# Empirical Findings on BDD Story Parsing to Support Consistency Assurance between Requirements and Artifacts

Thiago Rocha Silva
thiago.silva@ul.ie
Lero, the Science Foundation Ireland Research Centre for Software, University of Limerick (UL)
Limerick, Ireland

Brian Fitzgerald
brian.fitzgerald@ul.ie
Lero, the Science Foundation Ireland Research Centre for Software, University of Limerick (UL)
Limerick, Ireland

## ABSTRACT

Behaviour-Driven Development (BDD) stories have gained considerable attention in recent years as an effective way to specify and test user requirements in agile software development projects. External testing frameworks also allow developers to automate the execution of BDD stories and check whether a fully functional software system behaves as expected. However, other software artifacts may quite often lose synchronization with the stories, and many inconsistencies can arise with respect to requirements representation. This paper reports on preliminary empirical findings regarding the performance of two existing approaches in the literature intended to support consistency assurance between BDD stories and software artifacts. The first approach involves the parsing of BDD stories in order to identify conceptual elements to automatically generate consistent class diagrams, while the second approach seeks to identify interaction elements to automatically assess the consistency of task models and GUI prototypes. We report on the precision of these approaches when applied to a study with BDD stories previously written by Product Owners (POs). Based on the results, we also identify a set of challenges and opportunities for BDD stories in the consistency assurance of such artifacts.

## CCS CONCEPTS

• **Software and its engineering → Software verification and validation**.

## KEYWORDS

Behaviour-Driven Development, User Stories, User Requirements, Consistency Assurance, Software Artifacts.

**ACM Reference Format:**
Thiago Rocha Silva and Brian Fitzgerald. 2021. Empirical Findings on BDD Story Parsing to Support Consistency Assurance between Requirements and Artifacts. In *Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3463274.3463807



*EASE 2021, June 21–23, 2021, Trondheim, Norway*

ACM ISBN 978-1-4503-9053-8/21/06.
https://doi.org/10.1145/3463274.3463807

## 1 INTRODUCTION

In recent years, User Stories have been by far the most widely adopted requirements artifact employed by agile methods to specify user requirements [7]. A User Story proposes to specify a requirement based on a narrative describing the role, the feature, and the benefit that can be expected from that feature. Behaviour-Driven Development (BDD) [5] stories have enriched User Stories by proposing the additional specification of scenarios as acceptance criteria for each story, thus providing in a single structure the requirement itself along with examples describing the expected behavior of the system. A benefit of using BDD stories is that such scenarios can be made testable directly from their textual specifications, so the stories can be automatically executed to assess a fully functional software system.

However, BDD and its testing frameworks do not currently provide any support for assessing intermediate software artifacts. Since such artifacts represent an important source of information for teams to discuss or refine features, ensuring their consistency with requirements is a key factor for success. Assessing this consistency is nonetheless a challenge and may require a considerable effort from development teams. Most of the research intended to establish links between requirements and artifacts is centered on tracing requirements throughout the development process [9]. This problem has long been studied and a wide set of commercial tools have been developed using various approaches. Nevertheless, proposed solutions to traceability can simply identify whether a requirement is present or not in a given artifact, and do not necessarily permit effective assessment by checking the consistency and correct representation of such a requirement in a given set of artifacts.

To this end, we have been studying how BDD stories, as a requirements artifact, can be used to support the assessment of other software artifacts in order to verify the consistency between them. In this paper, we preliminarily investigate whether and to what extent two different approaches from the literature can effectively identify key elements on BDD stories to support consistency assurance on class diagrams, task models, and GUI prototypes. We applied both approaches to a study including BDD stories written by different Product Owners (POs) for booking business trips. We then analyzed the outcomes of each one of these approaches in terms of precision when identifying relevant elements from the stories.

