with the team while doing the task. Moreover, we discussed the goal of the study with the client and the team to make sure that their main goal was to complete the specification and test suite creation task regardless of making the CLARET approach work. Moreover, the use of CLARET should be assessed for bad effects on productivity and quality, being subject to replacement if needed. Finally, the results are very specific to the context of application and the team involved, presenting mostly valuable feedback for CLARET evolution and preliminary evidence of its practical application.

VI. Case Studies on MBT

To complement the evaluation described in Sections IV and V, and to better investigate the benefits and limitations of CLARET in practice, a series of case studies were performed in the context of a cooperation between our research lab and two different companies, Ingenico do Brasil Ltda and Viceri Solution Ltda[8]. CLARET was used in three industrial agile

[8]www.viceri.com.br

projects from different fields, SAFF, TCoM and BZC. Software engineers modeled the system requirements using CLARET and applied its tool set (Figure 1) to automatically generate their test suites. The SAFF project is an information system that manages status report of embedded devices; TCoM is a system that controls the execution and manages testing results of a series of hardware parts; and BZC is a system for optimizing e-commences logistic activities.

1) Procedure: Our case studies were conducted as discussed in Section III-C. For each project, engineers had several meetings with the clients during the Sprints; from initial meetings a requirements list was extracted. After that, the requirements were detailed iteratively at subsequent Sprints as use cases specifications using CLARET. By using the CLARET tool, engineers syntactically validated the specifications and generated their correspondent TGF models. These models were later inspected aiming at avoiding incorrect and/or incomplete specifications prior to release deployment. Finally, test suites were generated using the LTS-BT tool and registered in the TestLink environment. Prior to release dates, all test suites produced in the Sprints included in the release were run and their results analyzed in this study.

By the time we wrote this paper, the TCoM system had a single release version. While, due to a series of requirement changes, both the SAFF and BZC systems released two versions. As each version was individually specified and tested by using CLARET, we consider the results of each release version separately (SAFF 1, SAFF 2, BZC 1, BZC 2 - Table III).

2) Research Questions: [*RQ 1*] Do CLARET and the CLARET tool set enable requirements specification for RE and MBT? [*RQ 2*] Can test suites generated from CLARET use cases find system level defects?

3) Results and Discussion: Table III summarizes the number of created use cases per system and the size of the correspondent generated test suites.

Though requiring certain initial efforts for learning the CLARET syntax and how to use the supporting tools, the engineers did not have much trouble on using CLARET nor modeling requirements with it. No requirement could not be modeled by using our notation. Moreover, the support provided by the CLARET tool (syntax checking, TGF models generation) was greatly appreciated, since it provided a great help when validating the requirement artifacts. For instance, a requirement engineer mentioned: ‘‘*Since we work in a context with very volatile requirements, when requirement updates were needed, we used to go through all requirement and testing artifacts to check the impact of these changes and update them as well.* CLARET *sped up this process since now we just need to update the specification files and the changes automatically propagate throughout the other artifacts"*.

TABLE III
NUMBER OF USE CASES AND TEST CASES PER SYSTEM.

System	Number of Use Cases	Number of Test Cases
SAFF 1	17	56
SAFF 2	19	63
TCoM	12	32
BZC 1	23	95
BZC 2	55	283

Other practical benefit found by engineers was the use of the TGF models generated by CLARET. For instance, in several moments the requirement documents had to be revised due to inconsistencies found via TGF inspection. Moreover, an interesting fact to mention is that not only the CLARET specification files were used for guiding code development, but the TGF models as well. Developers found the TGF representation helpful to have a better overview of the system and to understand its main scenarios, which ended up guiding their coding efforts: *"By looking at the graph representation of the system I could visualize interaction of different flow paths that I did not realize before"*. While building models in agile development is avoided, having them can be handy.

Prior to releasing dates, software engineers led their teams to perform testing rounds. All test suites were automatically generated from CLARET artifacts and their execution was managed by using the TestLink tool. Since an extensive test suite would be impractical due to time restrictions, they applied a test generation strategy that creates an extensive test suite and reduced it without losing much testing power (this strategy is also provided by the LTS-BT tool). In our case studies, we worked with test suites that varied from 32 to 283 test cases (Table III). All test cases were manually run with an average running time of 5.21 minutes per test case.

