Disclosing the impact of BDD scenarios quality on Continuous Software Engineering

Gabriel Oliveira, Sabrina Marczak
Computer Science School, PUCRS
Porto Alegre, Brazil
gabriel.pimentel@acad.pucrs.br,
sabrina.marczak@pucrs.br

***Abstract*—Behavior-Driven Development (BDD) is a set of software engineering practices which uses a ubiquitous language, one that business and technical people can understand, to describe and model a system by a series of textual scenarios. Those scenarios serve not only as the project documentation but also as executable steps that specialized tools use to verify if the product has an acceptable set of behaviors. BDD tools can be used on continuous integration environment, thus enabling the documentation to be more effectively used during development and guaranteeing that changes on it are properly reflected on the product tests. Thus, in this position paper, we argue that BDD is a practice that supports Continuous Software Engineering by providing documentation based tests and straightening the boundaries between testing activities and coding. Our intuition leads us to believe that the value of those documentation based scenarios is connected with how well they convey and document the details discussed by the team about the behaviors needed to fulfill customer needs. Therefore, making sure that only "good" scenarios are used by constantly inspecting them should be an important activity during a software life cycle. Given the lack of studies addressing the problem of what makes a "good" BDD scenario, we take inspiration on the criteria used to evaluate other types of requirements (like use cases or user stories) to guide us on the reflection about how those concepts can be useful to BDD scenarios. Additionally, this paper reports on our experience with novice requirements writers and the pitfalls involved in the evaluation of scenarios during on on-going study.**

Keywords*—*BDD, Behavior-Driven Development, requirements quality, quality inspection.

I. Introduction

Behavior-Driven Development (BDD) is a set of practices that bring business analysts, developers, and testers together to collaboratively understand and define executable requirements, in the form of scenarios, together. According to Smart[1], those scenarios use a common language that allows for an easy, less ambiguous path from end-user requirements to usable, easy to automate tests. These tests specify how the software should behave and guide the developers in building a working software with features that really matter to the business. This set of practices tackles two common problems in software engineering: not building the software right and not building the right software. The first problem is covered due to the increase collaboration between members of the technical team to write well-crafted and well-designed software through the creation of executable examples. Those automated tests serve the dual purpose to demonstrate to clients that the new features have an acceptable set of behaviors and enhance the regression test suite that runs in a continuous fashion to safeguard the product from the development team's future changes. The second problem is covered due to the ubiquitous language used on those executable examples, that enhances the comprehension of business people about how the feature would solve their problems and reduces the chance of the team failing to understand what features the business really needs - thus ending up with a product that nobody needs.

Due to that ability to bring business and technical people together to collaboratively and continuously work on the same set of scenarios, we claim that BDD supports some of the Continuous * practices described by Fitzgerald and Stol [2]. The ability to have a model, built upon a set of scenarios, that serves as an executable documentation helps on shortening the knowledge gap between different roles among the team, thus improving Continuous testing activities. Additionally, the model serves as a technical documentation that maps the test coverage of each system feature, thus also helping with Continuous planning activities. Also, the complaint that acceptance test is perceived as too expensive [3] is mitigated by the simplification of acceptance tests creation - each line of documentation is a step to be taken on an acceptance test.

Even with those benefits to Continuous Software Engineering, little attention is being given to the quality of scenarios. It's well known that bad requirements are one of many potential causes of a project failure [4] and that the complete, accurate and concise documenting of requirements is of vital, perhaps paramount importance within software development, and errors made in this phase are often considered the most difficult to solve and most costly to fix [5]. Bad scenarios documentation can lead to misleading information that will negatively impact the tests ability to reflect the system coverage and the team confidence on them. Thereby, the Continuous testing and planning activities are negatively impacted as well.

Also, bad scenarios may impact directly on the quality of the acceptance tests execution. Neely and Stolt [6] report on the lack of trust on flaky tests, those that pass or fail based on race conditions or test execution ordering. Bad acceptance tests derived from bad scenarios may harm the team trust on Continuous Integration practice.

