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

Olivier Goletti, Kim Mens, Felienne Hermans

Aug. 2020

Abstract

In programming education, explicit strategies have been gaining traction lately. We found that an introductory programming course at UCLouvain that is based on a problem-based methodology could benefit from more explicit strategies. After analysing a previous run of this course for first year undergraduate students, we concluded that such strategies could improve the transfer of learning across the different weeks of the semester. In the course, tutors with close to no pedagogical background tested four instructional strategies. These strategies are aimed at decreasing cognitive load while providing explicit step by step procedures for different aspects of the course. The four strategies used were: explicit tracing, subgoal labeled worked examples, Parson's problems and explicit problem solving. Our goal was to explore how tutors could benefit in their mentoring from explicit strategies. Interviews with the tutors show that the easiest and most efficient strategies were best used. More time should be devoted with the tutors to explain and model the more elaborate strategies or they can be misunderstood and misapplied. This paper proposes four selection criteria for selecting an explicit strategy: easy to understand, straightforward to apply, useful on the long term and demonstrably efficient.

1 Introduction

Introductory programming courses are more and more taught using problem based learning methodologies [22]. At UCLouvain university, the introductory bachelor computer science (CS1) course is given to the future civil engineers and future majors in CS. This course is taught following a methodology inspired by problem-based and project-based learning (PBL).

Students in this CS1 course are mentored by tutors. Those are more senior students who just need to follow a small pedagogical training based on the PBL strategy used in the course. Our research question throughout this study was to explore *how tutors could benefit in their mentoring from explicit programming strategies*. The definition for *explicit programming strategy* that we use is: a "human-executable procedure for accomplishing a programming task." [17]. We focused on the impact on tutors because we wanted to assess how comfortable they were with adopting new instructional strategies. Since PBL and tutoring are commonly used in CS1 courses, the results of our study could be generalised.

We assume in this paper that the strategies work, so we did not try to measure their effect on students. These are the research questions we are interested in:

RQ1 How did the tutors apply and adapt the strategies?

RQ2 What strategies did the tutors prefer and why?

RQ3 What do the tutors think of using explicit programming strategies?

Our approach consists of selecting strategies from the literature that have already been tested in a CS education context and shown to be efficient. For this paper, we limit ourselves to four strategies:

1. **Explicit tracing**: An explicit step-by-step tracing strategy has been tested by Xie et al. [31] in a similar university-level CS1 context and shown to improve students tracing abilities.

2. **Subgoal-labeled worked examples**: Margulieux et al. [23] identified and used subgoal labels. They used them with worked examples to improve students' reading and writing performance on early applications of concepts in a Java-based CS1 course.

3. **Parson's problems**: require students to reorder scrambled pieces of code and have been studied on undergraduate students from an introductory CS course. This strategy leads to the same amount of learning in "significantly less time than fixing code with errors or than writing the equivalent code" [8].

4. **Explicit problem solving**: This strategy has been tested on high school students attending a web development summer camp. Loksa et al. [20] found that "the intervention increased productivity, independence, programming self-efficacy, metacognitive awareness, and growth mindset".

Considering the exploratory nature of our research questions, we conduct this study by regularly following the tutors in focus groups and interviewing them at the end of the semester. In this paper, we propose a qualitative analysis of the interviews made with the participating tutors. The impacts of these findings are then discussed.

2 Background

This section presents the theoretical concepts we use to better understand how tutors benefit from explicit programming strategies. First we present cognitive load theory. We use it both to analyse the problem-based learning methodology of the course presented next, and to select our strategies. We then present what explicit programming strategies are.

2.1 Cognitive Load Theory

Cognitive load theory (CLT) [30] is an instructional theory based on human cognitive architecture. Humans have two types of memory, the short term mermory (STM) which can only hold a few amount of novel elements for a short time

and the long term memory (LTM) which stores acquired knowledge in efficient and structured ways called schemata. This last process is called learning. The knowledge stored in LTM does not impact the limited STM.

The main impact of CLT on instruction design is the principle that one should try to reduce cognitive load when teaching new material. The impact of too much context or unnecessary information when teaching increase the extraneous cognitive load on the learner. The complexity inherent to a piece of knowledge to be learned is called the intrinsic cognitive load.

CLT also postulates that general skills such as problem solving are "used in familiar contexts but not in complex unfamiliar areas, it follows that the major difficulty faced by learners is likely to be in assimilating novel, complex information rather than learning general cognitive strategies" [28].

We selected the strategies for this study because they followed the instructional idea of CLT: diminishing the extraneous load, minimizing intrinsic load and helping to apply generic skills. The impact of CLT on instructional design leads for example to the use of worked examples, automatizing rules or using external representations [14].

2.2 Learning Transfer

Learning transfer can be defined as reusing a previously learned concept in a new context. This is a cognitive process compared to learning [3]. It has a lot of impact on how instruction can be designed. Many models of learning transfer exist [29] [2]. These models attempt at naming and decomposing the different phases from the learning of a new concept in a source task to its reuse in target task.

