2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)

Behavior-driven development as an approach to improve software quality and communication across remote business stakeholders, developers and QA: two case studies

André Scandaroli
Daitan
Brazil, Campinas
ascandaroli@daitan.com

Rodrigo Leite
Daitan
Brazil, Campinas
rleite@daitan.com

Aléxis H. Kiosia
Daitan
Brazil, Campinas
akiosia@daitan.com

Sandro A. Coelho
Daitan
Brazil, Campinas
scoelho@daitan.com

***Abstract*—In this work, we describe two of our experiences applying BDD: one at a large publisher of financial information and business news, and the other in a secure collaboration platform vendor. The team composition involves both a local team and an external service provider. We analyze two adoption processes, one driven by the product team, and the other driven by the development team. We cover the communication benefits and impacts on areas such as stakeholder alignment, software design, code refactoring, and bug fixing. We also discuss lessons learned, caveats and how we plan to expand this practice in the future.**

***Index Terms*—Distributed agile teams, Industrial offshoring and outsourcing experiences, Methods and processes, Social and Human aspects of software development.**

I. Introduction

Daitan[1] is a software service provider with headquarters in the US (San Ramon, CA) and operations sites in Canada (Victoria, BC) and Brazil (Campinas, SP). The projects involved in this paper, which clients are not mentioned openly due to non-disclosure agreement restrictions, fit in the category of nearshore services, where Daitan teams are located in Brazil and the client teams are located in the US, UK and APAC.

Techniques for reducing the communication gap between business and technical domains in software development are one of the main aspects of agile practices. Behavior-driven development [1] (BDD)[2] brings together strands from TDD and domain-driven design [2] into an integrated whole, making the relationship between these two powerful approaches to software development more evident. BDD aims to focus development on the delivery of prioritised, verifiable business value by providing a common vocabulary (also referred to as a Ubiquitous Language) that spans the divide between Business and Technology [3] [4].

This paper covers two experiences with two different projects that adopted BDD to some extent. Both are still ongoing projects. The former, project A, held with a large publisher of financial information and business news, and the latter, project B, held with a secure collaboration platform vendor. Both projects have a characteristic of leveraging nearshore services, having the contractor personnel based in Brazil and client personnel in the US, UK and APAC. In both cases, specific challenges motivating the adoption of BDD are detailed and explored. One of the key shared challenges is the distribution of team members in different geographic locations, having different time zones, different native languages, and other aspects inherent to the globally distributed teams nature that hardens proper communication and alignment in these teams [5] [6] [7].

The rest of the paper is organized as follows. Section II covers how each project team was structured as well as depicts the overall agile process in place in each team, highlighting motivators for BDD adoption. Section III describes the adoption of BDD in each project, highlighting what has been applied and how. Section IV lists which benefits the teams were able to identify because of the BDD adoption. Section V covers the trade-offs imposed by the BDD adoption as well as lists which improvements opportunities are available for each project. Section VI sums up both experiences and outlines key aspects, lessons learned and caveats.

II. Team Structure and Agile Process

Table I summarizes the relevant aspects of project A and B that we describe in the rest of the section.

A. Project A

For project A, the Daitan team is composed by 5 software developers in the Brazil operations site while the client team has the Product Owner (PO), a Technical Manager and a Systems Architect in two different locations (see Figure 1). Because in this project the development team have no specific members specialized in Software Quality Assurance (SQA), one of the major challenges of the development team is to make sure software is delivered with high level quality despite

[1]https://www.daitan.com
[2]https://behaviourdriven.org/

the lack of specialized SQA team members. Additionally, having teams distributed in three different locations, often poses a natural barrier to proper communication, among the most significant impacting aspects observed are long conference calls, lack of face-to-face engagement, eventual problems with audio quality, time zone differences, different English accents.

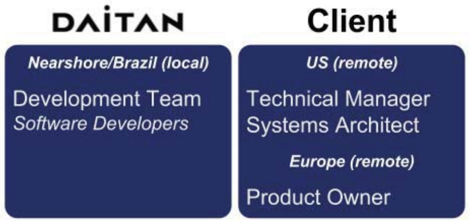

Figure 1. A depiction of Project A Team Structure

Project A agreed on using Scrum [8] [9] [10] to define their process, alongside with principles and practices borrowed from Agile Software Development[3], Lean Software Development [11], Kanban [12], Extreme Programming [13] (XP), DevOps [14], TDD, etc., in addition to BDD. Communication between the remotely distributed teams happens using online conference calls, instant messaging tools, online collaboration tools and emails.

