Definition of a behavior-driven model for requirements specification and testing of interactive systems

Thiago Rocha Silva
ICS-IRIT
Université Paul Sabatier – Toulouse III
Toulouse, France
rocha@irit.fr

***Abstract* — In a user-centered development process, artifacts are aimed to evolve in iterative cycles until they meet users' requirements and then become the final product. Every cycle gives the opportunity to revise the design and to introduce new requirements which might affect the specification of artifacts that have been set in former development phases. Testing the consistency of multiple artifacts used to develop interactive systems every time that a new requirement is introduced it is a cumbersome and time consuming activity, especially if it should be done manually. For that we propose an approach based on Behavior-Driven Development (BDD) to support the automated assessment of artifacts along the development process of interactive systems. In order to prevent that test should be written to every type of artifact, we investigate the use of ontologies for specifying the test once and then run it in all artifacts sharing the ontological concepts.**

***Index Terms* — Automated Requirements Checking, Behavior-Driven Development, Ontological Modeling, Prototyping, Multi-Artifact Testing.**

I. Introduction and Motivation

When designing new software systems, clients and users are keen to introduce new requirements along successive iterations. This characteristic has an impact in the future development as well as in previously developed artifacts. Requirements should be tested and verified against not only the software already produced, but also against the other permanent artifacts produced throughout the process. It leads us to a cycle of permanent production of multiple artifacts, in multiple versions, evolving all along of multiple phases of development.

The artifacts traceability problem has been studied by several authors and a wide set of commercial tools have been developed to address this problem in various approaches [3]. Nonetheless, solutions to promote vertical traceability of artifacts can simply track them among themselves, not allowing to effectively testing them against requirements specifications. It is a peaceful argument that testing plays a crucial role in the quality of the software under development. Moreover, the sooner the teams pay attention to test their software components and especially their requirements specifications, more effective will be the results towards a quality assurance of the product.

Lindstrom [5] declared that failure to trace tests to requirements is one of the five most effective ways to destroy a project. According to Uusitalo et al. [6], traceability between requirements and tests was rarely maintained in practice. This was caused primarily by failure to update traces when requirements change, due to stringent enforcement of schedules and budgets, as well as difficulties to conduct testing processes through a manual approach. In most cases, interviewees in industry longed for better tool support for traceability. Some also noted that poor quality of requirements was a hindrance to maintaining the traces, since there is no guarantee how well the requirements covered the actual functionality of the product.

In this context, Behavior Driven Development (BDD) [4] has aroused interest from both academic and industrial communities in the last years. Supported by a wide development philosophy that includes Acceptance Test-Driven Development (ATDD) [7] and Specification by Example [8], BDD drives development teams to a requirements specification based on User Stories [9] in a comprehensive natural language format. This format allows specify executable requirements, which mean we can test our requirements specification directly, conducting to a "live" documentation and making easier for the clients to set their final acceptance tests. It guides the system development and brings the opportunity to test Scenarios directly in the User Interface (UI) with the aid of external frameworks for different platforms.

However, this technique is currently limited and allows us to test requirements only against a Final User Interface (Final UI), using software robots that those external frameworks provide. Besides that, specifications using only Scenarios are not self-sufficient to provide a concrete perception of the system to the users and, at the same time, allow an overall description of the system in terms of tasks that may be accomplished. For that, the use of Prototypes and Task Models is well accepted as a good approach to address User-Centered Design (UCD), providing a concrete perception of the system under development and allowing an overall description of the tasks in executable Scenarios.

Moreover, domain ontologies are an effective means to represent concepts and relationships when integrating all of these techniques and approaches in a formal model. According to Gruber [10], ontologies describe concepts, relationships and behaviors between elements in a given domain. In the context of interactive systems development, we are studying the use of ontologies to create a flexible and reusable model that could support the description of an extensive set of artifacts, as well as their representations and behaviors for testing purposes.

II. Research questions and hypotheses

In this thesis work, we are trying to answer two main research questions: (i) "Ontologies are useful to support the development of testable User Stories?" and (ii) "Are User Stories self-enough to support the multi-artifact testing process?".

Based on these questions, we have two main hypotheses: (i) the use of a common ontological model makes easier the reuse of behaviors for the testing of interactive systems, and (ii) requirements expressed as User Stories can effectively support automated testing in a wide spectrum of artifacts, assuring traceability and consistency.

