Mining Jewels Together: Debating about Programming Threshold Concepts in Large Classes

Manuel Selva, François Broquedis

To cite this version:

HAL Id: hal-04383009
https://inria.hal.science/hal-04383009
Submitted on 9 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Mining Jewels Together: Debating about Programming Threshold Concepts in Large Classes

Manuel Selva
manuel.selva@inria.fr
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
38000 Grenoble, France

François Broquedis
francois.broquedis@grenoble-inp.fr
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
38000 Grenoble, France

ABSTRACT
Computer science has its threshold concepts as any academic discipline that students struggle to understand deeply. In this work, we report on using an active learning method called scientific debate to teach programming threshold concepts in large unplugged classes. This method involves students by having them defend their position with arguments and scales up with the number of students by leveraging collective intelligence. We present in detail how we apply scientific debate in a programming class. We also collect and discuss student exchanges during debates and gather feedback after the last debate. Students report they stay more focused and motivated during class with scientific debate compared to traditional transmissive lectures. They also indicate that they understood the goal of this new pedagogical contract. Our experience report could help programming teachers willing to replace/complement transmissive lectures. They also indicate that they understood the goal of this new pedagogical contract. Our experience report could help programming teachers willing to replace/complement transmissive teaching with a methodology involving the students.

KEYWORDS
active learning; threshold concepts; scientific debate; programming

1 INTRODUCTION
Threshold concepts are these jewels that, once crossed, strongly impact learners. Following their initial definition by Meyer and Land [14, 15], these concepts are troublesome, transformative, integrated, and irreversible. Said differently, threshold concepts are hard to grasp; they change how students perceive a given discipline; they allow making relations within a discipline, and students cannot forget them once learned. In the context of computer science, particularly in learning to program, the question of identifying threshold concepts has been studied [2, 23, 27, 32] since the introduction of the concept by Meyer and Land. Even if the question is not closed, these works identified and discussed several concepts as being threshold concepts or at least potential ones. For example, the program dynamics [27, 29] concept is a threshold concept. We refer to the following concise sentence [1] to give the reader an understanding of what this concept is about: “A running program is a kind of mechanism, and it takes quite a long time to learn the relation between a program on the page and the mechanism it describes.”

Once teachers identify a threshold concept, the question arises about how to help students cross it. Practicing a lot is a part of the answer. That being said, teachers have wondered how to speed up the learning curve regarding these concepts since the early days of teaching programming. As in any discipline with large cohorts of students, programming teachers also faced the question of using pedagogical methods scaling with the number of students. The traditional way of answering this question consists of carefully prepared transmissive lectures.

2 RELATED WORK AND BACKGROUND
Meyer and Land introduced the notion of threshold concepts twenty years ago [14, 15]. Since then, the computer science education community has worked on applying this theory to its field. Many studies focus on listing the threshold concepts of computer science [2, 27, 29, 32] and on defining a methodology to decide whether a computer science concept can be considered a threshold or not [22, 23, 33]. The conclusions suggest that one should take students, teachers, and professionals’ points of view into account to understand computer science threshold concepts better.

One of the main criticisms that one can formulate regarding transmissive lectures is that students are not scientifically engaged. Nevertheless, there exist active learning approaches aiming at involving students, making them actors of the class. Scientific debate [9] is one of these approaches developed in the context of teaching mathematics. In this socio-constructivist approach, students become actors in the class by having to defend with arguments their position on a problem during a debate. While the debate runs, the teacher’s role is only to animate discussions without taking a position on any argument students offer. The debate runs until students no longer bring new arguments or when the amount of time allocated for it has elapsed. At this time, the teacher can give the correct answer and explain why connecting her explanations to the students’ correct and wrong arguments. Even if mathematics education researchers have proposed the scientific debate approach before the seminal work on threshold concepts by Meyer and Land, it is a perfect fit for threshold concepts and later work on the method explicitly says that it tackles the “key concepts” of a discipline [10, 18].

