

An Analysis of Tutors' Adoption of Explicit Instructional Strategies in an 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 this paper we analyse in detail how tutors of an undergraduate-level introductory programming course use two explicit instructional strategies in practice with their students. The two strategies they used were an explicit tracing strategy and a subgoal learning strategy. We explored what triggered their use of these strategies, how faithfully they followed the proposed strategies, and how they adapted it in practice to their classroom setting. We rely on literature on fidelity of implementation to assess tutors' adoption of these strategies. The tracing strategy was much more used and with higher fidelity than the subgoal learning strategy. Tutors adopted both strategies with adaptations, simplifications and even combining them. From our observations we draw good and bad practices on the adoption of such explicit instructional strategies for future generations of tutors.

CCS CONCEPTS

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

KEYWORDS

explicit strategy; tracing; subgoal learning; fidelity of implementation; adaptation; observation; CS1

ACM Reference Format:

Olivier Goletti, Kim Mens, and Felienne Hermans. 2022. An Analysis of Tutors' Adoption of Explicit Instructional Strategies in an Introductory Programming Course. In *Koli Calling '22: 22nd Koli Calling International Conference on Computing Education Research (Koli 2022)*, November 17–20, 2022, Koli, Finland. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3564721.3565951>

1 INTRODUCTION

A lot of introductory programming courses at university use a setup with undergraduate and graduate teaching assistants (TAs) [9, 12]. In this study, we focus on undergraduate tutors and will refer to them as “tutors” in the remainder of this text. Previous research showed that tutors struggle to identify and use best practices [35]. Explicit instructional strategies are gaining traction lately and the

effort to bring research results into tutors' practice is worth exploring as advocated by Sue Sentance in her Koli Calling 2021 keynote [36].

In this study we looked at literature for effective instructional strategies that could lower cognitive load and favor learning transfer when used by tutors [15]. Many research-based explicit strategies have already been used in programming education [22, 23, 26, 29, 46], among which we selected two: Explicit Tracing and Subgoal Learning. Following criteria from earlier research on tutors' use of explicit strategies [18], we ensured through training material, training sessions and personalised follow-up during the course semester that both strategies were easy to use and straightforward to apply.

Based on research on educational change, *fidelity of implementation* and *adaptation* of research-based instructional strategies [1, 32], we used a mixed methodology to analyse recordings of tutor interventions during their lab sessions. Our aim was to understand how tutors adopt and adapt explicit instructional strategies in their classroom setting.

Based on previous work [18], we already have a good grasp of what tutors' views on using such explicit strategies are. The idea of our current study is to get a better insight on what tutors actually *do* in the classroom. The research questions that guide this work are:

- RQ1** What triggers the use of a specific instructional strategy by tutors?
- RQ2** What is the fidelity of implementation of the instructional strategies by tutors?
- RQ3** What adaptations of the strategies are made by tutors during instruction?

2 BACKGROUND AND RELATED WORK

The goal of this paper is to study the use of two explicit strategies by tutors. The selection of the two explicit strategies used in this study is based on cognitive load theory. This theory is presented as well as its increasingly important role in computing education research. We then present the educational change principle of fidelity of implementation we use to analyse the usage of the two strategies by tutors. Finally, we present the two strategies of explicit tracing and subgoal learning.

2.1 Cognitive Load Theory

Cognitive load theory (CLT) [38] is an instructional design theory based on human cognitive architecture. It acknowledges that our working memory is limited and can thus be overloaded by too many information elements when learning. Part of the cognitive load is intrinsic to the complexity of the learning concept,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Koli 2022, November 17–20, 2022, Koli, Finland

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9616-5/22/11...\$15.00

<https://doi.org/10.1145/3564721.3565951>

but some load can be attributed to instructional design or learning strategies. Too much context, too much or unnecessary information when teaching increases the cognitive load on the learner and hampers learning. The impact of CLT on instructional design is that a teacher should try to reduce load on the working memory. Many effects to take into account during instruction have been documented [33, 40, 44] related to minimizing cognitive load and helping to apply generic skills [39]. This can be achieved through the use of instructional strategies such as for example: the use of worked examples [43], automating rules, faded guidance [40] or using external representations [20]. This is one of the reasons why in our study we selected the two explicit instructional strategies of Explicit Tracing and Subgoal Learning which both help to decrease cognitive load. Both strategies are explained in more detail in subsection 2.3.

2.2 Fidelity of Implementation

When trying to change instructional practice through professional development, a teacher will ideally use the strategy and integrate it in their own practice. Research looks at their appropriation through two constructs: the *adaptation* that teachers will inevitably make and the *fidelity of implementation* of that strategy defined as the degree to which procedures are implemented as planned [32]. “*Adaptation and fidelity of implementation are different constructs and should be separately measured and related to outcomes*” [32].

Research on teachers’ adaptations and implementation of instructional strategies distinguishes *efficacy studies* from *effectiveness studies* [32]. Efficacy studies are lab studies focusing on the development stage and aiming to show that an innovation performed by experts in perfect conditions is indeed linked with better outcomes. Effectiveness studies look at natural classroom setups where adaptation is inevitable and where we try to understand when and how a practitioner adapts the innovation. Our current study is an effectiveness study where we observed how close to the prescribed strategies the tutors’ practice were and what adaptations they made to the strategies.

It is recommended to compare adaptations and fidelity to the desired outcomes, to be able to discriminate whether a failure to achieve these outcomes is due to an ineffective instructional strategy or too much divergence from a prescribed strategy. However, since our main goal is to describe and understand *how* and *when* tutors use and adapt the strategies, we didn’t look at the outcomes yet. There is a dire need in literature to more accurately document instructional strategies adaptation and fidelity of implementation, as confirmed by Borego et al. [1] who state that “*it is important to have complete and accurate descriptions and measures of how and when an instructional strategy is being implemented*”.

2.3 Explicit Instructional Strategies in Computer Science Education

Cognitive load theory informs instructional design and states that instructional strategies that explicitly teach problem solving steps are beneficial to novice learners [21, 40]. In computer science education in particular, explicit instructional strategies are often mentioned in recent literature: e.g., by explicitly teaching recurring patterns [6], or using explicit strategies for tracing [46], debugging [23],

code reuse [22] or problem solving [26]. The use of worked examples and subgoal-labeled material as techniques aiming to reduce cognitive load when learning has also been explored in numerous recent studies [18, 27, 29, 30]. As stated by Xie et al.: “*The implications for teaching are simple: help students practice an explicit strategy*” [46]. In particular, previous works show how both selected strategies are aligned with CLT instructional recommendations and promote learning transfer [15, 18].

