Practicing Abstraction Skills Through Diagrammatic Reasoning Over CAFÉ 2.0

Géraldine Brieven, Lev Malcev, Benoit Donnet
Université de Liège, Institut Montefiore, Belgium

***Abstract*—Shaping first-year students' minds to solve problems at different levels of abstraction is both important and challenging. Although abstraction is a crucial skill in problem-solving, especially in STEM subjects, students often struggle with abstract thinking. They tend to focus their efforts on concrete aspects of the problem, where they feel more comfortable and closer to the final solution. Unfortunately, this approach can cause them to overlook critical details related to the problem or its solution.**

To address this issue in our Introduction to Programming (CS1) course, we introduced a programming methodology that requires students to create a graphical representation of their solution and then derive the code from it. To enable them to practice this diagrammatic reasoning approach on a regular basis, we developed a learning tool called CAFÉ 2.0. It facilitates a semester-long activity in which students solve problems by submitting both a graphical representation of their solution and its implementation. Further to checking the final implementation, CAFÉ 2.0 also provides personalized feedback on how students have graphically modeled their solution and how consistent it is with their code. This paper presents an overview of the features of CAFÉ 2.0 and the methodology it currently supports in the context of our CS1 course. Then, using a survey and learning analytics, this paper evaluates students' interactions with CAFÉ 2.0. Finally, the potential for extending CAFÉ 2.0 to other STEM disciplines is discussed.

***Index Terms*—CAFÉ 2.0, Computer-Assisted Learning, Automated Feedback, Abstraction, Diagrammatic Reasoning, CS1**

I. INTRODUCTION

Entering Higher Education can be a challenging transition. Students are expected to autonomously digest complex and expansive academic topics. However, students often miss personalized feedback and have difficulties in self-regulating [1]. In our country, this trend is even more pronounced, as the open access is the norm in most Science, Technology, Engineering and Mathematics (STEM) fields, including Computer Science (CS). This leads to a situation where many students lack a strong mathematical foundation and struggle with abstraction, which are both crucial skills for solving problems and, therefore, mastering a STEM discipline [2].

This paper focuses on CS, which is closely related to STEM [3]. More specifically, we consider an Introduction to Programming course (usually referred to as CS1), that covers specific programming language concepts (C language) as well as abstraction. To mobilize abstraction skills, we teach an approach where the difficulty lies not in writing code, but in a *modeling phase* that is prior to a *development phase*.

To make this *modeling phase* accessible to first-year students while keeping it relevant, we adopt an "active" algorithm visualization technique. Although thinking through drawing is more intuitive and requires less formalization, it also relies on diagrammatic reasoning (DR), which involves abstraction [4] and is therefore an important competence in STEM [5], [6].

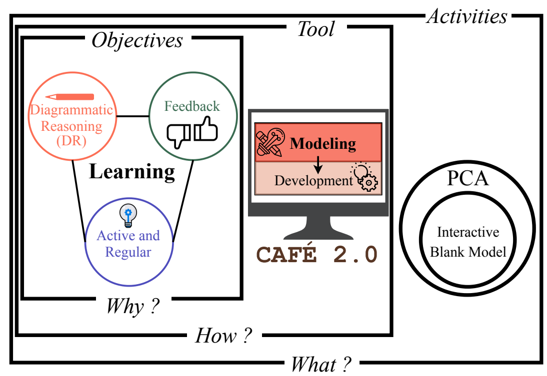


Figure 1. Motivation and context around CAFÉ 2.0 and our CS1 course.

In this paper, we present one type of visualization, aiming at modeling a *loop*, i.e., an iterative process repeating a certain number of times a block of instructions. We call it LOOP DRAWING (commonly referred to as *Graphical Loop Invariant* [7]). We integrate it in a programming methodology defined as Loop Drawing Based Programming (abbreviated as LDBP) [8]. To teach this methodology, we present it as a set of five solution parts that are clearly detailed in the theoretical and practical classes. Each of them belongs to the *modeling phase* or the *development phase*, or bridges them. This approach requires training and experience as it is new to students. But once mastered, it makes them approach programming from a new frame, privileging analysis over trial and error [9]. It results in clearer, less complex, and more effective solutions. That programming methodology is presented in Sec. II.

More generally, Fig. 1 summarizes the context and challenges described so far in the rectangle labeled "Objectives".

Encompassing these objectives, Fig. 1 expresses how to meet them. To ensure regular and active participation from a large group of students and provide them with personalized feedback, the most effective approach is to utilize a remote activity supported by an online system. We implemented such a learning tool and we called it "CAFÉ 2.0" (standing for "Correction Automatique et Feedback des Étudiants"[1]). Its features are described in Sec. III. Generally, CAFÉ 2.0 presents programming problems and scaffolds the expected solution

[1] which means "Automated Assessment and Feedback for Students"

through placeholders that map with the LDBP methodology's solution parts. Students must fill them and submit their answer. Then, they receive automated formative feedback and feedforward and, from that review, they can refine their solution and submit it again. Unlike most CS learning tools that are limited to automatic code simulation and evaluation [10], [11], [12], [13], [14], CAFÉ 2.0 focuses not only on program output, but also on the cognitive abstraction process involved in program construction. This feature makes relevant the expansion of CAFÉ 2.0 to other STEM problems whose solution is also constructed using a canvas to be filled in, before moving to lower levels of abstraction. Sec. VII shows how CAFÉ 2.0 could be extended and illustrates an example in Physics.

In practice, to make students use CAFÉ 2.0, it needs to be integrated into the course via activities, as shown in the outer rectangle labeled "Activities" in Fig. 1. In our course, the *Programming Challenge Activity* (PCA) was created [15]. In the context of the PCA, students must regularly solve statements. The modalities of the activity are detailed in Sec. IV.

