Experimenting with the Peanut Butter and Jelly Sandwich Challenge to Introduce Algorithmic Thinking and Test Case Writing

Kiev Gama
Informatics Center
Federal University of Pernambuco (CIn/UFPE)
Recife, Brazil
kiev@cin.ufpe.br

Andrew Diniz da Costa
Informatics' Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
acosta@inf.puc-rio.br

Hendi Lemos Coelho
Informatics' Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
hendi@les.inf.puc-rio.br

Ricardo Almeida Venieris
Informatics' Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
rvenieris@inf.puc-rio.br

Carlos José Pereira de Lucena
Informatics' Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
lucena@inf.puc-rio.br

ABSTRACT

New approaches that offer good learning experiences driven to computer science education have been applied in different places. One of the ways adopted is the application of dynamics in classrooms that challenge students to work in groups and make relations to situations of their lives. Besides, to improve content retention and students engagement, humor is one good element that should be applied in these dynamics. The "Peanut butter and jelly sandwich challenge" is an example that allows including the idea of challenging students using humor as a support to instructional content. This paper explains how that dynamic was applied to two students' groups. The first experience was offered in a mobile programming course that follows a boot camp style and involved a multidisciplinary group with students from three universities. The dynamic applied was used to present the relevance of algorithmic thinking. The second experience used the first case as motivation, adapting it to cover contents focused on test case writing applied to students of computer science. In both cases we present results gathered, such as learning impact for the students.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Applied computing** → **Collaborative learning**; • **Software and its engineering** → **Software testing and debugging**;

KEYWORDS

Algorithmic Thinking, Software Testing, Test cases, Humor in Classroom

ACM Reference Format:
Kiev Gama, Andrew Diniz da Costa, Hendi Lemos Coelho, Ricardo Almeida Venieris, and Carlos José Pereira de Lucena. 2019. Experimenting with the Peanut Butter and Jelly Sandwich Challenge to Introduce Algorithmic Thinking and Test Case Writing. In *XXXIII Brazilian Symposium on Software Engineering (SBES 2019), September 23–27, 2019, Salvador, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3350768.3353816

SBES 2019, September 23–27, 2019, Salvador, Brazil

ACM ISBN 978-1-4503-7651-8/19/09…$15.00
https://doi.org/10.1145/3350768.3353816

1 INTRODUCTION

Currently, several new approaches that look for improving the learning process of students related to Computer Science education have been explored in educational institutions around the world. In higher education, humor is an important element to improve content retention and students' engagement [16]. Laughter can have the potential to unleash creative thinking and to reduce social pressure [17]. The Instructional Humor Processing Theory [24] supports the idea that humor is highly appropriate when related to instructional content. It enhances the students' motivation and ability to process course content. The authors found that humorous messages caught students' attention, made content relevant and clearer, without distracting them from the instructional message.

A good example of humor in Computer Science education is the usage of the Great Peanut Butter Caper [18], which had many variations reported to be used in introductory courses focused on algorithmic thinking [1, 3, 7, 8, 21–23] and even in courses of human-computer interaction [12]. The main idea is having students to write down the steps necessary to make a peanut butter sandwich. When executing a student's algorithm, the instructor would provoke humorous situations: the lack of precision in an "open the pack of bread" instruction would end up in a ripped open bag, causing many bread slices to fall over the table. Ambiguity

can be explored in instructions such as "spread the butter on the bread": the instructor can rub a closed peanut butter jar on the bread, if there was no instruction to open the jar; or they can take an exaggerated portion of butter if it was not specified.

We present two learning experiences applying a Peanut Butter and Jelly (PB&J) Sandwich Challenge. The first one was applied with a multidisciplinary group that was learning algorithmic thinking. Building upon the positive outcomes identified in the first experience, we used the PB&J activity in a second experience as part of a regular software engineering course for senior students to introduce test case writing.

In the first experience, an important element from learning processes was applied, which concerns the competition and cooperation group dynamics. The concepts presented in [15] defines that cooperation and competition affects the dynamics of learning alliances, distinguishing two qualitatively different kinds of available benefits to participants in group learning experiences: the private one, when the participant or group unity earn unilaterally by picking up skills or knowledge from its partner and applying them to its own gain; and the common one, when the alliance brings gains to all parts. Intuitively the ratio of private/common benefits for a particular achievement will be higher when it has more opportunity to apply what was learned outside the scope of the alliance. In an academic learning activity, the main goal is to maximize gain for all participants. Thus, to introduce competition in learning activity needs to be done carefully, so that the competition itself does not overlap with the learning objectives.

In the second experience, as we did not find any reports in the literature about the usage and impact of a PB&J activity in test case writing. That approach is a way that improves the quality of test cases written by students, which often times have errors that make difficult to find hidden bugs [9] and to use data adequately and execute instructions written [10]. We decided to collect some measures and designed a study that is presented as part of this experience report. A subset of students from that class participated in that activity. We compared test cases written by those students against test cases written by students who did not participate in the activity, evaluating different characteristics (e.g., precision of inputs, clarity of outcomes, details of step-by-step execution).

Thus, this paper is structured as follows. Section 2 brings the background and related work. Section 3 describes the first experience we had using the PB&J activity that motivated using it on software testing. Section 4 details our experience and adaptation of the PB&J activity for test case writing. Next, section 5 analyses the data from the test case writing experience, followed by a discussion in section 6. Section 7 presents conclusions and future works.