To answer these questions and to verify our hypotheses, we study in this thesis a new ontological perspective for Behavior-Driven Development (BDD) to describe requirements in a Scenario-based approach [11], aiming multi-artifact testing since early in the design process. This approach aims to address the challenge of testing different artifacts throughout the development process of interactive systems, checking their correct correspondence with requirements, thus promoting as a consequence vertical and bidirectional traceability in the artifact level. To achieve this goal, a formal ontology model is provided to describe concepts used by platforms, models and artifacts that compose the design of interactive systems, allowing a wide description of interaction elements (and its behaviors) to support testing activities.

III. Technical challenges

A first challenge in the thesis assumption is that requirements are dispersed in multiple artifacts that describe them in different levels of abstraction. Thus, tests should run not only in the final product, but also in the whole set of artifacts to assure that they represent the same information in a non-ambiguous way, and in accordance with the whole requirements chain. A big challenge in this case is how to verify and check these artifacts, and mainly how to assure correctness and consistency between them and the other components of the requirements specification.

Another big challenge for testing is that requirements are not stable along the iterative processes of software development. Clients and Users introduce new demands or modify the existing ones all along the iterations and because of that, Regression Testing become crucial to assure that the system remains behaving properly and in accordance with the new requirements introduced. However, manual Regression Tests are extremely time consuming and highly error-prone. Therefore, promoting automated tests is a key factor to support testing in an ever-changing environment, allowing a secure check of requirements and promoting a high availability of testing.

A third challenge is that, despite being very profitable providing the testing component for requirements, Scenarios identified from BDD approaches in the Software Engineering processes become very cumbersome when specifying the whole set of cases in which the system is able to run. On the other hand, Scenarios extracted from Task Models in the UCD processes do not provide the testing component which requires a heavy charge of effort to implement automated tests. Thus, the challenge is how to use a combination of both practices to provide a practical method to extract useful and testable Scenarios as well as bringing the testing component for requirements specifications.

In short, these concerns bring us three main challenges: (i) formalize requirements in order to provide testability in an automated approach for multiple artifacts in ever-changing environments; (ii) provide vertical and bidirectional traceability of the requirements, ensuring reliability and consistency between artifacts; and (iii) assure a complete and testable ontological description of the requirements artifacts to support automated testing in an integrated way.

IV. State of the art

A. User Stories and Scenarios

User Stories have a large meaning in the literature. The Human-Computer Interaction community understands this concept as stories that users tell to describe their activities and jobs during typical requirements meetings. This concept of User Stories is close to the concept of Scenarios given by Rosson & Carroll [11] and widely used in UCD design. According to Lewis & Rieman [12], Scenario spells out what a user would have to do and what he or she would see step-by-step in performing a task using a given system. The key distinction between a scenario and a task is that a scenario is design-specific, in that it shows how a task would be performed if you adopt a particular design, while the task itself is design-independent, i.e., it is something the user wants to do regardless of what design is chosen. Given task models have already been developed, scenarios can also be extracted from them to provide the executable and possible paths in the system [14].

In the Software Engineering (SE) side, User Stories are typically used to describe requirements in agile projects. This technique was proposed by Cohn [9] and provides in the same artifact a Narrative, briefly describing a feature in the business point of view, and a set of Scenarios to give details about business rules and to be used as Acceptance Criteria, giving concrete examples about what should be tested to consider a given feature as "done". This kind of description handles a Behavior-Driven Development (BDD) assumption [4], in which the system is developed under a behavior perspective in the user point of view. This method assures for clients and teams a semi-structured natural language description, in a non-ambiguous way (because it is supported by test cases), in addition to promote the reuse of business behaviors that can be shared for multiple features in the system.

As we can realize, the approaches for Scenarios from UCD and SE share the same concept. Both of them provide a step-by-step description of tasks being performed by users using a given system. The main difference between them lies in the testing and the business value components present in the SE approach. Scenarios from UCD, despite describing events that a given system can answer, do not describe the expected behavior from the system when those events are triggered, besides not determine the business motivation to develop the feature being described. TABLE I. summarizes these characteristics.

