Behavior-Driven Development in Product Configuration Systems

Sara Shafiee[1] and **Lars Hvam** and **Anders Haug** and **Yves Wautelet**

Abstract. Product Configuration Systems (PCS) are increasingly used by companies to automate the performance of the sales and engineering processes. Since the benefits from such projects have huge variations, it is crucial to make the right decisions when scoping and developing PCSs. The development of PCS is influenced by both business interests and technical insights. Developers of PCS face various challenges while working in team, including different stakeholders such as business owners, developers, project managers, and product experts. The more diverse the team is, the more significant are the challenges. This paper suggests that Behavior-driven Development (BDD) may provide configuration teams with a specific structure to express scenarios (and thus constraints) on PCS in natural language. BDD may yield benefits such as a better expression of PCS constraints, more efficient communication of requirements and incorporation of the expressed rules in a software transformation process. In other words, applying BDD may eliminate unnecessary tasks when gathering knowledge, developing, and testing PCS projects. In this paper, we present a novel approach from an ongoing project on how to relate BDD to the development process of PCS while using Scrum-based methods.

1 INTRODUCTION

Product Configuration Systems (PCS) are expert systems that can support and facilitate the sales and engineering processes [1] by incorporating information about product features, product structure, production processes, costs and prices [2]. Thereby, configurators can support decision-making processes in the sales and engineering phases of a product [3]. Configurators enable companies to develop product alternatives to facilitate sales and production processes [4]. However, the companies that have managed to implement and utilize configurators also face various challenges [1], [5].

Even though advantages of PCS are evident, there are still some difficulties associated with high costs [2], [6] and considerable chances of failure [2], [7] in their implementation projects. This is because companies must overcome various challenges to implement and utilize configurators. For example, the development of a PCS often requires highly complex technical or commercial knowledge, which domain experts often have a hard time communicating to configuration experts [8]. Furthermore, the knowledge base that contains this knowledge has to be adapted continuously because of the changing components and configuration constraints [9], [10]. The difficulty of acquiring and modeling the required technical or commercial knowledge depends on whether it is available in a clear and formal form [11] which in turns may be contingent to company size [12], product complexity [13], degree of customization [14], or other factors such as knowledge management and scoping process [8]. The challenges to manage technical and commercial knowledge when implementing and maintaining a PCS may highly influence PCS costs and development time, as well as its lasting effectiveness.

The complexity in the development of PCSs comes from that it involves highly complex technical knowledge from domain experts, [9], [10]. Hence, Scrum and agile methods attracted attention in PCS development. The main reasons that can be mentioned as their ease of use, the constant communication with the stakeholders and the team and fast development time [10], [15], [16]. In spite of the potential benefits of Scrum, many organizations are reluctant to throw their conventional methods away and jump into agile methods. This to some extent may be attributed some of the issues experienced with the use of Scrum, including significant reduction of documentation, insufficient testing for mission/safety-critical projects, inadequate support for highly stable projects, only successful with talented individuals who favor large degrees of freedom, and inappropriate for large-scale projects [17].

As the discussion above indicates, it is very challenging to specify a PCS at analysis stage. In Scrum-based development this is traditionally done using user stories [18]. The latter nevertheless do present some drawbacks because their narrative alone is not enough to express the constraints related to PCS. This is precisely where BDD could offers complementary representation abilities destined to address user stories limitations. Because of the use of BDD next to user stories narrative, the PCS can be expressed without any other requirements representation artifact (traditionally the Product Variant Master is used for this [3]). Other positive aspects of the use of BDD to further describe user stories are their expression in (structured) natural language making them easy to understand by non-technical stakeholders. They thus present advantages for communication. Finally their dynamic and process oriented nature can be used in a systematic transformation process for the configurator design [19]. All of these elements bring potential positive contributions with the use of BDD and needs to be more formally studied. This paper presents a first attempt towards such a research.

