Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course

Olivier Goletti*
ICTEAM/INGI, UCLouvain
Louvain-la-Neuve, Belgium
olivier.goletti@uclouvain.be

Kim Mens
ICTEAM/INGI, UCLouvain
Louvain-la-Neuve, Belgium
kim.mens@uclouvain.be

Felienne Hermans
LIACS, Leiden University
Leiden, The Netherlands
f.f.j.hermans@liacs.leidenuniv.nl

ABSTRACT

In programming education, explicit strategies are gaining traction. The reason for this study was to improve an introductory programming course based on a problem-based methodology, by using more explicit programming strategies. After analysing a previous run of this course for first year undergraduate students, we concluded that such strategies could improve learning transfer for students across the different weeks of the semester. We introduced four instructional strategies to tutors with close to no pedagogical background: explicit tracing, subgoal labeled worked examples, Parsons problems and explicit problem solving. These explicit programming strategies aim to decrease cognitive load. Tutors tested these four strategies in the course. Our goal was to explore how tutors could benefit in their tutoring from explicit strategies. Interviews with the tutors show that the easiest and most effective of the tested strategies were best used. For the more elaborate strategies, more time should be devoted to explain and model them or they can be misunderstood and misapplied. We conclude that four criteria are key to successfully using an explicit strategy: easy to understand, straightforward to apply, useful on the long term and supported by literature.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **CS1**.

KEYWORDS

explicit programming strategies; cognitive load; problem-based learning

ACM Reference Format:
Olivier Goletti, Kim Mens, and Felienne Hermans. 2021. Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. In *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '21), June 26-July 1, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456348

*Also with LIACS, Leiden University.

ITiCSE '21, June 26-July 1, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8214-4/21/06…$15.00
https://doi.org/10.1145/3430665.3456348

1 INTRODUCTION

Introductory programming courses are increasingly taught using problem-based and project-based learning methodologies (PBL) [23]. At UCLouvain, an introductory bachelor computer science (CS1) course given to the future civil engineers and bachelors in CS, is taught following a methodology inspired by PBL. In this CS1 course, tutors facilitate students' work during lab sessions. Tutors are more senior students who follow a small pedagogical training on the PBL strategy used in the course. In this study we aim to explore **how tutors could benefit in their tutoring from explicit programming strategies**. LaToza et al. [17] define an *explicit programming strategy* as: a *"human-executable procedure for accomplishing a programming task"*. Prior work has shown that the strategies we use in this study are effective [8, 21, 25, 31]. Therefore, we focus our study instead on tutors to assess how comfortable they are with adopting new instructional strategies.

These are the research questions we are interested in:

RQ1 How did tutors apply and adapt the strategies?
RQ2 What strategies did tutors prefer and why?
RQ3 What do the tutors think about using explicit programming strategies?

Our study focuses on four strategies from research literature: 1) **Explicit tracing** [31], 2) **Subgoal-labeled worked examples** [25], 3) **Parsons problems** [8] and 4) **Explicit problem solving** [21]. We conducted this study by following the tutors in weekly focus groups and interviewing them at the end of the semester.

2 BACKGROUND

This section presents the theoretical concepts of cognitive load theory and learning transfer based on which we analysed the problem-based methodology of the course and selected the explicit programming strategies we use.

Cognitive Load Theory. Cognitive load theory (CLT) [30] is an instructional theory based on human cognitive architecture. CLT assumes a limited working memory. The impact of CLT on instruction design is that one should try to reduce load on the working memory when teaching new material. Too much context or unnecessary information when teaching increases the cognitive load on the learner. The four strategies in this study follow the instructional idea of CLT: minimizing cognitive load and helping to apply generic skills [29], for example through the use of worked examples, automatizing rules or using external representations [14].

Learning Transfer. Learning transfer can be defined as reusing previously learned knowledge in a new context [3]. It is important to note that learning transfer is an active process. Students attempting

to transfer benefit from prompts [3]. They can also transfer better if they are conscious that they are attempting to do so. This is typical in a problem-solving situation. The instructional implication of this is that *recall strategies*, which invite a learner to recall previously learned knowledge and solutions to similar problems, can facilitate learning transfer.

