Behavior-Driven Development - A case study on its impacts on agile development teams

Nicolas Nascimento
PUCRS, School of Technology
Porto Alegre, RS, Brazil
nicolas.nascimento@pucrs.br

Alan R. Santos
PUCRS, School of Technology
Porto Alegre, RS, Brazil
alan.ricardo@acad.pucrs.br

Afonso Sales
PUCRS, School of Technology
Porto Alegre, RS, Brazil
afonso.sales@pucrs.br

Rafael Chanin
PUCRS, School of Technology
Porto Alegre, RS, Brazil
rafael.chanin@pucrs.br

ABSTRACT

Software development practices which enhance software quality and help teams better develop collaboratively have received attention by the academic community. Among these techniques is Behavior-Driven Development (BDD), a development method which proposes software to be developed focusing primarily on its expected behavior. In this context, this paper investigates how BDD impacts agile software development teams. In order to achieve this, we have conducted a case study on a mobile application development environment which develops software using agile. In total, 42 interviews were performed. Our results indicate that BDD can have positive impacts, increasing collaboration among team members, and negative impacts, like difficulties in writing unit tests. We concluded that BDD has more positive than negative outcomes.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Applied computing** → **Interactive learning environments**; Collaborative learning.

KEYWORDS

Software Engineering, Behavior-Driven Development, Agile Development, Challenge Based Learning

ACM Reference Format:
Nicolas Nascimento, Alan R. Santos, Afonso Sales, and Rafael Chanin. 2020. Behavior-Driven Development - A case study on its impacts on agile development teams. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), Oct 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3391480

ICSEW'20, Oct 5–11, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05.
https://doi.org/10.1145/3387940.3391480

1 INTRODUCTION

The adoption of agile development makes software processes more flexible. If requirements or technology change, this methodology makes it easier to adapt and learn in order to keep delivering value to customers. Moreover, agile development focuses more on the human aspects of software engineering rather than its processes [7]. In addition, studies have shown that many factors play a role in developing software [15], among of which are human factors [11].

In this fast-changing environment, maintaining reliability and efficiently collaborating with stakeholders have been a few of the industry challenges and have led to the emergence of some development techniques, among of which is Behavior-Driven Development (BDD) [14]. BDD is a series of software development practices, proposed by Dan North [14], which aims at helping software development teams to build software which is both more reliable and more aligned with the needs of customers.

In order to further understand the impact of BDD in agile software development teams, we have performed a case study in a mobile application development course. The course teaches students using an active learning framework, Challenge Based Learning [13]. Our preliminary results indicate that BDD can have many positive impacts, such as better collaboration among team members and some negative impacts, such as difficulties in writing unit tests.

The remainder of this paper is organized as follows. Section 2 briefly contextualizes important concepts. In Section 3, we present the scientific approach used for obtaining, evaluating and analysing of the collected data. Following this, Section 4 depicts the results and Section 5 presents a brief discussion and highlights some important findings. Section 6 describes some threats to the study. Finally, Section 7 concludes the paper with some final thoughts and future work.

2 BACKGROUND

As agile is a more common theme in software engineering research, we have chosen to give a special focus on the two specific aspects of this work. Thus, appropriate background only about these themes is provided as follows.

2.1 Challenge Based Learning (CBL)

Teaching students can be done in multiple ways. Traditionally, learning is mostly based on lectures, a teacher-centered approach which

usually provides low levels of interaction. On the other hand, active learning is an approach that proposes high levels of interaction and stimulates students to perform not only low-order cognitive tasks, such as reading and writing, but also high-order ones, including debating, analysing and decision making [1, 6, 9].

There are several active learning methodologies that have been used in an educational setting. Problem Based Learning, Project Based Learning, Task Based Learning and Challenge Based Learning are just a few examples of these frameworks. *"The foundations of experiential learning can be found within the history of most cultures, but were formally organized and presented by David Kolb drawing heavily on the works of John Dewey and Jean Piaget"* [18]. Challenge Based Learning (CBL) [13] is a learning framework based on solving real world challenges.

The CBL process begins with the definition of a *big idea*, which is a broad concept that can be explored in several ways. The big idea has to be engaging and important to students. Once the big idea is chosen and the *essential question* is created, then the *challenge* is defined. From this point, students must come up with the *guiding questions* and *guiding activities and resources*, which will guide them to develop a successful solution. The next step is *analysis*, which will set the foundation for the definition of the *solution*. Once the solution is agreed upon, the *implementation* begins. Finally, *evaluation* is undertaken in order to check out the whole process and verify if the solution can be refined.

