# Extending Behavior-Driven Development for Assessing User Interface Design Artifacts

Thiago Rocha Silva, Marco Winckler, Hallvard Trætteberg

# Extending Behavior-Driven Development for Assessing User Interface Design Artifacts

Thiago Rocha Silva
Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Norway
*thiago.silva@ntnu.no*

Marco Winckler
SPARKS-i3S,
Université Nice Sophia Antipolis (Polytech), France
*winckler@unice.fr*

Hallvard Trætteberg
Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Norway
*hal@ntnu.no*

***Abstract* — This paper presents a scenario-based approach to specify requirements and tests by extending Behavior-Driven Development (BDD) with the aim of ensuring the consistency between user requirements and user interface design artifacts. The approach has been evaluated by exploiting user requirements specified by a group of potential Product Owners (POs) for a web system to book business trips. Such requirements gave rise to a set of User Stories that have been refined and used to automatically check the consistency of task models, user interface (UI) prototypes, and final UIs of the system. The results have shown our approach was able to identify different types of inconsistencies in the set of analyzed artifacts and consistently keep the semantic traces between them.**

***Index Terms* — Behavior-Driven Development (BDD); User Interface Design Artifacts; Automated Requirements Assessment.**

## I. INTRODUCTION

Modeling is recognized as a crucial activity to manage the abstraction and the inherent complexity of developing software systems. As a consequence, software systems tend to be designed based on several requirements artifacts which model different aspects and different points of view about the system. Considering that different phases of development require distinct information, resultant artifacts from modeling tend to be very diverse throughout the development, and ensuring their consistency is quite challenging [1]. To face this challenge, extra effort should be put on getting requirements described in a consistent way across the multiple artifacts. Requirements specifications should not, for example, describe a given requirement in a user interface (UI) prototype which is conflicting with its representation in a task model.

Behavior-Driven Development (BDD) [2] has aroused interest from both academic and industrial communities as a method allowing specifying testable user requirements in natural language using a single textual artifact. BDD describes User Stories (US) [3] and scenarios in a easily understandable way for both technical and non-technical stakeholders. In addition, BDD scenarios allow specifying "executable requirements", i.e. requirements that can be directly tested from their textual specification. Despite providing support to automated testing of user requirements, BDD and other testing approaches essentially focus on assessing fully interactive artifacts such as full-fledged (final) versions of user interfaces. Automated assessment of model-based artifacts such as task models, UI prototypes, etc. is not supported.

Motivated by such a gap, we have researched and developed an approach based on BDD and User Stories to support the specification and the automated assessment of functional aspects of user requirements on user interface design artifacts such as task models, UI prototypes, and final UIs [4]–[8]. This paper presents a refined version of this approach and summarizes the new results we got in a case study exploiting User Stories specified by potential Product Owners (POs) to automatically assess user interface design artifacts for a web system to book business trips. The following sections present the foundations of this work as well as the refined version of our approach and a brief discussion of the results obtained with this case study.

## II. FOUNDATIONS

### A. Behavior-Driven Development (BDD)

According to Smart [9], BDD is a set of software engineering practices designed to help teams focus their efforts on identifying, understanding, and building valuable features that matter to businesses. BDD practitioners use conversations around concrete examples of system behavior to help understand how features will provide value to the business. BDD encourages business analysts, software developers, and testers to collaborate more closely by enabling them to express requirements in a more testable way, in a form that both the development team and business stakeholders can easily understand. BDD tools can help turn these requirements into automated tests that help guide the developer, verify the feature, and document the application.

BDD specification is based on User Stories and scenarios which allow to specify executable requirements and test specifications by means of a Domain-Specific Language (DSL) provided by Gherkin. User Stories were firstly proposed by Cohn [3]. North [10] has proposed a particular template to specify them in BDD and named it as "BDD story":

```
Title (one line describing the story)
Narrative:
As a [role], I want [feature], So that [benefit]
Scenario 1: Title
Given [context], When [event], Then [outcome]
```

In this template, BDD stories are described with a *title*, a *narrative* and a set of *scenarios* representing the acceptance criteria. The *title* provides a general description of the story, referring to a feature this story represents. The *narrative* describes the referred feature in terms of the role that will benefit from the feature ("*As a*"), the feature itself ("*I want*"), and the

DOI: 10.18293/SEKE2019-054

benefit it will bring to the business ("*So that*"). The acceptance criteria are defined through a set of *scenarios*, each one with a title and three main clauses: "*Given*" to provide the context in which the scenario will be actioned, "*When*" to describe events that will trigger the scenario and "*Then*" to present outcomes that might be checked to verify the proper behavior of the system. Each one of these clauses can include an "*And*" statement to provide multiple contexts, events and/or outcomes. Each statement in this representation is called a *step*.