Problem-Based Learning. Problem-based learning (PBL) is a student-centered instructional methodology where students work in small groups on real life problems introducing new learning material [1]. The learning process is mostly self-directed: by solving problems and answering questions, they discover the theory by reading and studying by themselves. PBL has sometimes been criticized as an active teaching approach, because the minimal guidance it provides is in contradiction with CLT principles [15]. Other works nuance this critique by asserting that PBL does include scaffolding and guidance provided among others by tutors [11, 27].

Explicit Programming Strategies. Explicit programming strategies are gaining traction lately: e.g., by explicitly teaching recurring patterns [5], or using explicit strategies for tracing [31], debugging [17], code reuse [16] or problem solving [21]. Using worked examples, subgoal-labeled material or Parsons problems as techniques aiming at reducing cognitive load when learning has been studied in numerous recent studies [8, 24–26].

The four strategies explored in our study were conceived to be either recall strategies or to be explicit in the sense that, once automated, they will reduce the cognitive load associated with a specific task. We found them to be effective based on previous empirical results [8, 21, 25, 31]. These explicit programming strategies can be either a list of steps to reproduce, or meta-cognitive hints to help recall some specific technique. The goal of this paper is to explore how easily the selected strategies can be used by tutors that have close to no pedagogical background.

3 METHODOLOGY

The goal of this paper is to explore how tutors benefit from using explicit programming strategies in their instructional practice.

3.1 The CS1 Course at UCLouvain

The introductory programming course follows a methodology inspired by PBL, *"mostly to enhance deep and meaningful learning, to promote high-level capabilities, to develop student motivation and autonomy, and to promote team work"* [9]. The thirteen weeks course is followed by all first year undergraduate students majoring in computer science and civil engineering. One of the main pillars of the PBL methodology used in this course is tutoring. Although PBL has been criticized by proponents of CLT (cf. Section 2), in a previous run of this course, it was shown to be effective [10]. Nevertheless, in our current analysis of this CS1 course we investigate whether tutors could benefit from more explicit programming strategies. The output of our analysis is discussed after its structure.

Course Structure. UCLouvain's CS1 course is organised around small weekly projects called *missions*. Each mission introduces some new programming concepts and covers a theme. For example, the "strings and lists" theme is about DNA sequences; in the realisation phase for that mission students have to develop help methods for calculating a Hamming distance between sequences. The course uses Python and covers imperative programming and introduces programming with objects[1].

Each week starts with a one hour lecture. Students then have to read the theory syllabus and answer multiple choice and open questions provided in the exercise syllabus. During tutored sessions, students work in their usual group of six students and each tutor is assigned to a room of four groups. During a first one hour tutored session, the prepared answers to the exercises are shared and discussed. The tutor helps organizing the discussions, sending students to the blackboard. Students then have two days to submit their solution to a small weekly project. The sum-up session is the second and last tutored session of the week. During this one hour session, the tutor provides feedback on the submissions, discussing common mistakes and pitfalls the students have ran into. A summary exercise covering the concepts of the week is then solved by the room, before the cycle starts over again.

Course Analysis. A thorough inspection of the course material, based on the learning transfer model of Bracke [2], yielded three propositions to improve the course:

- **P1. The organisation of the learning objectives should be made more apparent throughout the entire course.** P1 is related to literature on explicit direct instruction and the way Hollingsworth et al. suggest to organize lessons [12].
- **P2. Transfer opportunities should be pointed at beforehand** in order to prepare the learner to recognize future situations in which (s)he could recall previous knowledge.
- **P3. More explicit recall strategies should be proposed in the course** to help retrieve known bits of knowledge in order to apply them to solve new similar situations.

P2 and P3 are in accordance with what CLT prescribes about instructional design. It promotes explicit strategies in the sense that we must automate content-related knowledge and so, familiarise learners to new content and enable them to transfer knowledge.

3.2 Four Evidenced-based Strategies

Our analysis of the course and the three propositions above led us to look for effective explicit programming strategies and use them in this research. In this section, we present each strategy and justify how they relate to our three propositions P1, P2 and P3: (1) **Explicit Tracing** [31] supplemented with the memory representation for more complex structures [7]; (2) **Subgoal labeled worked examples** [25]; (3) **Parsons problems** [8]; (4) **Explicit problem-solving** [21].

Explicit Tracing. Tracing code means executing a program in one's head or by hand. Research shows tracing is an important skill to read and write code, and that tracing is hard for novices [19].

The explicit tracing strategy we use is inspired by Xie et al. [31], who show a strategy that helps students trace programs while updating a memory table. Each variable encountered in the code has a line in the table and their values are systematically updated when executing the instructions of a given program. We also included the graphical representations for tracing arrays, lists and objects from Dragon et al. [7].