In summary, the contributions of this paper are:

- A brief review of the main approaches in the literature to support consistency assurance between requirements and class diagrams, task models, and GUI prototypes;

- Empirical findings regarding the performance of two of these approaches focused on parsing BDD stories to support consistency assurance on these artifacts; and
- An outline of the main challenges and opportunities involved in this process, as well as our ongoing and future research endeavors in this field.

# 2 BACKGROUND AND RELATED WORK

## 2.1 BDD Stories

There is increasing interest from both academic and industrial communities in BDD for supporting automated acceptance testing of user requirements [11, 22]. User requirements in BDD are specified as stories in natural language which are meant to be executable, so they can provide "living documentation" by dynamically informing developers and other stakeholders about the status of the system with respect to the acceptance criteria. BDD stories have been proposed by North [10] and include a User Story narrative and a set of acceptance criteria. The acceptance criteria are described as scenarios following a Given-When-Then format, where *"Given"* presents the current state of the system, *"When"* describes an event or stimulus to that system, and *"Then"* describes the expected resulting state 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". BDD scenarios correspond to state transitions, and when taken together, to a finite state machine.

BDD scenarios can be described at different levels of abstraction. They can specify steps using the domain vocabulary (usually at a higher abstraction level) or an interactive vocabulary (usually at a lower abstraction level). Chelimsky et al. [5] call these declarative and imperative scenarios respectively. While imperative scenarios are useful to go step-by-step through the multiple interactions required to perform a given task, declarative scenarios are more straightforward, wrapping all these interactions up into a single step usually referring to a domain concept. These two approaches impact different parts of the process in different ways.

## 2.2 Requirements and Artifacts Consistency

The study of automated approaches to explore links between software artifacts and BDD stories is very recent. Yang et al. [20] describe a model to predict when feature files (containing the BDD stories) should be modified before committing a code change in order to improve traceability between the BDD stories and the source code. Alferez et al. [2] propose to generate BDD acceptance criteria from requirements expressed through UML-based models. Others have tried to tackle the problem of assessing multiple artifacts by means of a scenario-based approach. Silva et al. [15] propose the parsing of BDD stories to identify relevant elements to be assessed on user interface design artifacts. The authors employ an ontology that models concepts describing the structure and the associated behavior of BDD stories, tasks, scenarios, and GUI elements. For assessing GUI prototypes [14], the approach suggests an inconsistency when steps are specified using interactive behaviors that are semantically inconsistent with the affected interaction element on the GUI, such as a selection to be made on a button, for example. For assessing task models [16], the approach proposes a sentence analysis of each interactive behavior in the BDD scenarios to identify potential tasks to be verified in the model. The relationship between user interface design and task modeling has been studied from a broader requirements perspective as well. Campos et al. [4] propose a tool-supported framework to link task models to an interactive application, defining a systematic correspondence between GUI elements and user tasks. The approach, however, requires a wide intervention in the source code of the application.

For the identification of conceptual elements on textual requirements specifications, the usage of Natural Language Processing (NLP) techniques is a fundamental approach. This is a long-term research topic for the RE community with the first works dating back the early 80's with Chen [6] and Abbott [1], who proposed using syntactic features of English sentences for Entity-Relationship (ER) modeling and program design. A theory of domain knowledge was also proposed by Sutcliffe and Maiden [18] to define the semantics and composition of generic domain models in the context of requirements engineering. These works paved the way for many others and helped to establish a set of heuristics to link syntactic elements from requirements expressed in natural language to conceptual elements in ontology [8], class [3, 12], and analysis [21] models. A recently published survey [23] provides a comprehensive review of the many contributions in this field.

Among the vast literature around the theme, we identified only a single approach specifically targeting the parsing of BDD stories with the aim of identifying conceptual elements. Soeken et al. [17] propose the parsing of BDD stories using a set of heuristics to automatically generate class diagrams. They claim regular nouns in sentences are usually realized as objects in the system, and therefore they can be represented by classes; adjectives usually provide further information about the respective objects, thus they can be represented by attributes of classes; and verbs usually describe actions in a scenario and can therefore be represented by methods of classes. The authors also propose the identification of associations and other relationships between these elements.