The CBL framework is flexible. In this sense, a wide variety of topics can be taught through CBL, including mobile application development (MAD), and it can be integrated with other frameworks, including Scrum and Lean.

2.2 Behavior-Driven Development (BDD)

Dan North [14] stated that BDD is a conjunction of multiple already existent concepts, which can be used together to help the development cycle. BDD is also sometimes associated with another development methodology called Acceptance Test-Driven Development (ATDD)[8], which also augments TDD[2].

As a development methodology, BDD emphasizes test cases which are written in a common language, derived from Domain Driven Design [5]. The specification of these test cases is done using scenarios (also known as BDD Scenarios), which should describe features of a system.

BDD Scenarios are used to further enhance the descriptive capabilities of user stories, which are commonly used as lightweight requirements specification in agile software development. Regarding formatting for specifying these scenarios, a structured language is usually applied, known as Gherkin [3]. Figure 1 shows a possible template, based on Gherkin, which can be used to specify scenarios. Besides on its description, and based on this template, BDD scenarios can be split into three core elements: *Given*, *When* and *Then*.

(1) *Given*: The context assumed for this scenario execution, *e.g.*: "Being logged in" or "Being on the home screen".
(2) *When*: An action or event which happens given the provided context, *e.g.*: "Press the login button" or "Type a character".
(3) *Then*: The expected outcome of the system for the provided action and context, *e.g.*: "Present a success alert" or "Redirect to Home Screen".

In addition to this, each element can have additional context. This is expressed in the template by the word *"AND"*.

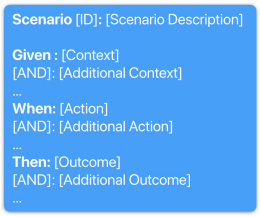

Figure 1: An example template based on [19] for specifying scenarios

In terms of key characteristics associated with BDD, Solis *et al.* [20] defined a few key characteristics which are inherent to BDD. Highlighting a few of those, BDD is composed of:

(1) **Ubiquitous Language** - During development, stakeholders and the development team should be able to cooperate and communicate using a common language. This language should contain enough terms so that any idea regarding the software product under development could be discussed. This can be done by establishing a dictionary at the beginning of the project.
(2) **Iterative Decomposition Process** - Development should happen iteratively with provided time slot as in the beginning of the development process both the customer and the development team are not certain about the requirements of the software being developed.
(3) **Plain Text Description with User Story and Scenario Templates** - Requirements specification in BDD should be performed using user stories augmented by BDD-scenarios. These specifications can be written in plain text and are only required to follow a pre-defined template, provided by the framework.
(4) **Automated Acceptance Testing with Mapping Rules** - After extracting behavior using stories augmented with scenarios, it becomes a task for the development team to properly translate this behavior into actionable tests. Ideally, these tests should be easily created from the plain text specification. This is usually achieve by using a BDD development toolkit.
(5) **Readable Behavior Oriented Specific Code** - When implementing the requirements specified previously, the generated code (methods, classes, etc) should indicate what its purpose is clearly. This implementation code should describe behavior and follow the ubiquitous language defined for the project.

In this context, our goal is to investigate the impact of BDD in agile software development teams. We have applied BDD in a two years mobile application development course that uses CBL to create real world projects. All the projects were executed by agile software development teams.

3 METHODOLOGY

In order to understand the impact of BDD in agile software development teams, we have performed a case study in the two-year mobile development course that teaches iOS, tvOS and watchOS development to undergraduate students. During the course students learn through CBL challenges as described by Nichols *et al.* [13].

Students are introduced to the development ecosystem of Apple platforms and learn by working on real world problems. Besides that, students can choose to focus on development or design aspects of mobile application development. They are expected to dedicate 20 hours per week at course activities.

As a research method, case studies can be used for software engineering research, as they allow the understanding of a certain phenomena in its natural occurring context [17] and are suited to evaluate a method and tool [10].

As our goal was to understand the impact of BDD, with few hypothesis being previously established, we have chosen to conduct an exploratory study. As such, case study methodology is suited to

be used as it enables the research to extract new insights and ideas, to understand what is happening and to generate new hypothesis for other researches [17].