[1] https://syllabus-interactif.info.ucl.ac.be/index/info1-theory

Explicit tracing is in alignment with CLT because it automates the process of executing code and uses external representations to lower cognitive load. It addresses P3 because it is an explicit recall strategy that will remind students how to execute code properly.

Subgoal Labeled Worked Examples (SLWE). "Worked examples demonstrate how to apply an otherwise abstract procedure to a concrete problem" [25]. Subgoal labels highlight the steps of a generic problem solving procedure. The idea of explicitly teaching the steps of recurring patterns in the resolution of many more or less similar problems is not new [6, 28]. This is the goal of SLWEs. Margulieux et al. [25] combined the idea of worked examples with labeling the steps behind common code structures. Subgoal labels identify those generic steps in order to emphasize them. The steps suggested by the labels can be reused when reading or writing similar code. Margulieux et al. provided SLWEs to students with increasing levels of difficulty, interleaved with practice problems.

SLWEs are linked to proposition P2 and the promotion of transfer opportunities. They also aim to automate the recognition of patterns like proposed in P3. Worked examples are also a strategy identified by CLT proponents to lower cognitive load for learners.

Parsons Problems. Parsons problems consist of mixing lines of a code broken into subgoals or even in single lines. The idea is for the student to solve an exercise by reordering all the pieces of the solution that have been mixed. Those puzzles are more engaging for students [8]. The difficulty can be adjusted by adding *distractors*, unnecessary lines that have to be left out. Distractors often reflect common mistakes that novices can make. A distractor can be paired with its corresponding correct line, for example using the same label. Using Parsons problems lead to the same amount of learning as fixing or writing the same code while taking less time [8]. The idea of reordering lines of code instead of writing them is heavily inspired by CLT and diminishes cognitive load. It is a way to practice the recognition of patterns shown in worked examples, and is therefore linked to our proposition P3.

Explicit Problem Solving. The fourth strategy concerns metacognition in problem solving. By explicitly giving students a list of steps to follow and corresponding reflection questions, the aim is to help them self-regulate the process of solving a problem. It is inspired by a study by Loksa et al. [21] in which the authors identified six steps to follow: Reinterpret problem statement; Search for analogous problems; Search for solutions; Evaluate a potential solution; Implement a solution and Evaluate implemented solution.

Explicit problem solving consists of explaining the six steps to the students, giving them a handout as a reminder and asking them in which phase there are when they ask for help in a lab session.

This strategy has as main goal to automate the self-regulation that will help a learner take advantage of metacognition. It is a way to diminish the difficulty of using higher level cognitive strategies in an unfamiliar context as suggested by CLT. It is also done in an explicit way like in proposition P3.

3.3 Study Setup

The goal of this paper is **to understand how tutors could incorporate explicit strategies in their tutoring** and what were their thoughts on the use of explicit strategies. Therefore we had to find those tutors, explain the strategies to them and interview them. This section details our methodology.

To the twenty-five tutors of our CS1 course, we distributed a survey with questions on how they saw their role as tutors for the course. Twelve responded and four among them accepted to test our explicit programming strategies. We met those four tutors once a week nearly each week of the semester. During these twenty-minute meetings, the four tutors and the first author would discuss the previously seen strategies. We discussed feedback, adaptation, best practices, interrogations, etc.

Since tutors had little pedagogical experience, we introduced the strategies to the tutors gradually. This was done to not overwhelm them with too much information at once. We also thought it would give them more time to fully focus on each strategy at a time. Every two other weeks, the first author proposed a new strategy, gave the paper it was taken from to the tutors, presented its motivations and objectives, explained how to apply it, gave a usage example and answered tutors questions.

Tutors were encouraged to use these strategies with their students and to modify and adapt them as needed. Regular feedback through the weekly meetings allowed us to know what tutors tested and how they adapted the initial proposed strategy. This approach was preferred instead of a more rigid "follow these steps" approach because we trust them in the end to *"weave it all together into something that works in the classroom"* [18].