### B. User Interface Design Artifacts

*1) Task Models:* Task models provide a goal-oriented description of interactive systems but avoiding the need of detail required for a full description of the user interface. Each task can be specified at various abstraction levels, describing an activity that has to be carried out to fulfill the user's goals. By modeling tasks, designers are able to describe activities in a fine granularity, for example, covering the temporal sequence of tasks to be carried out by the user or system, as well as any preconditions for each task [11]. The use of task models serves multiple purposes, such as better understanding the application under development, being a "record" of multidisciplinary discussions between multiple stakeholders, helping the design, the usability evaluation, the performance evaluation, and the user when performing the tasks. Task models are also useful as documentation of requirements both related with content and structure. HAMSTERS [12] is a tool-supported graphical task modeling notation for task modeling. In HAMSTERS, tasks can be of several types such as abstract, system, user, and interactive tasks. Temporal relationships between tasks are represented by means of operators. Operators can also be of several types such as enable, concurrent, choice, and order independent operators. The temporal operators allow extracting usage scenarios for the system. This is done by following the multiple achievable paths in the model, with each combination of them generating an executable scenario that can be performed in the system.

*2) User Interface (UI) Prototypes and Final UIs:* A UI prototype is an early representation of an interactive system. They encourage communication, helping designers, engineers, managers, software developers, customers and users to discuss design options and interact with each other. Prototypes are often used in an iterative design process where they are refined and become more and more close to the final UI through the identification of user needs and constraints. While the beginning of the project requires a low-level of formality with UI prototypes being hand-sketched in order to explore design solutions and clarify user requirements, the development phase requires more refined versions frequently describing presentation and dialog aspects of the interaction. By running simulations on prototypes, we can determine and evaluate potential scenarios that users can perform in the system [13]. The presentation aspect of full-fledged user interfaces frequently corresponds to how the user "see" the system. From the user's point of view, the presentation of a user interface actually is the system, so if some feature is not available there, then it does not exist at all. Mature UI versions are the source for acceptance testing and will be used by users and other stakeholders to assert whether or not features can be considered as done.

## III. The Proposed Approach

Our proposed approach for assessing the considered artifacts is illustrated in Figure 1, where User Story scenarios are used to ensure consistency in our target artifacts (task models, UI prototypes and final UIs). Therein are exemplified five steps of scenarios being tested against equivalent tasks in task model scenarios, and the equivalent interaction elements in UI prototypes and final UIs. In the first example, the step *"When I select '<field>'"* corresponds to the task *"Select <field>"* in the task model scenario. Such a correspondence is due to the fact that the step and the task represent the same behavior, i.e. selecting something, and both of them are placed at the first position in their respective scenario artifacts. The interaction element "*field*" that will be affected by such a behavior will be assessed on the UI prototype and on the final UI. In both artifacts, such a field has been designed with a *CheckBox* as interaction element. The semantics of the interaction in *CheckBoxes* is compatible with selections, i.e. we are able to select *CheckBoxes*, so the consistency is assured.

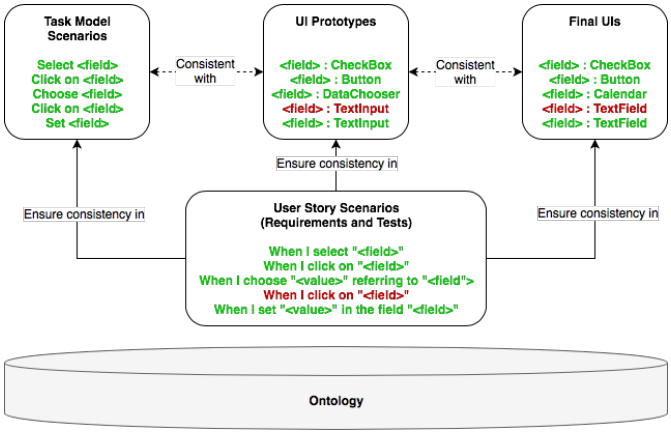


Figure 1. The approach for assessing the different UI artifacts.