2.3.1 Explicit Tracing. Tracing code is the process of executing a program in one’s head or by hand. Research shows that students who trace code better can explain code better [24, 25]. However, novice students learning to program often struggle with tracing. This struggle can be attributed to cognitive overload. Indeed, their limited working memory cannot handle tracking the value updates of the different variables of a program simultaneously, which leads to tracing errors [42]. It has been proposed to teach explicitly to students a proper tracing strategy using an external representation of the memory to help reduce their cognitive load [5, 42, 46].

In particular, Xie et al. [46] proposed a strategy they introduced to students before tracing six problems. The strategy helped students trace programs while updating a memory table. This strategy proposes step-by-step instructions to systematically trace the program. Each variable encountered in the code has a line in the memory table and their values are updated when executing line by line the instructions of a given program. Research shows that combining the explicit teaching of a tracing strategy combining both the use of an external memory representation and a systematic line-by-line tracing procedure may improve tracing performance [19, 46].

2.3.2 Subgoal Learning. Among the effects predicted by cognitive load theory, the worked example effect shows that exposing students to detailed solutions to problems improves their learning [45]. It has also been shown in programming that worked examples help students learn [11, 30].

While a worked example demonstrates an abstract solving procedure via the use of a concrete example [29], students still sometimes focus on the details or the context of the proposed examples. Labelling the steps or subgoals of the solving procedure draws the learner’s attention to the genericity of the solution steps [28, 30]. Subgoal learning, the idea of explicitly teaching the steps of recurring patterns in the resolution of many more or less similar problems is not new [7, 37]. The steps suggested by the labels can be reused when solving similar problems and favour learning transfer. Margulieux et al. [29] combined the idea of worked examples with labeling the steps behind common code structures, proposing subgoal labels worked examples (SLWEs). In the end, the subgoals are taught to the students by interleaving SLWEs and practice exercises [29] progressively diminishing the guidance as prescribed by CLT [21].

The SLWEs studied by Margulieux et al. suggests that “*subgoal materials helped learners to solve problems using the procedure more effectively during the early stages of learning even though no performance difference between groups was found on the exams*”.

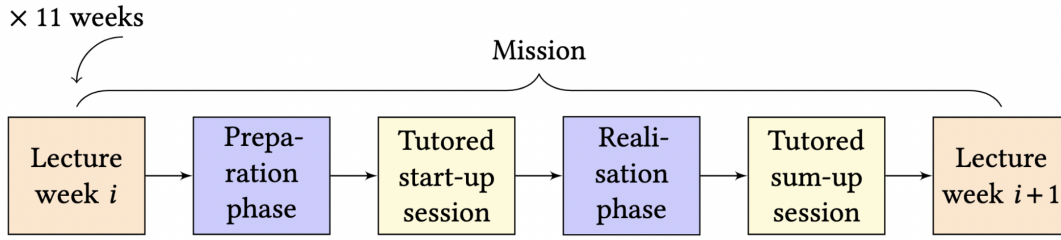


Figure 1: A week in the CS1 course at our university. Lectures in orange, individual work in blue, tutored labs in yellow.

3 STUDY DESIGN AND CONTEXT

3.1 CS1 course in the study

The introductory programming course studied is taught at a Belgian research-level university, in the French-speaking region of the country. The course lasts for thirteen weeks. It is mandatory for first year computer science and engineering undergraduate (bachelor) students with no assumed prior programming experience. This course typically targets more than 600 students and follows a methodology inspired by Problem-Based Learning [14]. In 2021, twenty eight senior students were recruited as tutors for this course.

As illustrated in Figure 1, the 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 and students have to develop help methods for calculating a Hamming distance between sequences. The course uses Python, covers imperative programming and introduces programming with objects¹.

Each week i starts with a one-hour lecture by the professor. Students then have to read the theory in the course notes and answer multiple choice and open questions (definitions, programming exercises, etc.), provided as assignments before the first tutored session. 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, and the cycle starts all over again with a new mission focusing on a new concept and a new theme. This weekly schedule is repeated over the 11 missions of the course.

3.2 Training Material

Prior experiences with training tutors with instructional strategies advised to put more effort in training them [18]. Moreover, it was observed that adoption is facilitated when tutors understand the underlying hypotheses and claims of instructional strategies. In that study, tutors also asked for more guidance on how and

when to apply the strategies. In response to that request, for this study two documents were prepared for training the tutors. Each of these documents explain the promises of the strategies, provide instructions on how to apply them as well as quotes from previous adopters and exercises to verify proper understanding of the strategies [16, 17]. In previous research [18], four strategies were progressively introduced to tutors throughout the semester of the course. In the current experiment, we decided to present our strategies at the start of the semester instead and limit ourselves to two strategies, in order to reach more depth. The two strategies were selected among the four strategies explored previously as follows. Explicit tracing was chosen because it seemed to have been the most effective and preferred strategy according to the tutors. And even though it seemed less effective to tutors, subgoal learning was also selected because we believe that with better training it could also be effective as supported by the literature presented in section 2.3.2. Tutors in our former study explicitly mentioned lack of training and lack of time as main obstacle for proper strategy use. We addressed that shortcoming in this paper by providing more training and guidance through dedicated training, dedicated training documents and follow-up during the semester. Also, both selected strategies have the advantage of being easily observable since they require tutors to make use of the blackboard.

3.2.1 Explicit Tracing. The document presenting the explicit tracing strategy is written in French and available online [17]. Figure 2 is an example of memory representation of a concept as given in the document. Annotations have been added to highlight the important parts and we translated the example to English for the sake of readability.

Our document introduces the explicit tracing strategy, gives step by step instructions on how to apply it as presented in §2.3.1, sums up its advantages and claims, gives advice on when to use it and introduces and exemplifies each memory representation of the different programming constructs introduced in the course.

The explicit tracing strategy we use is heavily inspired by Xie et al. [46] who introduced the strategy. The main steps are reused and explained in the handout document we prepared. However, since using an external representation when tracing code can take many forms [5], we decided to use a normalised memory representation based on the work of Dragon and Dickson [8, 10]. The examples and exercises proposed in our document are adapted to the Python programming language and specifically limited to the programming concepts seen in the CS1 course.