At the end of the semester, each of the tutors was interviewed for about one hour in a semi-structured way. Questions were prepared by the first author to not miss any specific point during the interview, as proposed by Kaufmann [13]. In order to answer **RQ1** on their use and adaptation of the strategies, we asked more detailed questions on what they recalled of each of the four strategies, how and when they applied it and its pros and cons. To answer **RQ2** on the tutors preference, we asked them to compare the strategies and rate them according to some criteria such as ease of application or effectiveness. The interview also included a few questions on their experience as a tutor, what changed in their practice throughout the semester and their thoughts on explicit methodologies. These questions aimed at answering **RQ3** on their views on explicit strategies. The weekly meetings were recorded and were used along with the interviews of the tutors as a source for the qualitative aspect of this study. The transcripts of the interviews were coded following the method proposed by Creswell [4]. We used codes emerging from the interviews, as well as codes induced by the themes of the different parts of the interview. The first interview was coded by two researchers and then the first author coded the three other interviews. In total, we coded 431 quotes from the interviews and the weekly meetings. The codes were then regrouped in categories that we used to answer our three research questions.

4 RESULTS

This section presents the results of the analysis of the interviews that we transcribed and coded. We answer each of our three research questions based on our coding of the interview material. Since the interviews were taken in French, the quotes provided below are translations provided by the first author of this paper.

4.1 Tutor's Use of the Four Strategies

In this section we analyse what the tutors reported on their use of each of the four strategies, by order in which they were introduced to the tutors. For each strategy, the categories are treated in order of number of coded quotes given in brackets. When no quote was coded in a category, it is left out. The final coded categories were: **Pros for tutors:** seen as a benefit of using a strategy; **Pros for students:** seen by tutors as helping the students; **Limitations** hindered students when using a strategy; **Application difficulties** hindered tutors; **Suggestions** made by tutors to improve a strategy.

4.1.1 Explicit Tracing. This strategy was the favorite of all four tutors. For tutors, it is easy to understand, simple to apply. It is useful for all students, simple to use, and helps code comprehension, testing and debugging code. Tutors considered explicit tracing as reassuring for the students. T4 said tracing would help later in their curriculum. Tutors also stated it reduces effectively the mental load for students. Tutors regarded the strategy as unfit for longer executions. They made some suggestions to improve this strategy.

Pros for students (38). Tracing was mostly used to help students identify where they misunderstood a statement or made an incorrect assumption. T3 said: *"Students can figure out by executing step by step that what they think is different from the result of the execution. By forcing oneself to write and trace, a student can come by himself to a conflict and then to the correction of the code."*

Tutors agreed it really helped students. For example T2 said: *"The idea is to write it instead of keeping it all in their head".*

Pros for tutors (25). Tracing was compared with debugging and thus felt familiar to tutors. T4 was shocked that it wasn't explicitly taught to students: *"I thought tracing code was already taught in [this course]... I think it is very useful to trace at the beginning."*

This strategy was the easiest to apply with the best results. For example, T1 described how the strategy helps students: *"It's the strategy I used the most [...] At first, one doesn't have the proper methodology to trace code. We just try to remember all variables and we just go too fast. But with the table we take our time, we update it after each statement, we calculate each expression separately, I think it helped them. I just did a reminder on this at Tuesday's lab because they asked me [...] It's just that, it is clear and it works well [...] They seemed to say they would use it during the exam."*

Limitations (12). The four tutors were unanimous to say that tracing takes a lot of time, especially for longer code.

Suggestions (7). Tutors suggested how to improve explicit tracing. E.g., one proposed to use several students "chained" to execute separate parts of the code or nested function calls. Another adaptation was to trace only chosen variables of interest in a program.

4.1.2 Subgoal Labeled Worked Examples. Tutors regarded subgoal labeled worked examples (SLWEs) more complicated to apply than the previous strategy. They had a hard time applying this strategy properly. They said they understood the underlying ideas but that it needed preparation and memorizing of the proper labels.

Application difficulties (20). Although tutors saw that identifying the subgoals for the students was helpful, they said it was difficult to articulate the difference between context steps and generic steps. Tutors could not relate to a similar strategy emphasizing the generality of the steps to write a loop. It was more implicit for the tutors so the strategy seemed *"overly theoretical"* (T3).

Tutors fell into a live attempt to find the proper steps to use a specific construct if they did not stick to the provided labels. Such a live exercise is difficult and is a reason why the strategy was designed [25]. T1 said on this topic: *"Well, if it's not explicit for us, it's difficult to explain it for the students... We know all that implicitly. But, it's never easy to explain like that if we haven't taken the time to sit down and say when I do a loop, step 1 is that, then that, etc."*

Because SLWEs need preparation and memorising, it demanded time from the tutors. This was a major hindrance in their first attempt at applying the strategy. Another difficulty was that subgoals were provided for many different constructs and are different when reading or writing. This led tutors to adapt the strategy by doing it orally and not as explicitly as proposed in the paper.