TABLE I. APPROACHES FOR DESCRIBING USER STORIES AND SCENARIOS

Approaches for User Stories and Scenarios	Key facts	Advantages	Shortcomings
User Stories and/or Scenarios by Rosson & Carroll [11]	Informal description of user activities contextualize d in a story.	Highly flexible and easily comprehensive for non-technical stakeholders.	Very hard to formalize, little evolutionary and low reusability.
User Stories and/or Scenarios by Cohn [9] and North [13]	Semi-formal description of user tasks being performed in an interactive system.	Highly testable and easily comprehensive for non-technical stakeholders.	Very descriptive and time consuming to produce.
Scenarios extracted from Task Models by Santoro [14]	Possible instances of execution for a given path in a task model.	Highly traceable for task models.	Dependency of task models and low testability.

In this thesis, we are interested in providing testing for the Functional aspects of interactive systems in the Acceptance level. Functional Testing identifies situations that should be tested to assure the appropriate behavior of the system under development in accordance with the requirements previously specified. The Acceptance Level makes reference to the tests made under the client/user point of view to validate the right behavior of the system. At this level, clients might be able to run their business workflows and to check if the system behaves in an appropriate manner. Considering these testing concerns and taking into account that the presented approaches do not solve the problem by themselves, a possible solution might address a combination of them.

B. Computational Ontologies

According to Guarino et al. [15], computational ontologies are a means to formally model the structure of a system, i.e., the relevant entities and relations that emerge from its observation, and which are useful to our purposes. Some approaches such as DOLPHIN [16], UsiXML [17] and W3C MBUI Glossary [18] have tried to define a common vocabulary for specific domains, although have not formalized it through a conventional ontology. According to the authors, DOLPHIN [16] is a software architecture that attempts to solve the problem of multiple definitions in the task modeling domain. The authors claim that multiple versions and expressions of task models used in user interface design, specification, and verification of interactive systems have led to an ontological problem of identifying and understanding concepts which are similar or different across models. This variety raises a particular problem in model-based approaches for designing user interfaces as different task models, possibly with different vocabularies, different formalisms, different concepts are exploited. The argument is there was not software tool able to accommodate any task models as input for a user-centered design process.

In a broader spectrum, UsiXML (which stands for USer Interface eXtensible Markup Language) [17] is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. UsiXML consists of a User Interface Description Language (UIDL) that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics. UsiXML describes at a high level of abstraction the constituting elements of the UI of an application: widgets, controls, containers, modalities and interaction techniques. More recently, W3C has published a glossary of recurrent terms in the Model-based User Interface domain (MBUI) [18]. It was intended to capture a common, coherent terminology for specifications and to provide a concise reference of domain terms for the interested audience. The authors' initial focus was on task models, UI components and integrity constraints at a level of abstraction independent of the choice of devices to implement the models.

The problem with these attempts to define concepts and relationships is they are incomplete and do not formalize an ontology model to be reused and adapted to other domains. In addition to that, they do not provide the testing component to directly support tests in the requirements artifacts.

C. Related Works

Requirements specified through an ATDD approach are relatively recent in academic discussions. Efforts to specify requirements in a natural language perspective are not so recent though. Language Extended Lexicon (LEL) [19] has studied this theme since the beginning of 90's. The authors propose a lexicon analysis in requirements descriptions aiming integrate scenarios into a requirements baseline, making possible their evolution as well as the traceability of the different views of the requirements baseline. The main focus is in using natural language descriptions to help the elicitation and modeling of requirements.

Soeken et al. [20] go in the direction of system design from a requirements specification provided in BDD. The authors propose a design flow where the designer enters into a dialog with the computer. In an interactive manner, a program processes sentence by sentence from the requirements specification and suggests creating code blocks such as classes, attributes, and operations. The designer can then accept or refuse these suggestions. Furthermore, the suggestions by the computer can be revised which leads to a training of the computer program and a better understanding of following sentences. Those works [19] and [20] use different approaches to process natural language; nonetheless do not consider constraints related to User-Centered Design (UCD) specifications.

Wolff et al. [21] discuss an approach for linking GUI specifications to more abstract dialogue models, supporting an evolutionary design process. These specifications are linked to task models describing behavioral characteristics. With this approach, prototypes of interactive systems are interactively generated, and then refined specifications are automatically generated using a GUI editor, which allows replacing of user interface elements by other elements or components. The authors present a design cycle from task model to abstract user interfaces and finally to a concrete user interface. It is an interesting approach to have a mechanism to control changes in interface elements according to the task they are associated in the task models. The approach however is limited, being applied only in the evolutionary process of UI elements in accordance to their representation in the task models. Apart from being applicable in a limited context, this approach does not provide the necessary testing component to check and verify user interfaces against predefined behaviors from requirements.