¹<https://syllabus-interactif.info.ucl.ac.be/index/info1-theory>

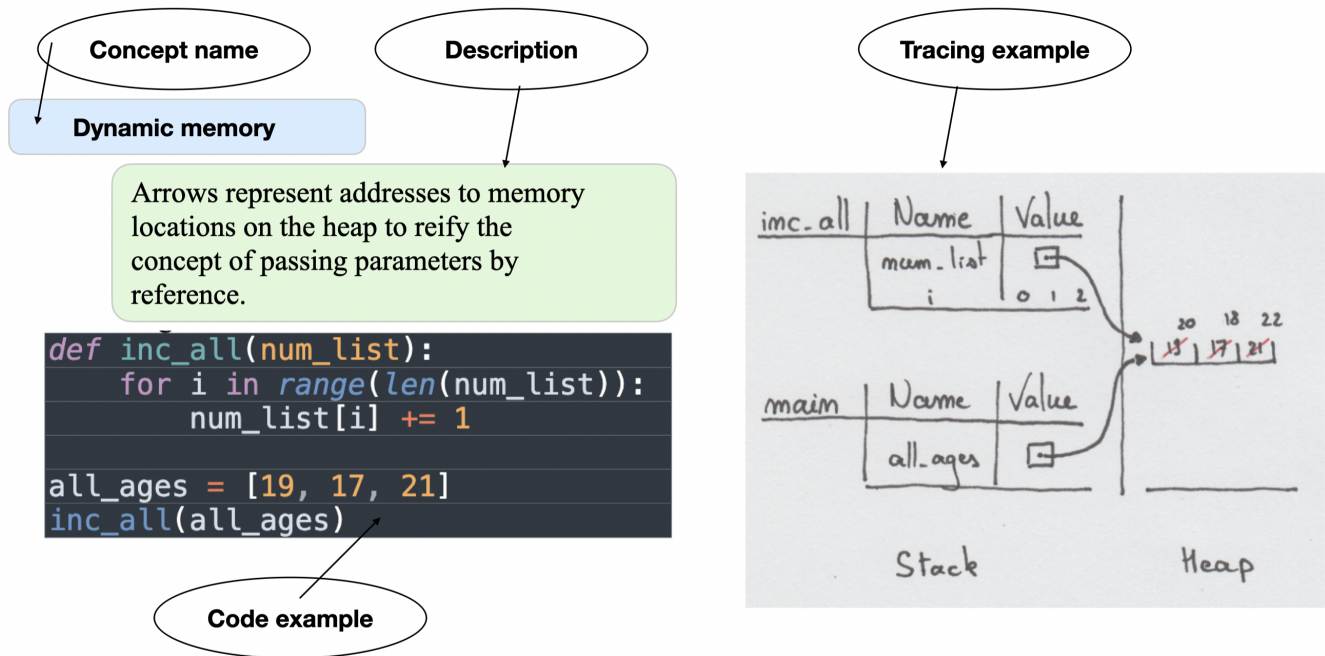


Figure 2: External representation of a list (inspired by Dickson et al. [8]) as handed out to our tutors. Translation, ellipses, arrows and highlighted parts have been provided for readability.

3.2.2 Subgoal learning. The document presenting the subgoal labeled worked examples is written in French and also available online [16]. Figure 3 is an example of an SLWE as given in the document. Annotations were added to highlight the important parts and we translated part of the example to English for the sake of readability.

The strategy is for the tutors to use the SLWEs in their teaching by presenting the SLWEs to the student to illustrate how to solve problems involving new programming concepts. Then tutors need to reuse the labels of the subgoals to annotate code when solving similar problems and go through the labels to help construct such a solution.

The worked examples presented in the training document are extracted from the CS1 course to illustrate the different concepts. They are annotated in color with the corresponding subgoal labels. The subgoal labels for the four first concepts (assignment, conditionals, loops and functions) are adaptations to the Python programming language of those proposed by Margulieux et al. in their article [29]. The others (list traversal, file reading and writing, dictionary creation and update, class creation, node insertion and removal in a linked list) have been created by the authors. To create these SLWEs we used a methodology similar to the Task Analysis by Problem Solving (TAPS) methodology proposed by Catrambone [3] and also used and recommended by Margulieux et al. [29]. This methodology is a kind of Task Analysis [4], an effective method to extract the steps and procedures that domain experts implicitly use when solving complex problems. For more details on the procedure we used to extract these subgoal labels, see Goletti et al. [16].

3.3 Participants

Tutors are senior students with a good knowledge of CS concepts, having passed their CS1 course with a grade above 70%. They are either third year undergraduate (i.e. bachelor) or first or second year graduate (i.e. master) students who facilitate CS1 students' work in small classes during their lab sessions. Tutors need to follow a mandatory training course in group management worth credits. After that, they are paid whenever they tutor a course. Tutors are accompanied throughout the semester in a weekly one-hour meeting with the course staff to discuss classroom management issues and to get briefed on the next week's objectives. The practice of using senior students as tutors is typical of the problem-based learning inspired methodology [13] used in the studied course. Tutors are also used in CS1 courses with similar setups at other universities [9].

Before the start of the semester, our research project was exposed to all tutors. Tutors were offered the possibility to be trained and to test the selected explicit strategies. There was a small financial incentive to participate in the study to compensate for the hours invested in preparation, training, focus group meetings and interviews. Among the twenty eight CS1 tutors, seven tutors volunteered and participated in the study. Five were tutors for the first time, the two others had prior experience tutoring courses and only one of them had experience tutoring this CS1 course and was tutoring it for the third time.

In the scope of this study, a training session took place at the start of the semester. The seven tutors were introduced in detail to the two strategies, the theoretical bases on which they are built and

