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 our reflection about how known quality attributes can be useful to BDD scenarios. Additionally, this paper reports on our experience with novice BDD scenarios writers and the pitfalls involved in the evaluation of scenarios during an on-going study.**

***Keywords*—Continuous verification, Continuous testing, Behavior-Driven Development, documentation quality.**

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. Smart [1] states that 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 Software Engineering practices described by Fitzgerald and Stol [2], also referred as Continuous *. 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's feature [3], thus also helping with Continuous Planning activities. Also, the complaint that acceptance test is perceived as too expensive [4] 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 is well known that bad requirements are one of many potential causes of a project failure [5] 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 [6]. 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, thus bringing problems to teams like the one on Rally Software, reported by Neely and Stolt [3]. 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 [3] report on the lack of trust on flaky tests, those that pass or fail based on race conditions or due to test execution ordering dependency. Bad acceptance tests derived from bad scenarios may harm the team's 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 on BDD scenarios while 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 perceptions about the on-going study and outlining some directions for future research.

II. Background

A. Behavior-Driven Development as a Continuous * Practice

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 expressed in a format known as Gherkin, that is designed to be both easily understandable for business stakeholders and easy to automate using dedicated tools. Smart [1] argues that 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). He also says that using conversation and examples to specify how one expects a system to behave is core to BDD.

It's a known fact that BDD scenario is a format to represent acceptance tests [7], as it fulfills the role of the Confirmation term defined by Jeffries [8]. 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 are representations of the Confirmation, a way to customers tell to developers how she will confirm that they have 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 is needed.

The importance of acceptance tests execution to Continuous Integration is well known. Humble and Farney [4] 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] [4].

[1]https://jbehave.org/

[2]https://cucumber.io/

For teams practicing BDD, the requirements and executable specifications are the same thing [1]. When the requirements change, the executable specifications are updated directly in a single place. Therefore, it helps to lower the burden of constantly updates on test scenarios when changes on the specification happens, thus making it easier for the team to have a Continuous Testing mindset, a desirable culture pattern described by Fitzgerald and Stol [2].

The use of acceptance tests as documentation is also highlighted by Neely and Stolt [3] on their experience at Rally Software. When planning stories that require modification of existing code, they state that the lengths of documenting existing tests can be used as guidance when discussing the story details. It helps to enhance the team's visibility about code's test coverage and also allows QA and developers to close existing gaps, boosting the team level of confidence on that part of the application. 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. To the best of our knowledge, 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 Criteria for other Requirements Formats

The Business Analyst Body of Knowledge (BABOK) newest edition [9] 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. It lists some characteristics a requirement must have in order to be a quality one, as follows: atomic, complete, consistent, concise, feasible, unambiguous, testable, prioritized, and understandable. A slightly different list is found on a prior version [10], as follows: cohesion, completeness, consistency, correction, viability, adaptability, unambiguity, and testability. Although the characteristics' meaning is defined, no measurement guidance is given.

Cockburn [11] seems to take inspiration on those attributes to define rules on how to validate use cases, a requirement format that 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). Those rules are summarized in a pass/fails questionnaire to be applied on use cases - good use cases are those that yield an "yes" answer to all of them.

Use cases quality is also discussed in details by Phalp and colleagues [6] who summarize prior work on that area and propose 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. With those quality attributes in mind, 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 [8]. For Cohn [12], a user story describes functionality that will be valuable to either a user or purchaser of a system or software. Lucassen et. al [13] 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.

Lucassen et. al [13] argue that 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 [12]. Due to that fact, Lucassen et. al [13] 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 Quality

Before creating a quality questionnaire similar to the one used on Cockburn's use cases [11] or the refined quality rules proposed by [6], we must understand what criteria are important to evaluate the quality of BDD scenarios. Therefore, we must list the quality attributes that would work with BDD scenarios, in a similar way that traditional attributes [10] [9] work with requirements documents and INVEST heuristic [12] works with user-stories. In preparation to creating that listing, we first seek to gain insight on whether the existing quality attributes would work with BDD scenarios, and if they do not, which ones would be a good fit for such purposes.

For that reason, we put together the traditional attributes from both BABOK editions [10] [9] and the INVEST framework [12] on an alphabetical ordered list and organized a study to evaluate how those attributes would be used to evaluate BDD scenarios. With that goal in mind, we created two fictional products (A and B) and asked graduate students to perform the following sequential tasks using the provided alphabetical ordered list of quality attributes:

TABLE I. Sample of study organization

ID	Product A format	Product B format	Product A evaluation	Product B evaluation
1	UC	US + BDD	Student 2	Student 4
2	US + BDD	UC	Student 1	Student 3
3	UC	US + BDD	Student 4	Student 2
4	US + BDD	UC	Student 3	Student 1

1) Develop functional requirements for product A with requirement format assigned;
2) Develop functional requirements for product B with another requirement format assigned;
3) Evaluate other student functional requirements for product A;
4) Evaluate other student (different from the item above) functional requirements for product B;

Each student was assigned a requirements format to perform tasks 1 and 2. They had to use either user stories and BDD scenarios (US + BDD) or use cases (UC). Expressing behaviors using BDD scenarios seems a comparable activity to perform the same task using use cases, for the following intuitive reasons: both seems to prefer a more declarative way to describe a behavior, rather than imperative, to stay detached from the implementation details [11] [1]; and a use case execution path seems to be a BDD scenario on a different format - on Cockburn's book [11], the main scenario is always described. With those reasons in mind, we judged it necessary to understand how the same person uses the same attributes on both formats in order to clarify their motives and rationale. We let the scope of the analysis - if one should evaluate each scenario or use case separately or together with the others from the same feature - open to interpretation to measure this aspect. Additionally, in order to avoid a person's writing style to affect the evaluation of a scenario or use case, we made sure that the same student would not read two requirements formats from the same colleague when performing tasks 3 and 4. Also, it was desirable that the student gets the opposite requirement format to evaluate a given product. Table I exemplifies how four students would perform the tasks given.