Limitations (7). A limitation mentioned by two tutors is that students would expect the solution to be given to them if the tutor often wrote it on the board to highlight the different subgoals.

Pros for students (7). Three tutors used the strategy and found that it was particularly useful for students who had more difficulties starting from a blank page. The idea of showing an example was fairly well adopted, especially during the introduction of a new concept. Tutors saw it as presenting a plan that students could refer to later on but stressed that one example is not enough and that balance should be found to avoid students expecting answers to be given. For example, T2 said: *"It's good to do this for the first time when a concept is discussed to show them how to solve a new type of exercise... It saves time rather than floundering... It provides them a well-solved reference exercise that shows the steps."*

4.1.3 Parsons Problems. This was the second favourite strategy of the tutors. They liked it because it was engaging for students of all levels. Another main advantage was that students could see more examples in less time. Nevertheless, tutors said it required time to prepare and it was not always clear when to use it.

Pros for students (28). The importance of good paired distractors was stressed since it forced students to justify their choice. One tutor tried unpaired distractors, but found that this was too hard for the students since they would try to fit all the lines of code in one solution. T3 describes its use: *"Line by line, without indentation, with distractors. To stimulate discussions in certain structures. For example to see if an* `else` *is mandatory or not."*

Tutors highlighted the benefit of Parsons problems to enable students to see more examples of solved programs in a short time, as reported in the original study [8]. E.g., T1 said *"[students are] really just thinking about the meaning. That way, they could see more different examples since they don't have to waste time writing. And learn faster, well that's the objective of the method."*

It was motivating for student not having to write all exercises by themselves. It mitigated the blank page syndrome. Tutors reported that students were more involved, more active and enjoyed the strategy; even those students who had more difficulties.

Limitations (10). Nevertheless all tutors agreed that Parsons problems, when done on paper, required time and preparation.

Application difficulties (10). The main difficulty for tutors was to identify exercises that would benefit from Parsons problems. Two tutors said they did not know when to use it and would have preferred to experience it by themselves first, to have more practice.

Suggestions (8). Some suggestions were made for Parsons problems. T3 suggested it could be used as a way of assessing student knowledge or diagnosing if a mistake is systematic, by giving them distractors on a specific misconception. T3 said that *"[with this strategy] we could see if we used a variable before assigning it or if we swapped two lines of code if it is systematic or a distraction error."*

Tutors also said that an automatic online solution might be more usable, but then they would lose the opportunity of discussion with and between the students in the classroom.

Pros for tutors (6). In order to properly invent a new Parsons exercise, T2 said tutors need to *"know how to trap students. To make good distractors, you have to know the corner cases that make a program not work"* and that he liked that. T3 said that it allowed him to put more or less difficulty in an exercise.

4.1.4 Explicit Problem Solving. This strategy was more controversial. While two tutors rated it as difficult to apply and understand, the other two used it successfully. It was straightforward to use for the students but difficult to understand because it was meta and at the same time described obvious steps of problem solving.

Pros for students (19). As with previous strategies, tutors noted that this strategy helped students who did not know where to begin. T3 said: *"some students need a course of action or they don't know what to do."*

The strategy was helpful to students. It needed to be applied systematically and if students didn't stick to it, they would burn steps and make mistakes. Self-regulation is difficult for students [20]. We can see the meta-cognitive impact when T2 said: *"They can see that it works when I ask them questions but cannot ask the question themselves."* or when T1 said: *"It's a bit like trying to work as if you are working in a group but alone."*

Tutors found this strategy complementary to the second strategy as they regarded that one as about translating a natural language resolution to code and this one about the whole process, starting from reinterpreting the statement in natural language.

This recall strategy especially matches a need identified in our analysis and was seen as such by T1: *"If you remember something that worked, it's very easy to reapply it. And taking the time to explicitly ask 'OK have I already done something that looks like that?' Sometimes it's quite silly but if you think about it for two or three minutes you find that it is the case."*

Limitations (11). Some difficulties were also mentioned. Tutors found it difficult for students to identify similar problems, especially when seeing so much new material in the course. But even tutors often only saw similar problems in a very narrow sense.