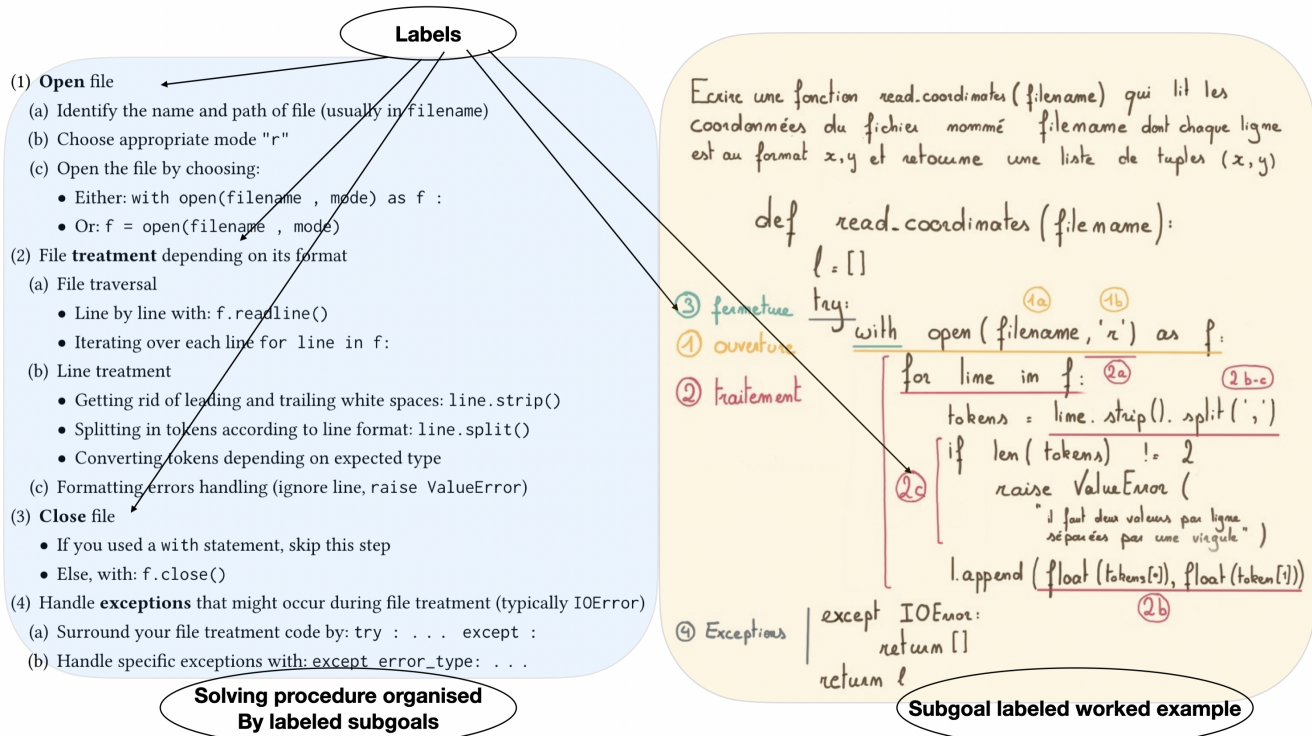


Figure 3: SLWE for file writing in Python. The presentation on the left is inspired by Margulieux et al. [29]. Subgoals' translation, ellipses, arrows and highlighted parts have been provided for readability.

their main objectives. The two aforementioned documents were handed out to the tutors. The strategies were modeled by the first author and then practised by the seven tutors to ensure proper understanding. The training session lasted two hours.

During the course semester, the seven participating tutors met weekly with the first author for one hour. The goal of those focus groups was to discuss, share and troubleshoot the use of the strategies. Each tutor would typically share how he or she used the strategies during the past week. During the first weeks of the semester, some class observations were made by the first author to give feedback on how the strategies were used. At the end of the semester, tutors were also interviewed in a semi-directed way to collect their impressions over the course and their practice of using the strategies. The training session, focus groups and interviews were recorded.

3.4 Fidelity criteria

In order to assess the fidelity of implementation of the strategies by the tutors, criteria have to be developed. Mobray et al. [31] propose to develop fidelity criteria by identifying "critical components ... based on an expert consensus process or the existence of a proven model which has been explicitly described" [31]. They suggest to use a model with proven efficacy, effectiveness or acceptance, and expert's opinions (surveys of experts or literature reviews). To develop our fidelity criteria (see Table 1), we used that method. Similarly to Borrego et al.'s study on research-based instructional strategies'

fidelity of implementation [1], which also used this method, we based ourselves on the literature describing each of the strategies and on our own expertise in using and applying the strategies with tutors [18].

3.4.1 Explicit Tracing. Two critical aspects come from the description of the explicit tracing strategy in Xie et al. [46] who affirm the strategy is effective for ameliorating tracing performance. We also find these aspects in other similar work on tracing [19]. They ground their strategy in two critical aspects:

- (1) source code needs to be traced **line-by-line** "like a computer does" to enforce a systematic tracing strategy for students and avoid errors. This aspect is supported by the importance of tracing [19, 24] and not letting students invent their own strategy [34];
- (2) an **external representation of memory** needs to be used to reduce the working memory load of remembering the values of variables. This aspect is grounded in CLT [40] and in the efficiency of sketching in computer science [5, 10].

These two aspects combine three elements: **line-by-line** tracing, using an **external representation**, but also the mapping process between the execution of a line and the **update of the state** of the program represented in the memory table. We translated these in the three explicit tracing fidelity criteria **ET1**, **ET2** and **ET3** listed in Table 1.

Table 1: Fidelity criteria for the considered explicit instruction strategies, derived from literature.

Strategy	Fidelity criteria
1. Explicit Tracing	ET1. The source code is traced line-by-line; ET2. An external representation of the memory is used; ET3. The values of the traced variables are explicitly updated.
2. Subgoal Learning	SLWE1. An SLWE from the document is explicitly introduced to the students; SLWE2. The tutor solves a similar problem in a worked example; SLWE3. The tutor reminds the students about the subgoal labels of a SLWE; SLWE4. The solving procedure of a SLWE is used to solve a practice exercise with the students.

Since the decision was made from the beginning that we wanted to focus on adaptations and not to enforce only exact use of the strategy, we decided to keep the external representation criterion unconstrained to a specific representation such as the one presented in Dragon et al. [10]. The strategy’s efficiency seems not to be based on that representation and the validation of the strategy for more advanced memory representation such as objects and data structures had not been established by Xie et al. [46].

3.4.2 Subgoal learning. Looking at the different studies using subgoal learning through the use of worked examples [28–30] the main aspect of this strategy is to interleave subgoal labeled worked examples (SLWEs) with practical exercises. This element is grounded in the fading guidance effect [40] and the interleaving of theory and exercises [41].

As explained in §2.3.2, SLWEs are built on the efficacy of showing worked examples. Therefore, the labels are first shown in a **given SLWE** provided in the document. During instruction a **worked example** needs to be solved by the tutor and the steps of the solution procedure need to be shown and addressed. Subgoal learning through the use of SLWEs adds to this the **use of labels** to highlight the generic structure of the solution procedure. Finally the last critical component is to reuse the solving procedure in a similar problem with the students in a **practice exercise**. We translated these elements in the four SLWE fidelity criteria **SLWE1**, **SLWE2**, **SLWE3** and **SLWE4** listed in Table 1.

3.5 Methodology

We used a mixed methodology in this study. We observed the tutors during their lab sessions and recorded those sessions for further analysis. During the semester, the first author observed tutors’ lab practices during three different weeks: week 5 on tuples and binary search, week 8 on classes and objects, and week 11 on linked lists. Due to schedule overlap, it was not physically possible to observe all 42 tutors’ interventions (7 tutors with 2 sessions per week on 3 different weeks), hence only 24 observations were made. During each observation, the first author sat in the classroom on the side and did not intervene. He took notes of tutor’s usage of the strategies and recorded the lab session for further analysis. In total, more than 26 hours of videos were analysed. In the rest of our analysis, we refer to tutors by the notations T1 to T7.