The same is true in the example with the second step *("When I click on '<field>'"*). There is a corresponding task *"Click on <field>"* at the same second position in the task model scenario, and the interaction element "*Button*", that has been chosen to address this behavior in both the UI prototype and the final UI, is semantically compatible with the action of clicking, thus the consistency is assured as well. In the third example, the step *"When I choose 'value' referring to 'field'"* is also compatible with the task *"Choose <field>"* in the task model, and with the interaction elements *DataChooser* and *Calendar*, respectively in the UI prototype and in the final UI. Notice that, despite being two different interaction elements, *DataChooser* and *Calendar* support a similar behavior, i.e. both of them support the behavior of choosing values referring to a field.

The example provided with the fourth step *("When I click on '<field>'"*) illustrates an inconsistency being identified. Even though there exists a corresponding task in the task model scenario, the interaction elements that have been chosen to address this behavior (*TextInput* in the UI prototype and *TextField* in the final UI) are not compatible with the action of clicking, i.e. such kind of interaction element does not semantically support such an action. The semantics of *TextInputs* (or *TextFields*) is receiving values, not being clicked. Such an example is provided with the fifth step *("When I set*

*'value' in the field '<field>'"*). For this step, the consistency is assured because *TextInputs* and *TextFields* support the behavior of having values being set on them. All this semantic analysis is supported by the use of an ontology that models the interaction elements and the interactive behaviors they support [14], [15].

The present strategy for assessment allows tracking some key elements in the UI design artifacts and check whether they are consistent with the user requirements. The solution has been implemented in Java integrating multiple frameworks such as JBehave, JDOM, JUnit, and Selenium WebDriver.

### A. Alternatives for Performing the Approach

Depending on the project phase, our approach can be applied in two ways. The first one is applied when the project is running, and artifacts have already been designed. In such a case, our approach can be used to assess such artifacts, indicating where they are not in accordance with the specified requirements. The second one refers to a project in the beginning, where no artifacts have been designed yet. In this case, by using the ontology, they can be modeled in a consistent way from the beginning, taking into account the possible interactions supported by each interaction element on the UI.

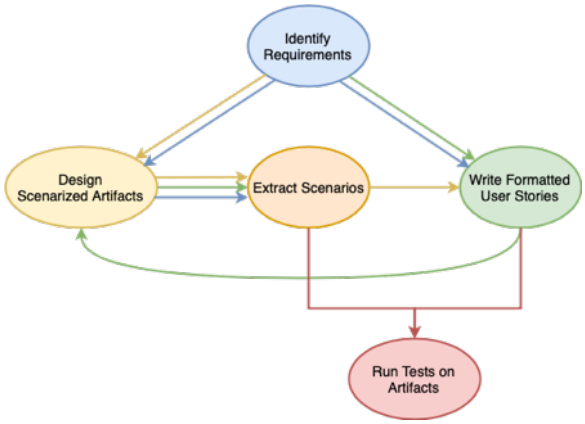


Figure 2. The graph of options for performing our approach (colors are used to visually identify the different paths).

Figure 2 illustrates the resultant graph of options considered. The colored lines indicate the possible paths to be taken in the workflow. The yellow path indicates the design of scenarized artifacts before writing formatted User Stories. The green path indicates the opposite, while the blue path indicates both activities in parallel. Notice that regardless the path chosen, the extraction of scenarios is only possible after having designed the scenarized artifacts, and the identification of requirements is a precondition for all the other activities. Finally, to run tests on the artifacts, it is required to have extracted scenarios and written the User Stories. The approach benefits from the independence for testing artifacts, i.e. tests can run on a single artifact or on a set of scenarized artifacts which will be targeted at a given time.

## IV. Case Study

To investigate the potential of the approach, we have conducted a case study with an existing web system for booking business trips. We have studied the current implementation of user requirements in this system, and by applying a manual reverse engineering, we redesigned the appropriate task models and UI prototypes for the system. Based on a set of User Stories collected in a previous study [16], we refined it to simulate the assessment of the resultant user interface design artifacts. The aim of this present study is to provide a preliminary evaluation regarding the extent of inconsistencies our approach is able to identify in the targeted artifacts.