We judge it necessary to better understand how we can prevent BDD scenarios, that brings many benefits to the development team, to suffer from those problems caused by bad documentation. We believe continuous verification practices, such as pre-defined checklists reported by Fitzgerald and Stol [2], may achieve that. Therefore, this paper proceeds

as follows. Section 2 reviews the set of practices involved on BDD, their importance to Continuous * practices from Fitzgerald and Stol [2] and reflects upon the lack of writing quality definition by observing some other requirements formats. Section 3 presents the study design we performed to acquire a deeper understanding of how quality attributes could be used to validate BDD scenarios and our impressions during this study. Section 4 concludes this paper by summarizing our opinions and outlining some directions for future research.

II. Background

A. Behavior-Driven Development

Behavior-Driven Development is an umbrella term that encapsulates a set of practices that uses scenarios as a ubiquitous language to describe and model a system [1]. Scenarios are expressing in a format known as Gherkin, that is designed to be both easily understandable for business stakeholders and easy to automate using dedicated tools. According to the author, bringing business and technical parties together to talk about the same document helps to build the right software (the one that meets customer needs) and to build it right (without buggy code). According to Smart, using conversation and examples to specify how you expect a system to behave is a core part of BDD.

Knowing that a BDD scenario is a format to represent acceptance tests, it fulfills the role of the Confirmation term defined by [7]. He described that the Card (most commonly written in user story format on agile methodologies) represent customer requirements rather than document them, has just enough text to identify the requirement and to remind everyone what the story is. The Conversation is an exchange of thoughts, opinions, and feelings. It is largely verbal but can be supplemented with documents. The best supplements are examples and the best examples should be executable. They're representations of the Confirmation, a way to customers tell to developers how she will confirm that they've done what is needed in the form of acceptance tests. That Confirmation, provided by those examples, is what makes possible the simple approach of card and conversation. When the conversation about a card gets down to the details of the acceptance test, the customer and programmer settle the final details of what needs to be done. When the iteration ends and the programmers demonstrate the acceptance tests running, the customer learns that the team can, and will, deliver what's needed.

The importance of acceptance tests execution to Continuous Integration is well known. Humble and Farney [3] states that the cost of properly created and maintained automated acceptance test suite is much lower than that of performing frequent manual acceptance and regression testing, or that of the alternative of releasing poor-quality software. Since BDD scenarios are easily automated by tools like JBehave[1] and Cucumber[2], the scenarios created and maintained can be called executable specifications [1] [3].

According to Smart [1], for teams practicing BDD, the requirements and executable specifications are the same thing; when the requirements change, the executable specifications are updated directly in a single place. Therefore, it helps to lower the burden on the team to constantly updating test scenarios after changes on specification and to improves the integration between the coders and testers tasks, thus helping improve on the Continuous testing mindset described by Fitzgerald and Stol [2].

The use of acceptance tests as documentation is also highlighted by Neely and Stolt [6] on their experience at Rally Software. When planning stories that require modification of existing code, they state the lengths of documenting existing tests can be used as guidance when discussing the story details, in order to provide visibility to the team about the test coverage of the code, allows QA and developers to close gaps and gives the team a level of confidence that that part of the application has been exercised by the automated test suite. As the use of acceptance tests can tighten the integration between planning and execution, it could help to fulfill the vision of Continuous planning described by Fitzgerald and Stol [2].

However, little attention is being given to the quality of that written documentation on BDD scenarios format. The only guide practitioners have to validate their scenarios are taken from tips based on few examples described by Smart [1] in his book, as follows: the scenarios steps expressiveness, focused on what goal the user want to accomplish and not on implementation details or on screen interactions (writing it in a declarative way and not on an imperative way); the use of preconditions on the past tense, to make it transparent that those are actions that have already occurred in order to begin that test; the reuse of information to avoid unnecessary repetition of words; and the scenarios independence. The author specifies examples of good and bad scenarios in order to demonstrate those characteristics.

We believe that, if the Confirmation is not a good representative of the details discussed in Conversations by the team and the customer, the simple approach of writing customer needs on Cards is not effective. With the lack of criteria to validate acceptance tests on BDD scenarios format, we look upon other requirements formats.

B. Quality on other requirements formats

The Business Analyst Body of Knowledge (BABOK) [8] says that a requirement is either a condition or capacity necessary to solve a problem or reach a goal for an interested party or some characteristic that a solution or component should possess or acquire in order to fulfill some form of contract. The 3rd edition [8] states that while quality is ultimately determined by the needs of the stakeholders who will use the requirements or the designs, acceptable quality requirements exhibit many characteristics. The second edition [8] describes eight characteristics a requirement must have in order to be a quality one, as follows: cohesion, completeness, consistency, correction, viability, adaptability, unambiguity, and testability. The third edition [9] brings nine: atomic, complete, consistent, concise, feasible, unambiguous, testable, prioritized and understandable.