After this, Sec. V presents the context in which we collected data in order to report how students perceived the LDBP methodology, CAFÉ 2.0 and the PCA, and how they distribute their attention between DR and coding tasks. These results are presented in Sec. VI. They mainly show that the methodology is hard to embrace, the PCA is engaging, and CAFÉ 2.0's feedback stimulates students' practice. By delving into a closer examination of their use of the LDBP methodology, we could also noticed that students tend to oscillate between the *modeling and development phases*.

II. OBJECTIVE : TEACHING ABSTRACTION THROUGH DIAGRAMMATIC REASONING (DR)

In STEM, computational thinking (CT) concepts are permanently mobilized in order to solve problems, and abstraction stands as the keystone of CT [16]. Although abstract thinking is a core skill for many scientists [17], there is no universal definition of abstraction [18]. Similarly, there is no standard method for teaching it [18]. The most suitable definition of abstraction for our teaching approach is the use of drawings to show important relationships between variables in a piece of code while ignoring unnecessary aspects of the problem [19]. As such, we believe that applying DR [20] is an appropriate way to introduce CT [21] and to allow students to develop their abstraction skills. Previous studies have also shown that developing spatial skills by asking students to construct their own visualization has a positive impact on their learning of STEM disciplines [22], [23]. More specifically, it also promotes success in CS1 courses [24], [25].

For this purpose, we defined the LDBP methodology. It consists in first designing the solution (through a *modeling phase*) so that its implementation (through a *development phase*) becomes straightforward. These *modeling and development phases* echo other frameworks from the literature. Namely, they map the two key tasks (modeling and implementing) of software development, identified by Zehetmeier et al. [26].

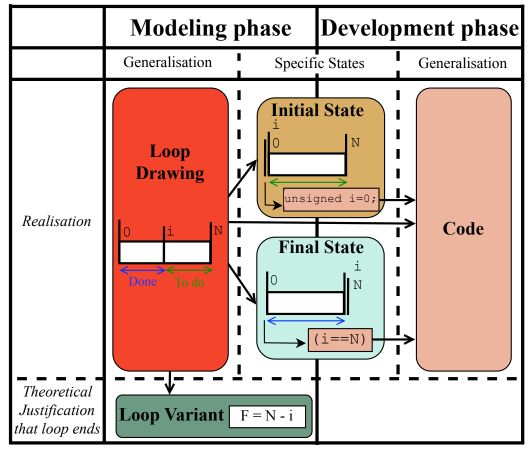


Figure 2. The five solution parts belonging to the two LDBP phases.

They also match with two of the stages of Loksa et al. [27], where students are expected to translate some abstract representations of a solution into code.

In particular, our LDBP methodology applies each time students need to implement a loop and it relies on drawing [8]. The drawing reflects the program loop's behavior that solves the problem and serves as the core of the *modeling phase*, as highlighted through Fig. 2. It encompasses all of the necessary information, in such a way that the split-attention effect [28] is avoided. Fig. 2 also shows that LOOP DRAWING should be manipulated to illustrate specific solution states: the INITIAL and FINAL STATES. This is similar to what is taught by Walker in his second stage [9], that aims at developing LOOP DRAWING from a problem statement and connecting it to the pieces of code development. To do this, students must clarify any and all relationships among variables using LOOP DRAWING. Then, they should exploit it before and after the loop, mapping so with our INITIAL and FINAL STATES.

Next, Fig. 2 shows the *development phase*, where students deduce a piece of code from these two states and integrate them in their whole code stemming from LOOP DRAWING.

Finally, besides this sequence of realisations, a theoretical justication (the Loop Variant) must be provided to ensure the loop ends. The Loop Variant is a function that calculates the number of elements that remain to be processed in the loop, based on the number of iterations that have been performed so far. This function must be decreasing, so that it becomes negative after a certain number of iterations. This indicates that there are no more elements to process, and the loop exits.

III. TOOL : CAFÉ 2.0

This section describes CAFÉ 2.0, in the context of our CS1 course. Its first module supports the problem-solving process, by going through the *modeling and development phases*, where the *modeling phase* relies on DR. Its second module consists in building a personalized feedback.

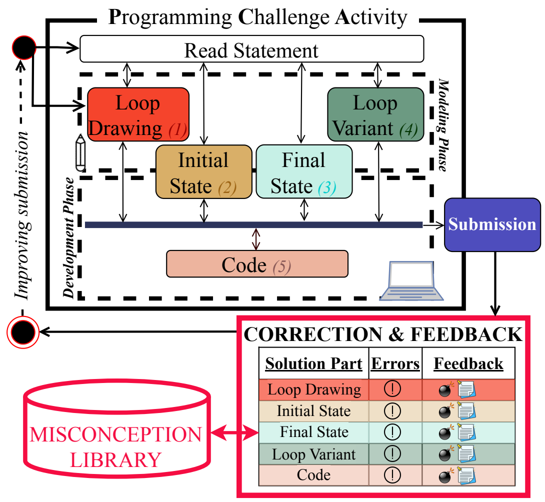


Figure 3. Activity Diagram showing how students are solving Challenge in CAFÉ 2.0, during the PCA.

$\text{Step}_1 \rightarrow$	$\text{Step}_2 \rightarrow$	$\text{Step}_3 \rightarrow$	$\text{Step}_4 \rightarrow$	$\text{Step}_5 \rightarrow$	Step_6
LOOP DRAWING →	INITIAL STATE →	FINAL STATE →	LOOP VARIANT →	CODE →	SUBMISSION

Figure 4. Expected problem-solving path following the LDBP methodology.

A. Problem-Solving Module

At a global level, CAFÉ 2.0 assists students in problem-solving by breaking down the expected solution into specific solution parts. They are depicted through the upper part of Fig. 3 and, more concretely, through Fig. 5, being a screenshot of CAFÉ 2.0. They are aligned with our methodology (see Fig. 2). We define a *step* as any update to a solution part, and a *problem-solving path* as a sequence of steps leading to the submission. The expected problem-solving path is provided in Fig. 4. Although students are taught to perform the solution parts in this order, CAFÉ 2.0 does not force them to do so. Due to this freedom, many problem-solving paths are possible. We do not lock that path as students may avoid that constraint anyway by using their own code editor, which would cause us to lose track of their actual problem-solving path.