In this work, we report and discuss our usage of scientific debate in a programming class at the license 3 level (in the European system). Section 2 presents related work and background. Section 3 lists with details the four problems we use to animate four debates. We then report in Section 4 on the student arguments we collected during the last three years for each of the four debates. Section 5 presents student feedback on this teaching methodology. We conclude in Section 6. We emphasize that we do not tackle the question of pinpointing threshold concepts in programming but focus on reporting our usage of scientific debate to tackle four concepts already identified as potential thresholds by the community.
example, using visualization software to teach the threshold concept known as program dynamics [28]. Shinners-Kennedy proposed classroom activities with mazes to address some computer science threshold concepts [24].

Other more general works focus on the high-level properties a class and associated teachers must have to teach computer science threshold concepts [4, 12, 13, 21, 27, 29]. Several key points emerged from these works. First, students must know the potential threshold concepts’ existence and importance [27, 29]. Second, while studying a threshold concept, students oscillate between a state where they think they got the concept and a state where they face facts showing it is not the case [4, 12, 13]. This oscillation period is called the liminal space. Because of this liminal space, the teaching approach must consider the affective aspect of threshold concepts [21]. Finally, teachers must find ways to reveal a threshold concept’s profound nature [27] or the “underlying game” [16] in Mayer and Land’s words.

Without a relationship to threshold concepts, Think-Pair-Share (TPS) [11] and Peer Instruction (PI) [3] are two lightweight active learning methods already in use in the context of large computer science classes. Both methods are similar to scientific debate, requiring students to think and debate on a given problem. In the case of PI, as for scientific debate, students must vote for a given answer, increasing scientific engagement. In the context of teaching programming, TPS can strongly impact learning outcomes, increases student engagement, and is easy to implement [7, 8]. PI fosters deep understanding [20, 25, 26] and helps build technical communication skills [19].

Legrand proposed an active learning teaching method called scientific debate [9, 10] to tackle threshold concepts in the context of mathematics. This method has mainly been used in mathematics [5, 18, 31]. Compared to TPS and PI, scientific debate explicitly targets threshold concepts. Said differently, scientific debate requires more student involvement, preparation on the teacher’s side, and more time in class. In our programming class, a single scientific debate occupies a full 90-minute lecture. The method leverages a short, nevertheless conceptually complex (since it relates to a threshold concept) and potentially controversial problem to engage all the students and highlight pre-conception to deconstruct the wrong ones.

3 FOUR SCIENTIFIC DEBATES

In our teaching context, we focus on four concepts identified by the community as candidates for being threshold. This section introduces our programming class and presents the four problems given to students for each of the four threshold concepts.

3.1 Organization of our programming class

The programming class title is Basis of Imperative Programming (BIP) and uses Python. It stands as the first programming class in our computer science curriculum. As the title suggests, its primary goal is to focus on concepts rather than mastering Python. While around 95% of the 240 students attending the class already have some programming experience, the audience’s level is very heterogeneous. Students with previous programming experience fit into three groups with three different backgrounds. In the group with the most significant programming experience, representing 15% of the 95%, students already followed 200 hours of programming courses, mainly in Python. In the group with the lowest experience, representing 20% of the 95%, students had less than 50 hours of programming courses before reaching our class. In the last group, students attended 120 hours of programming courses before.

The class comprises four sequential chapters corresponding to the four threshold concepts and lasts for a semester made up of 11 weeks. Chapter One lasts three weeks, chapter Two lasts three weeks, chapter Three lasts three weeks, and chapter Four lasts two weeks. Each chapter starts with a 90-minute lecture with all students, including the scientific debate, to introduce the threshold concept. The students then have one lab session and one unplugged session per week in groups of 30 students with a dedicated teacher. These per-group sessions aim to implement real programs on a machine and do paper exercises related to the chapter’s threshold concept to help students cross it while simultaneously learning technical skills. Each of the four 90-minute lecture with a scientific debate follows this structure:

1. an introduction by the teacher followed by the exposition of the problem and the proposed answers;
2. enough time (at least 10 minutes) for each student to think about the problem on their own and discuss it with neighbors;
3. a first individual vote followed by the posting of the result of the vote for the entire class;
4. a debate between students where each one defends her position;
5. a second individual vote followed by the posting of the result of the vote for the entire class;
6. another discussion time for students who have changed their minds to explain why;
7. a restructuring by the teacher, taking up the arguments heard and explaining why they are correct or not, called the institutionalization phase.