2 LITERATURE STUDY

2.1 Agile and Scrum

Scrum is an agile software development methodology. The *Agile Manifesto* outlines the values and principles that should be supported by the various agile processes applied in software development. Agile principles emphasize customer satisfaction, change and collaboration between domain experts and developers

[1] Mechanical engineering department, Technical University of Denmark, Denmark, email: sashaf@dtu.dk

[20]. Rubin [21] highlighted that with an agile approach, the team starts by creating a product backlog, which is a prioritized list of the features and other capabilities that need to be developed. Guided by the product backlog, team members address the most important or highest priority items first; priority is based on various factors, but delivered business value is most often the first priority. Scrum is an agile approach for developing innovative products and services [21].

Scrum facilitates cross-team coordination and collaboration [22]. Vlietland et al. (2016) determined that Scrum improves coordination through additional events, such as interteam sprint planning meetings, interteam daily Scrums, interteam product refinements and interteam sprint reviews. A Scrum development life cycle normally consists of short iterations of two to four weeks, an approach that enables swift feedback from software users and related stakeholders regarding the developed solution [19], [23].

2.2 Behavior-driven Development

In the movement of agile development, Test-Driven Development (TDD) has been around for a long time and can be traced back to eXtreme Programming practices developed in the late 1990s [24]. In particular, TDD employs so-called acceptance tests as the starting point for the development process to address some of the challenges related to Scrum. Following [25], these TDD relies on two simple principles:

- *Don't write any code until you've written a failing test that demonstrates why you need this code.*
- *Refactor regularly to avoid duplication and keep the code quality high.*

Behavior Driven Development (BDD) has been proposed as a result of the problems that arose with TDD when applying agile software practices [26]. It should be noticed that the language used for describing the tests, i.e. class names and operation names, plays an important role both for writing test cases and for finding bugs in case of a failed test. Inspired by [27], for this purpose BDD uses natural language as a ubiquitous communication mean to describe the acceptance tests by means of scenarios.

The cornerstone of TDD is the idea of writing a unit test before writing the corresponding code. However, BDD is much more than ensuring that every user story has a corresponding set of unit tests; BDD is also about writing specifications, as opposed to tests. In BDD, as an agile software development technique, acceptance tests are written in natural language in order to ensure a common understanding between all members of the project [28]. Consequently, as the first step, the sentences are mapped to actual source code [28].

The shift from TDD to BDD is subtle but significant. Instead of thinking in terms of verification of a unit of code, the focus is on specifying how that code should behave, i.e., what it should do [25]. In order to be sure that of building code that matters, there is a need for specifications that describe what the code should do and how to relate it directly to the business requirements.

Figure 1 shows an example of the BDD flow as it is employed in a specific tool [26].

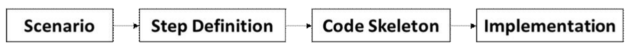


Figure 1. Behavior Driven Development flow

In BDD, as compared to TDD, the task is to write a specification of system behavior that is precise enough for it to be executed as code [29]. More specifically, the whole point of BDD is to ensure that the real business objectives of stakeholders are met by the software we deliver. If stakeholders are not involved, if discussions are not taking place, BDD is not going to work. BDD yields benefits across many important areas such as: (1) Building the right thing, (2) Reducing the risks, (3) Evolving the design [29].

The first Phase in BDD is to understand the business goals and defining features [25]. Vision statement templates make it possible to have a well-defined set of business goals. For a good product vision statement, Moore at al. [30] propose the following contents of a template:

- For (Who will benefit from this product?)
- Who (What do they need?)
- The (What sort of thing are you proposing?)
- That (What makes it so cool?)
- Unlike (What are you competing against?)
- Our product (Why customer prefer your solution?)

Secondly, the features have to be illustrated in natural language to execute the specifications. Consequently, the scenario has the structure [25]:

- Given [context, initial conditions]
- When [event occurs]
- Then [outcome]

There are several studies investigating how to automate all these scenarios such as Cucumber or Jbehave [25].

2.3 Requirement artifacts in traditional PCS projects: the Product Variant Master

Generic product structures can be illustrated using the so-called "product variant master" (PVM) notation, which in many cases has functioned as a common language between different product, process and IT experts [3]. Different definitions of the PVM notation have been proposed, and among them the definition of the PVM notation by Haug [11] is one of the most extensive and formalized. A principal example of the PVM technique used to model a toy car solution space is shown in Figure 2. On the left side of a PVM model, part-of-structure is shown, and on the right side, kind-of structure is shown. Classes (typically components and assemblies) are represented by circles and may include attributes and constraints (or rules).