Going deeper, at a local level, CAFÉ 2.0 also individually outlines each solution part. In Fig. 5, the first tab (labeled "Loop Drawing") contains a predefined canvas called the Blank Model. It depicts the shape that the correct LOOP DRAWING should have to solve a given problem. While solving the problem, the students must annotate the Blank Model so that the drawing becomes their own LOOP DRAWING. Two types of boxes must be filled in: (i) the red boxes, standing to host expressions composed by students (i.e., constants, variables, operations, or left blank – see `a` in box 2 on Fig. 5); (ii) the green boxes, standing to host labels that students must drag and drop from a list that is predefined by the supervisor (see the list on the left of the Blank Model in Fig. 5). By constraining the LOOP DRAWING, automatic semantic correction becomes possible. This is accomplished by pre-modeling the empty framework (i.e., Blank Model), utilizing the predefined boxes, where each box symbolizes a solution component.

The tab "Loop Variant" expects students to provide the Loop Variant function. The tabs "Initial State" and "Final State" allow students to transpose their LOOP DRAWING in specific states to get them closer to the final solution (being the code). Finally, the code part is simply represented through a template [29] where students have a placeholder (see "Code Editor" on Fig. 5). Moreover, they can compile and test the whole code through the "Playground Mode".

At that point, it is worth noting the two interests in decomposing students solution into predefined solution parts. First, it allows CAFÉ 2.0 to pave students' problem-solving path with respect to a given methodology. Next, it frames their solution, making feasible automated personalized feedback, mainly thanks to the Blank Model.

B. Correction and Feedback Module

When a student submits their solution (by clicking the "Submit" button on Fig. 5), CAFÉ 2.0 automatically corrects it and provides personalized feedback and feedforward. As shown in Fig. 6, the system identifies students' mistakes using error codes that are classified and characterized in a misconception library.

1) The misconception library: It is worth noticing that the concept of misconception is broadly used in the STEM literature [30], [31]. In this paper, we similarly use the terms "misconception", "error", and "mistake" to refer to "something that is done wrong". From a general point of view, any supervisor wanting to use CAFÉ 2.0 as automatic assessment and feedback system should define a rubric checklist [32] beforehand, forming so the *misconception library*. That rubric should be organized according to the parts a student's submission is made up of. For each solution part, typical mistakes should be identified. We built it based on previous experiences, like presented in other studies [33], [34]. Then, each mistake should be characterized by a unique error code, a nature (syntactic/semantic), a gravity factor (quantifying how serious the mistake is), a feedback message (explaining in details the error), and, optionally, a corresponding reference to the course (i.e., feedforward). Once the misconception library has been fed, some respective rule-based checks must be implemented and simply configured in order to catch each mistake in a given submission related to a given problem.

2) The Correction and Feedback Construction: When these last set-ups are ready, the system can process submissions. For a given student's submission, each solution part is digested by a dedicated checker module that detects any potential mistake defined in the misconception library.

In particular, for the LOOP DRAWING, each box is checked against the expected solution as well as the contents of the other boxes filled by the student. Each box is numbered so that they can be easily referenced in the feedback, making it richer while still being clear and easy for the student to digest. To define a Blank Model, a supervisor can use the Loop Drawing Editor implemented in CAFÉ 2.0. It offers patterns and predefined graphical components [8] that can be arranged to build a Blank Model. It is worth noting that students can

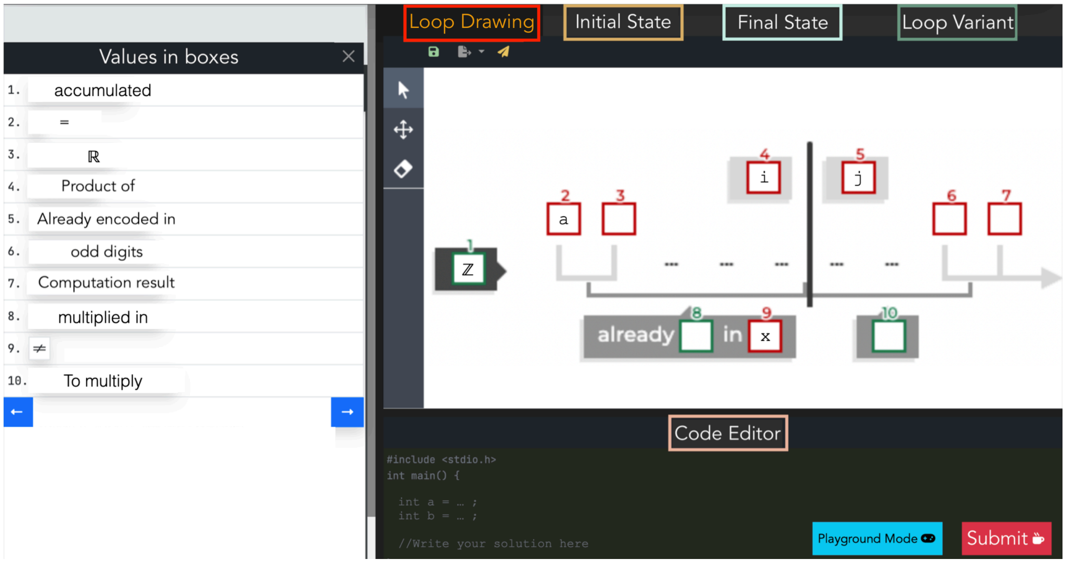


Figure 5. Screenshot of the PCA in CAFÉ 2.0. It also shows how CAFÉ 2.0 follows the LDBP methodology with tabs, one for each solution part.