We started the study by setting up an initial version of User Stories before reengineering initial versions of task models (in HAMSTERS) and UI prototypes (in Balsamiq) from the existing web system. After getting a first version of task models, we extracted a representative set of scenarios from them. By following our strategy for testing, we parsed and ran the initial version of User Stories against the initial set of extracted scenarios. As the strategy we follow for testing scenarios in task models parses all the steps of each scenario at once, the first round of results was obtained with a single battery of tests. Following this step, we ran the same initial version of User Stories against initial versions of Balsamiq prototypes. Unlike the strategy for testing task models, the strategy we follow for testing UI prototypes and final UIs parses each step of each scenario at a time, so if an error is found out, the test stops until the error is fixed. That requires to run several batteries of tests until having the entire set of scenarios tested. Consequently, at the end of running, the tested scenarios are fully consistent with the UIs. Finally, we analyzed the testing results and the main types of inconsistencies identified in each artifact.

In total, we set up for assessment 3 User Stories with 15 different scenarios, reengineered 3 task models (and extracted 10 scenarios from them), reengineered 11 UI prototypes, and tested 7 different final UIs. For scenarios extracted from task models, testing results return the equivalent position of each task in the US scenarios. For UI prototypes and final UIs, the expected result for each step is the presence on the UI of one and only one of the supported interaction elements designed to address a given interactive behavior.

TABLE I. Results after assessing the artifacts

| Artifact | Total (Steps Analyzed) | Results | |
|---|---|---|---|
| | | *Consistent* | *Inconsistent* |
| Task Models | 147 | 5 | 142 |
| UI Prototypes | 36 | 21 | 15 |
| Final UIs | 288 | 276 | 12 |

Table 1 summarizes the results obtained. For task models, the most common source of the 142 inconsistencies identified concerned the task gaps present in the beginning of the scenario. As the assessment is performed in the extracted scenarios which represent a sequential instance of the tasks in the task model, a task gap in the beginning causes a domino effect in the forthcoming tasks in the scenario. So even if the remaining tasks in the scenario are semantically equivalent to the respective steps, they will be shown as inconsistent once they will be found in wrong positions due to this gap. For UI prototypes, from the 15 inconsistencies identified, we noticed they were mainly due to interaction elements specified with different names in the step and in the prototype. For final UIs, the high number of consistent steps (276 out of 288) in the set of scenarios analyzed is due to the need of fixing the inconsistency found before moving forward to the next steps. This makes that the scenarios which

call previous ones, in order to reuse steps and reach a given state of the system, already have these steps fully consistent during the test. Most part of the inconsistencies on final UIs was due to interaction elements that do not carry a unique and single identifier (or carry a dynamically generated one) and, as such, cannot be reached during the test.

We could also remark that some of the inconsistencies identified showed to be more critical than others. While simple inconsistencies such as differences in names of tasks and fields, conflicts between expected and actual elements, and messages and elements not found are easy to solve, conflicts between specification and modeling, and different specification strategies for task models represent more critical problems. On UI prototypes, the presence of semantically inconsistent elements as well as more than one element to represent the same field are also critical problems. On final UIs, fields already filled-in denotates inconsistencies that exposes important design errors. During the test, we also noticed that some inconsistencies were due to a wrong specification of the step in the US scenario, and not to a problem in the design of the artifact itself. So, to fix these inconsistencies, steps of US scenarios needed to be modified during the battery of tests to obtain a consistent specification of user requirements and artifacts. An immediate consequence of this fact is that scenarios used to test a given version of an artifact may be different than the ones which were used to test another artifact previously. This makes regression tests essential to ensure that a given modification in the set of US scenarios did not break the consistency of other artifacts and ended up making some artifact (that so far was consistent with the requirements) inconsistent again.

As limitations of the approach, it is worthwhile to mention that its current version covers only the assessment of HAMSTERS task models, Balsamiq UI prototypes and web final UIs. The need of extracting scenarios from task models to perform testing in such artifacts, and tools that do not support yet the automatic classification of errors are other limitations.