The products were presented to the students in a product vision board format [14]. On that occasion, they had the chance to ask questions, straighten their understanding on how to solve the business problem presented, and collaboratively draft some high-level features. After this initial discussion, we would take their suggestions and create the final high-level features. Finally, a new discussion round was performed so the students could read the features and had the chance to validate them, negotiate their details, decide whether each feature should be part of the first release of the product or not. Those discussion sessions were planned to give them all the same understanding of what the product should be and how it should behave, as a means to guarantee a common ground for the students to produced use cases, stories, and BDD scenarios. After the last round of discussions, the students then proceeded to perform tasks 1 to 4 in an individual manners.

Product A aim was to develop a mobile app to help people with food allergy find places to eat free of ingredients that cause them allergy and distress. The user should be able to

have an easy way to indicate which kind of food allergy or restrictions one has. There were two target groups: Users with any kind of food allergy that are quickly (e.g., while driving, walking, chatting with another person, etc) looking for a place to eat; and Customers, owners of restaurants, whose company business reputation would be improved if they used this new client search channel. Product B was a social-network website that aims to bring low cost book readers and resellers (who sell second-hand books) together. Readers would fill their profiles with genre interests and literature thoughts in exchange of badges, a higher fame status, and occasional promotions directly to them. Those social network interactions would provide resellers with enough information to direct their marketing efforts and promotions to the right subset of users.

IV. Perceptions about the Study

The above mentioned study has finished its execution phase, with all the 15 students having performed their four assigned tasks, but the analysis of the data collected throughout the completion of tasks 1 to 4 and the students interpretation about quality attributes is still on-going. However, we could already take some degree of insight from our notes during the study and from a final 2 hours long discussion round with the students once they concluded performing their work.

First of all, the list of chosen attributes generated confusion on the evaluation of BDD scenarios. As it mixed traditional requirements characteristics from both versions of the BABOK [10] [9] and the INVEST heuristic provided by Cohn [12], some students could not see some attributes as different. For example, atomicity from BABOK [9] and independence from INVEST [12] were often seen as opposites, even if this is not always the case. One student reported that this confusion may have come from the difficulty to see bad examples of BDD scenarios, those specially built to demonstrate when an attribute fail.

Secondly, some attributes may not make sense to evaluate a single BDD scenario - completeness may represent how a set of scenarios demonstrate a complete set of examples to cover a user story, for example. As it was said before, we let the scope of the analysis open to interpretation on purpose. Thus, five students out of the 15 have joined the scenarios from a user story together before analyzing them, witch raises questions on what were the reasons that drove their decision. Follow up sessions need to be scheduled with those studies to search for the reasoning behind their evaluation approach.

Thirdly, the students reported that inputs/outputs are important on BDD scenarios to help on prioritization and testability. As they do no have use cases implementation details to help the development team estimate and analyze the new feature impact and on the system, using inputs/outputs on scenarios helps to improve prioritization and testability. Also, a few of them see that the lack of inputs/outputs can be seen as a lack of completeness attribute. Those perceptions are similar to the good examples given by Smart [1] on his book - however, the lack of input/output is not explicitly provided by him.

Fourthly, written rigour was judged as necessary by the students to re-enforce the team's common understanding during conversations with customers. As they stated, the writing of BDD scenarios is easier (due to the use of plain English language to represent behaviours) and could theoretically be done by anyone - however, they reported that someone who provides good examples is desirable to better demonstrate, through those example, to the development team how a new functionality can be put together with the existent ones.

V. Conclusion & Future Work

Our study perceptions seems to reinforces our belief that pre-defined list of quality attributes is important to guide the quality evaluation of BDD scenarios. A detailed data analysis is needed to confirm that thought. Still, those perceptions opened the debate questioning if a list of attributes alone is useful to achieve that goal or if other formats of checklist (e.g: direct questions, as those developed to use cases by Cockburn [11]) will yield better results. Additionally, the problem may be centered on the list of attributes itself - we know that some of them can be used to evaluate other requirements formats, but should they be used with BDD scenarios or other types of acceptance tests ? For example, execution stability is never mentioned as a quality concern, even with the known problems with flaky tests reported by Neely and Stolt [3]. Therefore, the concerns that

Therefore, our next steps to achieve our goal are to better understand how acceptance tests are evaluated, what problems may appear during their creation or use on a continuous engineering context and map those problems on BDD scenarios. This knowledge will give us confidence to build a question based checklist to avoid those problems.

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

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

[5] The standish group, "CHAOS." https://www.projectsmart.co.uk/white-papers/chaos-report.pdf, 2015. Visited in: Jan. 2017.

[6] 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.

[7] M. Gartner, *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley Professional, 2012.

[8] R. Jeffries, "Essential xp: Card, conversation, confirmation." http://ronjeffries.com/xprog/articles/expcardconversationconfirmation, 2001. Visited in: Jan. 2017.

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

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

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

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

[13] 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.

[14] R. Pichler, "The product vision board." http://www.romanpichler.com/blog/the-product-vision-board, 2011. Visited in: Jan. 2017.