2 BACKGROUND AND RELATED WORK

2.1 Test Cases written by Students

According to the IEEE [5], a test case consists on *"(A) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement; and (B) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item"*. Based on that, a well-designed test case is essentially composed of three parts: inputs, outputs and order of execution [6]. A test case must be reasonably capable of revealing information, typically in the form of a bug [14]. However, mastering in create test cases that help to reveal such information may be difficult.

Students programmers tend to write simple test cases covering just expected behavior, instead of writing tests that find hidden bugs [9]. This is what is known as "happy path", which leads to tests that work for common cases only. Students instead focus on typical-case scenarios without exploring situations that are likely to reveal bugs. However, this is not just a students problem. The same also happens with engineering tempted to write tests on the "happy path" when testing their code [2].

An analysis [10] of mistakes in test cases written by students of a Master-level testing course found other aspects to be improved, such as a better step-by-step description and better result evaluation. Again, this is not limited to a student problem since the authors reported similar problems in test cases written by industry testers. The study found that students tend to repeat their mistakes. If they missed one category (e.g., input data), they would repeat that mistake in other test cases.

2.2 The Peanut Butter Caper and its variations

Lewandowski and Morehead [18] presented the concept of Common Learning Experience (CLE), which are playful activities that engage students in problem-solving using a fun and interactive classroom environment. The Great Peanut Butter Caper is one of the CLEs they apply, which has been extensively used, adapted and reported by other authors focused on programming or algorithmic thinking [1, 3, 7, 8, 21–23]. Since they all provide similar perspectives, we will not exhaustively cover them here.

The experience of Rankin and Thomas [21] is particularly interesting. They successfully employed food-focused activities to help students developing algorithmic thinking in an all-female liberal arts college. As in a typical PB&J sandwich activity, it was used with the purpose of assisting students with understanding the concept of an algorithm. They began to understand the importance of providing explicit steps as something important in a well-defined algorithm with unambiguous instructions for completing a task in a finite amount of time.

Davis and Rebelsky [8] adapted that approach and used a PB&J in traditional CS1 and CS2 courses. An interesting contribution of the paper is the detailed guidance to instructors for planning and carrying out the activity, giving tips on the goals of the exercise, on what can be misinterpreted, among other instructions. The authors also suggest bringing students some notion of testing starting from algorithm design. They propose students to break the problem into smaller pieces, and solve and test each piece separately.

As presented on the different works detailed in this section, the reports on the usage of the PB&J activity are centered around algorithmic thinking. In the work we report in this article we have also used this approach for algorithmic thinking but expanded its usage to a different context in Software Engineering. We saw an opportunity to explore this approach in a different context which consists on test case writing.

3 LEARNING ALGORITHMIC THINKING

In this section we describe our first experience applying the PB&J sandwich activity to a multidisciplinary group of students. Thus, in subsection 3.1 we detail the context in which such activity was applied. Subsection 3.2 shows the learning goals of that activity. Subsection 3.3. explains how it was applied in details followed by subsection 3.4, which shares a perception identified during the dynamic about resilience and persistence. For last, subsection 3.4 shows results gathered.

3.1 Context

Our first attempt using the PB&J sandwich activity aimed to assist undergraduate students in learning what an algorithm is and logical thinking. When students begin studying algorithms, it is common to see them struggling to understand the relevance of splitting their ideas into functions, smaller parts that clearly state their intentions. That level of abstraction is not very obvious for those having contact with it for the first time and the long term advantages of smoother evolution, maintenance and testing are far from clear at the beginning. Besides, students also present the same recurring errors when writing test cases, such as obvious test paths, ambiguous steps, lack of detail about inputs and outputs.

The PB&J activity was initially applied in a mobile programming course that followed a boot camp style and involved a multidisciplinary group of 40 students from three universities. That group of students was composed of 17 women and 23 men. Besides, 25 students were in STEM courses (e.g., Computer Science and Engineering), while 15 students were in non-STEM courses (e.g., Design, Communication and Business). Table 1 shows in detail the number of students per course.

Table 1: Course with 40 participants of the PB&J sandwich activity.

Courses (Bachelor's degree)	Amount
Architecture	1
Business	1
Computing	17
Communication	2
Design	10
Chemical Engineering	2
Control and Automation Engineering	1
Law	1
Mechanical engineering	1
Production Engineering	4

3.2 Learning Goals

The goal of this experience was to use the PB&J activity to present the relevance of algorithmic thinking. In order to highlight that relevance, we presented why organizing code into functions helps evolving, maintaining and testing any software solution. When developers think that way, a lot of re-work can be avoided. In addition, another interesting topic experienced in this activity was related to the resilience and persistence required for programmers. Learning from experiences, trying and reiterating is important for the software development process. For last, but not least important, other aspect observed was the Dynamics of Learning Alliances [15], how competing and cooperative behaviors impact learning.

3.3 PB&J & Algorithmic Thinking

The PB&J activity lasted 40 minutes. It was applied in the first coding class of the course and divided into four parts, very similar to the stages proposed by Rankin and Thomas [8]. Our adaptation is described below:

First Part: Activity Instructions and Group Formation. We requested from students a list of instructions explaining how to make a butter and jelly sandwich. A picture illustrating how that sandwich should look like was presented to them. To prepare that script, students self-organized into groups of 3 to 5 people. In order to motivate these students and introduce some competitive behavior, the first three groups that presented a correct script earned a chocolate as a prize.