# 3 STUDY

## 3.1 Study Design

Since we identified only two approaches in the literature targeting the parsing of BDD stories to support consistency assurance either on class diagrams [17] or on task models and GUI prototypes [15], we investigated the performance of both when applied to a case study including a dataset of BDD stories written by potential Product Owners (POs) and obtained in a previous study [13]. The stories relate to the booking and management of flight tickets for business trips and include scenarios containing both domain and interactive behaviors (i.e., declarative and imperative steps). The dataset had 7 different stories with 20 scenarios, totaling 183 sentences. For each story, we parsed the scenario sentences to determine which elements we would be able to identify for potential assessment of the corresponding software artifacts. Our goal is to answer two research questions:

**RQ1**. How precise is the Silva et al. approach [15] to identify elements from BDD scenarios to support consistency assurance in task models and GUI prototypes?

**RQ2**. How precise is the Soeken et al. approach [17] to identify elements from BDD scenarios to support consistency assurance in class diagrams?

To answer these questions, we replicated both the Silva et al. approach [15] and the Soeken et al. approach [17] using the dataset mentioned above and focusing on the scenario part of the stories. It is important to notice that the goal of this study is not to use the stories for performing the assessment on previously designed artifacts, neither compare both approaches since they are focused on completely different types of artifacts. The goal is rather to evaluate the precision of both approaches for identifying and classifying their target elements in the BDD stories when applied to a particular case study.

## 3.2 Methodology

We started by applying the Soeken et al. [17] approach to the BDD scenarios from our dataset to identify the conceptual elements. To replicate the approach, we manually applied the heuristics proposed by the authors to identify the conceptual elements from the scenarios since the authors did not provide any kind of tool support. As proposed by the authors, we parsed each one of the BDD steps using the Stanford Parser and analyzed them with WordNet. The Stanford Parser parses sentences in different languages and returns a phrase structure tree (PST) representing the semantic structure of the sentence. We applied the parser in this work to process the BDD steps and identify conceptual elements which should be assessed in a class diagram. WordNet is a large lexical database which groups nouns, verbs, adjectives, and adverbs into sets of cognitive synonyms, each representing a lexicalized concept. We applied WordNet to determine the commonly used semantics of each word prior to assigning it to a particular element. Following the heuristics proposed by the authors, we identified candidates for classes, attributes, methods, and associations.

Next, we applied the Silva et al. [15] approach to the same dataset in order to identify tasks to be potentially assessed against task models, and interactive behaviors to be potentially assessed against GUI prototypes. The replication of the approach was straightforward since the authors developed a tool to parse the BDD stories and assess the artifacts [14, 16]. The tool receives the set of BDD stories as input and delivers the assessment results including the interactive tasks identified and their positions, as well as the interactive behaviors identified and their supported GUI elements.

Finally, after getting both results, we counted how many elements (classes, attributes, methods, associations, tasks, and interactive behaviors), as proposed by the authors, have been identified. We then calculated the precision of this identification considering the pertinence of these elements being present in potential class diagrams, task models, or GUI prototypes for that particular domain (the ground truth). As there were no actual artifacts to be assessed, we did not calculate the recall of the approaches.

# 4 RESULTS

Table 1 presents the results of parsing the BDD steps of a scenario from the dataset. The scenario in question is intended to assess a feature of listing travel authorizations. The imperative step *"When I type the 'Booking Reference'"*, for example, returns in the PST the verb (VB) *"type"* (a candidate method), an adjective (JJ) *"Booking"* (a candidate attribute), and a noun (NN) *"Reference"* (a candidate class). The PST also returns the adjectival modifier (amod) *"Reference-Booking"* which indicates that *"Booking"* (the adjective) would be an attribute of the class *"Reference"* (the noun). For this step, the ontology returns a candidate task *"Type 'Booking Reference'"* and the interactive behavior *"#type"* which is associated with the interaction element "Text Field" on GUIs. From these results, we can notice the verb *"type"* refers to an interaction with the GUI, so much so that it has been identified as an interactive behavior by the ontology. As part of an imperative step, this verb should have been discarded from the domain vocabulary, thus ignoring the assignment of a method named *"type"* to one of the classes. Concerning the adjective *"Booking"*, we notice it would not be an attribute of the candidate class *"Reference"*. Actually, in this case, it is more likely that *"Reference"* would be the attribute of a class named *"Booking"*.