It is important to note that leaning transfer is an active process. Students attempting to transfer benefit from prompts [3]. They can also better transfer if they are conscious that they are attempting to do so. This is typical in a problem-solving situation. The instructional implication is that strategies that invite a learner to recall previously learned knowledge and solutions to similar problems can facilitate learning transfer. We refer to these as *recall strategies*.

2.3 Problem-Based Learning

Problem-based learning (PBL) is a student-centered instructional methodology that is organized around real life problems where students work in groups of six to eight students. They work on problems and projects introducing new learning material. They need to answer questions and discover the theory often by reading and studying by themselves. They are supervised in their learning by tutors. The learning process is mostly self-directed [1].

However, PBL as an active teaching approach has sometimes been criticized, especially regarding the minimal guidance it provides that 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, 26]. This PBL methodology is used in the studied course and is described in more detail in Section 3.1.

2.4 Explicit Programming Strategies

In our study we explore four strategies. We selected these because they were conceived by their authors 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 chose them by exploring the literature on explicit strategies, that have been tested in a programming course setup and were promising. Explicit programming strategies can be either specific list of steps to reproduce, or meta-cognitive hints to help recall some specific technique.

The goal of such strategies is to optimize recall strategies to solve specific tasks and do this as easily as possible since we know it consumes the short span of STM. CLT is also quite present in CS education literature. We will detail in Section 3.2 how the selected strategies are indeed based on the same principle : lowering cognitive load for the learner.

Explicit programming strategies are gaining traction lately: De Raadt on teaching explicitly some recurring patterns in his curriculum [5] or Ko on using explicit strategies for tracing [31], debugging [17], code reuse [16] or problem solving [20]. Using worked examples or subgoal labeled material or Parson's problems are also techniques aimed at reducing cognitive load when learning and were studied by Morrison and Margulieux and their colleagues in numerous recent studies [24], [25], [8] and [23].

We mainly use specific strategies that have been adapted to CS teaching and not more generic organisational strategies such as those linked to Explicit Direct Instruction (EDI) [12]. The instructional advice given by EDI are based on some of the same assumptions as those behind the chosen strategies and are thus linked to the strategies used in this study.

In this paper, our goal is to explore how easily the selected strategies can be used by tutors that have close to no pedagogical background. We take their efficiency as granted.

3 Methodology

The goal of this paper is to explore how tutors can benefit from using explicit programming strategies in their instructional practice. In this section, we present the CS1 course studied in this paper, its setup and why we think it could benefit from more explicit strategies. We then present the strategies we selected and our motivations.

3.1 The CS1 Course at UCLouvain

A methodology inspired by problem-based and project-based learning (PBL) was adopted during a reform that impacted the first two years of the undergraduate civil engineering curriculum at UCLouvain university. The goal of this change throughout this curriculum was "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 introductory programming course follows those lines. Its setup is described in the next subsection. This course is followed by all first year undergraduate students majoring in computer science and civil engineering. One of

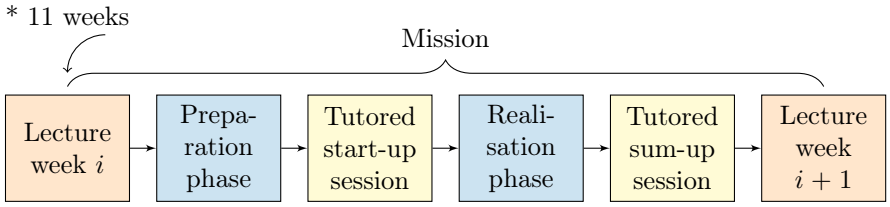

Figure 1: A one-week mission in the CS1 course at UCLouvain. Lectures in orange, individual work in blue, tutored sessions in yellow.

the three main pillars of the problem-based methodology used in this course is tutoring.

As discussed earlier, problem-based learning has been criticized by proponents of the cognitive load theory. However, Galand et al. have studied a previous run of the course we study in this paper and it has been shown to be efficient [10]. Nevertheless, our analysis of the CS1 course leads us to investigate whether this course could nonetheless benefit from more explicit strategies. The output of our analysis is discussed after its structure.

Course Structure

The CS1 course is organised around small weekly projects called *missions* and one such week is planned as illustrated in Figure 1. Each mission covers some programming concepts and has a theme, for example: the "strings and lists" mission has DNA sequences as a theme. The realisation phase for that mission ask to develop help methods for calculating a Hamming distance between sequences.

Each week i starts with a one hour lecture, typically on Friday. It is the opportunity to structure the concepts of the previous week and to introduce new ones. A preparation phase during the week-end is meant for individual student work. They have to read the theory syllabus and answer MCQs and open questions that are provided in an exercise syllabus. Each week a new fictional context is introduced and the work of that week will be situated in this context.

The first tutored session typically takes place on a Tuesday. Students work in their usual group of 6 students and one tutor is assigned to a room of four groups. During this one hour session, the prepared answers to the exercises are shared and discussed. The tutor helps organizing the discussions, sending students to the blackboard. He is not there to just give answers but to clarify theory concepts when needed and to ensure a correct solution is found by the groups.

The next phase starts at the end of this tutored session when the tutor introduces the students to the details of that week's mission. Each group is split in pairs who have to solve the small project on a computer. They have two days to submit their solution.