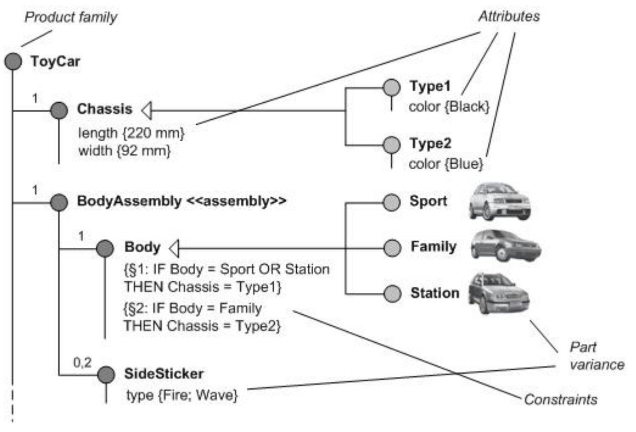


Figure 2. Product variant master (ToyCar example)

3 PROBLEM STATEMENT

Using Scrum for PCS development introduces both potential benefits and challenges (as for general IT projects in Table 1). One of the challenges is the level of knowledge complexity in PCS that has to be communicated clearly to all Scrum team members in terms of all attributes, constraints and acceptance criteria. An additional challenge from Scrum concerns the lack of visualization (modelling techniques [3]) for product structure. BDD supports Scrum with vision statements [30] to be able to demonstrate the product details and even user interface step by step. Another related challenge is the testing of the PCS as assessing PCSs implies to assess system features with respect to the many possible data and system outputs that might occur when a user is interacting with them. Testing PCSs is an arduous testing activity due to the wide range of user tasks and the different combinations of testing data. BDD, as an add-on of user stories, supports the testers with the detailed defined acceptance criteria.

In short, these concerns bring us three main guidelines for using Scrum methods in PCS projects:

- Formalize user requirements in such a way to provide testability in an ever-changing environment;
- Guarantee consistency between user requirements and their representation in multiple artefacts during development phase; and
- Lay on a validation approach that could be reused to ensure such a consistency for the artefacts along the project.

4 RESEARCH METHOD

The aim of this paper is to test the application of BDD in PCS projects in real case projects and gather the data regarding the BDD application. Firstly, we review the literature of BDD to gain deeper understanding of its definition and steps. Secondly, we would apply the findings from literature regarding BDD to the Scrum management in PCS. The authors' ultimate goal is to outline what the contribution of BDD to PCS can be and discuss its importance in promoting the collaboration and communication of knowledge within the organization.

Based on the mentioned challenges in PCS projects, BDD can be an effective solution to improve the definitions of different features and testing the codes in PCS projects next to the user stories. Hence, we posit the following three propositions:

Proposition 1: (expressiveness) BDD allows expressing all of the necessary constraints required for documenting a Configurator using Scenarios.

Proposition 2: (performance) BDD represents an effective approach for communicating the product specifications when implementing PCS as a replacement for PVM.

Proposition 3: (acceptance) BDD represents an effective and practical approach (requirements artifacts comparison) for unit testing of the implemented features in development and testing phases of PCS projects.

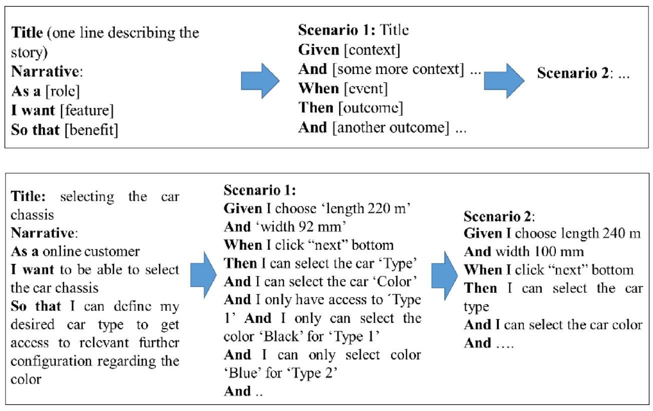
We use a qualitative exploratory design based on multiple data sources: requirements artifacts, workshops, interviews and participant observation [31], [32]. The study is ongoing in one case company using Scrum method for developing PCS for more than 4 years. Workshops are conducted to train the team for BDD to be implemented as part of the Scrum. Finally, feedback meetings are held as semi-structured interviews to collect knowledge about the team's satisfaction with BDD. Table 1 summarizes some of the details of the case projects.