## V. Conclusion and Future Works

This paper summarizes the new results we got by applying our approach for specifying and checking the consistency of user requirements on core user interface design artifacts. Compared to plain-vanilla BDD, this approach benefits from (*i*) an extension to assess other software artifacts than final UIs, and (*ii*) a common vocabulary to be reused for specifying interactive scenarios without requiring developers to implement the mentioned behaviors. Compared to other approaches for assessing requirements and artifacts, the term "test" is usually not employed under the argument that such artifacts cannot be "run", i.e. executed for testing purposes, so in practice they are just manually reviewed or inspected in a process called verification. Manual verification of the software outcomes is highly time-consuming, error-prone and even impracticable for large software systems. Fully interactive artifacts such as final UIs can in addition be validated by users who can interact with the artifact and assess whether its behavior is aligned with their actual needs. As within our approach we succeed automatically running User Stories on software artifacts for assessing their consistency with user requirements, we actually provide the "test" component for both verification and validation of artifacts in the software development. We consider this a big step towards the automated testing (and not only the manual verification) of software artifacts by means of a consistent approach allowing fully verification, validation, and testing (VV&T).

Future works include evaluating the impact of maintaining and successively evolving the mentioned artifacts throughout a real software development process, besides investigating the suitability of the approach for assessing a wider group of artifacts, especially those related to conceptual aspects of software modeling such as class diagrams. Concerning the tools, the development of a plugin to suggest and autocomplete steps in the User Story scenarios based on the interactive behaviors of the ontology is also envisioned.

## References


[1] M. Winckler and P. Palanque, "Models as Representations for Supporting the Development of e-Procedures," in *Usability in Government Systems*, Elsevier, 2012, pp. 301–315.

[2] D. Chelimsky, D. Astels, B. Helmkamp, D. North, Z. Dennis, and A. Hellesoy, *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf, 2010.

[3] M. Cohn, *User Stories Applied for Agile Software Development*. Addison-Wesley, 2004.

[4] T. R. Silva and M. A. A. Winckler, "Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems," in *2nd Workshop on Continuous Requirements Engineering (CRE), REFSQ 2016*, 2016, pp. 1–2.

[5] T. R. Silva, "Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems," in *Proceedings of the 24th International Requirements Engineering Conference (RE 2016)*, 2016, pp. 444–449.

[6] T. R. Silva, J.-L. Hak, and M. Winckler, "Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development," in *HCSE 2016 and HESSD 2016*, LNCS, vol. 9856, Springer, 2016, pp. 86–107.

[7] T. R. Silva, J.-L. Hak, and M. Winckler, "An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development," *Complex Systems Informatics and Modeling Quarterly*, no. 7, pp. 81–107, 2016.

[8] T. R. Silva and M. Winckler, "A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts," in *Proceedings of the 16th Brazilian Symposium on Human Factors in Computing Systems (IHC 2017)*, 2017, vol. 1, pp. 21–30.

[9] J. F. Smart, *BDD in Action: Behavior-driven development for the whole software lifecycle*, 1 edition. Manning Publications, 2014.

[10] D. North, "What's in a Story?," 2019. [Online]. Available: https://dannorth.net/whats-in-a-story/. [Accessed: 01-Jan-2019].

[11] F. Paternò, C. Santoro, L. D. Spano, and D. Raggett, "W3C, MBUI - Task Models," 2017. [Online]. Available: http://www.w3.org/TR/task-models/.

[12] C. Martinie, P. Palanque, and M. A. Winckler, "Structuring and Composition Mechanisms to Address Scalability Issues in Task Models," in *INTERACT 2011*, 2011, vol. 6948 LNCS, no. 3, pp. 589–609.

[13] M. Beaudouin-Lafon and W. E. Mackay, "Prototyping Tools and Techniques," in *Prototype Development and Tools*, 2000, pp. 1–41.

[14] T. R. Silva, J.-L. Hak, and M. Winckler, "A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces," *International Journal of Semantic Computing*, vol. 11, no. 04, pp. 513–539, 2017.

[15] T. R. Silva, J.-L. Hak, and M. Winckler, "A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems," in *Proceedings of the 11th IEEE International Conference on Semantic Computing (ICSC 2017)*, 2017, pp. 250–257.

[16] T. R. Silva, M. Winckler, and C. Bach, "Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners," *Cognition, Technology & Work*, 2019.