Another example is the declarative step *"And I check if the request is well registered"* which returns in the PST the verbs (VB) *"check"*, *"is"* and *"registered"* (candidate methods), and a noun (NN) *"request"* (a candidate class). The PST also returns the passive nominal subject (nsubjpass) *"registered-request"* which suggests *"register"* (the verb) could be a method of the class *"request"* (the noun). For this step, the ontology only returns *"Check request"* as a candidate task. Notice that, as expected from a declarative step, the ontology does not return any interactive behavior to be assessed on the GUI. From these results, we can notice the verb *"is"* would not be a method of any class and should be regarded as a determinant of the relationship *"request is registered"*. The other steps and scenarios have been parsed in a similar manner. We performed such analyses with all the BDD scenarios from the dataset which contained 132 sentences in total. Figure 1 summarizes the results of parsing the BDD steps.

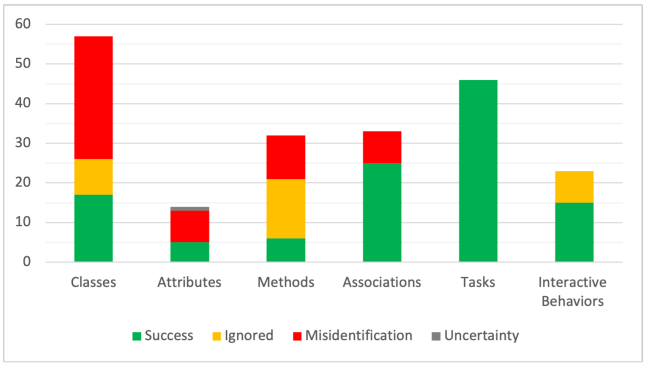


**Figure 1: Results of parsing the BDD steps.**

As expected, the majority of tasks and interactive behaviors (required for assessing task models and GUI prototypes respectively) were identified in the imperative steps. 46 tasks and 23 interactive behaviors were successfully identified when parsing the scenarios, with 15 behaviors being ignored since they only described calls to other scenarios. Concerning the identification of elements for assessing class diagrams, we counted 57 candidate classes, of which 17 were successfully identified, 9 ignored, and 31 misidentified. The

**Table 1: Results of parsing the BDD steps of a scenario.**

| Steps of a BDD scenario | Class Diagram (Candidate Elements) | Task Model | GUI Prototype |
|---|---|---|---|
| **Given** I go to the tab "Travel Authorization" | (VB *go*): method<br>(NN *tab*): class<br>(NNP *Travel*): class<br>(NNP *Authorization*): class<br>compound(*Authorization*-9, *Travel*-8): association<br>compound(*Authorization*-9, *tab*-6): association | *"Go to 'Travel Authorization'"*: candidate task | Interactive behavior *#goTo "Travel Authorization"*: Browser Window |
| **When** I type the "Booking Reference" | (VBZ *type*): method<br>(JJ *Booking*): attribute<br>(NN *Reference*): class<br>amod(*Reference*-7, *Booking*-6): class-attribute relationship | *"Type 'Booking Reference'"*: candidate task | Interactive behavior *#type "Booking Reference"*: Text Field |
| **And** I check if the request is well registered | (VBP *check*): method<br>(NN *request*): class<br>(VBZ *is*): method<br>(VBN *registered*): method<br>nsubjpass(*registered*-9, *request*-6): method-class relationship | *"Check request"*: candidate task | - |
| **Then** at this time, I can know for sure (or not) the request has been taken into account | (NN *time*): class<br>(VB *know*): method<br>(JJ *sure*): attribute<br>(NN *request*): class<br>(VBZ *has*), (VBN *been*), (VBN *taken*): methods<br>(NN *account*): class<br>nsubjpass(*taken*-19, *request*-16): method-class relationship | - | - |
| **And** it's shown a tab: authorized / non-authorized | (VBZ *is*), (VBN *shown*): methods<br>(NN *tab*): class<br>(VBN *authorized*): method<br>(VBN *non-authorized*): method | *"Show 'authorized / non-authorized'"*: candidate task | Interactive behavior *#show "authorized / non-authorized"*: Text |