3.1 Case Study Protocol

Our protocol follows the guidelines proposed by Kitchenham *et al.* [10]. Thus, the objective of this case study is **to identify some of the benefits and challenges of using BDD in agile software development teams**. In this sense, this paper aims at answering two research questions:

- **RQ1**: *"What are the positive impacts of developing software using BDD?"*
- **RQ2**: *"What are the negative impacts of developing software using BDD?"*

3.2 Data Collection

Regarding data collection, we have chosen to collect our data through semi-structured interviews [16]. The total number of interviews was 42. These 42 interviews were performed in two different stages in the case study: before using BDD (*e.g.*, pre-BDD) and after using BDD (*e.g.*, post-BDD). The interviews conducted in both stages were aided by interview questions. The pre-BDD interviews helped us to establish a baseline against which to compare the results of using BDD. These interviews were conducted right after teams had finished developing software in the context of CBL Nano-Challenge (*e.g.*, 1 week long). Thus, the post-BDD interviews helped us assess the benefits and challenges of developing software using BDD.

Participants in the study had been attending the course for six (6) months and thus had had the chance to work in software development projects using agile. In addition, some students had previous professional experience developing software.

The average team size in which the participants had worked on was 3.3334. Furthermore, most teams were working together for the first time.

It is also relevant mentioning that the CBL Nano-Challenges performed by participants followed the conceptual model proposed by Santos *et al.* [18]. In this model, students first perform the initial phases of CBL and then use the Scrum framework for developing a software solution.

Aside from the demographic questions, the interview questions used in this case study are presented in Table 1. It is important to note that questions which required the interviewee to know BDD were only asked in the post-BDD stage.

In an attempt to understand the impacts of BDD in each part of the software development lifecycle, we have split the interview questions into four groups, as follows:

(1) Questions 1, 2 and 3 - These questions focus on the project requirements phase, addressing elicitation, specification and the impact of BDD in this phase.
(2) Questions 4, 5 and 6 - These questions focus on *feature development*. A *feature*, as defined by Coad [4], is as function which is valuable for the client and which can be developed in up to two weeks. In our context, *feature development* indicates the process of proving an implementation for a *feature*. As such, they address the presence of ambiguities during the understanding of features which suffered changes and the impact of BDD in this phase.
(3) Questions 7, 8, 9 and 10: These questions focus on implementation quality. More specifically, they address overall quality, number of bugs, documentation and the impact of BDD in this phase.
(4) Questions 11 and 12: These questions address the overall impact of BDD, for the entire development lifecycle.

Table 1: Interview questions

#	Question	Phases
1	How did you elicit the project requirements?	Pre + Post
2	How did you specify the project requirements?	Pre + Post
3	What was the impact of BDD in the requirements elicitation/specification?	Post
4	Was there any ambiguity in the requirements?	Pre + Post
5	Was there any requirement developed which was very different or completely modified?	Pre + Post
6	What was the impact of BDD in the process of translating requirements to actual functionalities?	Post
7	How was the project in terms of code quality?	Pre + Post
8	How was the project was in terms of bugs?	Pre + Post
9	How did you document functionalities of the project?	Pre + Post
10	What was the impact of BDD in the implementation?	Post
11	What are the main benefits of developing using BDD?	Post
12	What are the main challenges of developing using BDD?	Post

In order to avoid biases, we have begun our data collection with this pre-BDD stage. This stage had a total of 27 interviews. The first 6 interviews were *meta* interviews, serving exclusively as a pilot study to improve the quality of the questions being used for actual interviews. This 6 interviews have enabled us to perform small improvements to the interview questions. After this, we have conducted 21 actual interviews. During this process, each participant was given the full context of the research and was told that participation was voluntary and it had no impact on their internal assessment process. Besides that, all interviews had their audio content recorded. This process, performed in both stages, is illustrated in Figure 2.

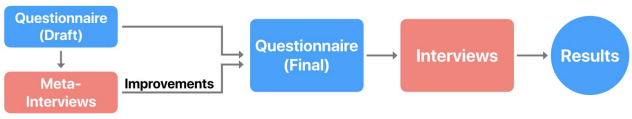

Figure 2: Process used to obtain results from interviews

The pre-BDD stage interviews were performed right after teams had developed software in the context of a CBL Nano-Challenge, which lasted one week. Teams developed software using agile and no interference was performed.