Second Part: Scripts creation and execution. When a group had a version of the script finished they presented it to one of the teachers (we had four participating) who would follow its steps, trying to make the sandwich. To guarantee that such script was being followed correctly by the teacher, each step was read in front of the group and executed so they could see if every given step was performed as the group expected. It was common to find steps that were not precise enough, such as "spread the butter on the bread". In this case, when a group did not inform that the butter jar should be opened in the first place, we tried to spread the butter using it closed on the bread. Actions like that made the learning environment fun. Another situation was when groups did not inform where to spread the butter over bread. To show the relevance of being precise with that information, we had chosen a small part of the bread to spread the butter. Thus, if any of the steps were wrong, that group would have to make changes and present a new script version with the correct steps. This part of the activity had a high engagement of the participants. They were divided into 10 groups and all groups presented at least three script versions. Thus, at the end of the activity, all groups presented a viable script that was unambiguous and sufficiently detailed, allowing to prepare a sandwich like the one presented at the beginning of the class.

Third Part: Linking with algorithms. During this part of the class, teachers explained to students how those scripts had relation with algorithms. To make such link, teachers illustrated how a script they had written in natural language could be converted into pseudocode. To make that, teachers took each written step and defined a function that could represent it, such as openBreadPacket(), takeBread(), openButterJar(), spreadButter(), etc. That was a way to explain the concept of functions, illustrating the relevance of organizing the logic development, dividing the problem into smaller parts to reach a goal, and consequently making it easier to test each function.

Fourth Part: Debriefing. In the last part of the class, individual feedback from students was gathered. From that feedback, we could realize that the approach was positively received by all students and that they could understand the algorithm concept. Besides, students with background in programming informed that such learning was

more engaging and accessible than previous approaches that they had experienced.

3.4 Resilience and Persistence

During the second part of the dynamic, it was clear that competition between groups had played an important part. Given that the humorous nature of the activity somewhat leveled the playing field - since experienced programmers had no advantage over the novices -, it's safe to say that the atmosphere of competition did not hinder the engagement of those with no previous experience writing code or background in Computer Science.

Cameras were spread in the room and while teachers were not executing some script, they walked around the groups to observe them. An additional collaborator also took pictures and recorded some moments while the dynamic was happening. From these registers it was possible to realize the excitement of those participating, the high level of cooperation inside the groups and how much they confronted mentors when those were not playing precise compilers. This is the kind of behaviour expected in any learning environment and even more desirable from those staring to learn how to write software. Being able to cope with the frustration that comes with many coming failures is a crucial mindset to acquire at the beginning of the learning journey.

Another interesting behavior observed was related to those who decided to wait for some of the other groups to execute their scripts, learning from that experience. Groups that decided not to be the first, leaving members observing the execution of the scripts were more assertive, benefiting from the experiences of others. That behavior also was emphasized as an important away to speed up their learning process. Sometimes the "eyes in the prize" strategy wasn't the smartest one.

3.5 Results

Teachers noticed greater engagement of students compared to a previous group, who did not have a class with the sandwich challenge. That previous group also had 40 students (see Table 2) from two universities, and they learned the same technical topics with three participant teachers of the sandwich activity. That greater engagement reflected in a faster evolution of students in software development, allowing 35 functional solutions to be developed by them in two months, comparing to 26 solutions developed by the previous group also in two months.

Table 2: Previous course with 40 participants

Courses (Bachelor's degree)	Amount
Chemistry	1
Computing	18
Communication	1
Design	13
Civil Engineering	1
Electric Engineering	2
Mechanical engineering	1
Production Engineering	3

One year after this activity to be applied, we returned to the students for an open interview that had the following questions.

(1) What do you remember from that activity?
(2) How did it help you in your coding learning journey?
(3) Did the dynamics help to think in tests?

Below we present important points gathered from each question made. These points follow the order of the questions presented above.

(1) For most of them, the activity was one of the highlights of the year, because it was funny, a good ice breaker for the students to get to know better each other, and some of them take the lessons learned on debriefing to life.
(2) The impact in their learning journey was more intense for the students with no coding experience, the reports are that understanding how to think algorithmically and associating it with common life tasks help to remove the feeling of detachment that surrounds the activity of coding. For the ones who had already coded before, it was interesting to understand how to travel between different levels of abstraction, it took them out of their comfort zones of a suitable IDE to program, making them think in the entire process before testing a single part. Everyone also agreed that it was interesting to realize how much of the assumptions we have, believing that these truths are already established, that the other, or in this case the compiler, has the same premises, which is not true.
(3) All of them reported that tests were not in their minds, but all of them also agreed that it is extremely important to try, err and try again, that making mistakes is part of the journey.

After the interviews, a group reflection was made. At this point everyone involved agreed that the competitive portion of this activity had discouraged them from thinking more carefully, observing the iterations of others to learn from their mistakes. It became all about moving fast and winning, more than observing and learning.

they get it!

4 THE TEST CASE WRITING EXPERIMENT

In a teachers' discussion about the outcomes of the activity presented in the previous section, we saw an opportunity to use such approach with software testing. We intended to perform that activity with our students and measure if it had any impact on the quality of the test cases they produced.

Thus, this section is organized as follows. In subsection 4.1 we detail the context in which such activity was applied. In subsection 4.2 we show the learning goals of such activity. Subsection 4.3 explains in detail the selection of the participants, and in subsection 4.4 describes how the activity was applied in details.