During the analysis, we coded segments of all observed sessions using both deductive and inductive coding. Deductive codes come from the strategies’ fidelity criteria introduced in Section 3.4. They

were used for determining when a strategy was used. The coded segments corresponding to fidelity criteria occurrences were counted and used to answer the second research question on the fidelity of implementation of the strategies by tutors (**RQ2**) in a quantitative manner. Those segments were also part of the answer to the third research question on adaptation of the strategies by tutors (**RQ3**). For this third research question, a more inductive coding emerged from the recordings and we categorised those codes using a thematic analysis [2]. For each strategy use, we also coded and categorised the trigger event in order to answer the first research question on what triggers strategy usage by tutors (**RQ1**). In order to code the tutor’s usage of the strategies, we took note of the timestamp, the context, the trigger, the fidelity criteria observed and the adaptations when appropriate. We also transcribed quotes of tutors speaking about the strategy and kept a screenshot of visible usage or adaptation of the strategies. This allowed us to easily come back to previous coded segments when the codes and categories had saturated. The coding of the different video segments was mainly done by the first author with the help of the second author to confirm that deductive coding was done in a consistent manner and that the inductive coding covered all observed triggers and adaptations. Two full lab sessions were coded by both the first and second authors and conflicts were discussed and resolved. Once all the material had been coded, themes emerged for the first and third research questions. Codes were regrouped in those themes through discussions between the authors.

4 ANALYSIS

The goal of this study was to understand how tutors apply explicit instructional strategies in their practice. Based on the analysis of the recordings of the tutors’ lab sessions, we answer our three research questions.

4.1 Triggers for Strategy Usage

Since we wanted to understand what triggers strategy usage (**RQ1**), during our analysis of the recorded observations we documented for each strategy use the event that triggered the use of the strategy. For this part of the analysis, the coding was induced by the observations and the categories emerged from the data.

4.1.1 Explicit Tracing. Among the different events that triggered the use of explicit tracing, we identified three sources depending on the origin of the strategy use:

Table 2: Measures of fidelity of implementation for tutors' usage of Explicit Tracing

Tutors	Uses	Number of fidelity criteria observed			Which fidelity criteria observed		
		3	2	1	ET1	ET2	ET3
7	73	31 (42%)	20 (27%)	22 (30%)	32 (44%)	72 (99%)	51 (70%)

Students Students are the principal triggers (total: 33 times).

Tutors would use the strategy when a student would ask a question (16 times), when students would be confused about the expected output of a program (7 times), when they observed a bug in a student's answer on the blackboard (5 times), when they would see a lack of understanding of an important concept (4 times) or when a conflict would arise between two students' predictions on the output of a program (1 time).

Tutors Tutors' decisions are the second most occurring trigger (total: 27 times). A tutor would take the initiative to trace a program because they prepared an example for a topic they would consider difficult or to emphasize an important concept (16 times). Tutors also asked students to trace code to verify their proper understanding (6 times). Tutors would also be triggered to use explicit tracing when they wanted to show that two alternative solutions were equivalent or not (4 times). On one occasion, a tutor traced a code snippet because he made an error of prediction himself and decided to take advantage of the situation to model the strategy.

Exercises Exercises themselves are the third trigger. Some exercises explicitly required the students to make sketches or to draw the state of the memory (13 times).

4.1.2 Subgoal learning. For subgoal learning, we only observed triggers from two of the three sources mentioned above.

Tutors Tutors would mainly decide to read an SLWE or use it to solve a similar problem, for introducing new concepts, or to summarize a concept at the end of a week (16 times). One tutor would also use the labels as a check list to verify that a proposed solution contains all the needed steps of the solution (1 time).

Students Students were a second encountered trigger. Tutors would use the labels or revisit an SLWE after having observed multiple instances of misunderstanding (3 times) or similar questions by different students on a corresponding concept (1 time).

4.2 Fidelity of Tutors' Strategy Implementation

During our analysis of the video recordings of our observations, systematic counting of the fidelity criteria was done to understand the fidelity of implementation of the strategies. While our goal was not to force tutors to respect the strategies to the letter, we wanted a measure of how often they did apply the strategy as presented. This data will help us answer the second research question on the fidelity of implementation of the strategies by tutors (RQ2). Since the criteria were deduced from research literature on the strategies, the codes for this part of the analysis were a priori codes. This part of our analysis was thus of more quantitative nature.

We used the following metrics. We counted how many times each strategy was used by each tutor per session. Then we calculate the percentage of number of criteria used for these strategy uses. We counted a strategy as used as soon as at least one of the criteria for the corresponding strategy was observed in the instructional practice of the tutor. One might argue that if not all criteria are present, the strategy is not properly used but that will be the whole point of looking at the adaptations made by tutors in Section 4.3 (RQ3).

4.2.1 Explicit Tracing. Our measures for the usage of the explicit tracing strategy are summarised in Table 2.

From the 24 observed lab sessions, we can see that overall tutors used explicit tracing 73 times. The duration of a tracing intervention ranged from 1 minute to 16 minutes with a mean of 3 minutes 27 seconds. Slightly less than half of the observations (42%) can be considered as expected strategy uses: the tutor would trace a program line by line, typically with the help of the students, and update variable values in a memory table. In nearly all observed strategy uses (99%), an external representation of memory was used (ET2). In fact, it was not observed in one case only. In that specific case, T4 traced the program line by line (ET1) but orally and without external representation. We did observe other instances of oral tracing, but those were quickly followed by the use of a memory table and were counted as using one. The different usage variations of this strategy are discussed in Section 4.3.1. When only an external representation was used (i.e., ET2 without ET1 nor ET3), it was to illustrate a specific state of the memory at a specific moment of the execution. It was often used to illustrate the outcome of an execution or the starting state before the execution. Often, such sketches would serve as a basis for more explanations by the tutors but without actual tracing or value updates. This has been observed 21 times. In all 51 occurrences of value updates (ET3), an external representation was used (ET2), but sometimes without a systematic line by line tracing (ET1). This occurred typically when a tutor would draw a sketch of the execution and update variables while making shortcuts in his or her explanations or when they would explain the steps of the program but without referring to the code.