Table 1. Scrum practice and defined roles and activities in Cases 3 and 4

Projects	***Case 1***	***Case 2***	***Case 3***
Time frame (months)	11	8	9
Number of iterations during the project	10	9	9
Roles	• Product owner: the stakeholders' manager • Project manager • Scrum master • Development team, including: 1 application manager, 1 project manager, 2 configuration engineers, 2 developers, 1 tester • End users		
Activities	• Backlog grooming • Sprint planning • Sprint backlog • Daily Scrum • Sprint review		
Artefacts (main specifications)	• Product goals and product backlog item (story) • Product backlog and stakeholders' requirements (list of user stories) • Testing (acceptance criteria in user stories)		
Planning approach	• Daily Scrum • Sprint planning • Sprint review • Feedback meetings		
Specific roles of meeting participants	• Same as the project roles, plus: • Product owner: 1 • Scrum master: 1 • Tester: 1		

5 OUTLOOK

It is yet an open question, how much BDD can contribute in the development phase of PCS projects. As PCS projects own their specific challenges, there is a need for further studies to test BDD influence specifically for PCS projects. The results from case studies and interviews will serve to verify or falsify the mentioned hypothesis when the study is accomplished. The structure of a user story presented to the case projects is demonstrated bellow:



Title (one line describing the story)
Narrative:
As a [role]
I want [feature]
So that [benefit]

Scenario 1: Title
Given [context]
And [some more context] ...
When [event]
Then [outcome]
And [another outcome] ...

Scenario 2: ...

Title: selecting the car chassis
Narrative:
As a online customer
I want to be able to select the car chassis
So that I can define my desired car type to get access to relevant further configuration regarding the color

Scenario 1:
Given I choose 'length 220 m'
And 'width 92 mm'
When I click "next" bottom
Then I can select the car 'Type'
And I can select the car 'Color'
And I only have access to ´Type 1' **And** I only can select the color 'Black' for 'Type 1'
And I can only select color 'Blue' for 'Type 2'
And ..

Scenario 2:
Given I choose length 240 m
And width 100 mm
When I click "next" bottom
Then I can select the car type
And I can select the car color
And ….

Figure 3. BDD structure of a user story presented for PCS projects in the workshops (Car example from Figure 1)

Concerning the adoption next to user stories narrative as a replacement for PVM and the vocabulary proposed in the ontology, an advantage is that requirements and tests in user stories are kept in a natural and high-level language. Based on Figure 3, the team should be able to demonstrate the prototype for user interface. The evaluation criteria of using BDD approach in the team can be considered as:

1. To support enterprise modeling within agile (user centric) methods
2. To estimate different scenarios, architecture and integration
3. Correct evaluation of available attributes and constraints for the product
4. To support the analysis of requirements communication
5. To analyze the compatibility with business modelling
6. To prototype the desired user interface

REFERENCES

[1] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-Based Configuration From Research to Business Cases*. Newnes: Morgan Kaufman, 2014.

[2] C. Forza and F. Salvador, *Product information management for mass customization: connecting customer, front-office and back-office for fast and efficient customization*. New York: Palgrave Macmillan, 2007.

[3] L. Hvam, N. H. Mortensen, and J. Riis, *Product customization*. Berlin Heidelberg: Springer, 2008.

[4] A. Felfernig, S. Reiterer, F. Reinfrank, G. Ninaus, and M. Jeran, “Conflict Detection and Diagnosis in Configuration,” in *Knowledge-Based Configuration: From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufman, 2014, pp. 73–87.

[5] L. L. Zhang, “Product configuration: a review of the state-of-the-art and future research,” *International Journal of Production Research*, vol. 52, no. 21, pp. 6381–6398, Aug. 2014.