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, discusses common mistakes with the room as well as the pitfalls the students have ran into. A summary exercise covering the concepts of the week is then solved by

the room. The cycle then starts over again. During the closing lecture, the core concepts are revisited in more detail and important points are highlighted.

Course Analysis

The first author conducted an analysis based on a thorough inspection of the course material. It yielded three main proposals that could favour learning transfer:

P1 **The organization of the learning objectives should be made more apparent throughout the entire course.** It should be revisited more explicitly during lessons and lab sessions so that it is clearer for the students where they are in the course structure and to make the overall connection between the different topics more clear. 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**, not when they occur but when the knowledge that could be transferred is learned. Students will more easily retain knowledge they know will prove useful later. And this is more true if they can identify such situations in which they can use it. P2 is very oriented towards future problem-solving and is kind of complementary with P3. It aims at preparing the learner to recognize future situations in which he will have to recall previous knowledge.

P3 **More explicit recall strategies should be proposed in the course.** The whole idea of transfer is based on the identification of already acquired knowledge and reusing it in a different situation in order to solve new problems and building new knowledge. Recall strategies are there to help retrieve such known bits of knowledge and how to apply them to solve those new similar situations.

P2 and P3 are both in accordance with what CLT prescribes about instructional design. It promotes explicit strategies in the sense that we must automate content related knowledge and doing so, familiarizing the learner to new content and allow them to transfer knowledge.

3.2 Four Evidenced-based Strategies

The analysis of the course and the three propositions that were made lead us to chose explicit programming strategies and propose them to our tutors. The four selected explicit programming strategies are described in the section, in order of their use during the semester. We justify each time how they relate to our three propositions: P1 on making learning objectives visible, P2 on pointing out possible transfer opportunities and P3 on using explicit recall strategies.

In this paper, we use these four strategies:

1. Explicit Tracing as proposed by Xie et al. [31] and supplemented with the memory representation for more complex structures from Dragon et al. [7]

2. Subgoal labeled worked examples as proposed by Margulieux et al. [23]

3. Parson's problems as proposed by Ericson et al. [8]

4. Explicit problem-solving derived from the strategy proposed by Loksa et al. [20]

3.2.1 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]. This seems to be partly due at least to a lack of proper strategy and the cognitive load of remembering all the values while at the same time struggling with the newly learned semantics of code constructs [19].

The explicit tracing strategy we use is inspired by Xie et al. [31], who show a strategy that helps students to 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 the given program. The student executes each instruction by hand, one after the other following the logic of the program. The objective of the strategy is to help reduce the cognitive load of keeping track of all changing values.

Xie et. al took inspiration from Dragon et al. [7]. We also included the graphical representations for tracing arrays, lists and objects from the latter.

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

3.2.2 Subgoal Labeled Worked Examples

"Worked examples demonstrate how to apply an otherwise abstract procedure to a concrete problem" [23]. Subgoal labels highlight the steps of a generic problem solving procedure. Subgoal labeled worked examples (SLWEs) allow students to abstract from the context of a specific example and emphasize the more generic and functional role of a part of the resolution.

The idea of explicitly teaching the steps of recurring patterns in the resolution of many more or less similar problems is not new. However, those patterns are seldom explicitly taught in classroom. If these patterns were taught they could help a student who could recognise them to translate a problem statement into code.

Already in the 80's, researchers were exploring the idea of what goal a student was trying to achieve while solving a programming exercise and which plan could serve them do so [27]. Recently, the idea of teaching explicitly such plans to teach students in an introductory programming course was tested in a study by de Raadt et al. [6].

This is the goal of SLWEs. It was proposed by Margulieux et al. [23] who combine the idea of worked examples with this idea of labeling the steps behind different code structures. Subgoal labels identify those generic steps in order to emphasize them. They reveal explicitly the structure behind the example. The steps suggested by the labels can be reused for reading or writing similar code. The labels corresponding to the steps of one type of problem form a kind of plan that can be learned by a student and reapplied later on.

Subgoals for evaluating and writing loops.	
A. Evaluate loops	B. Write loops
1. Identify loop parts a. Determine start condition b. Determine update condition c. Determine termination condition d. Determine body that is repeated 2. Trace the loop a. For every iteration of loop, write down values	1. Determine purpose of loop a. Pick a loop structure (while, for, do_while) 2. Define and initialize variables 3. Determine termination condition a. Invert termination condition to continuation condition 4. Write loop body a. Update loop control variable to reach termination

Figure 2: Subgoal labels for reading or writing a loop construct in Java [23].

In their study, they propose a set of subgoal labels for reading and writing common programming constructs like a conditional, a loop, etc. An example is given in Figure 2. When using the strategy, tutors had to reuse the identified labels of those constructs adapted to Python. In class when confronted with an exercise using the construct, they highlighted on the blackboard the code matching the labels and wrote down the label.

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

3.2.3 Parson's Problems

Parson's problems consist in mixing lines of a solved 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 also more engaging for students [8].

The difficulty can be adjusted in two ways: by leaving out the indentation, this kind of problems are called "2D Parson's problems", or 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.