4.2.2 Subgoal learning. Our measures for the usage of the SLWE strategy are summarised in Table 3.

The first observation from the data is that the subgoal learning strategy has been used much less than the explicit tracing strategy. Most of the 21 observed uses were pretty shallow with nearly half matching only one criteria (11 times) which often was only mentioning the labels (SLWE3: 7 times) or providing a detailed worked example solution (SLWE2: 3 times) on the board for the students or discussing/reminding the students of specific labels not to forget before solving an exercise. In 6 occurrences, mentioning

Table 3: Scores of fidelity of implementation for tutors usage of the SLWE strategy

Tutors	Uses	Number of fidelity criteria observed				Fidelity criteria observed			
		4	3	2	1	SLWE1	SLWE2	SLWE3	SLWE4
7	21	3 (12%)	1 (4%)	10 (40%)	11 (44%)	6 (24%)	8 (32%)	20 (80%)	12 (48%)

labels (SLWE3) was done while solving a practice example with the students (SLWE4). These cases represent the majority of occurrences of two criteria for the strategy. This is a typical use of the strategy that we expected when a tutor would solve with the students a similar problem as one illustrated in the training document using a similar solving procedure. In two occurrences though, the labels were discussed by the tutor (SLWE3) and a worked example to illustrate them (SLWE2) was provided. In three occurrences, we observed the four criteria, which means that the tutor would use the document (SLWE1) with the students, provide an equivalent worked example (SLWE2) or go through the worked example of the document while discussing the labels (SLWE3) of the corresponding solving steps and then apply the solving procedure to solve a similar exercise (SLWE4).

4.3 Tutors' Adaptation of Strategies

Finally, we also performed a qualitative analysis to understand and observe which adaptations were made by the tutors to the strategies they used and how far different tutors' instruction diverged from the proposed instructional strategies (RQ3). For that, we analysed expected and partial uses of the strategies. Figure 4 illustrates expected uses and adaptations of both strategies. The coding of the segments was done with the fidelity criteria and also directly induced by the observations. Our acceptance of what constituted an adaptation was broad and is detailed in the following. We present the observed adaptations of strategies by tutors and categorize them in three kind of adaptations: additions, effects and misuses.

Additions. These additions are small adaptations made by the tutors to the proposed strategy. We hypothesize those adaptations will mainly have a neutral or small positive effect on the outcomes of the strategy.

Effects. These effects are observed changes to tutors' instructional practice. They are not modifications to the strategy itself but observations of recurring instructional change in tutors' practice.

Misuses. These are adaptations made by tutors that we hypothesize might lessen the outcomes of the strategy.

4.3.1 Explicit Tracing.

Additions.

Inverse tracing In this interesting adaptation of the explicit tracing strategy, we observed multiple cases of inverse usage of an external representation. By this we mean, from a misunderstanding of a statement, a tutor would advise to use or use himself an external representation of the expected output or even a sequence of dynamic manipulation on a drawing and this would then serve as a basis to write the solution program to the statement. It is interesting to notice

that in several instances, this practice was quite dynamic in the sense that the external representation would be updated while discussing with the students the steps needed to achieve the expected output. This addition was observed 10 times.

Progressive tracing Another typical use of explicit tracing was a progression in the details added to the sketch. A tutor would typically explain the solution orally, then do a first drawing with very few details, sometimes with generic elements. And when students would still require explanations, they would fall back to a full line by line systematic use of the strategy and update the drawing systematically. This addition was observed 9 times.

Tracing of program variations Using an external representation was sometimes triggered by a difficulty identified by the tutor, through a misunderstanding or in response to a question from a student. In such cases, after tracing the program in question, another adaptation we observed was that tutors would ask students to evaluate expressions based on slight variations of the code and check whether they understood it properly with the help of an external representation. This addition was observed 8 times.

Coloring updates Tutors would sometimes use different colors to note updates in the traces to make them more visible. This addition was observed 6 times.

Effects.

Representation-based explanation An interesting positive effect observed nearly every time an external representation was used, is that this representation served as a basis for further explanation. Tutors would often gesture towards the memory representation, replaying with their hand the evolution that brought the memory in its actual state. This effect was observed 47 times. An example is given in Figure 4a.

Collaborative tracing Instead of just verbalising aloud their thought processes while tracing, tutors often involved their students in the evaluation of the expressions and updating the memory table. Tutors would often ask the class questions such as: "Do I enter this while loop?" (T5), "What is the new value of this variable? What do I write here?" (T4) while pointing in the table towards a value to update. This effect was observed 29 times.

Encouraging discussions In multiple instances, tracing code stimulated relevant questions from the students, explanations from the tutors, etc. Students would ask about small variations and exchanges between students and the tutor would occur naturally. This effect was observed 9 times.

Code annotation Tutors would sometimes write down intermediate values directly in the code. This often happened



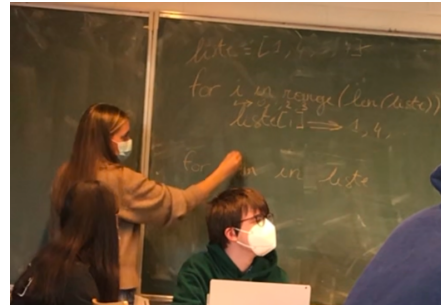
(a) Expected use of the Explicit Tracing with Representation-based explanation effect



(b) Oversimplified memory representation (not all variables are sketched)



(c) Expected use of the Subgoal Learning through (colored) code annotations



(d) Unlabeled subgoals during an explanation of alternative solutions for list traversals

Figure 4: Some examples of both strategies used by tutors. (a) and (c) are expected uses while (b) and (d) are observed adaptations. Permissions to publish these pictures have been granted by the tutors and students.

for compound expressions that did not appear directly in the table. This was sometimes used in conjunction with a memory table, but sometimes not. This effect was observed 6 times.

Program explanation While tracing, tutors would sometimes give more natural language explanations about the code. As they built the memory table on the board, copying variable names and initial values or when they executed specific lines of a program, tutors would naturally explain the role and reasons for those variable names or the executed statements. Tutors seemed naturally driven to give more comments about the names and constructs used in traced code. This effect was observed 6 times.

Misuses.

Oversimplified memory representation Often, tutors would not use the proposed memory representations and would simplify a lot their drawings. Sometimes, this translated in incomplete drawings, oversimplified notations (e.g., object instances without attribute names), or very partial sketches. Those less detailed drawings sometimes evolved in more refined traces as with progressive tracing. This misuse has been observed 32 times. An example is given in Figure 4b.

Tracing aloud Tutors would sometimes do the tracing out loud, without the code available to the students or at least without them following. The tutor executes the code in his head and just give the different values aloud as if it were

evident. This often triggered a progressive tracing effect. This misuse was observed 6 times.

4.3.2 Subgoal learning.

Additions. No addition was observed per se, but an important variation is discussed in §4.3.3.