Since test suites covering system-level scenarios could be generated from CLARET documents and applied by the team, we can answer our first research question (*RQ 1*) by stating that, in fact, CLARET and its tool set enable requirement specification as a central artifact for RE and testing. Our findings corroborate with the ones from sections IV and V.

Figure 4 summarizes the test execution results. For SAFF 1, SAFF 2, and TCoM most test cases passed (59%, 73% and 63%), while for the BZC versions the rates dropped to 34% and 31%. All found bugs were registered and later fixed by the teams. The great number of failing test cases for BZC was already expected since BZC is a project in its early stages and no systematic tests were conducted before the study using CLARET. However, it is interesting to notice that even for systems that had been tested before (SAFF and TCoM), new bugs were found by using CLARET.

In all five executions there were a few test cases that could not be run (blocked). Blocked test cases refer to specification changes that had been implemented to the code but had not been updated yet in the specification documents. This is a common challenge in agile development where changes are continuously embraced and effectiveness of change management procedures are key to reduce this rate of blocked test cases. This is target for further research.

As we can see, in all cases faults were found by using the test cases generated from the CLARET specification. In SAFF 1, 36% of all test cases failed because the system was not according to its specification. Slightly lower, but still high, rates

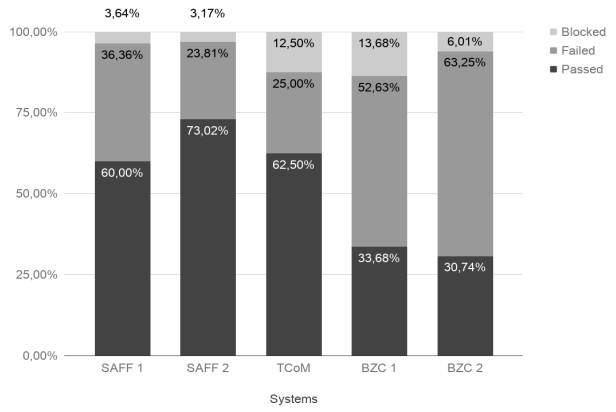

Fig. 4. Test execution results per system.

were found for the other two Ingenico systems (SAFF 2 - 24%, TCoM - 25%). In the BZC system the rate of failing test cases was quite high (53% and 63%). The found failing test cases refer to defects related to exception flows not treated by the developers, missing field scope validations, and/or unanticipated user interaction. All defects were registered as bugs and fixing patches were applied to new versions of the software. Thus, we can answer the second research question (*RQ 2*) by stating that suites created from CLARET artifacts are effective for testing, since real system level defects were identified. In fact, this testing phase was crucial since it helped to prevent the delivery of faulty versions of the software. Notice that we cannot state that all possible system level defects were detected since the study was conducted in a real development environment making it impractical to experiment alternative solutions to compare with.

Our case studies evidence the practical benefits of CLARET and the proposed tool set and process. By using CLARET, and its tool-set, requirement and software engineers were able to model all requirements from five versions of three industrial systems from different fields (SAFF, TCoM and BZC). Moreover, test suites created from specification artifacts were able to find defects in an apparently stable software. By using CLARET and the proposed methodology, the engineers created manageable and useful specification artifacts, and avoided extra costs of finding later defects.

Finally, the test case generation algorithm applied by the LTS-BT tool systematically captures all possible flows in the structure of the model without interpreting the semantics of the steps (natural language description) or adding information to the model. In this sense, the CLARET specification is the source from which meaningful information for the test cases is gathered. If use cases are not sufficiently described, test cases will probably have its execution blocked for the lack of information. Therefore, bugs could only be detected in the studies because the flow that originates the test case that fail is described in the specification and therefore could be generated by the LTS-BT tool and executed by the team.