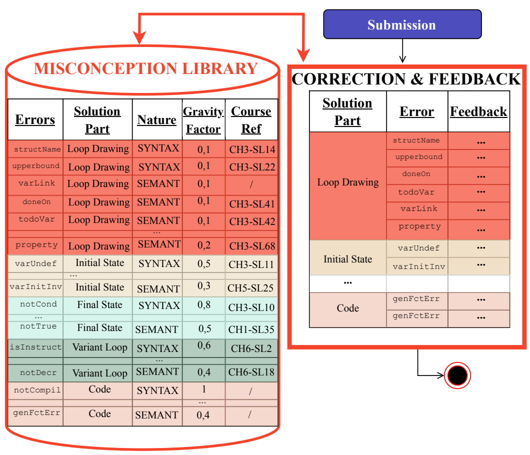


Figure 6. Correction and Feedback module of CAFÉ 2.0.

also use it to define their own LOOP DRAWING by directly filling in the components, independently from any activity.

Next, the checks related to the INITIAL STATE, the FINAL STATE, and the Loop Variant cover the scan of the LOOP DRAWING's specific states students derived as well as their consistency with the final code and the other fields (e.g., variables introduced in the INITIAL STATE have to be initialized in the code accordingly).

Finally, regarding the code, it first gets compiled. Then, different tests are run, each of them corresponding to a general or a particular scenario.

If an error is detected, the student's final grade is affected by the gravity factor that characterizes the error. In addition, the corresponding feedback message and course reference are added to the list of comments that are ultimately provided to the student. This list of comments is split on a per-solution-part basis. Fig. 6 shows an example of misconception library from which mistakes were detected in a given submission.

Finally, once the feedback has been received, no particular order in the problem-solving path is expected as each student has their own errors to fix, categorized per solution part. From a theoretical standpoint, CAFÉ 2.0 implements "Answer-until-correct" (AUC) [35] feedback approach allowing students to refresh their solution as many times as needed. It is comparable to Singh et al. [36] approach, where students receive a numerical value indicating the number of required changes, along with suggestion(s) on how to correct the mistake(s).

IV. ACTIVITY : THE PCA

Three times during the semester, students are given a Programming Challenge [15]. This usually requires an additional four out-of-class hours. Their content for Academic year 2022–2023 is described in Table I. Each Challenge consists in a problem potentially split in several subproblems (defined in the last column in Table I). For each subproblem (relying on a loop), a Blank Model and a code template are provided (Fig. 5). Regarding the modalities, a Challenge starts on Wednesday, 06:00PM and finishes on Friday, 08:00PM. During this 2-days timeframe, a student can submit up to three times their solution, each one receiving an automated feedback. The latest submission determines the final mark and each Challenge accounts for 2% of the final mark for the course. After this certificative period, students are free to keep training, but it will not affect their final grades.

V. METHOD

During the academic year 2022–2023, 97 students (N=97) registered to our CS1 course. Among them, 53% (resp. only 20%) had more than 2h/week (resp. 4h/week) of Physics (resp. Maths) in Secondary school. A tiny proportion of students (5%) had some prior programming experience.

From a content perspective, our course strongly focuses on the LDBP methodology, described in Sec. II. Fifteen hours of theoretical and traditional exercises sessions are dedicated to

Table I
DESCRIPTION OF THE CHALLENGES SUPPORTED BY CAFÉ 2.0 DURING THE ACADEMIC YEAR 2022–2023.

Challenge	Starting Date	Topic	Details
1	Oct. 19th	Counting the odd digits for all numbers in $[a, b]$	outer loop (spans $[a, b]$) inner loop (count odd digits in a given number)
2	Nov. 9th	Compressing an integer array into another	a single loop

3	Nov. 30th	Displaying all numbers in array `T` that follow a property G	`uint power(uint a, uint b)`
			`uint is_prime(uint x)`
			`uint is_g(uint x)`
			`void display(int T[], int N)`

it (roughly half of the course schedule). Moreover, students are also exposed to the three Challenges described in Sec. IV.

At the beginning of the semester, students were informed they would be part of a research study and they got the opportunity to opt-out of the study. All students gave their consent and no one opted-out throughout the semester.

Perception data was collected via an anonymous survey, following the final exam. 74 students shared their opinion. Further, additional data was collected via CAFÉ 2.0 and it was anonymized prior to our analysis. In particular, any action performed by a student was logged, e.g., login/logout, opening/closing a tab, updating the Blank Model, etc. Further, all logs are timestamped.

Prior to analyzing the logs to track students' actions on CAFÉ 2.0, the data was pre-processed to remove any unexpected usage traces. Specifically, we addressed the issue of external IDE (Integrated Development Environment) usage by identifying instances of copying and pasting. This allowed us to monitor the amount of isolated pastes in CAFÉ 2.0's code editor, thereby enabling us to recognize possible employment of external IDEs. A paste is considered isolated if there is no recorded corresponding copy action in CAFÉ 2.0. Our preliminary analysis revealed 72 submissions out of 469 in which at least one isolated paste was identified. As a result, these submissions were excluded from the dataset.

VI. PRELIMINARY EVALUATION

The purpose of this section is to showcase the value of CAFÉ 2.0. To fulfill this objective, we first present students' perspectives on the three goals behind CAFÉ 2.0 (see Fig. 1). Subsequently, we observe how students implement the LDBP methodology, using learning analytics. The impacts of CAFÉ 2.0 on students' performance is left for future work.

A. Perception

Considering students' opinion related to the LDBP methodology, Fig. 7 shows a large majority (63%) appear to mainly like coding. On the opposite, only 34% of students find LOOP DRAWING useful, 27% are not sure about that claim, and 39% explicitely express they do not take any benefit from it. That demonstrates the difficulty students may encounter in grasping the LDBP methodology. That statement gets confirmed seeing how students self-assess some of their skills: (i) about half of the students consider they are rigorous enough; (ii) 35% of them feel their code is of good quality; (iii) similarly,

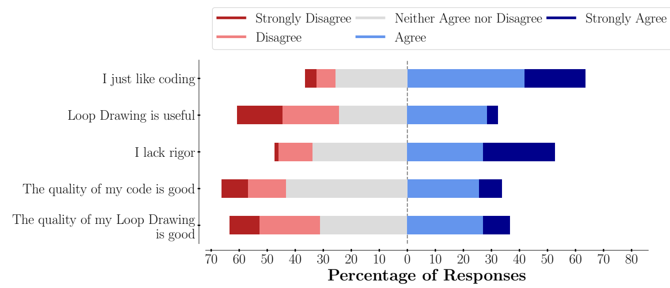


Figure 7. Perception of the LDBP methodology ($N = 74$).