In the post-BDD stage, we have also conducted 27 interviews. Maintaining the improvement procedure used in the first stage, we have conducted 6 *meta* interviews in a pilot study. These *meta* interviews, besides helping to improve the overall quality of questions in our interview questions, helped us to add two additional questions at the end of the interview. These additional questions directly address the benefits and challenges of using BDD. After adding these questions, we proceed to conduct the next 21 interviews. We have performed the same procedures as the pre-BDD stage and these interviews were recorded as well.

The post-BDD stage interviews were performed right after teams had developed software in the context of a CBL Nano-Challenge (different from the pre-BDD stage). Teams were introduced to BDD

practices through lecturing and practical activities before starting their Nano-Challenge. At any time during the development phase of the projects, participants could reach to a senior instructor in case any doubts or questions aroused. In this context, the BDD-scenarios were created in any tools participants were comfortable working on. The only restriction was related to the executable specification aspect of BDD, in which case Quick[1] & Nimble[2] were used, due to their iOS-specific nature.

Another important point is the distribution of participants in the interviews. We have created three groups in which participants were distributed. The first group had 10 people whose participation was restricted to the pre-BDD interviews. The second group had 11 people and its participants were interviewed in both interview phases. The third group had 10 people who were only interviewed in the post-BDD phase. Figure 3 illustrates the distribution of participants in the interviews.

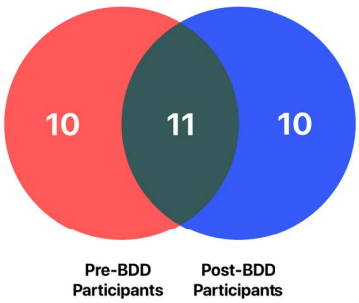

Figure 3: Distribution of participants in the two interviews phases

These groups aimed at increasing the insights generated and allowed us to reduce any bias that the participants might have towards the interviews questions.

After obtaining the results from both pre-BDD and post-BDD phases, we have proceeded to the analysis phase. The analysis performed was qualitative and followed the guidelines proposed by Runeson and Höst [17]. Our analysis aimed at generating hypothesis from the data, thus, we had little to none prior hypothesis.

In this phase, we first performed transcription of the audio recorded in the interviews. In total, we have generated 8 hours, 17 minutes of recorded material. The transcripts from these recording generated 20.207 words. We have chosen to perform this process manually. This was performed as it gives the researcher the opportunity to further extract insights from the interview data during the transcription process. These transcription were arranged in a spreadsheet to ease the analysis process.

Using the spreadsheet, we began further analysis of the data. The analysis was done using two strategies: clustering and categorization. We first clustered answers which presented similar results, counting the number of occurrences. Afterwards, the generated clusters were categorized (which could be one or more). As a final step, we have grouped similar categories. These strategies were used for both pre-BDD and post-BDD results.

[1] github.com/Quick/Quick
[2] github.com/Quick/Nimble

Finally, using the available data and generated analysis, we were able to extract insights from the data. The results of our analysis are presented in the following section.

4 RESULTS

From a demography standpoint, all participants were actively attending undergraduate courses (they were around the 5th and 6th semester). Their average age is 23.9 years old, and all participants have less than 2 years experience in software development (having at least six months of experience on software development).

Following demographyc results, we proceed to gather results from the interviews. In order to ease analysis, we have split our results into four groups:

(1) **Project requirements** - The process of elicitation and specification of project requirements;
(2) **Feature development** - The process of translating project requirements in actual implementations;
(3) **Implementation quality** - The overall quality of code being developed;
(4) **BDD impact** - The reported impact of BDD in the development process, both positive and negative.

4.1 Project Requirements

In terms of project requirements, we have first assessed the regular processes used by development teams and then performed a comparison of these processes after the introduction of BDD.

These was done by analysing the first three (3) questions of the interviews as they directly address aspects such as elicitation and specification of requirements. It is important to note that the last question, from this group of three, was only answered by participants who were interviewed in the post-BDD phase.

4.1.1 Elicitation. Prior to the introduction of BDD, the most reported (10 out of 21) way in which requirements were elicited by the teams was by building up on an individual idea of a team member who had an app idea prior to the team formation. The second most used technique was by properly understanding scope limitations, such as the timeframe.

After the introduction of BDD, the most reported (13 out of 21) way for requirements elicitation did not change and was still by following an individual idea from a team member. However, the second most reported way was through the interaction of team members (12 out of 21). This may indicate that BDD promoted more interactions among team members, thus helping in the elicitation process. However, this is just an indicative, as some reports are complementary to others and could be used in conjunction.