4.1 Context

Our context differs from the initial activity that worked as a motivation. It took place in the second semester of 2018 on a Software Engineering (SE) course for senior undergraduate Computer Engineering students. The course has lectures on requirements engineering, software architecture, software testing, configuration management and project management using some agile principles. Also, students need to develop a group project that is carried out throughout the semester, where they can practice the techniques taught in the lectures. The project has to target a real problem, and they have to deliver a functioning system at the end of the semester, with some deliverables (e.g., sequence diagram, test case) being gradually built as they acquire the required knowledge.

The PB&J activity happened one week before the students had an introductory class in software testing. In this SE course, the part that covering software testing provides notions on functional black-box testing [6, 19]. Our focus with the class experiment reported here was on manual testing, where a human tester interacts with the system to execute the tests to ensure its behavior is as expected, often following a written test plan that contains a set of test cases [11]. Although there is a trade-off between manual and automated testing, they are of equal importance since bugs they tend to reveal are different [20]. Understanding well the dynamics of test case writing focused on manual testing works a preliminary step before going to test automation.

4.2 Learning Goals

The goal with this instruction was to use the PB&J activity as a playful approach to reinforce the importance of algorithmic thinking and introduce to students the characteristics of a good, clean and testable code, creating functions that have clear inputs and outputs, are simple and unambiguous, respecting its level of abstraction. The take-home lessons for students participating in this activity would be the importance of writing code with all these characteristics, so the test case can be repeatable by someone else or automated.

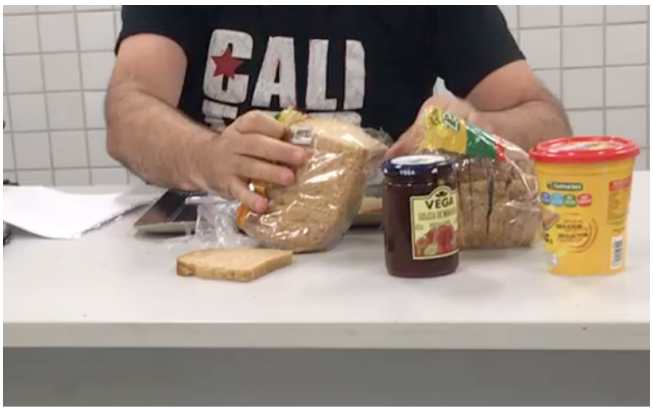
4.3 Participants selection

There were 34 students in this class (33 male and 1 female). Only 14 of them (41% of the class), all male, participated in the optional class where the PB&J activity was performed. Since this is an introductory class in Software Engineering and its many disciplines (e.g., software testing), all students were novice testers. We intentionally ran the PB&J activity as an optional class during a week where students had exams from many courses and ended up skipping the SE class to find extra time to prepare for taking their exams. The idea was to have one group of students that would participate in the PB&J activity and another that would not participate. This allowed us to use the latter as a control group in the comparison we made.

4.4 PB&J and Test Case Writing

The participants only knew there would be an activity related to testing, but they did not have any information on what it was. One teacher was responsible for running the activity. The whole exercise took 40 minutes and the steps followed were the same

Figure 1: Teacher rips open the bag of bread literally following the instructions written by students.



ones as described in the previous section. The significant differences happened on the third and fourth parts, as described below. Another difference in relation to the first exercise is that we took out the competition aspect (i.e., no prize given), for no special reason, and reduce from three to two rounds of

First Part: Activity Instructions and Group Formation. The teacher announced the instructions to the students, indicating that they should have 5 minutes to write a script with the necessary steps to make a butter and jelly sandwich. Students had to organize themselves in groups from 3 to 4 people. They were suggested to work with their course project groups, but were free to form groups with any student.

Second Part: Scripts creation and execution. The process was very similar to what was previously described in section 3: students wrote the scripts, and the teacher had to execute them for making a sandwich. Humor and exaggeration by the teacher when executing ambiguous instructions with lacking steps were key aspects to illustrate the gaps in the scripts. Naturally, the ambiance was very good with many students laughter and astonishment faces when the teacher executed instructions such as "open the bag" but torn it apart (Figure 1), instead of gently opening it the correct way. As an iterative approach, there were two additional rounds, each one taking 3 minutes, where students would try to improve their scripts. In the end, they were all sufficiently detailed for making a sandwich correctly.

Third Part: Linking with test cases. After the conclusion of the script's execution, the teacher discussed the relation of the exercise with the context of test case writing. He briefly presented the idea of a test case and initiated a discussion on the purpose of the exercise around the concepts of inputs, outputs and order of execution, which are essential aspects of a test case. He highlighted the importance of specifying what information is important as input (e.g., the amount of peanut butter to use), the correct sequence of unambiguous steps and what to expect from that execution. He used the instruction of spreading "enough" butter or jelly on the bread, as an example of not being clear about the expected output. He also emphasized the importance of correctly and unambiguously

a writing sequence of steps that should be executed by someone else or even automated.

Fourth Part: Debriefing. On this last part, the teacher conducted individual discussions with groups asking them about any particular questions they would have but were not covered. The main purpose of the individual discussions, in fact, was identifying possible scenarios for potential test cases that could exist in their projects and make them think about them.

5 ANALYSIS

Three weeks after the PB&J activity, all 34 students from the class took their Software Engineering exam which had a traditional question that is repeated every semester: write one test case for your group project. We perform our analysis based on the responses to that question. We compared the results of the test cases written 14 students that participated in the activity against the results of the control group (the other 20 students). As an additional control group, we used test cases from exams of the two previous semesters (2nd semester of 2017 and 1st semester of 2018). In the next subsections, we describe the criteria that were used in the evaluation and the results obtained.