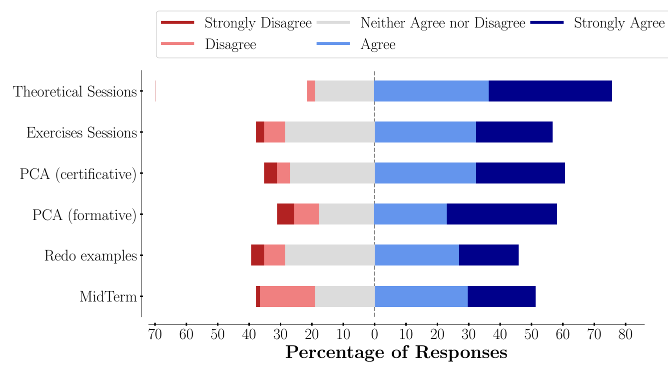


Figure 8. Perception of the PCA with respect to other activities ($N = 74$).

38% of students think that they are able to design a good LOOP DRAWING. These last opinions are in favor of a lack of awarness regarding abstraction (it is aligned with previous results [37], [38]). By correlating the five statements, we have discovered a positive and more significant correlation between self-rated code quality and LOOP DRAWING design quality (r=0.42, p=0.0002), suggesting that LOOP DRAWING helps in better coding. To a lower extend, we also noticed that students who find LOOP DRAWING useful tend to give a higher rate to their ability to design it (r=0.30 and p=0.0116), which expresses the importance of motivating the effectiveness of the LDBP methodology.

A second inquiry in the survey assesses how the PCA, supported by CAFÉ 2.0, affects students' motivation in learning, in comparison to other (more conventional) activities. Fig. 8 shows that the PCA comes as the second most stimulating activity, after the theoretical lessons. 60% of the respondents (strongly) agreed that the PCA was motivating, 25% had no opinion, and 15% did not embrace that online experience.

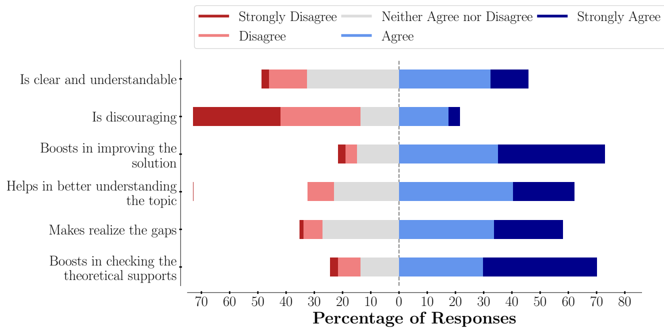


Figure 9. Perception of the feedback ($N = 74$).

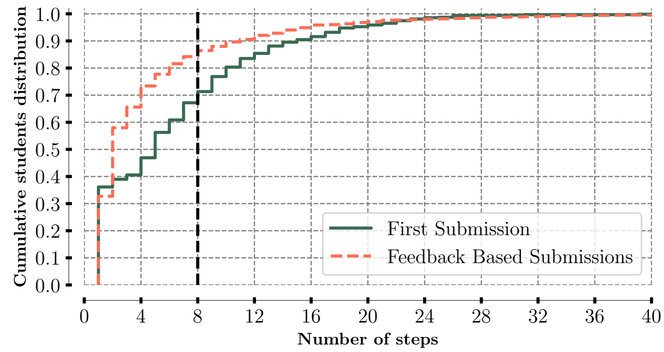
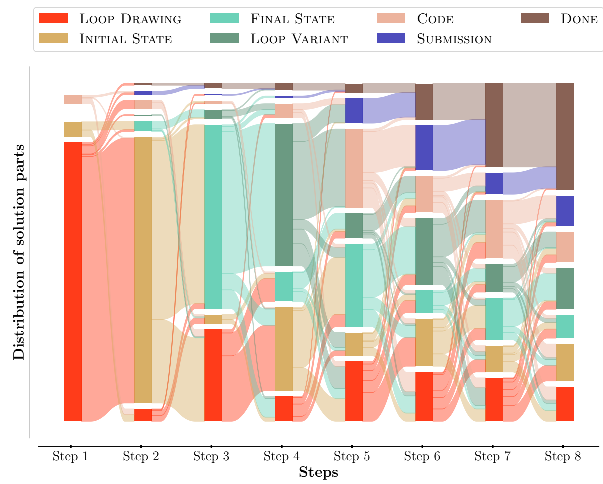
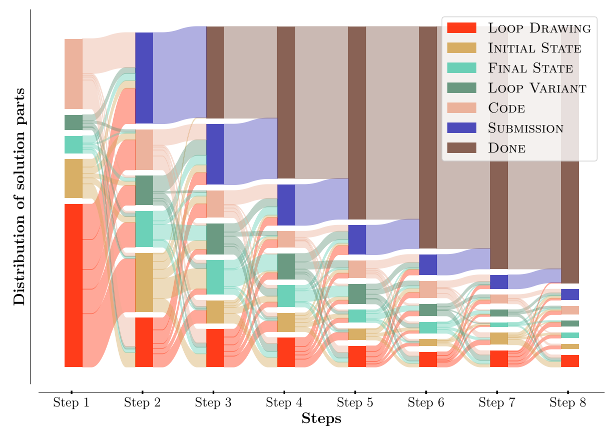


Figure 10. Number of steps articulating the problem-solving paths.