VII. LESSONS LEARNED

The empirical studies presented in this paper focused on the application of the CLARET approach to address current challenges on the practice of RE and MBT in the agile development and evolution of industrial applications. Even though the results obtained are specific to our context of application, a number of important lessons have been learned that can be largely applied. We discuss them in this section.

a) Agile developers may not be skilled on writing requirements: One of the challenges to introduce the CLARET notation for requirements specification was the lack of sufficient experience of the participants on writing requirements, especially use cases. As RE is not an established practice in the agile community, developers may invest most of their training time on improving their programming and testing skills.

b) Writing use cases near release time is more effective: Since agile development may start with a very general list of requirements such as product backlog, in the first Sprints usually there is not enough information to write on a detailed requirements document. Even if there is information, the documents will be subject to continuous changes so that the team will end up not being able to cope with the update burden. As near a sprint is to release time, as more effective is to perform requirement detail writing in its scope.

c) Agile developers get engaged on writing requirements for automated test generation: With the value of automated test case generation, developers can get motivated to write requirements documents. Writing test cases manually is a hard task that present similar challenges to writing requirements. Moreover, requirements documents are more concise than individual test cases and get together nicely all facets of a functionality, providing a self-contained system overview.

d) Requirements documents are needed during test case execution: Different from test cases, RE documents provide a concise and self-contained view of the system under testing. Particularly when performing test execution manually, the tester is challenged to setup test execution and also deciding on test outcomes. Therefore, RE documents can be a valuable source of information during test execution.

e) Keeping RE documents updated requires considerable effort: Even with all benefits of having CLARET documents as a central artifact for RE and MBT and having it largely accepted by the development team and the client, there are still challenges to be faced to keep CLARET documents updated during systems evolution. As test suites are executed, it is necessary to keep execution history since this information may be used for test case selection. This requires tracing test cases from previous and current generated test suites. Also, a number of test cases get blocked because the implementation has been dropped out. So, it is important that the RE team get this feedback to update the documents accordingly.

VIII. RELATED WORK

Exploratory studies on the alignment of requirements and testing have already been presented in literature. For instance, Hotomski et al [14] performed a study based on interview on

current practices for managing requirements and acceptance testing documentation. They found that acceptance tests are usually written from requirements that are not detailed enough, acceptance tests are not regularly maintained and testers and technical are not involved in RE activities resulting in misunderstandings. Moreover, Bjarnson et al [4] performed a case study on agile requirements engineering by considering the use of test cases as requirements. They found that this practice poses challenges on RE activities such as tracing requirements to testing. These findings support the need to develop further studies on bringing closer requirements and testing activities.

Use case specifications in NL have been considered a suitable input for test case generation and automation. Sarmiento et al. [24] propose a test case generation approach from use cases expressed in a semi-structured NL. Also, Wang et al. [29] propose an approach for use case modeling devoted for system tests generation, named UMTG. For test generation, UMTG takes as input use case specifications and a domain model. To make test case automation possible, UMTG performs NL processing and constraint solving over a restricted form of use case specification. Moreover, Yue et al. [30] present RTCM, a test case generation framework based on NL that supports test case automation. Use cases are specified by following a template and a restricted NL with keywords for guiding test case generation. The framework is supported by a tool that performs transformations on models, including a specific test generator platform.

Formal specification notations have also been applied to complement NL use case specifications and/or interpret them in terms of models from which test cases can be generated and properties analyzed [20], [21]. On the other hand, CLARET is based solely on the textual specification of use cases using natural language to specify use case fields.

Gherkin[9] is a notation used in the Cucumber tool that is concerned with the creation of behavior-based test cases. Despite being based on user stories, the Gherkin notation induces a more detailed specification of behavior, while CLARET specifications are less controlled and focus on higher abstraction levels. Similarly the C&L tool [23] supports test case generation from natural language specifications. While it focuses on the use of natural language, it demands the existence of a domain specific language to support the creation of activity diagrams from which test cases are generated.

Our approach does not rely on the existence and/or creation of models as well as it poses less constraints on use case writing at the cost of not allowing for a deeper analysis of test cases beyond the network of flows and actors. The goal is to suit agile development where costs of detailed modeling are prohibitive, while manual test case execution is a common practice. Moreover, CLARET specifications are intended to be used as central artifact for MBT and RE. Apart from specification of actors and alternate/exception flows to suit test case generation, we do not use control words among the NL sentences. Furthermore, test cases are sequences of actions and expected results in NL along with NL pre and postconditions.