5.1 Test case evaluation criteria

We analyzed each test case question from the exams of all four groups, according to well-defined criteria for characterizing test cases [10]. To minimize teacher bias, we did not identify the participants of the PB&J activity until all exams were evaluated.

We aimed at measuring the clarity of inputs, expected outputs, and step-by-step description, the elaboration of a non-obvious (i.e., not a "happy path") test case, and the total steps. While the last characteristic was inspired in an aspect that can change as a result of a PB&J activity focused on algorithms [7], the other four correspond to some of the categories used in a study that analyses mistakes in test case writing [10]. The authors of these studies we used as references described the criteria to evaluate these characteristics in a subjective way. Therefore, we detail the following criteria we used for each characteristic when evaluating the students' test cases:

- **Input data**. We considered it as "precise" or "imprecise". There must be explicit data for user input (e.g., digits, letters, special characters) [10]. If a test case provided just superficial instructions on what type of data (e.g., enter an invalid date, send a request to the server with invalid data), we considered it as "imprecise".
- **Expected outputs**. A test case has to specify the expected result or visible outcome so that the outcome of the test case could be compared with it [10]. Thus, we looked for explicit statements of expected results or post-conditions [10], marking it as "clear". Otherwise, it would be marked as "unclear".
- **Step-by-step description**. The test case must often be described in small and very detailed steps, the same as writing code [10]. Test cases with precise steps were marked as "sufficient" while those with ambiguous interpretations would be marked as "insufficient".
- **Variation of test case**. The criteria considered the creativity on the test case avoiding the "happy path". This characteristic was referred to in [10] in the case of only using the most obvious test case or input, which has little impact on the main functions of the system. Some examples of errors being provoked with a "non-obvious" variation of a test case: disconnecting the system from the network and trying to use it; using corrupt files; sending requests with missing fields.
- **Number of steps**. This was a straightforward metric, where we counted the total steps defined in a test case.

5.2 Results

This subsection details the results presented in Table 3. We have two groups representing the students that studied in the semester where we applied the PB&J exercise: 2018.S2 PB&J group and 2018.S2 control group. The former corresponds to the group of students that did the Peanut Butter and Jelly activity, and the latter corresponds to those who did not (i.e. the control group). As already mentioned, two other groups (2017.S2 and 2018.S1) representing the two previous semesters were used as additional control groups. We compare the results of the different semesters and confront them with the findings of the study by Eldh et al. [10] which we use as a reference in this study:

- **Input data**. The results on the precise specification of input data provided in test cases show similar values on the previous semesters (12.9% and 18.2%), with a significant evolution on the control and PB&J groups of semester 2018.S2 (30% and 35.7%, respectively). The finding of Eldh et al. [10] presented 30% of the students providing valid input.
- **Expected outputs**. The analysis of Eldh et al. [10] found that 28% of students missed giving an evaluation of what was expected. This number was very close to semesters 2017.S2 (29%) and 2018.S1 (30.3%) and the 2018.S2 control group (35%). There was a significant difference in relation to the PB&J group, which had low value (7.1%), meaning that the majority of test cases from that group detailed well the expected outputs.
- **Step-by-step description**. Eldh et al. [10] reported 47% of students providing insufficient details that could be unambiguously followed by a human. We found a slightly lower rate in our analysis, with 41.9% and 51.52% for semesters 2017.S2 and 2018.S1, respectively. Those numbers confirm the finding of [10], which says that beginners write too little information in the test case. There was a significant improvement on semester 2018.S2, with insufficient definitions dropping to 30% and 14.3% on the control group and PB&J group, respectively.
- **Variation of test case**. Eldh et al. [10] observed less than 5% of students having variations that would consist of non-obvious test cases. This was the largest discrepancy in relation to our measures. The semester of 2018.S2 had 15% (control group) and 28.6% (PB&J group) "non-obvious" test cases trying to force errors. The group that participated in

Table 3: Analysis of the test cases written by students from three different semesters.

	2017.S2 (*n=31*)	2018.S1 (*n=32*)	2018.S2 control gr. (*n=20*)	2018.S2 PB&J *n=(14)*
Input data				
Precise	12.9%	18.2%	30%	35.7%
Imprecise	87.1%	81.8%	70%	64.3%
Expected outcome				
Clear	71.0%	69.7%	65%	92.9%
Unclear	29.0%	30.3%	35%	7.1%
Step-by-step description				
Sufficient	58.1%	48.48%	70%	85.7%
Insufficient (ambiguous)	41.9%	51.52%	30%	14.3%
Variation of Test Case				
Non-obvious	32.3%	18.2%	15%	28.6%
"Happy path"	67.7%	81.8%	85%	71.4%
Number of steps				
Average	2.97	3.24	3.58	5.77
Std Deviation	1.85	1.50	1.68	3.35
Median	3	3	3	5
Mode	1	3	3	3

the PB&J activity did slightly better. However, the highest rate was the one from semester 2017.S1 (32.3%) while 18.2% of test cases from 2018.S2 tried to avoid the "happy path". The common fact among all of those analyses is that the vast majority of test cases continues to test the "happy path".