Parson's problems are rooted in the idea that learning is favored by mixing examples with practice [8]. This paper advances that two-dimensional Parson's problems with paired distractors 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 also heavily inspired by CLT and diminishing cognitive load. It is a way to practice the recognition of patterns shown in worked examples, and is therefore linked to our proposition P3.

3.2.4 Explicit Problem Solving

The fourth strategy concerns metacognition in problem solving. By explicitly giving students a list of steps to follow and associated 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. [20] in which the authors identified:

1. Reinterpret problem statement
2. Search for analogous problems

3. Search for solutions
4. Evaluate a potential solution
5. Implement a solution
6. Evaluate implemented solution

Explicit problem solving consists of explaining the 6 steps to the students, giving students 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 a 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 like suggested by CLT. It is also done in an explicit way like in our proposition P3.

3.3 Study Setup

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

A survey was distributed to the 25 tutors of our CS1 course with questions on how they saw their role as tutors for the course. 12 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 meetings, for about twenty minutes, the four tutors and the author would discuss the previously seen strategies. We discussed feedback, adaptation, best practices, interrogations, etc.

We introduced the strategies to the tutors gradually. Since they have close to no pedagogical experience, it was done to not overwhelm the tutors with too much information at once. We also thought it would give them more time to fully focus on each strategy one at a time. Every two other weeks, a new strategy was proposed (~20min). When a new strategy was proposed, the paper it was taken from was shown to the tutors and a brief explanation of the strategy was given according to this scheme:

1. Presentation of the motivations and objectives of the strategy;
2. Explanation of how to apply the strategy;
3. A usage example was given;
4. Questions and clarifications.

The tutors were encouraged to use these strategies with their students and to modify or adapt them if they wish. Regular feedback through the weekly meetings allowed us to know what they 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].

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.

At the end of the semester, each of the tutors was also interviewed for about one hour in a semi-structured way. A list of questions had been prepared by the first author to guide the interview and in order to not miss any specific point following the method proposed by Kaufmann [13]. In order to answer RQ1 on their use and adaptation of the strategies, we asked more detailed questions on 1) what they recalled of each of the four strategies, 2) how and 3) when they applied it and its pros and cons. To answer RQ2 on their preference, we asked them to compare them and rate them according to some criteria like the easier to apply, or the most efficient one. 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 guide for these semi-structures interviews is available in Appendix A.

During the thirteen weeks of the course, the tutors had the opportunity to test the four strategies and were interviewed. The transcripts of the interviews were coded following the method proposed by Creswell [4]. We used both codes emerging from the interviews, as well as codes induced by the themes of the different parts of the interview like explained in the previous section. The first interview was coded by two researchers and then the first author coded the three other interviews. In total, we coded 431 quotes coming mainly from the interviews, but also 10 from the recording of the weekly meetings. The codes were then regrouped in categories. We use them 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 the three research questions based on the coding we did of the interview material.

Since the interviews were conducted in French, the quotes provided below are translations provided by the first author of this paper.

4.1 Tutor views on the Four Strategies

In this section we will analyse what the tutors reported on their use of these strategies one by one by order in which they were introduced to the tutors. The final coded categories that were used were:

- Pros for tutors: what tutors saw as a benefit of using the strategy;
- Pros for students: how tutors saw the strategy was helping the students;
- Limitations: what tutors saw as obstacles for the students when using a strategy;
- Application difficulties: what tutors saw as obstacles for them to apply the strategy correctly;
- Suggestions: adaptations or comments made by tutors on improving the strategy.

4.1.1 Explicit Tracing

This strategy was the favorite of all four tutors. For the tutors, it is easy to understand and simple to apply. It is useful for all the students, simple to use, it helps code comprehension, testing and debugging code. Tutors have seen this strategy as reassuring for the students. One tutor said that the strategy would help them later in their curriculum. Tutors also said it reduces effectively the mental charge for the students. Tutors found that the strategy was unfit for longer executions. They even made some suggestions about to improve the use of this strategy.

Pros for tutors Among the four strategies, tutors said explicit tracing was their favorite. It was compared with a debugger and felt familiar to the tutors. One tutor was even shocked that it wasn't explicitly taught to the students:

> "Tracing code for example, I thought it was already taught in [this course]... I think it is very useful to trace at the beginning." (T4)

This strategy was also the easiest to apply with the best results. For example, a tutor 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." (T1)

Pros for students Tracing was mostly used to illustrate with an example for a student the origin of a bug.

> "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." (T3)

This last quote shows that it helped the most to identify where the students misunderstood a statement. Or when it was indeed a prediction error based on too high cognitive load.

> "The idea is to write it instead of keeping it all in their head" (T2)

> "For the students it's much easier because retaining 3 a 4 variables in their head, at their level, it's impossible. It's extremely difficult. At the slightest distraction, they forget a variable update and it doesn't work." (T1)

Limitations The four tutors were unanimous to say that tracing took too much time. Especially for longer codes.

Suggestions Tutors also made suggestions to improve explicit tracing. For example, one proposed to use several students "chained" to execute separate parts of the code or nested function calls. Another adaptation was to only trace chosen variable of interest in codes with objects where all the instance variables are not always necessary. For arrays and objects, they asked for and recommended after testing it more graphical representations like proposed by Dragon et al. [7].