This organization, respecting the principles of the scientific debate method allows students to think individually about the problem first. The individual vote aims to promote scientific engagement among all students. The debate is the heart of the class, where students are encouraged to explain what they think to others. Its purpose is to let them defend their position with arguments on their terms. The results of the new vote provide feedback on how convincing the students’ arguments were. Students are again encouraged to speak up and explain which arguments have convinced them and why. Finally, the teacher’s role is to take up all the arguments heard, indicating which are correct and which are not, and then explaining why. For wrong arguments, the teacher explains where can be the origin of the misconception.

3.2 Program dynamics

The first threshold concept tackled in the class is the program dynamics one we presented in the introduction. The problem presented for debate to the students is the following: What does the Ruby program in Listing 1 print on screen? Along with the program and the question, the teacher proposes the following answers to students:
Listing 1: Problem to debate about program dynamics

```python
def f():
    print "welcome 
print " cool"
return
end
print " in BIP 
print "for the season *
end
def h(phrase):
pos = 0
while pos < phrase.length:
    ch = phrase[pos].chr
    if ch == ")" then break
    elif ch == "(" then next
end
print ch
end
def i(x, y):
    return x * y
end
```

Listing 2: Problem to debate about data structures

```python
def do_stuff(nb):
r, m = [], 0
for _ in range(nb):
    r.insert(m, i)
    m += 1
r = open("out.data", "w")
for elem in r:
    print(elem, file=f)
return r
```

### 3.3 Data structures versus abstract data types

The second threshold concept we address is data structures [17, 32]. The question students debate on is how many CPU instructions are required to execute the Python program shown in Listing 2, considering the average overhead induced by executing one Python statement is approximately 100 CPU instructions. We then propose the following answers:

- A. 100 × (nb × x) with x < 1000
- B. 100 × (nb × x) with x < constant
- C. 100 × (nb × x) with x depending on the results of randint
- D. More than the 3 answers above
- E. Something else

This simple program allows students to intensely discuss data structures even if they have never heard about the term. Indeed, most have already used Python list objects and know they can insert and remove elements from them. Nevertheless, only some have already wondered what is happening behind the scenes. Explicitly asking for counting CPU instructions force them to think about what happens internally and introduces the distinction between the operations available on a Python list (the abstract data type) and how the interpreter implements it (the data structure).

Another advantage of this simple program is that running it in front of students with a not-so-large number, such as 500, 000, shows them the tangible impact of using an inadequate data structure: the program runs for 30 seconds!

During institutionalization, the teacher explains that what Python calls lists are dynamic arrays. Again, we use visual representation to support this explanation to have students see the shift of all elements of one position to the right when inserting at index zero.

### 3.4 References

The third threshold concept we tackle is references [2, 30, 32]. For this one, students must guess the output of the Python program shown in Listing 3. The proposed answers are the following:

- A. white and blue columns
- B. white and blue lines
- C. white and blue chessboard
- D. Something else

Again, despite its apparent simplicity, this program allows students to dig into the concept of references deeply. Indeed, many students have already encountered the * operator. However, few have ever been interested in what its use implies and what a Python list contains in the interpreter.
The last concept we study is recursive functions, also identified as about recursive functions. Indeed, to be able to anticipate what will happen for the code shown in Listing 4 with the following propositions:

- If one can only understand recursive functions with a good understanding of program dynamics in the presence of recursive calls is supported again by a visual tool we designed to show the evolution of the call tree on the first call to the recursive function with $i$ equals 5.