4.1.2 Specification. Prior to the introduction of BDD, participants reported using many different techniques in order to specify software requirements. The three most reported ones were *Verbalization* (9 out of 21), where requirements were not specified in any written format and were only defined through conversations, *Unstructured annotations* (7 out of 21), where requirements were specified using an informal written format, and *To-do lists* (5 out 21), where requirements were specified by defining the tasks which would be performed directly. One interesting results from this phase was that only two (2) participants reported using lightweight specification, such as User Stories.

After the introduction of BDD, every participant performed their specification using User Stories and Scenarios. One interesting

result from this interview phase was that participants found it difficult to understand that scenarios were augmentation of user stories and some reported thinking scenarios encompassed user stories.

4.1.3 Requirements - BDD Impact. Regarding the specific impact of BDD in the project requirements phase, participants have reported many positive aspects and some negative aspects.

Starting by the positive aspects, the three most reported positive aspects were *feature clarification*, where both team and individual found it easier to correctly understand what was supposed to be developed, *ease of development*, where the overall software development process was facilitated, and *project alignment*, where the team managed to develop a common idea of the project.

In terms of negative aspects, participants reported four (4) aspects, which were *lack of impact*, *poor execution*, *initial confusion* and *difficulty to change development mindset*. The number of negative reports, in general, was low (5 reports, against 31 positive).

4.2 Feature Development

Following a similar procedure to the previous section, we have first assessed how participants performed the translation of requirements when implementing features in their regular process. After that, we compared the results from this stage with the ones from post-BDD.

This step was performed by analysing the answers of three questions from the interview questions (questions number 4, 5 and 6). It is important to mention that the last question analysed in this subsection was only answered by participants of the post-BDD phase.

4.2.1 Ambiguity. Prior to the introduction of BDD, the majority of the participants (16 out of 21) reported that their projects had some type of ambiguity when translating the requirements into concrete features. The most frequent types of comments on the ambiguities were that they were related to the application mechanics (5 out of 16), had low impact on the development process (5 out of 16) or were related to the application interaction.

After the introduction of BDD, there were less reported ambiguities, with the most frequent report (13 out of 21) being that there were not ambiguities in the translation process. In addition, the ambiguities reported by the some participants was related to the creation of BDD scenarios, the creation of tests and implementation.

4.2.2 Feature Changes. In terms of features which were specified and changed during the development process, while in the pre-BDD phase, the majority of the participants (11 out of 21) reported not having changed any of the specified features. The remaining participants reported that their project changed during the development phase. Some of these changes included application mechanics, technological limitations and design updates.

In the post-BDD phase, results were similar. Not changing any aspect of the features remained the most reported item by the participants (13 out of 21), followed by changes in the BDD scenarios and the creation of new scenarios.

4.2.3 Feature Development - BDD Impact. Regarding the specific impact of BDD in this phase of development, results are very diverse. Participants described a total of 13 different positive aspects and 9 negative aspects. These aspects are presented in Table 2 and Table 3 respectively. It is important to note that these results were obtained only from the post-BDD interviews.

Table 2: Positive aspects reported by participants

Aspect	Occurrences
Better understanding of features	8
Ensure correct execution	4
Team alignment	2
Reduction of unexpected changes	2
Reduction in implementation problems	1
Broader vision	1
Improvement in the development process	1
Facilitation in the requirements communication process	1
Project guide	1
Facilitation in the development process	1
Reduction in re-work	1
Reduction in user story breakdown difficulty	1
Facilitation of communication among team members with different expertise levels	1

Table 3: Negative aspects reported by participants

Aspect	Occurrences
Difficult to execute	3
Difficult to adopt initially	1
More suited for large projects	1
Meaningless	1
Equivalent to a User Story	1
Value only visible at the end of the process	1
Need for previous planning	1
Easy for some scenarios, difficult for others	1
Low gain (considering learning curve)	1

4.3 Implementation Qualtiy

To address implementation quality, we have followed a similar procedure to the two previous steps, comparing the pre-BDD and post-BDD reports regarding this specific aspect.

This step of the analysis considered results from four questions of the interview questions (numbers 7, 8, 9 and 10). As with the other phases, the last question was only answered by participants in the post-BDD phase.