4.1.2 Subgoal Labeled Worked Examples

Tutors said in the interviews that subgoal labeled worked examples (SLWEs) were more complicated to apply than the previous strategy. Tutors 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 Although tutors saw what that identifying the subgoals for the students was helpful, they said it was difficult to articulate the difference between in context steps and generic steps. They could not relate to a similar strategy emphasizing the generality of the steps to write a loop. It was more implicit for them so the strategy seemed "overly theoretical" (T3).

Notably because they had to stick to the provided subgoals labels that were provided (adapted to Python when needed). If they did not, the fell into a live attempt to explain as an expert the steps to follow in order to read or write a specific construct. And this is the prime reason why the strategy was designed, because it is difficult to do such an exercise. In the paper of the strategy [23], a long process was indeed setup in order to "extract" and identify the proper subgoals underlying the different programming constructs. This is illustrated in the next quote.

> "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 ok when I do a loop, step 1 that's it, then that ... " (T1)

Limitations Because SLWEs needed preparation and memorizing, it demanded more time for the tutors. This was a major hindrance in their use of the strategy. And this difficulty to remember the correct labels was due to lack of preparation for their first attempt at applying the strategy and also the fact that subgoals were provided for many different constructs and different when reading and writing.

Another 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.

> "They get used to it and expect the solution. Use with moderation !" (T2)

And we can also note that the tutors had a tendency to adapt the strategy because of lack of time or of preparation. They would do it orally and not as explicitly as proposed in the paper.

Pros for students 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.

> "It's good to do this for the first time that 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. ... " (T2)

4.1.3 Parson's Problems

Using Parson's problems was the second favorite strategy of the tutors. Tutors liked this strategy because it was engaging for students of all levels. Nevertheless, tutors said it required time to prepare and it was sometimes not easy to know when to use. Another main advantage was that students could see more examples in less time.

Pros for tutors In order to properly invent a new Parson's exercise, tutors found it was needed to think about common mistakes students would make. But found this exercise fun.

> "[A tutor needs to] know how to trap these students. To make good distractors, you have to know the corner cases that make a program not work." (T2)

One tutor also said that it allowed him to put more or less difficulty in an exercise.

Pros for students The importance of good paired distractors was stressed since it forced students to justify their choice. One tutor tried with unpaired distractors but found that this was too hard for the students or students would try to fit all the lines of code in one solution.

> "Line by line without indentation with distractors. To stimulate discussions in certain structures. For example to see if an else is mandatory or not." (T3)

Parson's problems allowed tutors to also see the benefits for the students of seeing more examples of solved codes in a short time, like reported in the original study [8].

> "That way they don't loose too much time writing stuff ... 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 but ..." (T1)

Not having to write all exercises by themselves was motivating for the students (labs are mostly done on paper). It mitigated the blank page syndrome. They also reported that students were more involved, more active and enjoyed the strategy even those who had more difficulties.

Limitations All tutors agreed that Parson's problems, when done on paper, required time and preparation.

Appropriation difficulties The main difficulty for the tutors was to identify exercises that would benefit from Parson's problems. Two tutors said they did not know when to use it. They said they would have preferred to experience it by themselves before, to have had more practice.

Suggestions Some suggestions were also made for Parson's problems. A tutor 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.

> "[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." (T3)

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

4.1.4 Explicit Problem Solving

Explicit problem solving was a bit controversial. For two tutors it was rated as difficult to apply and to understand but the other used it with success. It was straightforward to use for the students but difficult to understand because it was meta and at the same time so obvious.

Pros for students As with the previous strategies, tutors noted that this strategy helped student who did not know where to begin.

> "some students need a course of action or they don't know what to do." (T3)

The strategy still helped the students in the end. 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. We can see the meta-cognitive impact in the comments of the tutors:

> "They can see that it works when I ask them questions but cannot ask the question themselves." (T2)

> "It's a bit like trying to work as if you are working in a group but alone" (T1)

They found this strategy complementary to the second strategy as tutors saw that one as about translating a natural language resolution to code and this one was about the whole process, starting with reinterpreting the statement in natural language.

This recall strategy especially match a need identified in our analysis and was seen as such by the tutors:

> "If you remember something that worked, it's very easy to put it 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 for 2-3 minutes. You find that it is the case." (T1)

Limitations Some other 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 close definition.

Strategy appropriation is difficult Even though they saw the need to explicit the steps of problem solving, they found it sometimes too obvious and students too. Sometimes, tutors reported that they were not sure if it would help. They had to use it to be convinced of the usefulness of a strategy before using it.