[6] A. Haug, L. Hvam, and N. H. Mortensen, “Definition and evaluation of product configurator development strategies,” *Computers in Industry*, vol. 63, no. 5, pp. 471–481, Jun. 2012.

[7] S. Shafiee, K. Kristjansdottir, and L. Hvam, “Business cases for product configuration systems,” in *7th international conference on mass customization and personalization in Central Europe*, 2016.

[8] S. Shafiee, K. Kristjansdottir, L. Hvam, and C. Forza, “How to scope configuration projects and manage the knowledge they require,” *Journal of Knowledge Management*, vol. 22, no. 5, pp. 982–1014, 2018.

[9] A. Felfernig, G. E. Friedrich, and D. Jannach, “UML as domain specific language for the construction of knowledge-based configuration systems,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 4, pp. 449–469, 2000.

[10] S. Shafiee, L. Hvam, A. Haug, M. Dam, and K. Kristjansdottir, “The documentation of product configuration systems: A framework and an IT solution,” *Advanced Engineering Informatics*, vol. 32, pp. 163–175, 2017.

[11] A. Haug, “The illusion of tacit knowledge as the great problem in the development of product configurators,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 26, no. 1, pp. 25–37, Dec. 2010.

[12] C. Forza and F. Salvador, “Product configuration and inter-firm coordination: An innovative solution from a small manufacturing enterprise,” *Computers in Industry*, vol. 49, no. 1, pp. 37–46, Sep. 2002.

[13] A. Myrodia, K. Kristjansdottir, S. Shafiee, and L. Hvam, “Product configuration system and its impact on product’s life cycle complexity,” in *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2016, pp. 670–674.

[14] E. Sandrin, “An empirical study of the external environmental factors influencing the degree of product customization An Empirical Study of the External Environmental Factors Influencing the Degree of Product Customization,” no. December 2016, 2017.

[15] B. Selic, “Agile Documentation, Anyone?,” *IEEE software*, vol. 26, no. 6, 2009.

[16] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[17] J. Cho, “A Hybrid Software Development Method for Large-Scale Projects: Rational Unified Process with Scrum,” *Issues in Information Systems*, vol. 10, no. 2, pp. 340–348, 2009.

[18] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, “Unifying and extending user story models,” in *International Conference on Advanced Information Systems Engineering*, 2014, pp. 211–225.

[19] Y. Wautelet, S. Heng, S. Kiv, and M. Kolp, “User-story driven development of multi-agent systems: A process fragment for agile methods,” *Computer Languages, Systems and Structures*, vol. 50, pp. 159–176, 2017.

[20] F. Paetsch, A. Eberlein, and F. Maurer, “Requirements engineering and agile software development,” *Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pp. 308–313, 2003.

[21] K. S. Rubin, *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[22] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, “The agile requirements refinery: Applying SCRUM principles to software product management,” *Information and Software Technology*, vol. 53, no. 1, pp. 58–70, 2011.

[23] J. Vlietland, R. Van Solingen, and H. Van Vliet, “Aligning codependent Scrum teams to enable fast business value delivery: A governance framework and set of intervention actions,” *Journal of Systems and Software*, vol. 113, pp. 418–429, 2016.

[24] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[25] S. J. Ferguson, *BDD in Action: Behavior-driven development for the whole software lifecycle*. Manning, 2015.

[26] D. North, “Behavior Modification: The evolution of behavior-driven development,” *Better Software*, vol. 8, no. 3, 2006.

[27] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[28] M. Soeken, R. Wille, and R. Drechsler, “Assisted behavior driven development using natural language processing,” in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2012.

[29] S. Nelson-Smith, *Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code*. O’Reilly Media, Inc., 2013.

[30] G. A. Moore and R. McKenna, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*. HarperBusiness, 2014.

[31] A. H. Van de Ven, “Nothing is quite so practical as a good theory,” *Academy of Management Review*, vol. 14, no. 4, pp. 486–489, 1989.

[32] D. M. McCutcheon and J. R. Meredith, “Conducting case study research in operations management,” *Journal of Operations Management*, vol. 11, no. 3, pp. 239–256, Sep. 1993.