4.3.1 Overal code quality. In order to assess the impact of BDD in the implementation quality, we have gathered the overall reported code quality of participants prior and after the introduction of BDD. This generated a three categories (*good*, *average* and *bad*), which summarize reports of the participants. As some of participants did not directly state their implementation quality, we have used the overall report as a matching mechanism to associate the report with one of the generated categories. Table 4 presents the distribution of overall reports from participants.

4.3.2 Bugs. In terms of number of reported bugs, prior to the introduction of BDD, the majority of the participants reported code had either many bugs or some bugs (14 out of 21). In the post-BDD phase, results were somewhat similar, with a slight diminish in the

Table 4: Code quality reported by participants

Overall Qualtiy	Pre-BDD	Post-BDD	Difference
Good	6	11	+5
Average	8	6	-2
Bad	7	4	-3

number of reports of many bugs and a slight increase in the reports of no bugs. Table 5 presents a comparison of the results from these phases.

Table 5: Bug category for projects

Overall Qualtiy	Pre-BDD	Post-BDD	Difference
Many bugs	9	5	-4
Some Bugs	8	9	+1
No Bugs	4	7	+3

4.3.3 Documentation. In terms of documentation used during the development of projects, in the pre-BDD interviews, the vast majority (19 out of 21) of participants used some type of documentation. However, the most reported type of documentation was code comments. The results for this phase are presented in Table 6.

Table 6: Documentation types reported (Pre-BDD)

Type	Occurrences
Code comments	15
Code	8
List of tasks	3
No documentation	2
Product Backlog	1
Diagrams	1

In the post-BDD phase, results have changed slightly. Among of these changes is the usage of BDD Scenarios as documentation (6 reports out of 21). The results from these phase are present in Table 7.

Table 7: Documentation types reported (Post-BDD)

Type	Occurrences
Code comments	9
BDD Scenarios	6
Executable Specifications (BDD Tests)	5
User stories	5
No Documentation	4
List of tasks	1

4.3.4 Implementation Quality - BDD Impact. Finally, in order to generate a broad vision of the impact of BDD in the implementation quality, we have used question number 11 of the interview questions. Participants reported many positive (16) and negative (14) influences of BDD in the implementation. Tables 8 and 9 respectively present the positive and negative aspects reported by participants.

One interesting result from this question was that the majority of those who reported negative aspects of BDD did not think it is bad. Rather, they report not being able to properly use it.

Table 8: Positive impacts of BDD in implementation

Item	Occurrences
Improvement in implementation quality	5
Organization & Planning	5
Clear implementation	3
Facilitated implementation	2
Right (client expectations) implementation	2
Reduction in re-work	2
Reduction in ambiguities	1
Modularization	1
Custom execution	1
Simultaneous Development and Testing	1
Correct (functional) implementation	1
Product-oriented vision	1
Improvement in task division	1
High-level of impact in implementation	1
Perception of the need for good scenarios	1
Suited for documentation	1

Table 9: Negative impacts of BDD in implementation

Item	Occurrences
No changes	3
Meaningless	2
Poor tests	2
Difficult in terms of UI	1
Difficult for unexperienced developers	1
Project too simple for BDD	1
Bigger projects can benefit more	1
Other paradigms provide better guidance	1
Low impact	1
Slower implementation	1
Testing becomes difficult	1
High learning curve	1
Lack of team commitment (to BDD)	1
Lack of time	1

4.4 BDD Impact

Finally, regarding the perceptions of BDD, we have gathered results from the last two (2) questions of the interview questions performed with participants. These questions were only performed in the post-BDD phase, as participants had been asked specific questions about their perception of BDD. In this sense, Table 10 presents the positive aspects of BDD reported by participants. In contrast, Table 11 presents the reported negative aspects of BDD.

5 DISCUSSION

The results from our case study present a view of the impact of introducing BDD to the development lifecycle of agile software development teams in the context of a software development course. Aiming at answering the proposed research questions of this work, we further discuss these results in the following subsections.