Martinie et al. [22] propose a tool-supported framework for exploiting task models throughout the development process and even when the interactive application is deployed and used. To this end, they introduce a framework for connecting task models to an existing, executable, interactive application. According to the authors, the main contribution of the paper lies in the definition of a systematic correspondence between the user interface elements of the interactive application and the low level tasks in the task model in a tool-supported way. This task-application integration allows the exploitation of task models at run time on interactive application. The problem with this approach is that it only covers the interaction of task models with Final UIs, not covering other types of possible requirements artifacts that can emerge along the process. It does not even indicate how other set of artifacts could be supported. Another problem is it requires much intervention of developers to prepare the code to support the integration, making difficult to adopt in applications that cannot receive interventions in the code level.

Buchmann & Karagiannis [23] present a modelling method aimed to support the definition and elicitation of requirements for mobile apps through an approach that enables semantic traceability for the requirements representation. According to the authors, instead of having requirements represented as natural language items that are documented by diagrammatic models, the communication channels are switched: semantically interlinked conceptual models become the requirements representation, while free text can be used for requirements annotations/metadata. The work is oriented to provide support for requirements representation by means of a knowledge-orientation. The authors claim that the method can support semantic traceability in scenarios of human-based requirements validation, but using an extremely heavy modeling approach which it is not suitable to check requirements in a high level of abstraction. Besides that, the method is not focused in providing a testing mechanism through common artifacts, but only in validating the requirements modeled within the approach.

Finally, Käpyaho & Kauppinen [24] describe a case study to explore how prototyping can solve the challenges of requirements in an agile context. Authors' findings indicate that prototyping can help with some challenges of agile requirements such as lack of documentation and motivation as well as poor quality communication, but it also needs complementary practices to reach its full potential. These practices include using ATDD (Acceptance Test-Driven Development), among other ones. The authors conclude that one of the biggest benefits from prototyping is that the prototypes act as tangible plans that can be relied on when discussing changes. Prototypes also seem to improve motivation to do requirements work as they force participants to discuss changes to requirements more concretely.

These findings point initially towards a gap integrating different requirements artifacts throughout a design process. Some methods address concerns in scenarios descriptions, other ones in prototype or task modeling, however none of them solve the problem of multi-artifacts integration in order to provide means to test them, assuring correctness and consistency along the development.

V. Research Methods and Evaluation

Research methods for this thesis were initially based on literature reviews and observations in the industry to establish the thesis scope. Based on the findings, we are proposing an approach to address the stated problem. This approach is planned to be validated following empirical methods to assure its adherence to the problem statement. To check the results against our hypothesis, we envision 3 main validations:

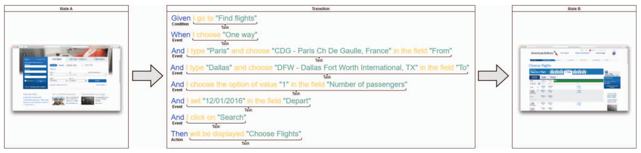
- (i) through a case study to evaluate how effective is reusing behaviors described in the ontology to test an interactive system;
- (ii) through a case study to evaluate the User Stories support for testing Task Models, Prototypes and Final User Interfaces; and
- (iii) through a controlled experiment aiming to verify the effectiveness and the workload of the approach when providing multiple design solutions and testing a predefined set of artifacts and requirements.

This strategy aims to cover the more frequent set of artifacts used to build interactive systems: User Stories and Scenarios, Prototypes, Task Models and Final User Interfaces. The case studies are planned to be conducted for the Web and Mobile environments whilst the experiments are planned to be conducted in laboratory with real requirements collected in the industry.