- **Number of steps**. The number of steps is significantly higher in the PB&J group from 2018.S2 in comparison to all other groups that did not have that activity but have similar values. Although the other three semesters have different averages, their median value is the same (3) while the median value of the PB&J group is 5. However, the number of steps in that group has a high standard deviation (3.35), since a few test cases had around ten detailed steps.

Apart from the impact of the PB&J activity in test case writing, which is an activity particular from our study, a critical difference when comparing to the work of Eldh et al. [10] is that their analysis was based on test cases written for the same system. In our case, there are 6 to 7 different projects each semester. Therefore, this could justify potentially discrepant analysis values. However, on the contrary, our numbers were closer to theirs. When comparing the results of our analysis in the four common characteristics shared with the reference study [10], our results were mostly equivalent except for the variation of test cases.

5.3 Students Grades

Since the participation on the activity was voluntary, one could argue that self-selection bias can take place. If this would be the case, there was a possibility of only students of better attendance or those who perform better would volunteer. This argument can be reinforced especially because this class took place in a week where students would be skipping classes due to other exams. Thus, this may lead to the thought that less dedicated students would not join the exercise and only students who would naturally perform better attended to the optional PB&J class.

To verify if this was the case, we compared the average of grades from both groups to see if there was a major difference that would distinguish a group as performing better than the other (Table 4). We considered the grades from the first exam, which happened weeks before the exercise, and the grades from the second exam, which happened three weeks after the exercise and included the question on test case writing.

When looking at their grades, the group of students that volunteered to participate in the PB&J activity had, in fact, a slightly lower average grade (about 10%) than the group that did not participate in that exercise. So, the self-selection bias, in this case, did not necessarily cause an uneven balance in terms of a group being composed of students that perform better than the ones in the other group.

Table 4: Comparison of grades (maximum of 10) from students of the control and PB&J groups.

	Control group (*n=20*)		PB&J (*n=14*)	
	Avg	**SD**	**Avg**	**SD**
1st exam	6.1	1.8	5.5	1.4
2nd exam	6.5	1.7	6.9	1.7

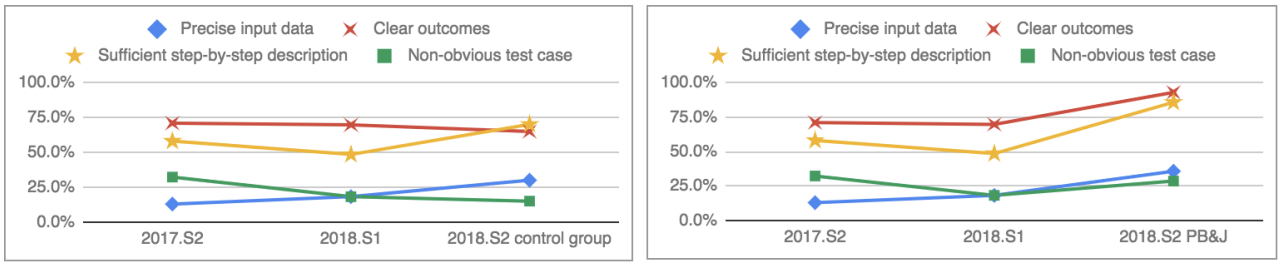
5.4 Students Feedback

The 14 students who participated in the PB&J activity received a feedback form in the week after the software testing class, with three open-ended questions. There were eight responses out of 14 participants. The first question was about their opinion on what was the main message in the PB&J exercise. Most of the respondents mentioned the importance of clear and well-detailed steps. One of them mentioned *"the importance of good writing of test cases, with details and without ambiguity in writing"*.

The second question concerned their perspective on the connection between the exercise and test cases. They all agreed that there was a clear connection between the PB&J and test case writing. One of the students stated that *"the connection I got was that when you are going to perform some test case, try to include as many details and variables as possible"*. Another student mentioned that it made him think from the perspective of a software tester who would manually execute such steps on a test case to test a system.

The third question asked if playful interventions such as the PB&J exercise would distract students from the course's learning objectives. Most of the responses mentioned that such types of classes are complementary to traditional lectures. They also mentioned it as a way to improve learning. According to one of the statements, such types of activities *"help to make clear some aspects that are more subtle and may go undetected in a lecture, like the importance of clearly specifying a step in the test case, even if it seems trivial to the writer. I don't believe that this kind of dynamic is distant*

Figure 2: Line charts comparing control group (left) with the group that followed the PB&J activity (right).



from learning objectives". One of the comments had a suggestion of explicitly bringing more testing concepts to the PB&J activity: *"It was great but I believe in this lesson I missed learning more concepts from white and black box testing and so on"*.

The students feedback we collected reinforces the Instructional Humor Processing Theory [24] about humor in the classroom creating a positive effect, making content relevant and more clear and not distracting students from the instructional message.

6 DISCUSSION AND LESSONS LEARNED

The line charts (Figure 2) shows the PB&J group with increases in the evaluation of the characteristics of a good test case. The step-by-step description, clear outcomes and the number of steps were the characteristics having the most significant increase when compared to the control groups. When directly comparing the PB&J group with the control group of the same semester (2018.S2), the participants of the PB&J activity performed better in all 5 characteristics, although "precise input data" and "non-obvious test case" did not have major improvements.

The visible increase in practically all of the characteristics when comparing the previous semesters with the PB&J group and control group of 2018.S2 may suggest that. However, students had three weeks between the PB&J activity and the exam. During the project postmortem meeting in the end of the semester, many students that were absent in the PB&J activity said their colleagues told them how it was. Thus, it may have influenced the quality of their test cases.