Application difficulties (9). Even though tutors saw the need to make the steps of problem solving explicit, they found it sometimes too obvious and so did students. Some tutors reported that they were not sure if it would help. They had to be convinced of the usefulness of a strategy before using it, as T4 stated: *"I didn't test it for lack of time and because I didn't really see the point."*

4.2 RQ1: Use and adaptation of the strategies

The overall impression is that tutors found the strategies useful and effective. Tutors used a strategy more easily if they could understand the motivation behind it. They would reuse a strategy when it helped the students, didn't take too much time and when it increased the students' motivation. The tutors did not hesitate to test the strategies. They adapted it to their own practice or to what they understood. Still, they sometimes reported not being sure on when to use a strategy and a need for more practice before using it with the students. T4 even said that for one strategy they felt lost: *"Do not just leave the tutors facing the paper because there is quickly a way to get lost and make mistakes."*

Tutors really need to understand the goal of a strategy to be convinced it has value and to use it properly. Otherwise, they would have a tendency to focus on the context and not on the general principle a strategy emphasises. Especially with more abstract strategies like the subgoal labeled worked examples, they needed to understand them well. Otherwise, they would struggle to find the proper labels on the spot, which is kind of the main reason for using that strategy.

A difficulty encountered was that lack of time and preparation were a brake on instructional changes. It is known that TA's and by extensions also tutors suffer from class management issues [22]. It is well possible that in this case, since they had little pedagogical background and since three of them were tutoring this course for the first time, it hampered their pedagogical confidence and hence the degree to which they could adapt instructional experiments.

4.3 RQ2: Preferred Strategies

When comparing strategies, tutors found that explicit tracing was the easiest to understand and apply as well as the most effective one. Using Parsons problems was considered easy and effective even though it took more time to prepare. The third most effective strategy was explicit problem solving according to tutors and it was also third easiest to understand. Finally, subgoal labeled worked examples was the most difficult to understand and least effective according to the tutors.

The strategies seen as more effective were not necessarily easier to apply. For example Parsons' problems was difficult to apply according to tutors. They liked to see that a strategy was useful. Even though some strategies were seen as broadly applicable, tutors mainly saw them as useful for students in difficulty. This was a major motivation for tutors to try the strategies.

Overall, we can say that when adopting strategies tutors preferred those that were easy to understand, straightforward to apply, useful for students on the long term and supported by literature.

4.4 RQ3: On Explicit Programming Strategies

Tutors found the explicit nature of the strategies effective and *"less demanding for the students"* (T1). To counterbalance explicit strategies, they said that students still need occasions to explore strategies by themselves and figure out what works best for them. They said that explicit strategies should be shown especially when introducing new material but that students should then have some less guided time to try them out and adopt or adapt them. T2 said: *"I think it is good to do things that are very explicit for a student, to*

show him how to do an exercise properly and that he can adapt it for himself."

Two tutors also feared that students would not search by themselves anymore because of explicit strategies. T1 said: *"The disadvantage is that they may not search by themselves, it will be much less personal. It will be like that. Because we said it works. But not because they have tested it and ... Maybe there are other methods that work well and they may never find out on their own."*

The tutors seem to agree that more open exercises are also beneficial for students and that less explicit methods would force students to learn how to "figure it out". They clearly associate implicit with letting the students work it out by themselves, like T2: *"Less explicit, it is rather when they have more freedom, when we give them the statement ... and we do not give a course of action... It is good from time to time to leave the students a little more on their own."*

To summarize, tutors view explicit strategies as an effective way to teach and automate good practices for students. But they have a more balanced view than expected on this topic. Indeed, for most of the tutors, an inquiry-based, less guided methodology still has its place in the course. This might be due to their own experience of the course and also by their feeling that students need to learn to search by themselves.

4.5 Discussion

The goal of this paper was to explore how tutors can benefit from using explicit programming strategies. In this section, we discuss two noteworthy aspects that were not covered by our research questions but that have been reported by the tutors: the benefits of the weekly meetings and the problem of students to translate a problem statement into a program.