> "I didn't test it for lack of time and because I didn't really see the point." (T4)

4.2 Answer RQ1: Use and adaptation of the strategies

The overall impression is that the tutors found the strategies useful and efficient. They use a strategy more easily if they can understand the motivation behind it. They will reuse a strategy if it helps the students, if it doesn't take too much time and if it impacts the motivation of the students. The tutors did not hesitate to test the strategies. They adapted it to their practice or to what they understood. Still, they reported sometimes not being sure on when to use a strategy and a need for more practice before using it with the students. They needed to see how and when to apply them like in this quote:

> "It's just that I had a really hard time judging when it is time to do it. And when you realize it in the middle of the session it's a bit of a problem, you don't know how to do it during the session." (T1)

One tutor even said that for some 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." (T4)

The tutors really need to understand the goal of the strategy to be convinced it has value and to use it properly. Otherwise, they had a tendency to focus on the context and not on the general principle the strategies highlighted. Especially with a more abstract strategy like the subgoal labeled worked examples, they needed to understand them well. Otherwise, the expert blind spot (ie. showing explicitly the steps an expert follow but cannot articulate) which is kind of the main cause behind the usage of the strategy was working against them.

A difficulty encountered was that lack of time, preparation and poor class management was a brake on instructional changes. It is already known that TA's and by extensions also tutors suffers from class management issues [21]. It is well possible that in this case, since they have close to no pedagogical

Easiest to understand for the tutor	1 3 4 2
Easiest to apply for the tutor	1 2-4 3
Most effective for the students	1 3 4 2

Table 1: Comparison of the strategies. 1) Explicit tracing; 2) Subgoal labeled worked examples; 3) Parson's problems; 4) Explicit problem solving

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 play with instructional experiments.

The next section on RQ2 will summarize the most important criteria of a strategy for the tutors.

4.3 Answer RQ2: Preferences in Strategies

When comparing the strategies, tutors found that the easiest to understand, to apply and the most effective one was explicit tracing. Using Parson's problems was considered easy and efficient even though it took more time to prepare. The third most efficient was the explicit problem solving strategy and it was also third easiest to understand. Finally, subgoal labeled worked examples was the most difficult to understand and the least effective according to tutors. We asked them to order the strategies and the corresponding results are in Table 1.

Maybe was the first strategy preferred because it was the most straightforward to apply and because it was also very close to the actual day to day practice of the tutors who are also CS students themselves. It seems to be a strategy that the tutors found by themselves or because it was mentioned in their curriculum but without being never explicitly taught. It is also, like they mentioned, very close to what a debugger do for them. So familiarity was also a big factor

From these comparisons and the quotes and categories in Section 4.1 we can say that. The best strategies were not necessarily the easiest to apply for example the third one about Parsons' problems was difficult to apply according to them. Tutors 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. So this was a major motivation for the tutors to try the strategies. In the end, we can see that the tutors preferred easy to understand, easy to apply and efficient strategies. It also has to be somehow reusable for the students in their later programming life.

To summarize we can say that for a strategy to be adopted by a tutor, it has to be easy to understand, straightforward to apply, useful on the long term and demonstrably efficient.

4.4 RQ3: On Explicit Programming Strategies

The tutors found the explicit nature of the strategies efficient and that "it is less demanding for the student" (T1):

> "But I find that the explicit requires a lot less motivation ... less struggling ... to understand things when it is explicit. [On the] tracing, we clearly know how to do it, we don't need to do a lot ... many failures to find ... a method that works. So I think that it can help." (T1)

Regarding the balance with less explicitness, they said that students still need occasions to explore strategies by themselves and figure out what works best for them. That explicit strategies should be shown especially when introducing new materials and that the students should then have some less guided time to try them out and adopt or adapt them.

> "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 him." (T2)

> "The two are complementary. You have to explain to them how to do it at least once." (T4)

> "It is one method among others that we give them and that they will try to adapt for themselves." (T3)

There was also a fear of two tutors that the students would not search by themselves anymore because of explicit strategies.

> "The disadvantage is that [because of the explicit strategies] 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 ... it's something that they like, with which works well for them... Maybe there are other methods that work well and they may never find out on their own" (T1)

The tutors also seem to agree that more open exercises are also beneficial for students and that less explicit method would force students to learn how to "figure it out". They clearly associate implicit with letting the students work it out by themselves.

> "Less explicit, it is rather when they have more freedom, when we give them the statement in the broad sense of the term and we do not give a course of action." (T2)

> "It is good from time to time to leave the students a little more on their own." (T2)

> "We say to ourselves, that they will not get there because they are having trouble and we have more experience. Teaching them to have a hard time can also have good sides." (T2)

To summarize, they view explicit strategies as an efficient way to teach and to automate good practice for students. But they also 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.

5 Discussion

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

First, tutors reported that they liked very much to discuss during the weekly meeting about the different strategies which allowed them to see how it was used by other and encourage them to try them out. This setup was inspired by the idea of communities of practice. Having regular meetings allow learners to hear and learn from each other. The following quote illustrates the feeling of T1 on this topic:

> "Seeing each other every week, ... we realized for certain strategies that it was easier to use than we thought [speaking about Parson's problems], so that was nice." (T1)

Second, we want to come back on the comments made by all four tutors when speaking about the second strategy (subgoal labeled worked examples). Tutors mentioned that, however 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 previous step 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. "

.

Giving labels to the students was a step too far even though it helped guide them to a solution. The tutors mentioned that the fourth strategy on explicit problem solving helped the students with this first step. So maybe, just introducing this strategy first would suffice. However, extra attention need to be put on giving tools and strategies to tutors to help them explain to students how to express or translate in natural language a problem statement.