### 4 STUDENTS EXCHANGES DURING DEBATES

This section reports and discusses the arguments given by students during each of the four debates. We collected these arguments during three class instances: fall 2020, fall 2021, and fall 2022.

#### 4.1 Program dynamics

Here is the list of the arguments we got from students formulated in our teachers’ words:

1. The character $!$ cannot be printed since the corresponding print statement is located right after a return statement.
2. The character $.$ must be printed after the new season since the print statement is located right after the call to function $g$.
3. The answer must start with $welcome$ since it is the first print statement.
4. The answer cannot start with $welcome$ since the first print statement executed is the first one not in a function.
5. I do not know anything about error handling, but something must repeat because of a division by zero and retry.
6. with fun is not printed since $x$ is 0 when reaching this line.

These arguments point out that students attending the class already have previous programming experience and know the existence of the program dynamics concept. Nevertheless, arguments (2) and (3) demonstrate that many still need to cross this threshold. In particular, only one-third of the students provide the correct answer on the first vote each year.

While students may have several other misconceptions, they only reported two wrong arguments, as mentioned above. One reason is that getting student participation in this first debate is challenging. Indeed, this debate occurs during the first class of the first day of our computer science curriculum when students had just arrived. Students barely know each other at this time, and speaking in front of others to defend a position is intimidating. Moreover, the pedagogical method is new for them since they only followed transmissive lectures in the context of large classes before reaching our class.

#### 4.2 Data structures versus abstract data types

Here are the arguments we collected for the second debate:

1. It depends on what we insert and, consequently, the ratio between even and odd numbers.
2. The running time of $randint(0, nb)$ increases linearly with the value of $nb$.
3. The running time of $len$ increases linearly with the list size.
4. The running time of $len$ is constant.
The first conclusion we can draw, considering the students’ answers, is that the four concepts we tackle are thresholds. Hence, in our context, it is mandatory to insist on these concepts even if students already have non-negligible programming experience. Indeed, if most of them had crossed the four thresholds, we would have expected to get a significant fraction of correct answers to each of the four problems, and the debates would have been sterile. That was not the case. Said differently, students need time to cross threshold concepts. That is why we believe the first programming class of a computer science curriculum must insist on these thresholds even if many students already have previous programming experience.

The second conclusion from our experience is that the teacher must propose a well-calibrated problem to debate a given threshold successfully. It must be difficult enough to get different opinions from the audience, but not too difficult either, as students may struggle to have opinions on something too difficult to understand. More importantly, the problem must focus on the threshold and avoid as many details as possible. For example, the feedback we had animating the debate on recursive functions is that we should remove the * operator to initialize the list with zeros. Indeed, some students got lost on this point they knew to be important from the previous debate on references instead of focusing on the dynamics of recursive functions.

Another evident, nevertheless crucial, element regarding the success or not of a debate is the class’s atmosphere. Indeed, it is crucial for students to feel and know they can freely give their arguments without being judged. Here, the teacher plays a central role in ensuring everyone respects every opinion during the debates. While this statement generally concerns the scientific debate method, we believe it is crucial in our programming class context. One reason is that we often have few students genuinely passionate about programming in front of us that may already have crossed the thresholds. These students may be intimidating to others if they always give the correct arguments during the debate. It happened several times, but thanks to the excellent class atmosphere and the fact that many students did not have already crossed the threshold, these situations led to exciting debates between the few students with the correct arguments and the others, among which wrong arguments convinced some.

5 STUDENTS FEEDBACKS

For five years, our university has asked students to evaluate classes when they finish. Students can leave comments about the course in a free text field on the evaluation form. We introduced scientific debate in 2020. Also, because of the pandemic that year, only the first two scientific debates occurred. Table 1 summarizes the results of these evaluations concerning the large classes of the BIP course. We accounted for the number of positive and negative feedback explicitly concerning the large classes. The neutral line is the number of students whose feedback only concerns group lab or unplugged sessions and does not mention large classes.