The choice for BDD on project A came initially from the development team to help requirements clarifications and proper understanding before moving on to coding, in addition to help the test coverage of new features as well as bugs found. The basis for the development team suggesting BDD as a tool is due to various studies showing successful cases in test-driven development [15].

At first the client team showed some indifference in adopting BDD in addition to TDD as there were no evident and immediate value perceived. Nonetheless, the team was allowed to apply the technique.

B. Project B

For project B, the Daitan team is composed by 6 software developers and the Product Manager. The client team is composed by a Technical Manager and by the Systems Architect, in addition to 4 SQA engineers in two different locations (see Figure 2).

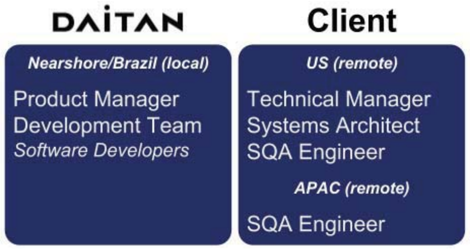

Figure 2. A depiction of Project B Team Structure

[3]https://agilemanifesto.org

The choice for BDD in project B came from a necessity to find a common language between the business and engineering team as they were facing communication problems.

Like all start-ups at first, the client wanted to ship the product as soon as possible to avoid losing market share. A perfect product from the quality standpoint was not a priority.

During the product introduction phase, the client had no substantial problems with their customers, but when the product entered the growth phase, including more and more new features, the product became less stable. Customers began to complain about product reliability and then the growth within installed base slowed down.

An initiative was created to identify the quality issues in the product and the low coverage of the test cases was the first issue found. That issue demanded the whole client company to consider the time for building the proper test cases in the product release planning to guarantee the quality of the product. The initiative worked at the beginning of the process, where the developers started creating unit tests for each part of the code. However, unit tests were not enough to improve the quality at the desired level.

Another problem was distance, since part of the quality team was remote (1 in the US and 3 in APAC), making communication more difficult. As the Product Specification was not always precise or contained ambiguity made the quality team have problems to write the test cases or test cases that were not covering what they had to.

A second initiative was created, and further analysis identified the problem as related to communication issues between the product managers and the developers. Most of the time, the product managers were talking to the customers to understand their problems, but when the developer teams received the requirement, in many cases the information had a wrong interpretation, which led developers to code business rules and test cases that did not reflect the original goal.

The business team had to find out a way to create a bridge over the chasm between the Product and Engineering teams. During the research, they figured out the product managers spoke a business language, and developers spoke a technical language, so if there were a common language between them, the problem would be solved. The solution found was the BDD.

III. BDD Adoption

A. Project A

As mentioned earlier, the client team was a bit indifferent with adopting BDD as suggested by the development team. The client was more keen to have unit test coverage and working software instead of investing valuable time in identifying BDD scenarios beforehand.

At some point in the initial sprints, the development team started applying techniques to identify acceptance criteria for each new user story, and automating BDD scenarios from them. The development team agreed that acceptance criteria should be listed in the definition of ready of each new

Table I
CHARACTERIZATION OF THE TWO PROJECTS

Aspect	**Project A**	**Project B**
Agile Process in use	Scrum based	Scrum based
Key initial motivator for adopting BDD	Lack of specialized SQA	Business vs technical teams communication gap
Business complexity [16] (high, medium, low)	Medium	Medium
Technical complexity [17] [18] (high, medium, low)	Medium	Medium
Technical team size	5 (local) 1 (remote)	6 (local) 4 (remote)
Technical team location	Local and remote	Local and remote
Business team size	2 (remote)	1 (local)
Business team location	Remote only	Local

user story or bug, and that all acceptance criteria must have been properly covered with automated BDD scenarios in the definition of done.

One key challenge of this project was that the engagement of the client team in the acceptance criteria identification was rather low, which imposed a lot of effort from the development team to hash out all possible unknowns as early as possible. Some impacts of the lack of engagement from the client team, especially related to business definitions, did appear in a few reworks and refactoring that had to happen later, as some criteria have not been properly uncovered ahead of time.