(a) First submission (N_1=135 – 312 steps sequences).



(b) Feedback based submissions ($N_{2 \rightarrow n}$ = 263 – 575 steps sequences).

Figure 11. Programming Challenges solving path followed by students.

Specifically, the inclusion of both the formative and certification periods seems to be beneficial to the practice of the course. A closer look reveals a slightly stronger preference for the formative period. It is likely that some students were able to benefit from the PCA independently, without the pressure of being graded. However, some did not take advantage of the formative periods to train. One possible reason for the lower activity outside the certification period could be that students became stuck despite receiving feedback, which hindered their progress and prevented them from completing the Challenge.

Finally, special interest was paid to the way students perceived the automated feedback. Fig. 9 reflects that CAFÉ 2.0 feedback is well received by the students. Half of the respondents found it clear and understandable, 30% had some misunderstanding of it, and 20% could not understand it well. Nevertheless, the majority of respondents (74%) felt encouraged to improve their solution after receiving the feedback. Likewise, more than 60% of respondents could identify their gaps, focus on the appropriate theoretical support to fill them, and better understand the topic. Finally, it is also interesting to note that, although some students struggled to digest the feedback, few of them felt truly discouraged.

B. How Students Embrace the LDBP *Methodology?*

This section targets to draw the different problem-solving paths students follow. According to the LDBP methodology, students should achieve the five solution parts in the suggested order (Fig. 3 and Fig. 4). However, as explain in Sec. III, CAFÉ 2.0 does not force them to do so.

To assess how the methodology was actually handled, we distinguish between the first submission of a Challenge (N_1 = 135 submissions) and subsequent ones after feedback ($N_{2 \rightarrow n}$ = 263 submissions). The latter are referred to as "feedback based submissions". The motivation behind this distinction is that the solution parts that students choose to work on are strongly influenced by the feedback they received for all the submissions following the first one. Conversely, when students prepare their first submission, they all start from the same point, which better frames the discussion.

Fig. 10 distills the number of steps students take to build their solution. 70% of them need a maximum of eight steps for their first attempt, while 10% go from one part to another more than 14 times. As for the feedback based submissions, fewer steps are required: 70% of students need less than five steps to tune their solution. Based on these observations, our analysis will focus on the first eight steps students take for each subproblem in each submission.

Fig. 11 presents two Alluvial diagrams[2], one for each

[2]An Alluvial diagram is a type of flow chart that represents changes in a network structure over time. In that sense, it helps identify patterns and trends (see https://en.wikipedia.org/wiki/Alluvial_diagram).

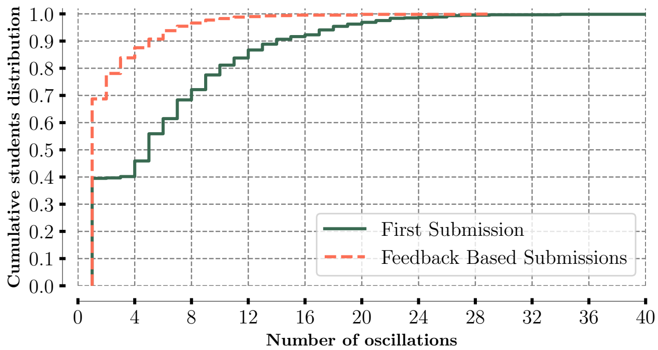


Figure 12. Number of oscillations between the *modeling and development phases* (per submission).

submission type.

From Fig. 11a, we can see that students address their attention to the four parts of the *modeling phase*, before moving further. More generally, the LOOP DRAWING step occurs the most frequently. Some students also appear to move back and forth between consecutive parts. Typically, from step 1 to 4, 25% of students confine their attention on LOOP DRAWING and the INITIAL STATE. Regarding their last updates, 50% of students submit their whole solution right after performing the code part, while the others review their *modeling phase*. Finally, 10% of students submit the first version of their solution without any code.

These results suggest that students embrace the methodology since they first focus on the parts included in the *modeling phase* and keep going back to them further in their problem-solving path. However, these results are limited to the eight first steps. Therefore, in order to capture to which extent students move back and forth between the *modeling and development phases*, Fig. 12 highlights the cumulative distribution of students across the number of oscillations between these phases. The main outcome is that students do not follow a linear path. Instead, they seem to permanently try to reconcile their abstract representations and their code. Quantifying that finding, 60% of students oscillate more than five times between their design and their code.

For the feedback based submissions, Fig. 12 shows that the frequency of oscillations is much lower than for the first submissions. Typically, 80% of students move less than four times between an abstract representation and their code. More precisely, Fig. 11b depicts that they do not go (anymore) through the whole flow. Half of the students submit the new version of their solution after two steps, the latest. 35% of them even just review their LOOP DRAWING, without adapting the code accordingly. Among the others, 80% of them skip reviewing their code after updating their representation(s), which may lead to some dissonances between their *modeling and development phases*. On the contrary, many students (30% for the transition from step 2 to 3 typically) move back to LOOP DRAWING after managing the code, which illustrates some reverse engineering.

To summarize, students oscillate between the *modeling and development phases*. A similar behavior was observed by Friebroon et al. [39], [40], showing that the process of problem solving involves back and forth transitions between levels of abstraction. In our case, this is especially true for the first submission. For the next ones, students mainly go through LOOP DRAWING and its states. When looking at the mistakes captured by the system, most of them occurred for the *modeling phase*, which explains such a behavior. More generally, these results also show the interest of CAFÉ 2.0. Students focus on the graphical model of their solution, thus training their abstraction skills through diagrammatic reasoning.

VII. EXTENDING CAFÉ 2.0

This section projects CAFÉ 2.0 as an interdisciplinary learning tool, aiming to train abstraction through DR, in general.

A. General Considerations

To integrate a new problem profile in CAFÉ 2.0, the following requirements must be met:

Requirement 1: The problem solving process should be paved by a sequence of solution parts.