First, tutors reported that they liked the weekly meetings to discuss the different strategies. This allowed them to see how they were used by the others and be encouraged to try them out. This setup was inspired by the idea of *communities of practice*. Having regular meetings allowed learners to hear and learn from each other. T1 said: *"Seeing each other every week, ... we realized for certain strategies that it was easier to use than we thought, so that was nice."*

Second, we want to come back to the comments made by all four tutors when speaking about the second strategy (subgoal labeled worked examples). They mentioned that this strategy was good to translate from a natural language solution to code by using the labels for writing constructs. The students mostly struggled with the step before however, which was reinterpreting the problem statement into a proper solution. For example, T4 said: *"It is useful if you already know the program that needs to be built. The problem is more knowing that you have to make a loop."* The tutors mentioned that the fourth strategy on explicit problem solving helped the students with this first step. So maybe, just introducing this explicit problem solving strategy first would suffice. However, extra attention needs to be put on giving the right tools and strategies to tutors to help them explain to students how to express or translate a problem statement into natural language.

4.6 Threats to Validity

A first threat to validity of this study is the small amount of tutors who participated. But our main goal was to explore tutors' use of explicit programming strategies and to discover some directions for follow-up research. It would for example be interesting to take the proposed criteria for a strategy and test them on more strategies and more tutors. A second threat is in the way the strategies were selected. Other interesting strategies could exist that may lead to other results. In particular, we might want to find an explicit strategy helping students to translate a problem statement into a natural language description of the solution. Finally, tutors were encouraged to use the strategies but were not forced to do so. One can imagine that a more controlled experiment could lead to more quantifiable results on whether a strategy is indeed properly used and adopted by the tutors.

5 CONCLUSION

In this exploratory study, we studied how four tutors used and experienced the use of four explicit strategies in a CS1 university course. The strategies were chosen because they were shown to be effective and explicit, and in accordance with the three propositions that were drawn from our analysis of the course and following instructional propositions made by cognitive load theory.

The four strategies were gradually presented to and tested by the tutors in the thirteen week course during which regular meetings were held. At the end of the semester interviews were conducted, transcribed and analysed to answer three research questions. We observed that the tutors liked these new instructional strategies even though it was sometimes difficult for them to use them properly. They neither had sufficient time to prepare their lab sessions with these strategies nor the pedagogical experience to be confident enough to try more complicated strategies.

Based upon our interview analysis, we propose four criteria for a strategy to be more easily adopted by tutors with little pedagogical background. A strategy has to be easy to understand, straightforward to apply, useful on the long term and supported by literature. Tutors consider explicit strategies effective and useful. In particular, tutors preferred explicit tracing according to these criteria. Nevertheless, they believe a trade-off is to be found regarding more inquiry-based strategies so that students are left the opportunity to find out by themselves the best strategies for them to use.

We believe our results can be generalised to other introductory programming courses with a similar setup. In particular, since problem-based learning and tutoring are widely adopted in CS courses, our work could support them using more explicit programming strategies. Our research points in the direction of more actionable materials that could be given to tutors. A short training with the highlights of a strategy and an example of how and when to use it properly could help them a lot.

This exploratory study still leaves a lot of research questions unanswered. A first question is whether trained tutors will continue to use the strategies. Since they are students themselves, will they use them in their own curriculum? Another question would be to study the impact on students and their views on the strategies. Does it help them? (How) do they use and adapt them? Finally, a more quantitative study could be designed as a controlled experiment where part of the students are taught by tutors who received some training on explicit strategies and see whether this has an observable impact on their students' learning or grades.

REFERENCES

[1] Howard S. Barrows. 1996. Problem-Based Learning in Medicine and beyond: A Brief Overview. *New directions for teaching and learning* 1996, 68 (1996), 3–12.

[2] Danièle Bracke. 2004. *Un Modèle Fonctionnel Du Tranfert Pour l'éducation.* Presses Université Laval, 77–106.

[3] National Research Council. 2000. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition.* National Academies Press.

[4] John W. Creswell. 2012. *Analyzing and Interpreting Qualitative Data* (4th ed ed.). Pearson, Boston, Chapter 8, 236–246.

[5] Michael De Raadt, Mark Toleman, and Richard Watson. 2007. Incorporating Programming Strategies Explicitly into Curricula. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88.* Australian Computer Society, Inc., 41–52.

[6] Michael de Raadt, Richard Watson, and Mark Toleman. 2006. Chick Sexing and Novice Programmers: Explicit Instruction of Problem Solving Strategies. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.* Australian Computer Society, Inc., 55–62.