The development team leverages the BDD mindset during Scrum planning and refinement sessions, which helps foster objective discussions when clarifying requirements. These sessions happen in conference calls, where there is a natural barrier of communication especially when subjects are complex. The objectiveness imposed by the BDD approach are substantial for the efficiency and effectiveness of the sessions.

After a few sprints, when the product already had a good amount of features in it, the development team and the client team could observe the benefits of having applied BDD early in the project, especially when some significant architectural refactoring took place.

The main development language in use is Java and the development team chose Cucumber[4] as the BDD framework. The process defined to devise the BDD scenarios and carry on with development can be described as follows:

1) Product owner gathers the requirements from the customer.
2) Product owner generates requirements document with sample outputs and inputs.
3) The development team creates user stories based on those requirements.
4) The development team refines the user stories, adding acceptance criteria to the user story card using Gherkin[5].
5) Product owner and development team review the user stories in order to agree on the acceptance criteria, and to make adjustments to them as needed.
6) The development team adds BDD scenarios from the user story to the Cucumber suite and proceeds with traditional BDD/TDD development process, consulting the result of BDD scenarios execution for failed and passed scenarios, until development is done.

BDD scenarios are also added for functional bugs. In this case the process above is slightly different in steps 1 and 2, since the product owner already creates a bug card detailing the issue, from where the development team engages the process.

Another slight deviation from the process above happens when the development team gets more acquainted with a new set of requirements, which enables discussions of new scenarios not previously documented. This is done during a few Scrum ceremonies, such as plannings, reviews and refinements, between the development team and the PO. This kind of collaborative and open communication between development team and PO is the primary goal of BDD which leads to substantial improvement in the communication and collaboration between distributed teams.

In this project, products are essentially APIs, hence with respect to BDD scenario automation, the project has usually one Gherkin feature per endpoint. For each action on that endpoint, we have at least one Gherkin scenario.

With time, the team learned how to write steps, generic enough, so they could be reused on different scenarios, thus increasing the team speed even more.

B. Project B

In the beginning, there was resistance from both sides (product managers and developers) since each claimed more work. The product managers needed to study how to write the same business requirements following the BDD specification, which is a more verbose language. On the other hand, developers needed to study a BDD framework and understand how to create test cases. They realized that the process was more difficult than writing unit tests and would require more time.

However, as the top management proposed the idea since the quality of the product was compromised, a top-down demand was made for the whole client company to create the BDD culture and everybody accepted it.

As the product was developed using the Java language, they chose JBehave[6] as the BDD framework and a few rules were added to the process:

1) Product manager gathers the requirements from customer

[4]https://docs.cucumber.io
[5]https://docs.cucumber.io/gherkin
[6]https://jbehave.org

2) Product manager transforms requirements in user stories in the product specification document, detailing their acceptance criteria
3) Product manager writes a BDD table with the requirements related to the user stories.
4) Developers transcribe the BDD table on the story file provided by JBehave.
5) Developers map the steps of the stories in the Java Test Steps class.
6) Developers configure JBehave and JUnit[7] to integrate the story files with the Java Test classes.
7) Developers run the stories using their IDE (Integrated Development Environment), such as Eclipse[8].
8) Developers view the report result indicating whether the test has passed or failed.

The Product Specification is a document where the product manager writes the requirements following the user stories format and depicts the BDD table with the scenarios that reflect the stories. The BDD table follows a specific structure that helps the developers team to create the JBehave story files:

1) ID: an identifier number.
2) Given: it represents a precondition to an event.
3) When: it represents the occurrence of the event.
4) Then: it represents the outcome of the event.

The development team reads the Product Specification document to understand the requirements and transcribes the BDD table to the story file provided by the JBehave framework.

IV. COLLECTED BENEFITS

A. Project A

In an aspect that might be more related to testing in general than to BDD, with the tests written, the development team can be sure of the completeness and stability of the feature. All members of the project, be it a developer or a business person, remote or local, can not only be sure the user story is done when the new test scenarios are passing, but also that the new feature is not breaking any of the existent scenarios. Having a well established BDD culture improves the correctness ratio [4] and the confidence of all members that user stories are really done.