[9]https://github.com/cucumber/cucumber/wiki/Gherkin

IX. Concluding Remarks

This paper presents CLARET, a notation for specifying NL use case that can be used in both RE and MBT as part of agile development. With CLARET, requirement engineers are able to build artifacts that are easy to read and useful for system test case generation. Moreover, we introduce a tool set that enable a consistent and manageable integration of RE and MBT activities. With CLARET, we intend to reduce the burden of building requirement artifacts that can also be used for testing. Our user study and survey with professional developers have shown that CLARET is more expressive and creates specification artifacts that are easy to read and understand, when compared to free-written use cases. Moreover, industrial case studies show the practical benefits that CLARET brings regarding integration of RE and MBT. In the studies, RE documents were used as part of RE and MBT practices bringing benefits to agile development context and real faults were detected by using test cases generated from CLARET artifacts. Although results are specific to the context of investigation, this studies present practical evidence on the benefits and challenges of using of RE and MBT aligned with agile practices which has not yet been extensively reported in literature to the best of our knowledge.

As future work we plan to: i) extend our evaluation by having a larger number of participants using CLARET and/or collecting subjective measures regarding the quality of generated CLARET artifacts. This evaluation would help us to have stronger conclusions regarding the benefits of using CLARET and its process; ii) provide guidelines for building expressive CLARET artifacts. As the steps of CLARET use cases are described by using NL, their expressiveness might vary according to aspects such as the experience of who is writing them. Thus, we plan to conduct an investigation with a group of experienced professional requirement engineers and testers in which they would help us to define a set of guidelines to guide the definition of expressive and useful use case steps; and iii) conduct an investigation on the impact that requirement changes have on specification and testing artifacts. Currently, when a requirement changes, we need to update its correspondent CLARET use case and regenerate all test cases for this artifact. However, sometimes testing history is important and cannot be discarded. Thus, there is a need for a solution for reducing the number discarded test models in a MBT/Agile context.

Acknowledgements

This research is partially supported by cooperation between UFCG and two companies Viceri Solution LTDA and Ingenico do Brasil LTDA, the latter stimulated by the Brazilian Informatics Law n. 8.248, 1991. First author is supported by National Council for the Improvement of Higher Education (CAPES)/Brazil. Second author is supported by National Council for Scientific and Technological Development (CNPq)/Brazil (Process 311239/2017-0).

REFERENCES

[1] CLARET website. https://sites.google.com/a/computacao.ufcg.edu.br/claret/. Accessed: 2018-05-21.

[2] R. V. Binder, B. Legeard, and A. Kramer. Model-based testing: Where does it stand? *Queue*, 13(1):40:40–40:48, Dec. 2014.

[3] E. Bjarnason, P. Runeson, M. Borg, M. Unterkalmsteiner, E. Engström, B. Regnell, G. Sabaliauskaite, A. Loconsole, T. Gorschek, and R. Feldt. Challenges and practices in aligning requirements with verification and validation: A case study of six companies. *Empirical Softw. Engg.*, 19(6):1809–1855, Dec. 2014.

[4] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström. A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology*, 77:61 – 79, 2016.

[5] E. G. Cartaxo, W. L. Andrade, F. G. O. Neto, and P. D. L. Machado. LTS-BT: A tool to generate and select functional test cases for embedded systems. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1540–1544, New York, NY, USA, 2008. ACM.

[6] T. Chow and D.-B. Cao. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6):961 – 971, 2008. Agile Product Line Engineering.

[7] K. Curcio, T. Navarro, A. Malucelli, and S. Reinehr. Requirements engineering: A systematic mapping study in agile software development. *Journal of Systems and Software*, 139:32 – 50, 2018.

[8] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos. A survey on model-based testing approaches: A systematic review. In *Proc. of the 1st ACM Int. Work. on Emp. Assessment of Soft. Eng. Languages and Technologies*, pages 31–36, 2007.