Table 10: Positive aspects of BDD

Aspect	Occurrences
Better Comprehension of feature under development	4
Team alignment	3
Eased task division	3
Correct (functional) development	3
Right (client expectations) development	3
Reduction in ambiguities	3
Eased project comprehension	3
Clearer project	2
Eased project contextualization	2
Eased project organization	2
Faster development	2
Awareness of final product requirements	1
Conjunct development and testing	1
Eased project execution	1
Eased project testing	1
High market usage	1
Improved requirements	1
Improved documentation	1
Improved prioritization	1
Improved project planning	1
Value perception in larger projects	1
Automatic testing	1
Reduction in 'course' changes	1

Table 11: Negative aspects of BDD

Aspect	Occurrences
Tests writing	8
Test-driven development	8
Scenario creation	4
Initial process	3
Reduction in development time	3
Lack of experience with BDD	2
Small scope	2
Team engagement with BDD	2
Scenario comprehension	1
Up-front need of complete project comprehension	1
Team size	1
Task distribution	1
Scenario precision	1
Limitation in test amount	1

5.1 RQ1 - What are the positive impacts of developing software using BDD?

The initial phases of development, where project requirements are generated and specified, seems to have suffered positive influence. An example is the report from participants about the elicitation phase, where the second most used technique was the interaction of team members. This result is aligned with proposed benefits of BDD [19]. Moraes [12], in her master thesis, has found that using BDD encourages team members to collaborate. This could indicate that the elicitation process has changed due to the increase in team member collaborations. In addition to this, participants have reported multiple positive aspects which may be due to the use of BDD, such as an increase in feature understanding.

Project requirement specification has also drastically changed as, prior to the introduction of BDD, participants would most-likely specify project requirements in informal ways, such as verbalization and unstructured annotations. The combination of user-stories and scenarios was the specification method used by all participants in the second phase of interviews. This result is not particularly surprising as participants were using BDD in their development lifecycle and this specification method is proposed in the BDD framework [19].

Reports from participants also indicate that the process of feature development was improved. The majority of the post-BDD participants have reported a more clear understanding of the project after adopting BDD. This could be connected with the reduction in the number of ambiguities, which were also reported by participants in the post-BDD phase.

In terms of implementation quality, BDD seems to have had a positive impact on the development of projects. The most reported positive aspect of BDD were its clarification of features which need to be developed. This means that the development team would spend less time debating a feature and more time actually implementing it as the creation of scenarios, where everyone is allowed to participate and contribute, helps to ensure a collective understanding of the user stories.

Moreover, regarding the implementation process, results from the documentation types reported show that both BDD scenarios and BDD executable specifications were considered as documentations by participants, even though code comments remained as the main documentation type. One interesting result from this was that no participant reported code as being the documentation of the project, whereas 8 participants reported this in the pre-BDD phase (Table 6).

Finally, analysing the results from the second to last question, BDD seems to have helped teams develop software in multiple ways. By allowing team members to better comprehend features under development and by improving team alignment, BDD helped teams have a clear vision of the project. Besides that, participants reported felling their implementation was both correct, meaning that it had less bugs and had a better overall organization. Finally, another interesting result is an eased process of task division, a common activity in agile teams.

5.2 RQ2 - What are the negative impacts of developing software using BDD?

Regarding negative impacts of BDD, in the beginning of the development lifecycle, there were negative aspects reported, with problems being due to poor execution of BDD, initial confusion and a difficult to change (we are excluding the inability to see value in the methodology, which is probably related to lack of commitment of the development team). A possible cause for these problems is the fact the agile teams were using BDD for the first time and naturally went though the learning curve of the framework. This should also be considered for other negative impacts, as some might have happened because some of the participants were using BDD for the first time.

Results from the question related to the feature development were similar in terms of quantity. The number of reported features which did not change during development was very similar both in the pre and post-BDD phases (an increase of 13 reports against 11). In addition, participants reported having trouble to both create and

change BDD scenarios. This may indicate that the increased levels of interaction among team members was not enough to decrease the number of features which were not completely clear during development. Furthermore, many of the negative aspects reported by participants were related to lack of experience using BDD, which could indicate more experience would remove this issue.

In terms of implementation, the BDD process seems to have been difficult to be executed, as reported by some participants. In addition to this, both the overall code quality and the number of bugs seem to have improved from the process of BDD. However, this is difficult to assume as there were no pre-established criteria for those metrics and their reports could be subjetive to what a participant take into account when classifying their code as *"good"* or *"bad"*.

Finally, results from the last question, which directly addresses the negative aspects of BDD, have shown that writing tests and developing based on tests (a shared practice between BDD and TDD) were the main challenges. In addition, the first experience with BDD can be challenging, as the learning curve is somewhat large. Furthermore, results indicate that developers have a feeling where performing the processes of BDD, such as writing executable specification and creating scenarios, can decrease the time of development. Even with this, results indicate that participants were able to perceive the value of these processes and are not completely unhappy with them.