Furthermore, the scenarios serve as documentation of the expected behavior of the feature. This can be used to easily align the coverage of the requirements with the whole team. Having to design these scenarios beforehand can raise questions, which helps to refine the feature, uncover missing requirements and make sure all members of the team are truly aligned, generating high quality deliveries, despite not having specialized SQA team members and face-to-face engagement.

The speed of the team tends to grow over time as well, because if the steps are generic enough, they can be reused on other Gherkin scenarios, which also helps when there are more deep changes within the code without changes on the behavior, since a simple change on the affected step will be reflected on all related scenarios.

In summary, by using BDD, the development team improved the communication between local and remote members, while also improving the alignment with business people, and delivery speed, by high re-usability and high-acceptance of the tasks.

B. Project B

One of the benefits that the BDD culture brought to the client was to have all members of the team aligned regarding a feature.

Product managers tend to write requirements following a business language that works for the business team, but not necessarily for the development team. The BDD methodology created a culture that made the product managers write better user stories, establishing easier communication with the developers and building a bridge between the two worlds.

In the beginning, developers did not like the idea to write test cases in a more verbose way, but, when they realized they were creating a shield to protect them from themselves, they changed their mind. It is beneficial to have an environment where developers are not afraid to break the product when adding new features. Apart from the benefits that the common language brings to communication, the test suite became stable and matured what gives more confidence to developers.

Also, with the adoption of the BDD culture, communication among the remote teams improved as now they have a definite source of truth in an understandable language. The SQA team does not need to wait for the product manager to clarify the product specification as it now has a BDD scenarios section.

Regardless of where a team member is (local or remote), once we have a product specification document following the BDD language, the teams no longer find any problems regarding communication.

Furthermore, the BDD test cases were integrated into the CI/CD [14] infrastructure pipeline, responsible for accepting the new code into the master branch. If the new code breaks the BDD scenarios tests, it is not merged.

After applying this new methodology, the product becomes more stable, and features got more comprehensive by product managers, developers and SQAs as they found a ubiquitous language.

V. TRADE-OFFS AND IMPROVEMENT OPPORTUNITIES

A. Project A

Given the initial skepticism of the client team, the development team was challenged with some trade-offs during the adoption of BDD. For instance, the lack of engagement from the client led the design of the BDD scenarios to become a responsibility of the development team. In the beginning, that generated less generic scenarios and steps that are very dependent on the code, i.e. steps were designed using terms that could be related much more to the internal behavior of the application than to the behavior seen by users. For instance,

[7]https://junit.org
[8]https://www.eclipse.org

if the application uses a third-party API, business people will have no use for a Gherkin step describing that this API should be available.

With the design of the BDD scenarios emerging from the development team, often those scenarios are poorly defined, be it due to lack of details in the requirement document or to complete lack of descriptions, in case of bugs. Had the BDD scenarios been defined before the requirements by the client team, these uncertainties could have been extinguished before reaching the development team, saving time and guaranteeing the expected deliverables. It is worth noting that often the BDD scenarios designed by the development team were not always exact. Business people sometimes expected different behaviors than what was delivered. As an improvement on the process, if the BDD scenarios are well defined before the feature reaching the development team, this unexpected factor should be close to zero, because, according to Smart [4] it is easier to a business people to comprehend scenarios written in Gherkin, being so, they know exactly what is going to be delivered.

When designing the CI/CD infrastructure, the client team did not cover anything related to BDD, as it was not part of their process. Being so, the build pipeline does not contain automated tests on every commit, for instance. Without this step on the pipeline, the automated test suite must be triggered manually on each user story.

Should those negative aspects be solved, the development team could gain speed, due to the fact that features would be done as expected, without need for fixing differences between what was expected by business people and what was understood by the development team. These negative aspects are, in most cases, the exception. Project A's development team tries to drive the process, by documenting and validating the designed BDD scenarios before starting the development and, despite not having a CI/CD pipeline contemplating tests as it should, triggering the BDD scenarios test suite before considering a user story as done is the common practice.

Finally, by involving all member on the BDD culture, the scenarios could be used to create structured requirement documents where all members involved, be it technical team or business team, would be able to express their needs.

B. Project B

Despite the benefits of the new culture that the BDD brought, sometimes the old habits come back, and the Product Manager writes stories following the old language, generating ambiguity.