Effects.

Shared vocabulary A positive effect of the SLWE strategy is that since the explanations of the solution steps use a specific vocabulary, tutors would use themselves but also encourage students to use a more precise vocabulary when talking about their code, so that they could associate it more easily with the corresponding SLWE. A tutor, when distinguishing arguments from parameters for the function concept would say *“It is important to use the proper vocabulary”* (T5). This effect was difficult to count exactly but was observed in a lot of recordings.

Collaborative code building On a few occasions, a tutor used the strategy to solve a new problem with the students based on the labels (SLWE4). But they did it cooperatively, with a high degree of interaction and students helping to build a correct solution on the board. The tutors sometimes built correct code only by themselves too. This notable effect was observed 6 times.

Explanation of alternative solutions Since some subgoal labels correspond to steps only to follow as the result of an alternative, when discussing those with the students, tutors had to explain them and give explanations on the different alternatives. Using the strategy thus favoured discussion and understanding on those specific points. *“If we use a for loop with a range, the i variable will successively have the value from 0 to the end of the list. l[i] will have exactly the same value as the element of the list and this can also be achieved by iterating with the form for e in l. [...] Who can tell me when we should prefer one or the other form?”* (T5). This effect was observed 6 times. An example of this effect is shown in Figure 4d.

Misuses.

Unlabeled subgoals The labels provide a shared vocabulary. But in some observations, tutors would not use the labels, or only just a few. In those cases, tutors would discuss about subgoals linked to a concept but without mentioning the corresponding name of the label or without referring to the generic problem solving procedure. Some examples were tutors discussing list traversal with the students without mentioning the provided labels in the document, or a tutor discussing the importance of naming for classes and attributes without referring to the corresponding SLWE. This misuse was observed 12 times. An example is shown in Figure 4d.

Tutor-defined labels On a few occasions, we also observed tutors using other labels than those provided in the SLWE document. It was because they did their own attempt at labeling the important solving steps for a given concept or because they misunderstood the strategy as a “divide and

conquer” strategy and would develop subgoals for solving a problem, label them, and then annotate the solution code with those different subgoals. In those cases, subgoals and labels would not be complete or generic. This misuse was observed two times.

Oral annotation One tutor tried to apply the strategy by reading and mentioning the labels and subgoals while students were writing their solution on the board. This caused confusion since students in the classroom were trying to read the code first. Explicitly annotating the code as exemplified in the document was more effective.

4.3.3 Combined strategies.

Subgoal learning with external representation On four occasions observed in three different sessions given by three different tutors, tutors would reuse the labels of a specific concept and illustrate the different steps and cases covered by the labels with an external representation. On one of those occurrences, for writing the code solution for inserting a node in a linked list, the adaptation was combined with the *“Inverse tracing”* variation of the explicit tracing strategy and the code of the solution was built based on the sketches.

5 DISCUSSION

The goal of this study was to understand how and when tutors apply explicit instructional strategies. From our observations of tutors’ use of Explicit Tracing and Subgoal Learning, we propose some pieces of advice to practitioners and discuss threats to validity.

5.1 Advice to practitioners

From what we observed in tutors’ practice of the two selected strategies, it seems that most of the adaptations were small and still aligned with the main idea of the strategies. However tutors might also misuse instructional strategies. We advise teachers responsible for introductory programming courses to pursue the effort of looking for research-based strategies and making them available to their teaching assistants and student tutors so they can reach their classroom and benefit their students. Tutors will adopt the strategies if they are research-based, easy to use and straightforward to apply. For the sake of bringing effective research-based explicit instructional strategies into their classrooms, we recommend teachers responsible for tutors and teaching assistants to:

Train tutors with research-based strategies by providing them hands-out documents and training sessions on the strategies so they can prepare and gain time during lab sessions.

Raise awareness of the triggers that we listed and that will help tutors anticipate which exercises will need further explanation and when to use which strategy.

Allow for innovation by providing freedom to adapt the strategies. The idea of combining strategies was a nice example of such innovation. So was the inverse usage of external representation.

Provide targeted exercises that explicitly or implicitly incite students to use the strategies. This will favor strategy adoption by the tutors.

5.2 Threats to validity

We studied only a small number of participants in order to be able to conduct a thorough qualitative analysis of the observations. This allowed us to have a first glance of the possible adaptations made by tutors but does not allow us to generalise our findings yet.

While we proposed some advice to practitioners based on our experience and observations, we did not measure the outcomes of the different strategy uses and their variations. Further research measuring the outcomes of the different variations might be needed here.

6 CONCLUSION

In this study, we analysed how seven tutors adopted two explicit instructional strategies in a CS1 course. The two chosen strategies were selected because they are research-based, recognised as effective instructional strategies and designed to reduce the cognitive load of the learners. We based ourselves on the education change literature to analyse the fidelity of implementation and the adaptations that tutors made to the strategies. Based on that, we derived fidelity criteria for both strategies based on the corresponding literature. We used a mixed methodology to treat three research questions by coding tutors' session recordings: we did a qualitative analysis of the observations to understand what triggers strategy usage by the tutors; we quantified based on our criteria the fidelity of implementation of both strategies; we further qualitatively analysed the different adaptations made by the tutors to the strategies.

The first result we can draw from the analysis of our observations is that Explicit Tracing was more used than Subgoal Learning. We identified three main categories that could trigger either strategy. Events that made tutors use a strategy originated either from: the **students** (their questions, their mistakes and their misunderstandings); the **tutors** (their preparation, the importance of concepts and their modelling of solutions and alternatives); the **exercises** (when prompting to trace). Both strategies were adopted by tutors with adaptations, simplifications and even by combining them. The fidelity of implementation of the explicit tracing strategy was higher than for the subgoal learning strategy. Tutors used overall a lot of external representations with a variety of adaptations. In their main use, they progressively transitioned from an oral explanation to a detailed line-by-line tracing depending on students' needs and understanding. The subgoal learning strategy took more time to apply and was often applied partially. A significant number of adaptations for both strategies have been documented. These adaptation were categorized as **additions**, **effects** and **misuses**. We hypothesize most of these additions and side-effects will marginally favor the expected outcomes of the strategy. The main obstacle to the use of both strategies seems to have been a lack of time since we can see that a full use of the subgoal learning strategy could take up to half an hour. While many adaptations were made, some seemed to be an incomplete use of the strategies or a misunderstanding by the tutor of the strategy. Based on our analysis, we proposed some pieces

of advice to teachers responsible for introductory programming courses.