Threats to Validity

This paper suffers from some threats to validity. The first limit we can see to this study is the small amount of tutors that participated. But the goal here was mainly to explore tutors use of explicit programming strategies and to propose some leads for follow-up research. It would indeed be interesting to take the proposed criteria for a strategy and test them on more strategies and more tutors.

The second point is in the way the strategies were selected. It might be that other strategies exist and could be tested with other results. As seen in the beginning of the Discussion section, we might want to find an explicit strategy helping students to translate a problem statement to natural language.

Finally, the 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.

6 Conclusion

In this exploratory study, we have proposed the use of four explicit strategies for tutors in a CS1 course at UCLouvain university. The four strategies have been chosen because they were shown to be efficient and explicit and in accordance with the propositions that were drawn by our analysis of the course and the instructional propositions made by cognitive load theory.

The four strategies were gradually presented and tested by four tutors in the course over a period of thirteen weeks during which regular meetings were held. At the end of the semester interviews were conducted, transcribed and analyzed to answer three research questions.

What we can see is that the tutors like to use new instructional strategies even though it is difficult for them to do so properly. They do not have neither the time to prepare there lab sessions including strategies nor the pedagogical experience to be confident enough to try more complicated strategies.

We propose four criteria for a strategy to be readily adopted by tutors without a pedagogical background. It has to be easy to understand, straightforward to apply, useful on the long term and demonstrably efficient. Following those criteria, the preferred strategy was the explicit tracing.

Tutors seem to consider explicit strategies efficient and useful. Nevertheless, they think a trade-off is to be found regarding more inquiry based strategies and that students still need to be left with the opportunity to find out by themselves the best strategies for them to use.

We think that our results are still somewhat generalizable to other introductory programming courses with the same kind of setup. Indeed, since problem-based methodology and tutoring are widely used in CS courses, we hope other teachers and researchers will try to include more explicit programming strategies in their course. Our research seems to point 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 to use it properly and when could help them a lot.

We are well conscious that this is only an exploratory study. It leaves a lot of unanswered research questions.

The first one that came to our mind would have been to ask the students about what they themselves think about the strategies. Does it help them ? Do they use them ? And if so during all the course or just at the beginning ? Do they adapt them ? And is there a link between their views of when to use or adapt a strategy and what the tutors expressed in this study ?

Another aspect would be to see whether trained tutors continue to use the strategies ? Since they are themselves students, will they use them in them own curriculum ?

Another aspect, more quantitative, could be to assess the efficiency of the process on students learning. We could design a controlled experiment where part of the students are taught by tutors who received some training on the strategies and see whether it has some impact on their learning or their grades.

Acknowledgements

Work partially conducted with the support of the Erasmus+ programme of the European Union.

Co-funded by the Erasmus+ Programme of the European Union

References

[1] Howard S. Barrows. "Problem-Based Learning in Medicine and beyond: A Brief Overview". In: *New directions for teaching and learning* 1996.68 (1996), pp. 3–12.

[2] Danièle Bracke. "Un Modèle Fonctionnel Du Tranfert Pour l'éducation". In: *Le Transfert Des Apprentissages: Comprendre Pour Mieux Intervenir*. Presses Université Laval, 2004, pp. 77–106.

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

[4] John W. Creswell. *Educational Research: Planning, Conducting, and Evaluating Quantitative*. Prentice Hall Upper Saddle River, NJ, 2002.

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

[6] Michael de Raadt, Richard Watson, and Mark Toleman. "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., 2006, pp. 55–62.

[7] Toby Dragon and Paul E. Dickson. "Memory Diagrams: A Consistant Approach Across Concepts and Languages". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. New York, NY, USA: ACM, 2016, pp. 546–551. ISBN: 978-1-4503-3685-7. DOI: 10.1145/2839509.2844607.

[8] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. "Solving Parsons Problems Versus Fixing and Writing Code". In: *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. Koli Calling '17. New York, NY, USA: ACM, 2017, pp. 20–29. ISBN: 978-1-4503-5301-4. DOI: 10.1145/3141880.3141895.

[9] Mariane Frenay et al. "Project-and Problem-Based Learning in the Engineering Curriculum at the University of Louvain". In: *Management of Change*. Brill Sense, 2007, pp. 93–108.

[10] Benoît Galand, Mariane Frenay, and Benoît Raucent. "Effectiveness of Problem-Based Learning in Engineering Education: A Comparative Study on Three Levels of Knowledge Structure". In: *International Journal of Engineering Education* 28.4 (2012), p. 939.

[11] Cindy E. Hmelo-Silver, Ravit Golan Duncan, and Clark A. Chinn. "Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller, And". In: *Educational psychologist* 42.2 (2007), pp. 99–107.

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

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

[14] Paul A. Kirschner. "Cognitive Load Theory: Implications of Cognitive Load Theory on the Design of Learning". In: *Learning and Instruction* 12.1 (Feb. 2002), pp. 1–10. ISSN: 0959-4752. DOI: 10.1016/S0959-4752(01)00014-7.