[7] Toby Dragon and Paul E. Dickson. 2016. Memory Diagrams: A Consistant Approach Across Concepts and Languages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16).* ACM, New York, NY, USA, 546–551. https://doi.org/10.1145/2839509.2844607

[8] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. 2017. Solving Parsons Problems Versus Fixing and Writing Code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17).* ACM, New York, NY, USA, 20–29. https://doi.org/10.1145/3141880.3141895

[9] Mariane Frenay, Benoît Galand, Elie Milgrom, and Benoît Raucent. 2007. Project- and Problem-Based Learning in the Engineering Curriculum at the University of Louvain. In *Management of Change.* Brill Sense, 93–108.

[10] Benoît Galand, Mariane Frenay, and Benoît Raucent. 2012. Effectiveness of Problem-Based Learning in Engineering Education: A Comparative Study on Three Levels of Knowledge Structure. *International Journal of Engineering Education* 28, 4 (2012), 939.

[11] Cindy E. Hmelo-Silver, Ravit Golan Duncan, and Clark A. Chinn. 2007. Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller, And. *Educational psychologist* 42, 2 (2007), 99–107.

[12] John R. Hollingsworth and Silvia E. Ybarra. 2017. *Explicit Direct Instruction (EDI): The Power of the Well-Crafted, Well-Taught Lesson.* Corwin Press.

[13] Jean-Claude Kaufmann. 2016. *L'entretien Compréhensif-4e Éd.* Armand Colin.

[14] Paul A. Kirschner. 2002. Cognitive Load Theory: Implications of Cognitive Load Theory on the Design of Learning. *Learning and Instruction* 12, 1 (Feb. 2002), 1–10. https://doi.org/10.1016/S0959-4752(01)00014-7

[15] Paul A. Kirschner, John Sweller, and Richard E. Clark. 2006. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist* 41, 2 (June 2006), 75–86. https://doi.org/10.1207/s15326985ep4102_1

[16] Amy J. Ko, Thomas D. LaToza, Stephen Hull, Ellen A. Ko, William Kwok, Jane Quichocho, Harshitha Akkaraju, and Rishin Pandit. 2019. Teaching Explicit Programming Strategies to Adolescents. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19).* ACM, New York, NY, USA, 469–475. https://doi.org/10.1145/3287324.3287371

[17] Thomas D. LaToza, Maryam Arab, Dastyni Loksa, and Amy J. Ko. 2020. Explicit Programming Strategies. *Empirical Software Engineering* 25, 4 (July 2020), 2416–2449. https://doi.org/10.1007/s10664-020-09810-1

[18] Raymond Lister. 2016. Toward a Developmental Epistemology of Computer Programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education.* ACM, 5–16.

[19] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, and Otto Seppälä. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 119–150.

[20] Dastyni Loksa and Amy J. Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* ACM, 83–91.

[21] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems.* ACM, 1449–1461.

[22] Jiali Luo, Laurie Bellows, and Marilyn Grady. 2000. Classroom Management Issues for Teaching Assistants. *Research in Higher Education* 41, 3 (2000), 353–383.

[23] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion).* Association for Computing Machinery, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[24] Lauren Margulieux, Richard Catrambone, and Mark Guzdial. 2013. Subgoal Labeled Worked Examples Improve K-12 Teacher Performance in Computer Programming Training. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 35.

[25] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. 2019. Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '19.* ACM Press, Aberdeen, Scotland Uk, 548–554. https://doi.org/10.1145/3304221.3319756

[26] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research.* ACM Press, 21–29. https://doi.org/10.1145/2787622.2787733

[27] Henk G. Schmidt, Sofie MM Loyens, Tamara Van Gog, and Fred Paas. 2007. Problem-Based Learning Is Compatible with Human Cognitive Architecture: Commentary on Kirschner, Sweller, And. *Educational psychologist* 42, 2 (2007), 91–97.

[28] Elliot Soloway. 1986. Learning to Program= Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (1986), 850–858.

[29] John Sweller, Paul Ayres, and Slava Kalyuga. 2011. *Cognitive Load Theory, Volume 1 of Explorations in the Learning Sciences, Instructional Systems and Performance Technologies.* Springer, New York.

[30] Jeroen J. G. van Merriënboer and John Sweller. 2005. Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review* 17, 2 (June 2005), 147–177. https://doi.org/10.1007/s10648-005-3951-0

[31] Benjamin Xie, Greg L. Nelson, and Amy J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* ACM, 344–349.