Requirement 2: Problem solving should run through two phases : the abstraction one and the concrete one.

Requirement 3: The abstraction phase should rely on diagrammatic reasoning.

Requirement 4: The graphical representation should be dynamic, in such a way that it can be manipulated to illustrate different solution states (general ones and specific ones).

Requirement 5: The graphical representation should be made up of predefined graphical components. They can stand as placeholders or movable elements students must handle when they are designing a solution.

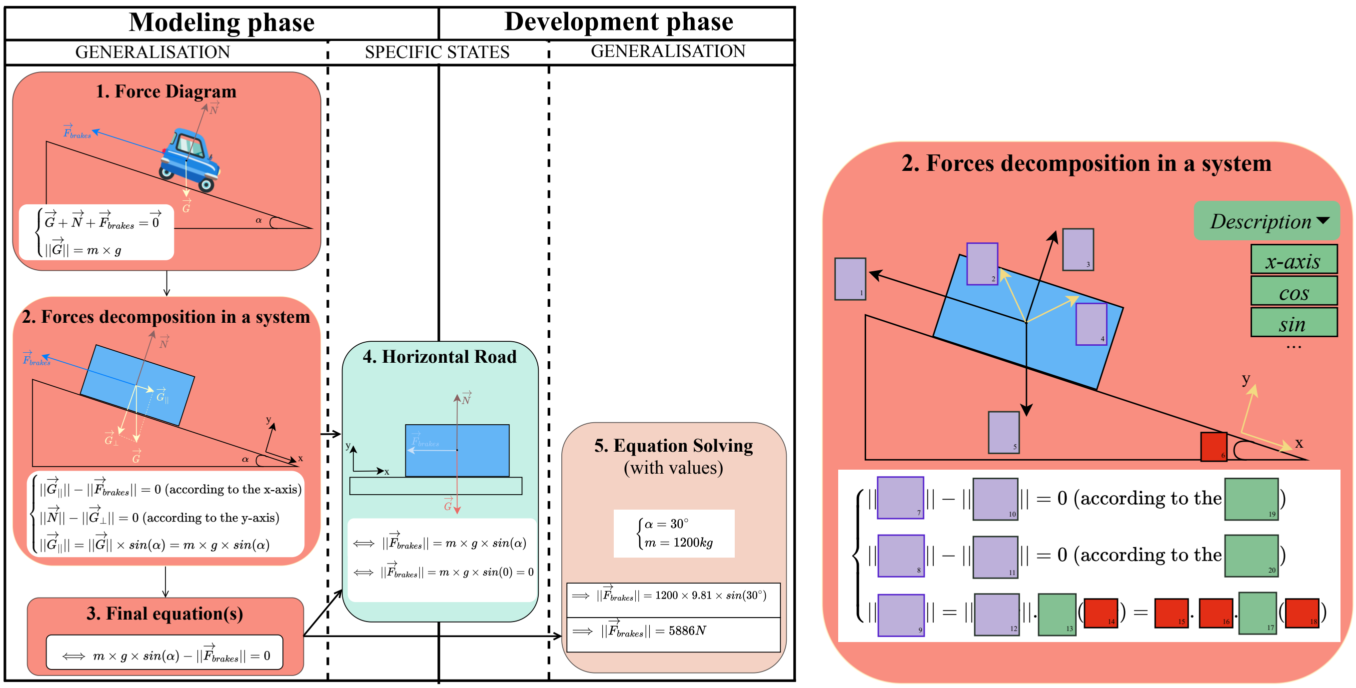
In addition, regardless of the discipline, an activity must be designed to make CAFÉ 2.0 an integral part of the course. Finally, it should be noted that first-year students are the most appropriate target group to avoid modeling solutions that are too complex. Moreover, it is likely that first-year students are the ones most in need of this type of support.

B. Application to Physics

To emphase the interdisciplinary potential of CAFÉ 2.0, we match it to a specific problem profile picked from another field than CS. In particular, we are considering the following Kinematics problem in Physics :

A car of mass $m = 1200kg$ is parked on a slope of $\alpha = 30°$. We would like to compute the magnitude of the friction forces so that the car is at rest.

1) Problem-Solving Setting: Considering this specific type of problem, five solution parts could be defined: (i) Representing the situation and identifying the forces acting on the object of interest (here the car); (ii) Choosing a system and decomposing the various forces so that they follow the system's directions; (iii) Deriving the mathematical expression(s) that allow one to formulate the friction forces; (iv) Transposing the



(a) Problem-Solving Flow in Physics

(b) Blank Model in Physics.

Figure 13. How CAFÉ 2.0 can fit to a Physics Introduction course.

general representation in a particular case (where the problem may be possible to solve intuitively); (v) Using numerical data to compute the solution. Fig. 13a illustrates these solution parts and maps them to the two phases supported by CAFÉ 2.0 (*Requirement 2*). Fig. 13a also shows that the problem should be solved sequentially (*Requirement 1*). Further, the Force Diagram satisfies *Requirement 3*. Finally, this drawing is manipulated at step 4 meeting so *Requirement 4*.

2) Correction and Feedback Setting: To automate correction and feedback, a misconception library needs to be defined and fed. This library should cover most of the misconceptions students may fall into while solving the problem, as explained in Sec. III-B and illustrated in Fig. 6. For example, considering our Kinematics problem, for the second solution part where the forces should be decomposed according to the system, a typical mistake could be that the students did not direct all the arrows according to the orthonormal system that is set. Another example of an error could be that the second equation does not reflect the drawing above.

3) Solution Parts Modeling Setting: Similar to the Blank Model, it is important to hide the key components from the expected pictorial representation of the situation. In this way, students have to identify and connect them on their own, while still having benchmarks thanks to the provided canvas. Furthermore, on the supervisor's side, these components have to be modeled (by having a specific semantic and relation with each other) in order to enable automated personalized feedback. For the Kinematics problem of interest, a relevant empty diagram could be the one exposed by Fig. 13b. This figure is based on different types of graphical components. Each of them is mapped to a specific color. As in the Blank Model, red boxes should contain variables while green ones expect a description selected from a drop-down list. In addition to them, purple boxes represent forces and movable arrows are illustrated in yellow. Everything else is fixed.