VI. Contributions

A. Definition of an Ontology

We have started defining an OWL ontology for Web and Mobile platforms and associating the most common behaviors that each UI element in these environments can answer. These behaviors are being described using a natural language convention, useful later to specify Steps of Scenarios to set actions in these elements. For that, we have started modeling concepts describing the structure of User Stories, Tasks and Scenarios. Following this, we have modeled the most common Interaction Elements used to build Prototypes and Final User Interfaces (FUIs) in the Web and Mobile environments. The dialog component that allows us to add dynamic behavior to Prototypes and navigation to FUIs was modeled as a State Machine. In this level, a Scenario that runs on a given interface is represented as a Transition in the machine, while the interface itself and the other one resultant of the action were represented as States. Scenarios in the Transition state have always at least one or more Conditions (represented by the "Given" clause), one or more Events (represented by the "When" clause), and one or more Actions (represented by the "Then" clause). These elements always trigger instances of tasks that are represented as the Steps of Scenarios.



```
Given I go to "Find flights"
When I choose "One way"
And I type "Paris" and choose "CDG - Paris Ch De Gaulle, France" in the field "From"
And I type "Dallas" and choose "DFW - Dallas Fort Worth International, TX" in the field "To"
And I choose the option of value "1" in the field "Number of passengers"
And I set "12/01/2016" in the field "Depart"
And I click on "Search"
Then will be displayed "Choose Flights"
```

Fig. 1. State Machine representing a Scenario transition

The ontological model describes only behaviors that report Steps performing common actions directly in the User Interface through Interaction Elements. We call it Common Steps. This is a powerful resource because it allows us to keep the ontological model domain-free, which means they are not subject to particular business characteristics in the User Stories, instigating the reuse of Steps in multiple Scenarios. Steps might be easily reused to build different behaviors in different Scenarios. Specific business behaviors should be specified only for the systems they make reference, not affecting the whole ontology.

Technically and with this structure, the current version of the ontology bears an amount of 422 axioms, being 276 logical axioms, 56 classes, 33 object properties, 17 data properties and 3 individuals. The ontology could be extended in the future to support behaviors for other environments or platforms.

B. User Stories Modeling

The Fig. 2. presents the conceptual model that explains how testable requirements are formalized in the ontology. A requirement is expressed as a set of User Stories (US) as in the template proposed by Cohn [9] and North [13]:

```
Title (one line describing the story)
Narrative:
As a [role]
I want [feature]
So that [benefit]
Acceptance Criteria: (presented as Scenarios)
Scenario 1: Title
Given [context]
 And [some more context]...
When [event]
Then [outcome]
 And [another outcome]...
Scenario 2: ...
```

User Stories are composed by a Narrative and a set of Acceptance Criteria. Acceptance Criteria are presented as Scenarios and these last ones are composed by at least three main Steps ("Given", "When" and "Then") that represent Behaviors which the system can answer. Behaviors handle actions on Interaction Elements in the User Interface (UI) and can also mention examples of data that are suitable to test them. Notice that these concepts are part of the ontology described in the previous section.

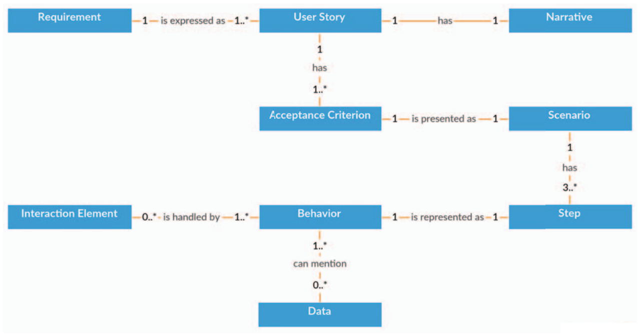
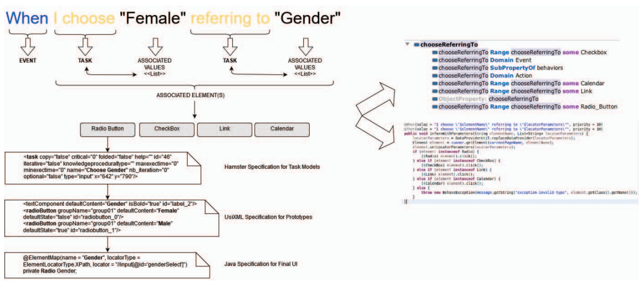