6 THREATS TO VALIDITY

Our study was conducted with a limited number of respondents and from the same iOS development course. In addition, our results are drawn based on participants viewpoint - students (development teams), thus being a subjective opinion. Therefore, the results reported in this study are dependent on the participants' honesty, perceptiveness and judgment.

It is also important to notice that part of project participants were attending a training course without previous experience with other software engineering practices and approaches. Also, as the translation and coding processes were manually performed and even ensuring they were correctly executed, errors could have been made and consequently have influenced our results.

Finally, as both projects were performed sequentially and participants were developing software in an education context, it is possible that the experience from the first project somehow improved their development skills, changing their development practices.

7 CONCLUSION

This paper has performed an assessment of the impacts of BDD in agile software development teams. The case study explored the positive and negative impacts of BDD in a two-year mobile development course where participants were working in an agile software development environment.

We have found indicatives in our study that BDD can help development teams in most of the phases of the software development lifecycle. Project requirements creation may benefit as teams showed more levels of interaction in contrast to when BDD was not applied. During feature development, developers can benefit manly by better understanding of feature encompasses. Quality implementation can also benefit as participants reported BDD improves quality implementation and better documenting their projects.

However, inherent aspects of BDD, including writing tests and creating scenarios has been reported as a challenge by participants. In addition, lack of experience and commitment with BDD is a possible influenced for this negative result.

Overall, our results indicate that BDD can provide more benefits than harms to the development lifecycle and, as future work, we will perform another case study to address whether more experienced developers can further improve the software development lifecycle by using BDD.

REFERENCES

[1] K. Ahmad and P. Gestwicki. 2013. Studio-based Learning and App Inventor for Android in an Introductory CS Course for Non-majors. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. ACM, New York, NY, USA, 287–292.

[2] Kent Beck. 2003. *Test-driven development: by example.* Addison-Wesley Professional.

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

[4] Peter Coad, Jeff de Luca, and Eric Lefebvre. 1999. *Java modeling color with UML: Enterprise components and process with Cdrom.* Prentice Hall PTR.

[5] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley Professional.

[6] M. Fetaji and B. Fetaji. 2009. Analyses of mobile learning software solution in education using the task based learning approach. In *Information Technology Interfaces, 2009. ITI '09. Proc. of the ITI 2009 31st Int. Conf. on.* 373–378.

[7] H. K. Flora and S. V. Chande. 2013. A Review and Analysis on Mobile Application Development Processes using Agile Methodologies. *International Journal of Research in Computer Science* 3, 4 (July 2013), 8–18.

[8] Markus Gärtner. 2012. *ATDD by example: a practical guide to acceptance test-driven development.* Addison-Wesley.

[9] P. Gestwicki and K. Ahmad. 2011. App Inventor for Android with Studio-based Learning. *Journal of Computing Sciences in Colleges* 27, 1 (Oct. 2011), 55–63.

[10] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. 1995. Case studies for method and tool evaluation. *IEEE software* 12, 4 (1995), 52–62.

[11] Paul Luo Li, Andrew J Ko, and Jiamin Zhu. 2015. What makes a great software engineer?. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1.* IEEE Press, 700–710.

[12] Lauriane Moraes. 2016. *An Empirical Study on the Use of BDD and its Support to Requirements Engineering.* Master's thesis. Pontifical Catholic University of Rio Grande do Sul, Brazil.

[13] M. Nichols, K. Cator, and M. Torres. 2016. *Challenge Based Learning Guide.* Digital Promise, Redwood City, CA, USA.

[14] Dan North et al. 2006. Introducing bdd. *Better Software* 12 (2006).

[15] Paul Ralph and Paul Kelly. 2014. The dimensions of software engineering success. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, 24–35.

[16] Colin Robson. 2011. *Real world research.* Vol. 3. Wiley Chichester.

[17] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

[18] A.R. Santos, A. Sales, P. Fernandes, and M. Nichols. 2015. Combining Challenge-Based Learning and Scrum Framework for Mobile Application Development. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'15).* Vilnius, Lithuania, 189–194.

[19] John Ferguson Smart. 2014. *BDD in Action.* Manning Publications.

[20] Carlos Solis and Xiaofeng Wang. 2011. A study of the characteristics of behaviour driven development. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications.* IEEE, 383–387.