majority of misidentified candidate classes was found in scenarios mixing both declarative and imperative steps (23 out of 33). We also identified 14 candidate attributes, of which 5 were successfully identified, 8 misidentified, and 1 uncertainty. As for classes, almost all of the misidentified candidate attributes have been found in scenarios mixing both declarative and imperative steps (8 out of 9). For methods, we identified 32 candidates, being 6 successfully identified, 15 ignored, and 11 misidentified. Verbs were considered ignored as candidate methods when they matched the interactive behaviors present in the ontology, which was always the case in scenarios with only imperative steps. Finally, for associations, we identified 33 candidates, of which 25 were successfully identified and 8 misidentified.

Table 2 presents the total number of elements identified by the Soeken et al. and Silva et al. approaches. For the purpose of calculating the precision, elements classified as ignored in Figure 1 were counted as misidentified since they should have been either discarded (in case of conceptual elements) or signalized (in case of interactive behaviors) by the respective approaches as discussed above. On the other hand, elements classified as uncertainty were counted as success since these approaches assume designers should manually check the final results in order to decide whether or not to consider the suggested elements in the final model.

**Table 2: Precision identifying correct elements in the case study.**

| | Identified | Correct | Precision |
|---|---|---|---|
| Classes | 57 | 17 | 29.82% |
| Attributes | 14 | 6 | 42.86% |
| Methods | 32 | 6 | 18.75% |
| Associations | 33 | 25 | 75.76% |
| Tasks | 46 | 46 | 100.00% |
| Interactive Behaviors | 23 | 15 | 65.22% |

The approach proposed by Silva et al. clearly performed well to identify tasks for assessment on task models (100% precision) and interactive behaviors for assessment on GUI prototypes (65.22% precision) (**RQ1**). The approach proposed by Soeken et al. overall performed badly to correctly identify elements to be assessed on class diagrams, ranging from a precision as low as 18.75% for methods and 29.82% for classes to a moderately high precision (75.76%) for associations. An intermediate precision of 42.86% has been observed for the identification of attributes (**RQ2**). As the original papers did not report on the precision of their respective approaches, we could not compare our results to theirs.

# 5 DISCUSSION

## 5.1 Challenges and Opportunities

**Differentiation between classes and attributes**. As noticed from the results, there is a high level of misidentification for classes and attributes provided by the Soeken et al. approach. This is mainly due to the use of simplified heuristics. We noticed that many potential attributes, for example, are actually nouns that are either adjectivized in the sentence or semantically referenced to another noun which is the potential class to which such an attribute would refer. That leads to some nouns being actually candidate attributes of other classes instead of new classes themselves. The analysis of the relationships returned by the compounds in PST has been shown to be insufficient to distinguish them. WordNet helped to distinguish some nouns employed more frequently as adjectives of other nouns (which could signify an attribute instead of a class), however its effectiveness appeared to be limited in many cases.