However, the developers have the freedom to criticize, making feedback and allowing the Product Manager to updates the Product Specification document with user stories and BDD scenarios better well written.

BDD takes more time to develop than Unit tests, which causes developers to consider this time in the estimates so as not to create a false expectation with a delivery.

Sometimes the development of BDD scenarios takes more time than the development of the feature itself, which frustrates younger developers a little, but with the explanation of the benefits, the acceptance becomes less traumatic.

The BDD culture of the client is so consolidated that when the deployment pipeline fails due to a test case, it generates stress on the part of the business board team. Today quality is one of the central pillars in the client.

In this situation, it is commonly agreed that the developers should stop what they are doing to fix the problem.

However, it is not awkward as the whole client company has embraced the culture, and it benefits everybody.

VI. CONCLUSIONS

As detailed in the previous sections, there are common benefits collected from both adoptions, such as:

- increased self and mutual confidence for both the remote and local teams;
- organic documentation of product features provided by the BDD scenarios;
- decreased communication gap between remote and local teams;
- improved product quality and stability.

When working with remote teams, there are a number of barriers that can burden the communications: different languages, different cultures, different time zones, or the simple fact that working with distributed teams without frequent face-to-face engagement can discourage collaboration. BDD solves domain language and culture barriers, being specific and direct on the requirements by leveraging a structured language (e.g. Gherkin) to enforce objectiveness.

Considering that local and remote team members are in direct contact mostly during Scrum ceremonies, and not always being available on other communication channels (e.g. due to different time zones), defining BDD scenarios during these ceremonies help aligning requirements and expectation.

There was an evident series of trade-offs essentially observed when the adoption of BDD is mostly bound to the technical team:

- BDD scenarios can become too technical, which removes the benefits of proper understanding of features across all contexts (business and technical);
- eventual misalignment in expectations, due to scenarios not being clear for both contexts during requirements specification.

It has also been observed some level of skepticism from the technical team with regards to the increased effort using BDD, when the adoption was mostly bound to the business team.

Considering the findings above, adopting BDD, especially in projects that present a considerable level of complexity, is always beneficial, however, for full effectiveness, both business and technical teams must be fully engaged.

ACKNOWLEDGMENT

We would like to thank Daitan, our employer, for providing us the time and resources to share this experience and write this paper.

References

[1] D. North, “Introducing bdd,” http://dannorth.net/introducing-bdd, 2006, accessed: 2019-02-18.

[2] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[3] C. Solis and X. Wang, “A study of the characteristics of behaviour driven development,” in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, Aug 2011, pp. 383–387.

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

[5] M. Kajko-Mattsson, G. Azizyan, and M. K. Magarian, “Classes of distributed agile development problems,” in *2010 Agile Conference*. IEEE, 2010, pp. 51–58.

[6] P. J. Hinds and M. Mortensen, “Understanding conflict in geographically distributed teams: The moderating effects of shared identity, shared context, and spontaneous communication,” *Organization science*, vol. 16, no. 3, pp. 290–307, 2005.

[7] H. Kaur, H. M. Haddad *et al.*, “Distributed agile development: A survey of challenges and solutions,” in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2015, p. 42.

[8] K. Schwaber, *Agile Project Management With Scrum*. Redmond, WA, USA: Microsoft Press, 2004.

[9] K. Schwaber and J. Sutherland, “The scrum guide,” https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf, 2017, accessed: 2019-02-18.

[10] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, 1st ed. Addison-Wesley Professional, 2009.

[11] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[12] D. J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[13] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[14] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[15] M. Kropp and A. Meier, “Qualitative study of successful agile software development projects,” *IMVS Fokus Report*, 2014.

[16] Y. Yavari, M. Afsharchi, and M. Karami, “Software complexity level determination using software effort estimation use case points metrics,” in *2011 Malaysian Conference in Software Engineering*. IEEE, 2011, pp. 257–262.

[17] S. Chawla and G. Kaur, “Comparative study of the software metrics for the complexity and maintainability of software development,” *International Journal of Advanced Computer Science & Applications*, vol. 4, 2013.

[18] T. Honglei, S. Wei, and Z. Yanan, “The research on software metrics and software complexity metrics,” in *2009 International Forum on Computer Science-Technology and Applications*, vol. 1. IEEE, 2009, pp. 131–136.