Fig. 2. Conceptual Model for testable requirements

C. Multi-Artifact Testing

Fig. 3. gives a general view of how testing integration can occur in multiple artifacts, given an example of behavior. In the top of the figure is presented an example of a Step of Scenario describing the behavior "*choose ... referring to ...*". In the example, a user is choosing the gender "Female" on the UI element "Gender" in a form. This task is triggered when an event "When" occurs in the Scenario. To be tested, this task is associated to values for data ("Female") and UI element ("Gender"), indicating a possible and executable Scenario that can be extracted from that task. Following the ontology, the behavior addressed by this task can be associated to multiple UI elements such as Radio Button, Check Box, Link and Calendar components. The arrows in the right side of the figure indicate two implementations of this ontology, highlighting these associations. First in an OWL version at the top and then converted in Java code in the bottom. Considering that the UI element Radio Button has been chosen to attend this behavior, a locator is triggered to trace this element throughout the artifacts, thus allowing us to reach it for testing purposes. The figure shows this trace being made through a HAMSTERS Specification for Task Models [22] (in the task "Choose Gender"), through a UsiXML Specification for Prototypes [17] (Radio Button "Gender" with the data options "Male" and "Female"), and finally through a Java Specification for Final UIs (@ElementMap "Gender" with the XPath reference "//input[@id='genderSelect']").


When I choose "Female" referring to "Gender"
EVENT
TASK
ASSOCIATED VALUES <<List>>
TASK
ASSOCIATED VALUES <<List>>
ASSOCIATED ELEMENT(S)
Radio Button
CheckBox
Link
Calendar
Hamster Specification for Task Models
UsiXML Specification for Prototypes
Java Specification for Final UI

```
<task copy="false" critical="0" folded="false" help="" id="46"
iterative="false" knowledgeproceduraltype="" maxexectime="0"
minexectime="0" name="Choose Gender" nb_iteration="0"
optional="false" type="input" x="642" y="790"/>
```

```
<textComponent defaultContent="Gender" isBold="true" id="label_2"/>
<radioButton groupName="group01" defaultContent="Female"
defaultState="false" id="radiobutton_0"/>
<radioButton groupName="group01" defaultContent="Male"
defaultState="true" id="radiobutton_1"/>
```

```
@ElementMap(name = "Gender", locatorType =
ElementLocatorType.XPath, locator = "//input[@id='genderSelect']")
private Radio Gender;
```

```
chooseReferringTo
    chooseReferringTo Range chooseReferringTo some Checkbox
    chooseReferringTo Domain Event
    chooseReferringTo SubPropertyOf behaviors
    chooseReferringTo Domain Action
    chooseReferringTo Range chooseReferringTo some Calendar
    chooseReferringTo Range chooseReferringTo some Link
    ObjectProperty: chooseReferringTo
    chooseReferringTo Range chooseReferringTo some Radio_Button
```

```
@When(value = "I choose \"$elementName\" referring to \"$locatorParameters\"", priority = 10)
@Then(value = "I choose \"$elementName\" referring to \"$locatorParameters\"", priority = 10)
public void informWithParameters(String elementName, List<String> locatorParameters) {
    locatorParameters = DataProviderUtil.replaceDataProvider(locatorParameters);
    Element element = runner.getElement(currentPageName, elementName);
    element.setLocatorParameters(locatorParameters);
    if (element instanceof Radio) {
        ((Radio) element).click();
    } else if (element instanceof CheckBox) {
        ((CheckBox) element).click();
    } else if (element instanceof Link) {
        ((Link) element).click();
    } else if (element instanceof Calendar) {
        ((Calendar) element).click();
    } else {
        throw new BehaveException(message.getString("exception-invalid-type", element.getClass().getName()));
    }
}
```

Fig. 3. Identifying behaviors through multiple artifacts

Tools like Webdriver, JBehave and JUnit can therefore be used to conduct the testing automation, running directly in the artifacts that compose the requirements specification, validating them and keeping the trace between themselves, Scenarios in the User Stories and the instantiated ontology, which leads to a genuine and "live" documentation.

VII. Progress

We have started this thesis by making a large systematic review in the literature about prototyping and tools that support this activity. It has been made to explore the state of the art in this theme, searching mainly for solutions that other works have already given for processing Scenarios in the Prototyping context, and eventually for the problem of testing Prototypes and Final UIs in an evolutionary perspective. Part of this work has been published in Silva et al. [1] and the final results have been submitted as a survey for publication in a journal.

In a second moment and based in our findings, we started working on the ontology and on the core mechanism to address the problem of promoting the testing component for multiple artifacts. The first ideas were published in Silva & Winckler [2]. Afterward, we started working on applying the initial proposal for Prototypes, Task Models and Final UIs as primary artifacts. The results of this work have been submitted for publication in a conference and in a journal.