A good example is a sentence specified by one of the POs: *"When I inform the data concerning the traveler (name, given name, birthdate, phone, mail), and eventually the loyalty card (Flying Blue and Season Ticket)"*. It brings many potential attributes such as name, given name, birthdate, phone, mail, and loyalty card, but due to the lack of a predefined structure for the sentence, the attributes will not be identified as such since they are all marked by PST as simple nouns (NN). For this case, PST has successfully identified relationships only between *"traveler"* (a potential class) and *"name/loyalty card"*. Many other candidate classes have also been misidentified due to nouns completely meaningless to the business domain. We also missed the identification of some methods, especially when PST returned a word as being a proper noun in the phrase when the word was actually a verb according to WordNet (which would correctly suggest a method). That happened, for example, when parsing the steps *"And I click on 'Finalize/Decline the trip'"* with the words *"Finalize"* and *"Decline"*. Particularly for the identification of attributes, the inclusion of refined heuristics such as the ones identified in [3, 12] and other works could be especially helpful.

**Identifying test data**. Usually, the content between quotation marks is regarded as such. Identifying to which attribute such data refer is however a tricky problem. When proper names are used, one of the possible strategies is looking for a relationship "dep" or more specifically "dobj" (the accusative object of the verb) in the PST composed by the verb (VB) and the respective proper noun (NNP or NNPS). In many cases, however, test data are not proper nouns. Some of them such as *"TOULOUSE/PARIS"*, for indicating respectively a departure and a destination, have been identified by PST as a noun (NN) with an appositional modifier (appos) *"destination, TOULOUSE/PARIS"* which is not enough to correctly identify the data *"TOULOUSE/PARIS"* as a binomial departure/destination. Furthermore, it is not trivial to automatically identify when test data refer to a data domain or the actual data.

**Declarative steps are indeed richer for identifying conceptual elements**. While imperative steps often contain more elements relating to the user-system interaction (thus favoring the identification of interactive behaviors as proposed by Silva et al.), declarative steps often make much more reference to concepts of the business domain (thus favoring the identification of conceptual elements). For example, when applying the Soeken et al. approach to imperative steps, the majority of verbs were to be discarded since they made reference almost exclusively to interactive behaviors on the GUI. When parsing them, the Silva et al. approach is able to provide good results for assessing task models and GUI prototypes. The extent of common interactive behaviors provided by the ontology is, however, a key factor for adoption in real-world projects. For task models in particular, good results are highly subject to the abstraction level at which the designer chose to specify the tasks.

**Manual analysis of the suggested elements is still necessary**. It is clear the need of additional manual work to analyze the elements suggested by the approaches, especially for class diagrams, in order to refine them and arrive at a reliable subset for assessment on the artifacts. That means we are still far from a fully automated approach that works well for BDD stories. The Soeken et al. approach could be enriched with additional and more refined heuristics from other generic NLP approaches to deliver better results and require less manual intervention at the end of the process. This is a key factor for adoption on the assessment of conceptual artifacts at the same level of automation that task models and GUI prototypes are assessed by the Silva et al. approach.

**The particular characteristics of the BDD stories have not yet been explored for conceptual artifacts**. Parsing free text of natural language requirements is often challenging especially because the sentences are usually context-dependent, and it is difficult to automatically identify what is in the scope of the system and what is not. This information is often tacit and not automatically inferable [3]. BDD stories, however, present some particularities that could be taken into account when applying NLP heuristics since they are written in a stricter format. BDD stories feature a state machine language to describe its scenarios, clearly identifying the context to which the scenarios apply, the action to be assessed, and the expected outcomes. This means that the sentences in each one of these components may carry specific information that should be relevant to identify particular elements of the conceptual model. These characteristics are leveraged by Silva et al. when identifying elements for assessment on task models and GUI prototypes, but not properly catered for at present by the Soeken et al. approach.