Negative feedback on the large unplugged classes has significantly decreased since we introduced scientific debate.

<table>
<thead>
<tr>
<th></th>
<th>2018</th>
<th>2019</th>
<th>2020</th>
<th>2021</th>
<th>2022</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>neutral</td>
<td>44</td>
<td>38</td>
<td>49</td>
<td>37</td>
<td>27</td>
</tr>
<tr>
<td>negative</td>
<td>29</td>
<td>21</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Students global feedback over the years
To get more insights from students this year, we asked for explicit feedback on the four debates. We did that at the end of the last debate to collect feedback from the 153 attending students. In this survey, we asked them to numerically indicate their level of agreement with 19 propositions listed on the left part of Fig. 1. A [t] at the beginning of a proposition means teacher. For example, the first proposition means the teacher clearly explained motivations. A [s] means I had enough individual time to think about the problem as a student. The numerical agreement values proposed were: (1) No reviews (2) Strongly disagree (3) Disagree (4) Agree (5) Strongly agree.

As shown in Fig. 1, we gathered the 19 propositions into three groups: (1) propositions about teacher explanations before the debate; (2) propositions regarding what happens during the debate; (3) propositions concerning the institutionalization phase after the debate. In the first group of propositions, the survey asked if the teacher gave clear explanations of different things and if he insisted on the importance of thinking individually to have his own opinion. In the second group, the survey asked students if they had enough time to think and discuss, if they could express their arguments without fear and judgment, and if the teacher wrote and discussed all the arguments and kept the debate focused. In the third group, the survey asked if the teacher linked arguments given during the debate with its explanations and if she kept away details. Finally, still in the third group, students were asked if they perceived the complexity of the threshold and if they thought they still had remaining work to cross it.

From the results of the first group propositions and the [s] understood complexity one in the last group, it is clear that students understood both the pedagogical purpose of scientific debate and the existence of programming threshold concepts and their complexity. It is a good result since, as stated in Section 2, students must be aware of the threshold concepts’ existence and importance [29].

The second conclusion from the answers in Fig. 1 is that students were comfortable with this new pedagogical method. As teachers, one of our biggest fears when introducing the method was having few or no student debates. Nevertheless, except for the first debate, as already mentioned in Section 4, it was easy for us to maintain involved participation from students.

Finally, the results in Fig. 1 suggest that we must take more care to highlight diverging arguments and give more time for individual reflection. Indeed, the two associated propositions have the highest number of disagree and strongly disagree responses.

Another survey part was tree free forms for positive points, negative points, and enhancement proposals. In the positive points form, 42 out of the 153 students say the class format helps them stay focused during the class. In the negative part, 16 said the reflection time before the debate was too short, while 2 said it was too long. In the enhancement proposal, 11 students suggested doing more large classes, which we had never seen before introducing scientific debate.

Finally, the survey asked students to give a global note from 1 (bad) to 6 (excellent) on the scientific debate sessions. The average value from the 153 answers is 5.5/6, which confirms that students understood the method and found it beneficial.

6 CONCLUSION

This paper presented our usage of the scientific debate method in the context of a programming class. The feedback provided by students up to now encourages us to pursue in that direction since it positively impacts students’ motivation and concentration.

Also, even if this could have been identified by other means, students’ vote results and arguments during debates indicate that many students have not crossed programming thresholds when they reach our curriculum. It is a logical conclusion of the nature of threshold concepts: they take time to be crossed. From our teacher’s perspective, this suggests that we need to keep a closed eye on students’ understanding of these concepts.

As a final note, the move to scientific debate has been a stimulating experience from the teacher’s point of view. It forced us to sit together to identify and agree on the thresholds. Also, scientific debate requires us to adopt a different role than what we are used to in transmissive lectures. Although it was not easy initially, this was a challenging experience for us teachers.