Ongoing work is currently being conducted to verify potential problems and inconsistencies when working with multiple design options and complex task models. We are also developing a tool to support the creation, visualization and execution of the tests. Next steps include establish the case studies and experiments planned to validate the proposed approach.

VIII. Accepted publications

[1] T. R. Silva, J. L. Hak, and M. Winckler. "A Review of Milestones in the History of GUI Prototyping Tools." INTERACT 2015 Adjunct Proceedings: 15th IFIP TC. 13 International Conference on Human-Computer Interaction, Bamberg, Germany, 2015.

[2] T. R. Silva, and M. A. A. Winckler. "Towards automated requirements checking throughout development processes of interactive systems." Joint Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Gothenburg, Sweden, 2016.

References

[3] S. Nair, J. L. de la Vara, and S. Satyaki. "A review of traceability research at the requirements engineering conference re@ 21". IEEE International Requirements Engineering Conference (RE), p. 222-229, 2013.

[4] D. Chelimsky et al. "The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends". Pragmatic Bookshelf, 2010.

[5] R. D. Lindstrom. "Five ways to destroy a development project (software development)". IEEE Software, p. 55-58, 1993.

[6] J. E. Uusitalo et al. "Linking requirements and testing in practice". IEEE International Requirements Engineering Conference, p. 265-270, 2008.

[7] K. Pugh. "Lean-Agile Acceptance Test-Driven Development". Pearson Education, 2010.

[8] G. Adzic. "Specification by Example: How Successful Teams Deliver the Right Software". Manning Publications, 2011.

[9] M. Cohn. "User stories applied: For agile software development". Addison-Wesley Professional, 2004.

[10] T. Gruber. "A Translational Approach to Portable Ontologies". Knowledge Acquisition 5.2, p. 199-229, 1993.

[11] M. B. Rosson, and J. M. Carroll. "Usability engineering: scenario-based development of human-computer interaction". Morgan Kaufmann, 2002.

[12] C. Lewis and J. Rieman. "Task-centered user interface design: A Practical Introduction". 1993.

[13] D. North. "What's in a story?". http://dannorth.net/whats-in-a-story/, Accessed: Jun. 2016.

[14] C. Santoro. "A Task Model-based Approach for Design and Evaluation of Innovative User Interfaces". Presses univ. de Louvain, 2005.

[15] N. Guarino, D. Oberle, and S. Staab. "What is an ontology?". Handbook on ontologies, Springer Berlin Heidelberg, p. 1-17, 2009.

[16] Q. Limbourg, C. Pribeanu, and J. Vanderdonckt. "Towards uniformed task models in a model-based approach". Interactive Systems: Design, Specification, and Verification, Springer Berlin Heidelberg, p. 164-182, 2001.

[17] Q. Limbourg et al. "USIXML: a language supporting multi-path development of user interfaces". EHCI/DS-VIS 3425, p. 200-220, 2004.

[18] J. Pullmann. "MBUI - Glossary - W3C". https://www.w3.org/TR/mbui-glossary/, Fraunhofer FIT, Accessed: Jun. 2016.

[19] J. C. S. do Prado Leite et al. "Enhancing a requirements baseline with scenarios". Requirements Engineering 2.4, p. 184-198, 1997.

[20] M. Soeken, R. Wille, R. Drechsler. "Assisted behavior driven development using natural language processing". International Conference on Modelling, Techniques and Tools for Computer Performance Evaluation. Springer Berlin Heidelberg, p. 269-287, 2012.

[21] A. Wolff et al. "Linking GUI elements to tasks: supporting an evolutionary design process". Proceedings of the International Workshop on Task models and diagrams. ACM, p. 27-34, 2005.

[22] C. Martinie et al. "A generic tool-supported framework for coupling task models and interactive applications". Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, p. 244-253, 2015.

[23] R. A. Buchmann, and D. Karagiannis. "Modelling mobile app requirements for semantic traceability". Requirements Engineering, p. 1-35, 2015.

[24] M. Kapyaho, and M. Kauppinen. "Agile requirements engineering with prototyping: A case study". IEEE 23rd International Requirements Engineering Conference (RE), p. 334-343, 2015.