**There is a clear need for some kind of controlled natural language to specify BDD stories**. Particularly when identifying conceptual elements, many of the misidentified ones were due to the free format used especially in declarative steps. The results suggest that some predefined structure for the BDD stories as well as a guiding vocabulary may be very helpful to correctly identify elements for assessment on artifacts. The use of a controlled natural language (CNL) [19] associated with a domain-specific language (DSL) could provide such a structure and guide the writing of testable stories. This could avoid misspecification by limiting the extent of the vocabulary available, which could facilitate the identification of useful elements for assessment. Such a solution would benefit software development processes in two ways. First, by providing a controlled vocabulary in natural language to be used by technical and non-technical people to specify testable user requirements in a consistent and comprehensive way, which could reduce communication gaps. Second, by providing a more straightforward identification of key elements for the automated assessment of requirements on the final product and on a range of software artifacts produced throughout the project.

### 5.2 Threats to Validity

The first threat to the validity of this study is that the stories have been originally written in French by the POs in our previous study and were, for the purpose of the present study, translated into English to facilitate the use of the tools. We acknowledge this might have somehow affected the results due to the imprecision invariably added to a translation. Another identified threat refers to the Soeken et al. approach being originally intended to generate the class diagrams, instead of assessing them. It is still to be investigated if the heuristics used to generate such models from BDD stories would be different from those used to just identify the elements and then assess previously designed models.

## 6 CONCLUSION AND FUTURE WORK

This paper reports preliminary findings on the effectiveness of the current approaches for parsing BDD stories aiming at supporting consistency assurance between requirements and artifacts. In summary, the results show that the automatic identification of conceptual elements is definitely not a trivial problem and can lead to many classification errors. In this field, the Soeken et al. approach performed with low precision, indicating there are still many challenges, especially associated with the correct distinction between classes and attributes, and the identification of methods. The identification of elements for the assessment of task models and GUI prototypes, despite still presenting some challenges, seems to be better addressed with the Silva et al. approach which performed well in the case study analyzed.

At present, we are investigating a specific set of generic NLP heuristics to be used on the parsing of BDD stories for the assessment of class diagrams. We aim specifically at incorporating more refined heuristics for the identification of classes, attributes, and methods. A comprehensive case study is being conducted to compare the effectiveness of such generic heuristics and support the development of a new tool-supported approach for parsing BDD stories and assessing their consistency with class diagrams with better precision. We are also currently working on the development of a high-level DSL that proposes a controlled natural language to specify testable and consistent user requirements through BDD stories. Such a DSL will combine the benefits of easy communication provided by BDD with vocabulary predictability to allow the identification of useful elements in the stories for the assessment of software artifacts.

## ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754489.

## REFERENCES

[1] Russell J. Abbott. 1983. Program Design by Informal English Descriptions. *Commun. ACM* 26, 11 (1983), 882–894. https://doi.org/10.1145/182.358441

[2] Mauricio Alferez, Fabrizio Pastore, Mehrdad Sabetzadeh, Lionel C. Briand, and Jean-Richard Riccardi. 2019. Bridging the Gap between Requirements Modeling and Behavior-Driven Development. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 239–249. https://doi.org/10.1109/MODELS.2019.00008

[3] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2016. Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation. In *2016 ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 250–260. https://doi.org/10.1145/2976767.2976769

[4] José C. Campos, Camille Fayollas, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. 2016. Systematic Automation of Scenario-Based Testing of User Interfaces. In *8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 138–148. https://doi.org/10.1145/2933242.2948735

[5] David Chelimsky, Dave Astels, Bryan Helmkamp, Dan North, Zach Dennis, and Aslak Hellesoy. 2010. *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf.

[6] Peter P. S. Chen. 1983. English sentence structure and entity-relationship diagrams. *Information Sciences* 29, 2-3 (1983), 127–149. https://doi.org/10.1016/0020-0255(83)90014-2

[7] Garm Lucassen, Fabiano Dalpiaz, Jan M. E. M. van der Werf, and Sjaak Brinkkemper. 2016. The Use and Effectiveness of User Stories in Practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 205–222. https://doi.org/10.1007/978-3-319-30282-9_14