Using the Peanut Butter and Jelly sandwich to introduce test case writing demanded minimal adaptations from its original usage reported for CS introductory classes. The most significant difference concerned the contextualization and discussions. We had some reflections on the approach and compiled some lessons learned shared next:

- **Focus on non-obvious paths**— The results did not lead to a significant increase in the percentage of test cases that avoid "happy paths", which are the obvious test cases that are more difficult to lead to errors. This is something to be explicitly addressed when performing the exercise again.
- **Number of steps**— Some students took the compromise of providing well-detailed test cases very seriously and provided numerous steps (more than 10) in their test cases sometimes just for simple activities such as logging into a system and accessing a list of items. During the activity, it should be enforced that the number of steps is not as important as being clear. The three test cases with more than 10 steps did not specify precisely the input data.
- **Introduce more testing concepts**— One of the students indicated in the feedback form that they missed learning some testing concepts. This is something feasible that can be adapted. For instance, the PB&J sandwich algorithm could be tested using white-box techniques presented during the activity.
- **Involve students in the execution of script**— Only the teacher executed the PB&J sandwich script. It would be important to assign to some of the students the task of executing the scripts of other groups, so they could "test" the script made by a colleague and check if it is detailed enough.
- **Explore the possibility of automation**— The initial focus of the activity was on manual testing, but this activity has very good potential for introducing the notion of automated testing as indicated in our initial activity describe in section 3.
- **More playful activities in the classroom**— The experience of introducing playful activities was initially seen as risky in the case the link between the course content and the activity not being made. Taking risks is important to move forward. The successful result of this activity stimulates the authors to keep improving it and also experiment with new, playful approaches.

6.1 Limitations

The threat on the decision of tricky borderline cases identified in the reference study [10] also applies in our case. In addition, the researcher bias in the evaluation may have posed a threat, with potential inclination to evaluate better the semester having the PB&J activity. However, during the corrections he could not identify which student participated or not in that activity. There is also a potential self-selection bias of the results from the intervention being introduced in an optional lecture format, which was discussed in subsection 5.3.

Another threat to the validity of these results consists of the heterogeneity of test cases written on the exam. Since students had to write a test case of their project, they would come up with different scenarios. Therefore, they are not being evaluated on the same test case that is being written.

Although there was an overall visible increase in test case quality, as previously discussed, it may be explained by the contact that students participating in the PB&J activity had with students who did not participate. The social ties potentially allowed them to inform the absent students about the details of the activity, thus influencing their perspective on the subject.

Students are suggested to have rotating roles in their project teams. However, they typically do not follow that suggestion. Therefore, a person responsible for creating the test artifacts in their project would perform better in the exam question about test cases even if they did not participate in the PB&J activity.

6.2 Applying PB&J in other SE topics

After applying the PB&J dynamic in the context of algorithm thinking and test case writing, we realized that it could be applied to a subset of topics in SE. For instance, test design, requirements specifications and use cases. Taking the last example, use cases (UC) describe a scenario and typically a sequence of user-system interactions. Many times these interactions are poorly described. The same PB&J logic can be applied here. For instance: present UC to students and tell them to write one; do a PB&J exercise and discuss it; write the UC again; discuss how the UC improved. Thus, it is possible to explain the relevance of good/bad use cases drawing a parallel with PB&J instructions.

The main steps that we believe that can be reused to difference SE situations are:

(1) Bringing some funny dynamic that can engage students;
(2) Motivating them to work in groups, exchange experiences and have a good time;
(3) Relate the PB&J dynamic to a SE topic, by discussing the lessons learned could apply to that context.

7 CONCLUSIONS AND FUTURE STEPS

As an attempt to improve the learning of algorithmic thinking and the quality of test cases produced by students, we used a playful approach based on a Peanut Butter and Jelly sandwich activity. To instil algorithmic thinking, we applied the activity in a multidisciplinary group with students from three different universities including some that had never coded before. To introduce test cases we performed the activity in a Software Engineering course but with adaptations to the particularities of test cases, exploring important concepts such as inputs expected outcomes and order of execution.

The first learning experiment applying the PB&J activity to algorithmic thinking was important to understand how we could include humor during the dynamic, and how we could adapt it to other learning scenarios, as we made to students write adequately test cases. The fact that we had students with a very diverse background also gave a glimpse of the impact that the activity had in each different profile.

To verify if the activity had any significance on the algorithmic learning, we verified the engagement of the students during the course, and after one year we made an open interview with them to map the influence of such dynamic to their learning. This time interval gave students perspective over their learning journey from that very beginning to the current point in time and also made them more comfortable to give honest feedback without the fear of being assessed.

The fact that people with no computer science background was engaged in the activity as much as those that in some cases already have a couple of years of experience, creates a great opportunity for further exploration in terms of inclusion of non-stem students in particular and other underrepresented groups as well in the field of software development.

In the second learning experiment, we could apply the PB&J activity in the context of test case writing, without major adaptations in the steps executed in the algorithmic thinking activity. We could find evidence that the PB&J dynamic can help in a broad range of activities across the curriculum without much modification.

To verify the impact on the quality of the students test cases, we compared the test cases of a group that participated in the PB&J activity against the results of students who did not participate in the activity, which worked as control groups: one from the same class of students and two from classes of previous semesters.