Both editions [8] [9] define what each characteristic means, but does not provide any measurement guidance. Cockburn [10] seems to take inspiration on those attributes to define rules on how to validate use cases, a requirement format that

[1]https://jbehave.org/

[2]https://cucumber.io/

captures a contract between the stakeholders of a system about its behavior and describes the system's behavior under various conditions by interacting with one of the stakeholders (the *primary actor*, who want to perform an action and achieve a certain goal).

Use cases quality is also discussed in details by Phalp et. al [5], that summarizes prior works on that area and proposes refined rules based on discourse process theory, such as avoiding the use of pronouns, use active voice over passive one, achieve simplicity trough avoiding to use negative forms, adjectives and adverbs and use of discourse cues and the effect of readers background and goals. Also, desirable quality attributes of use cases are listed, that may be suited for certain project phases but not others as follows: standard format, completeness, conciseness, accuracy, logic, coherence, the appropriate level of detail, consistent level of abstraction, readability, use of natural language and embellishment. The authors create rules, that should be enforced and must be obeyed, and guidelines, which indicate an ideal that cannot always be followed, that could best produce those attributes.

Most agile methodologies tend to not use traditional requirements or use cases, but represents requirements using user stories, that fill the Card role described by Jeffries [7]. For Cohn [11], a user story describes functionality that will be valuable to either a user or purchaser of a system or software. Lucassen et. al [12] summarize that user stories only capture the essential elements of a requirement: *who* it is for, *what* it expects from the system, and, optionally, *why* it is important.

According to Lucassen et. al [12], the number of methods to assess and improve user story quality is limited. Existing approaches to user story quality employ highly qualitative metrics, such as the heuristics of the INVEST (Independent-Negotiable-Valuable-Estimable-Scalable-Testable) framework described by Cohn [11]. Due to that fact, Lucassen et. al [12] define additional criteria to evaluate user stories on their QUS Framework, as follows: atomic, minimal, well-formed, conflict-free, conceptually sound, problem-oriented, unambiguous, complete, explicit dependencies, full sentence, independent, scalable, uniform and unique.

III. Evaluating BDD scenarios trough quality attributes

Paragraphs ideas/order:

- Attribute list used and motivation
- What was needed when evaluating (presence/absence of attribute on a qualitative explanation, attribute interpretation)
- Explain study design (2 products, one requirement format per product, cross design)
- First product description (allergy app)
- Second product description (allergy app)
- Participants evaluation of each other scenarios - design motivation on who evaluates who

IV. Discussion & Conclusion & Future Studies

Paragraphs ideas/order:

- writing difficulties/easiness between use cases and scenarios
- rigour on writing importance,
- solution knowledge importance (write scenarios are easy, but not writing good scenarios that the team could use)
- bad attributes to analyze scenarios (maintainability, completeness, atomicity)
- results (?)

References

[1] J. Smart, *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle*. Manning Publications, 2014.

[2] B. Fitzgerald and K.-J. Stol, "Continuous software engineering and beyond: Trends and challenges," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, 2014.

[3] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[4] The standish group, "CHAOS," 2015.

[5] K. Phalp, A. Adlem, S. Jeary, J. Vincent, and J. M. Kanyaru, "The role of comprehension in requirements and implications for use case descriptions.," *Software Quality Journal*, 2011.

[6] S. Neely and S. Stolt, "Continuous delivery? easy! just change everything (well, maybe it is not that easy)," in *Agile Conference*, 2013.

[7] R. Jeffries, "Essential xp: Card, conversation, confirmation," 2001.

[8] IIBA, *A Guide to the Business Analysis Body of Knowledge (BABOK Guide) 2nd Edition*. International Institute of Business Analysis, 2009.

[9] IIBA, *A Guide to the Business Analysis Body of Knowledge (BABOK Guide) 3rd Edition*. International Institute of Business Analysis, 2015.

[10] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[11] M. Cohn, *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc., 2004.

[12] G. Lucassen, F. Dalpiaz, J. VanDerWerf, and S. Brinkkemper, "Forging high-quality user stories: Towards a discipline for agile requirements," in *International Requirements Engineering Conference*, 2015.