[8] Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan M. E. M. van der Werf, and Sjaak Brinkkemper. 2017. Extracting conceptual models from user stories with Visual Narrator. *Requirements Engineering* 22, 3 (2017), 339–358. https://doi.org/10.1007/s00766-017-0270-1

[9] Sunil Nair, Jose Luis de la Vara, and Sagar Sen. 2013. A Review of Traceability Research at the Requirements Engineering Conference RE@21. In *2013 21st IEEE International Requirements Engineering Conference (RE)*. 222–229. https://doi.org/10.1109/RE.2013.6636722

[10] Dan North. 2021. What's in a Story? https://dannorth.net/whats-in-a-story/

[11] Lauriane Pereira, Helen Sharp, Cleidson de Souza, Gabriel Oliveira, Sabrina Marczak, and Ricardo Bastos. 2018. Behavior-driven development benefits and challenges: Reports from an industrial study. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*. 1–4. https://doi.org/10.1145/3234152.3234167

[12] Vidhu B. R. V. Sagar and S. Abirami. 2014. Conceptual modeling of natural language functional requirements. *Journal of Systems and Software* (2014). https://doi.org/10.1016/j.jss.2013.08.036

[13] Thiago R. Silva, Marco Winckler, and Cédric Bach. 2020. Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners. *Cognition, Technology and Work* 22, 3 (2020), 437–457. https://doi.org/10.1007/s10111-019-00566-3

[14] Thiago R. Silva, Marco Winckler, and Hallvard Trætteberg. 2019. Ensuring the Consistency Between User Requirements and GUI Prototypes: A Behavior-Based Automated Approach. In *IFIP Conference on Human-Computer Interaction*, Vol. 11746. Springer, 644–665. https://doi.org/10.1007/978-3-030-29381-9_39

[15] Thiago R. Silva, Marco Winckler, and Hallvard Trætteberg. 2019. Extending Behavior-Driven Development for Assessing User Interface Design Artifacts. In *The 31st International Conference on Software Engineering & Knowledge Engineering (SEKE 2019)*. KSI Research, 485–488. https://doi.org/10.18293/SEKE2019-054

[16] Thiago R. Silva, Marco Winckler, and Hallvard Trætteberg. 2020. Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach. *Proceedings of the ACM on Human-Computer Interaction* 4, EICS (2020), 77:1–32. https://doi.org/10.1145/3394979

[17] Mathias Soeken, Robert Wille, and Rolf Drechsler. 2012. Assisted Behavior Driven Development Using Natural Language Processing. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Vol. 7304 LNCS. Springer, 269–287. https://doi.org/10.1007/978-3-642-30561-0_19

[18] Alistair Sutcliffe and Neil Maiden. 1998. The Domain Theory for Requirements Engineering. *IEEE Transactions on Software Engineering* 24, 3 (1998), 174–196. https://doi.org/10.1109/32.667878

[19] Alvaro Veizaga, Mauricio Alferez, Damiano Torre, Mehrdad Sabetzadeh, and Lionel Briand. 2021. On Systematically Building a Controlled Natural Language for Functional Requirements. *Empirical Software Engineering* (2021).

[20] Aidan Z. H. Yang, Daniel A. da Costa, and Ying Zou. 2019. Predicting co-changes between functionality specifications and source code in behavior driven development. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 534–544. https://doi.org/10.1109/MSR.2019.00080

[21] Tao Yue, Lionel C. Briand, and Yvan Labiche. 2011. A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering* 16, 2 (2011), 75–99. https://doi.org/10.1007/s00766-010-0111-y

[22] Fiorella Zampetti, Andrea Di Sorbo, Corrado A. Visaggio, Gerardo Canfora, and Massimiliano Di Penta. 2020. Demystifying the adoption of behavior-driven development in open source projects. *Information and Software Technology* 123 (2020). https://doi.org/10.1016/j.infsof.2020.106311

[23] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol-Valeriu Chioasca, and Riza T. Batista-Navarro. 2021. Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study. *Comput. Surveys* 54, 3 (2021).