[9] I. K. El-Far and J. A. Whittaker. Model-based software testing. In *Encyclopedia on Software Engineering*. Wiley-Interscience, 2001.

[10] E. Farchi, A. Hartman, and S. Pinter. Using a model-based test generator to test for standard conformance. *IBM Systems Journal*, 41(1):89–110, 2002.

[11] W. Grieskamp, N. Kicillof, K. Stobie, and V. Braberman. Model-based quality assurance of protocol documentation: Tools and methodology. *Softw. Test. Verif. Reliab.*, 21(1):55–71, Mar. 2011.

[12] B. Hois, S. Sobernig, and M. Strembeck. Natural-language scenario descriptions for testing core language models of domain-specific languages. In *Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on*, pages 356–367. IEEE, 2014.

[13] B. Hoisl, S. Sobernig, and M. Strembeck. Comparing three notations for defining scenario-based model tests: A controlled experiment. In *Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the*, pages 95–104. IEEE, 2014.

[14] S. Hotomski, E. B. Charrada, and M. Glinz. An exploratory study on handling requirements and acceptance test documentation in industry. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 116–125, Sept 2016.

[22] J. Peleska, A. Honisch, F. Lapschies, H. Löding, H. Schmid, P. Smuda, E. Vorobev, and C. Zahlten. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In *Testing Software and Systems*, pages 146–161, 2011.

[15] I. Inayat, L. Moraes, M. Daneva, and S. S. Salim. A reflection on agile requirements engineering: Solutions brought and challenges posed. In *Scientific Workshop Proceedings of the XP2015*, XP '15 workshops, pages 6:1–6:7, New York, NY, USA, 2015. ACM.

[16] ISO/IEC 14977:1996. *Information technology – Syntactic metalanguage – Extended BNF*. ISO/IEC Copyright Office, Geneva, 1996.

[17] D. N. Jorge, W. L. Andrade, P. D. L. Machado, and E. L. G. Alves. Claret - central artifact for requirements engineering and model-based testing. In *Brazilian Conference on Software (CBSoft Tools)*, 2017.

[18] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez. The epsilon book. *Structure*, 178:1–10, 2010.

[19] A. Marques, F. Ramalho, and W. L. Andrade. Comparing model-based testing with traditional testing strategies: An empirical study. In *IEEE ICST Workshops/AMOST 2014*, 2014.

[20] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jezequel. Automatic test generation: a use case driven approach. *Software Engineering, IEEE Transactions on*, 32(3):140–155, March 2006.

[21] S. Nogueira, A. Sampaio, and A. Mota. Test generation from state based use case models. *Formal Aspects of Computing*, 26(3):441–490, 2014.

[23] E. Sarmiento, J. C. S. D. P. Leite, and E. Almentero. C&L: Generating model based test cases from natural language requirements descriptions. *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, pages 32–38, 2014.

[24] E. Sarmiento, J. Sampaio do Prado Leite, and E. Almentero. C amp;l: Generating model based test cases from natural language requirements descriptions. In *Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on*, pages 32–38, Aug 2014.

[25] S. Sobernig, B. Hoisl, and M. Strembeck. Requirements-driven testing of domain-specific core language models using scenarios. In *Quality Software (QSIC), 2013 13th International Conference on*, pages 163–172. IEEE, 2013.

[26] The Eclipse Foundation. Open Up Checklist on Use Case Completeness and Consistency. http://www.eclipse.org/downloads/download.php?file=/technology/epf/OpenUP/published/openup_published_1.5.1.5_20121212.zip. Accessed: 2018-05-21. From main index, follow Team tab, Guidance, Checklists, Use Case.

[27] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kauffman, first edition, 2007.

[28] B. Walter, J. Hammes, M. Piechotta, and S. Rudolph. A formalization method to process structured natural language to logic expressions to detect redundant specification and test statements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 263–272, Sept 2017.

[29] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal. Automatic generation of system test cases from use case specifications. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 385–396, New York, NY, USA, 2015. ACM.

[30] T. Yue, S. Ali, and M. Zhang. Rtcm: A natural language based, automated, and practical test case generation framework. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 397–408, New York, NY, USA, 2015. ACM.