According to our analysis, the PB&J activity positively influenced the quality of students test cases. Test cases written by students who participated in the activity had a significantly higher level of detail and less ambiguity than the other groups. They were more explicit about the inputs and expected outcomes and also had the average number of steps higher than the other groups. However, it did not influence the creation of non-obvious test cases —those that make easier to find errors —since it was just equivalent to the control group that scored better in that characteristic. According to the feedback that was given, students enjoyed the activity very much and had the perception that playful activities help to learn new concepts without distracting from the primary purpose.

As some improvements in the activity tailored to test case writing, we suggest finding a way to explore variations of test cases, which would end up with a non-obvious test more capable of finding errors. Also, emphasizing to students that the important thing is not having too many steps but rather having detailed and unambiguous instructions.

In special, we had a high interest in identifying if the adaptation of the PB&J to writing test cases would have a positive impact. We see this activity as very versatile, being feasible to teach many activities other than algorithmic thinking. For instance, it can be very useful in the context of use case writing. Below we mention some additional future works that we intend to do:

- Applying the PB&J activity to teach other contents related to software test, such as automated test.
- Going deeper in the fourth part, where the relation between the sandwich's recipe and algorithms, in general, is made. Maybe splitting the group by coding proficiency would bring some insight into how far into basic concepts we could go.
- Offering other dynamics in classrooms that request the code creation to execute actions to achieve some goal. Known approaches that offer similar ideas are Swift Playground [13] offered by Apple and Code.org [4].

REFERENCES

[1] Karen Anewalt. 2008. Making CS0 fun: an active learning approach using toys, games and Alice. *Journal of Computing Sciences in Colleges* 23, 3 (2008), 98–105.

[2] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41.

[3] Dennis J Bouvier. 2003. Pilot study: living flowcharts in an introduction to programming course. *ACM SIGCSE Bulletin* 35, 1 (2003), 293–295.

[4] Code.Org. 2019. Code.Org. https://code.org Last accessed 15 May 2019.

[5] Software & Systems Engineering Committee et al. 2008. IEEE standard for software and system test documentation. *Fredericksburg, VA, USA: IEEE Computer Society* (2008).

[6] Lee Copeland. 2004. *A practitioner's guide to software test design.* Artech House.

[7] Cheryl L Coyle and Heather Vaughn. 2008. Making Peanut Butter and Jelly Sandwiches: Do Students from Different Disciplines Approach This Exercise Differently?. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 52. SAGE Publications Sage CA: Los Angeles, CA, 624–628.

[8] Janet Davis and Samuel A Rebelsky. 2007. Food-first computer science: starting the first course right with PB&J. *ACM SIGCSE Bulletin* 39, 1 (2007), 372–376.

[9] Stephen H Edwards and Zalia Shams. 2014. Do student programmers all tend to write the same software tests?. In *Proceedings of the 2014 conference on Innovation & technology in computer science education.* ACM, 171–176.

[10] Sigrid Eldh, Hans Hansson, and Sasikumar Punnekkat. 2011. Analysis of mistakes as a method to improve test case design. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation.* IEEE, 70–79.

[11] Vahid Garousi and Mika V Mäntylä. 2016. When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology* 76 (2016), 92–117.

[12] Juan Pablo Hourcade, Olga I Garcia, and Keith B Perry. 2007. Learning observation skills by making peanut butter and jelly sandwiches. In *CHI'07 Extended Abstracts on Human Factors in Computing Systems.* ACM, 1753–1758.

[13] Apple Inc. 2019. Swift Playgrounds. https://www.apple.com/swift/playgrounds/ Last accessed 15 May 2019.

[14] Cem Kaner. 2003. What is a good test case. In *Software Testing Analysis & Review Conference (STAR) East.*

[15] Tarun Khanna, Ranjay Gulati, and Nitin Nohria. 1998. The dynamics of learning alliances: Competition, cooperation, and relative scope. *Strategic management journal* 19, 3 (1998), 193–210.

[16] Neelam Kher, Susan Molstad, and Roberta Donahue. 1999. Using humor in the college classroom to enhance teaching effectiveness in'dread courses'. *College Student Journal* 33, 3 (1999).

[17] Debra Korobkin. 1988. Humor in the classroom: Considerations and strategies. *College teaching* 36, 4 (1988), 154–158.

[18] Gary Lewandowski and Amy Morehead. 1998. Computer science through the eyes of dead monkeys: learning styles and interaction in CS I. In *ACM SIGCSE Bulletin*, Vol. 30. ACM, 312–316.

[19] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing.* John Wiley & Sons.

[20] Rudolf Ramler and Klaus Wolfmaier. 2006. Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In *Proceedings of the 2006 international workshop on Automation of software test.* ACM, 85–91.

[21] Yolanda A Rankin and Jakita O Thomas. 2016. Leveraging food to achieve 100% student retention in an intro CS course. *Journal of Computing Sciences in Colleges* 32, 2 (2016), 127–134.

[22] Samuel A Rebelsky. 2005. The new science students in too much, too soon an abbreviated, accelerated, constructivist, collaborative, introductory experience in CS. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 312–316.

[23] Susan H Rodger. 2002. Introducing computer science through animation and virtual worlds. In *ACM SIGCSE Bulletin*, Vol. 34. ACM, 186–190.

[24] Melissa B Wanzer, Ann B Frymier, and Jeffrey Irwin. 2010. An explanation of the relationship between instructor humor and student learning: Instructional humor processing theory. *Communication Education* 59, 1 (2010), 1–18.