Future work

Regarding future work and questions raised by our analysis, three main research ideas surfaced. First, a more controlled experimental setup with an explicit comparison between tutors trained and not trained for strategy usage could provide insight into the prevalence of tutors' use of the strategies like Borrego et al. [1] did in their study. However this is a big experiment to put in place, with lots of observations to analyse and variables that could influence the experiment. Maybe trained observers could help in such an experimental setup with an observation grid to fill out immediately during class observation. Main variations of the strategies could also be tracked during such an experiment. An initial list of possible variations could be drafted based on the observations we made in the current paper. Second, specific post-tests designed to measure outcomes could be used to evaluate whether strategy usage and their variations indeed deliver the expected outcomes. In our case that would mean a tracing test for the explicit tracing strategy and isomorphic problem solving tests for the subgoal learning strategy. Finally, tutors' feedback again pointed to the time constraints of introducing students to the SLWEs. It is not uncommon for TAs to mention time constraints as a threat to best practice [35]. Integrating them in the course material and observing the impact on strategy use in classroom would be an interesting lead to explore. We hypothesize that in such a setup, subgoal learning would be used more by the tutors.

REFERENCES

- [1] Maura Borrego, Stephanie Cutler, Michael Prince, Charles Henderson, and Jeffrey E. Froyd. 2013. Fidelity of Implementation of Research-Based Instructional Strategies (RBIS) in Engineering Science Courses. *Journal of Engineering Education* 102, 3 (2013), 394–425. <https://doi.org/10.1002/jee.20020> _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/jee.20020>.
- [2] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (Aug. 2019), 589–597. <https://doi.org/10.1080/2159676X.2019.1628806> Publisher: Routledge _eprint: <https://doi.org/10.1080/2159676X.2019.1628806>.
- [3] Richard Catrambone. 2011. Task Analysis by Problem Solving (TAPS): Uncovering Expert Knowledge to Develop High-Quality Instructional Materials and Training. In *Learning and Technology Symposium, Columbus, GA*.
- [4] Richard Clark, David Feldon, Jeroen J. G. Van Merriënboer, Kenneth Yates, and Sean Early. 2008. Cognitive Task Analysis. In *Handbook of Research on Educational Communications and Technology*. Chapter 43, 577–593.
- [5] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using tracing and sketching to solve programming problems: replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on international computing education research*. 164–172.
- [6] 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.
- [7] 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.
- [8] Paul E Dickson and Toby Dragon. 2021. A Memory Diagram for All Seasons. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 150–156.
- [9] Paul E Dickson, Toby Dragon, and Adam Lee. 2017. Using undergraduate teaching assistants in small classes. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 165–170.
- [10] Toby Dragon and Paul E. Dickson. 2016. Memory Diagrams: A Consistent 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>
- [11] 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>
 - [12] Jeffrey Forbes, David J. Malan, Heather Pon-Barry, Stuart Reges, and Mehran Sahami. 2017. Scaling introductory courses using undergraduate teaching assistants. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on computer science education*. 657–658.
 - [13] 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.
 - [14] 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.
 - [15] Olivier Goletti. 2021. Promoting Learning Transfer in Computer Science Education by Training Teachers to Use Explicit Programming Strategies. In *ICER '17*. ACM, Virtual Event USA, 411–412. <https://doi.org/10.1145/3446871.3469776>
 - [16] Olivier Goletti, Florian De Pierpont, and Kim Mens. 2022. Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec Python. In *Didapros 9-Didastoc*.
 - [17] Olivier Goletti and Kim Mens. 2022. Atelier : Utiliser des stratégies d'instruction explicites dans l'enseignement de la programmation. <https://dial.uclouvain.be/pr/boreal/object/boreal:263637>
 - [18] Olivier Goletti, Kim Mens, and Felienne Hermans. 2021. Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. In *ITiCSE '21*. ACM Press, Virtual Event, Germany, 7. <https://doi.org/10.1145/3430665.3456348>
 - [19] Matthew Hertz and Maria Jump. 2013. Trace-based teaching in early programming courses. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 561–566.
 - [20] 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](https://doi.org/10.1016/S0959-4752(01)00014-7)
 - [21] 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/s15326985sep4102_1
 - [22] 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>
 - [23] 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>
 - [24] 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.
 - [25] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin* 41, 3 (2009), 161–165.
 - [26] 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.
 - [27] 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.
 - [28] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research*. 71–78.
 - [29] 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>
 - [30] 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>
 - [31] Carol T. Mowbray, Mark C. Holter, Gregory B. Teague, and Deborah Bybee. 2003. Fidelity Criteria: Development, Measurement, and Validation. *American Journal of Evaluation* 24, 3 (2003), 315–340.
 - [32] Carol L. O'Donnell. 2008. Defining, conceptualizing, and measuring fidelity of implementation and its relationship to outcomes in K–12 curriculum intervention research. *Review of educational research* 78, 1 (2008), 33–84. Publisher: Sage Publications.
 - [33] Fred Paas, Alexander Renkl, and John Sweller. 2003. Cognitive load theory and instructional design: Recent developments. *Educational psychologist* 38, 1 (2003), 1–4.
 - [34] David N. Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.
 - [35] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2021. Challenges Faced by Teaching Assistants in Computer Science Education Across Europe. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 547–553.
 - [36] Sue Sentance. 2021. Teaching computing in school: is K-12 research reaching classroom practice?. In *21st Koli Calling International Conference on Computing Education Research*. 1–2.
 - [37] Elliot Soloway. 1986. Learning to Program= Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (1986), 850–858.
 - [38] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.
 - [39] 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.
 - [40] John Sweller, Jeroen J. G. van Merriënboer, and Fred Paas. 2019. Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review* (2019), 1–32. Publisher: Springer.
 - [41] J. Gregory Trafton and Brian J. Reiser. 1993. Studying examples and solving problems: Contributions to skill acquisition. In *Proceedings of the 15th conference of the Cognitive Science Society*. Citeseer, 1017–1022.
 - [42] Vesa Vainio and Jorma Sajaniemi. 2007. Factors in novice programmers' poor tracing skills. *ACM SIGCSE Bulletin* 39, 3 (2007), 236–240.
 - [43] Tamara van Gog, Fred Paas, and John Sweller. 2010. Cognitive Load Theory: Advances in Research on Worked Examples, Animations, and Cognitive Load Measurement. *Educational Psychology Review* 22, 4 (Dec. 2010), 375–378. <https://doi.org/10.1007/s10648-010-9145-4>
 - [44] 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>
 - [45] Mark Ward and John Sweller. 1990. Structuring effective worked examples. *Cognition and instruction* 7, 1 (1990), 1–39.
 - [46] 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.