[15] Paul A. Kirschner, John Sweller, and Richard E. Clark. "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching". In: *Educational Psychologist* 41.2 (June 2006), pp. 75–86. ISSN: 0046-1520, 1532-6985. DOI: 10.1207/s15326985ep4102_1. URL: http://www.tandfonline.com/doi/abs/10.1207/s15326985ep4102_1 (visited on 09/10/2018).

[16] Amy J. Ko et al. "Teaching Explicit Programming Strategies to Adolescents". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. New York, NY, USA: ACM, 2019, pp. 469–475. ISBN: 978-1-4503-5890-3. DOI: 10.1145/3287324.3287371.

[17] Thomas D. LaToza et al. "Explicit Programming Strategies". en. In: *Empirical Software Engineering* 25.4 (July 2020), pp. 2416–2449. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-020-09810-1.

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

[19] Raymond Lister et al. "A Multi-National Study of Reading and Tracing Skills in Novice Programmers". In: *ACM SIGCSE Bulletin*. Vol. 36. ACM, 2004, pp. 119–150.

[20] Dastyni Loksa et al. "Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 1449–1461.

[21] Jiali Luo, Laurie Bellows, and Marilyn Grady. "Classroom Management Issues for Teaching Assistants". In: *Research in Higher Education* 41.3 (2000), pp. 353–383.

[22] Andrew Luxton-Reilly et al. “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. New York, NY, USA: Association for Computing Machinery, July 2018, pp. 55–106. ISBN: 978-1-4503-6223-8. DOI: 10.1145/3293881.3295779.

[23] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. “Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1”. en. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '19*. Aberdeen, Scotland Uk: ACM Press, 2019, pp. 548–554. ISBN: 978-1-4503-6895-7. DOI: 10.1145/3304221.3319756.

[24] Lauren E Margulieux, Richard Catrambone, and Mark Guzdial. “Subgoal Labeled Worked Examples Improve K-12 Teacher Performance in Computer Programming Training”. en. In: (2013), p. 7.

[25] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. “Subgoals, Context, and Worked Examples in Learning Computing Problem Solving”. en. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM Press, 2015, pp. 21–29. ISBN: 978-1-4503-3630-7. DOI: 10.1145/2787622.2787733.

[26] Henk G. Schmidt et al. “Problem-Based Learning Is Compatible with Human Cognitive Architecture: Commentary on Kirschner, Sweller, And”. In: *Educational psychologist* 42.2 (2007), pp. 91–97.

[27] Elliot Soloway. “Learning to Program= Learning to Construct Mechanisms and Explanations”. In: *Communications of the ACM* 29.9 (1986), pp. 850–858.

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

[29] Jacques Tardif. *Le Transfert Des Apprentissages*. Editions logiques, 1999.

[30] Jeroen J. G. van Merriënboer and John Sweller. “Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions”. en. In: *Educational Psychology Review* 17.2 (June 2005), pp. 147–177. ISSN: 1573-336X. DOI: 10.1007/s10648-005-3951-0.

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

A Interview Guide

Feeling on being a tutor

- How would you describe your experiment as a tutor this semester ?
- How would you define 'being an efficient tutor' ?

  Has your vision changed on what it is to be an efficient tutor ?

  How efficient do you think you are ?

  Has it changed during this semester ?

Context

- Could you describe the level of your students ?

  The atmosphere in your classroom ?

  Has it evolved during the semester ?

  Thanks to what ?

  What about your own feeling as a tutor ?

Explicit v. PBL

- Could you give an example of a strategy that you use/know which is more explicit ?
- Could you give an example of a strategy that you use/know which is more discovery based ?
- How would you say they are different ?

For each strategy

(those questions will be asked for the four strategies):

- Can you redefine/reexplain the strategy in your words?
- What did you think about this strategy ?
- Had you heard of it before ? Can you compare it to a strategy you did use before ?
- Have you read the paper for this strategy ?
- Were the motivations and objectives clear for this strategy ?

  What would you say is the minimal knowledge required to understand this strategy ? (regarding CS, regarding
- Was it easy to apply during lab session ?

  Have you felt obliged to apply/try it ?

  How did you feel about testing this specific strategy ?

  Was it adapted to this course ?

- Could you give an example of use for this specific strategy ?

  Do you think it was well applied ?

  What was the effect of this strategy on your students ?

  Do you think it was beneficial for the students ?

  in which way ? (engagement, ease of use, efficiency, CLT, class management?)

  for which students ? (high achieving vs difficulties ?)

  Do you think it was beneficial for you ?

- What do you think about the claims of the strategy ? Was the effect as described in the paper/in our discussion ?

Strategies comparison

- Could you order the 4 strategies (and comment on this):

  from the easiest to the hardest to understand

  from the easiest to the hardest to apply

  from the least to the most efficient

  from the one applying to all students to the one only applying to a subset of students

  from the one that should come earlier to the later in the semester

- If you had to share your preferred strategy with another tutor which one would it be ?

- Will you continue to use them ? Or other ones ?

Closing

- Do you have comments on the overall process of this experiment ?

- What has this experiment brought to you ?