VIII. RELATED WORK

Fig. 14 highlights the three dimensions of CAFÉ 2.0 (aligned with the objectives introduced in Fig. 1): it implements diagrammatic reasoning (Sec. II) within an activity (Sec. IV) and provides automated feedback (Sec. III). Several tools sharing those features are presented[3], with most being related to programming. This is because CT skills, approached via DR in this paper, are often seen as directly linked to CS [3].

To the best of our knowledge, the only tool that trains DR and provides automated feedback at the same time is Sowiso [41], by calling GeoGebra [42]. Sowiso proposes exercises for practicing STEM disciplines (mainly Mathematics, but also some topics in Physics). Sowiso enables teachers to create their own classroom and visualize their students performance. However, it does not include the exercises in any activity or game. Sowiso is also expanding to teaching programming (in Python), but without relying on DR.

Generally, training DR and automating the review of a graphical model constructed by a student is challenging [43]. This may explain why many learning tools solely focus on either (i) DR (e.g., BlueJ [44]), (ii) design and simulation (e.g., Geogebra [42], Codex [45], Scratch [46]), or (iii) automated

[3]Providing an exhaustive list of all the similar learning tools is out of the scope of this paper.

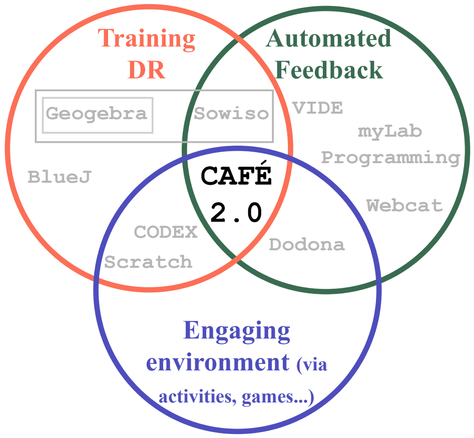


Figure 14. Objectives of CAFÉ 2.0 and other tools meeting them.

feedback on the final solution, without considering abstract thinking, upstream to the code (e.g., VIDE [14], Dodona [13]).

Enabling solution simulations can help students determine if they are on the right track. Codex [45] is a gamified application that offers challenges to solve, along with a visualization of the execution of their pseudocode. Scratch [46] provides a visual programming environment [47] in which students manipulate graphical elements and figures to create programs that generate animations. Like for the LOOP DRAWING, boxes are objects treated as entities and organized to represent computational relationships [48]. This approach is suitable for students with limited computer programming knowledge [49].

Further than simulating specific instances, many tools offer programming exercises, with automated feedback [50]. Most of them use "test-based correction", i.e., the student's code is corrected through unit tests. Dodona [13] typically provides problem statements and corresponding unit tests that are specifically designed to cover edge cases. However, it does not assist students with exploratory modeling and visualization. Dodona only checks the final solution, which is the code. Students are encouraged to use Dodona as it supports activities, similar to CAFÉ 2.0 which supports the PCA. Activities are crucial to promote engagement [51].

IX. CONCLUSION

This paper introduces CAFÉ 2.0, a system that offers personalized feedback. Currently, CAFÉ 2.0 supports an online activity in which students solve programming problems, based on a graphical model called LOOP DRAWING.

For each new problem statement, the supervisor must provide its definition and corresponding solution. The solution should be expressed through distinct parts, each presented on a canvas (either a template or a Blank Model) with designated placeholders for students to complete. Defining the semantics and potential errors associated with each solution part in advance is crucial. This helps guide students in solving the problem and allows for automated assessment. Additionally, all students' actions are recorded, allowing for tracking of their problem-solving strategies.

This paper focuses on the sequence of solution parts that students go through, in order to see if students actually adopt the problem-solving process supported by CAFÉ 2.0. Knowing that each solution part is related to a *modeling or development phase*, we can see that students tend to oscillate between modeling and coding tasks. Further research could explore the amount of time students spend on these tasks and the corresponding impact on their problem-solving skills. These results could be compared to the ones obtained by Böttcher and Grellner [52], who answered similar questions in the context of their own courses.

More generally, we could also use CAFÉ 2.0 to study students' problem-solving strategies in other areas by extending it. CAFÉ 2.0 is applicable to introductory courses that use abstract graphical representations to construct solutions. In Physics, for example, students often solve Kinematics problems with the help of force diagrams. We have already taken steps to extend CAFÉ 2.0 to this problem profile, envisioning it as a tool for practicing abstraction through diagrammatic reasoning in various STEM contexts.

SOFTWARE ARTEFACT

CAFÉ 2.0 is written in Python 3. It requires the Pandas library for working properly. CAFÉ 2.0 source code is available at this URL: https://gitlab.uliege.be/cse.

ACKNOWLEDGMENTS

This work is supported by the CyberExcellence project funded by the Walloon Region, under number 2110186.

REFERENCES

[1] G. Brieven, S. Liénardy, and B. Donnet, "Lessons learned from 6 years of a remote programming challenge activity with automatic supervision," in *Proc. European Distance and E-Learning Network (EDEN)*, June 2022.

[2] P. Henderson, T. Cortina, and J. Wing, "Computational thinking," in *Proc. ACM Technical Symposium on Computer Science Education (SIGCSE)*, March 2007.

[3] Y. Li, A. Schoenfeld, A. Disessa, A. Graesser, L. Benson, L. English, and R. Duschl, "On computational thinking and STEM education," *Journal for STEM Education Research*, vol. 3, no. 1, August 2020.

[4] K. Tylén, R. Fusaroli, J. Philipsen, J. Rączaszek-Leonardi, S. Østergaard, and F. Stjernfelt, "Diagrammatic reasoning: Abstraction, interaction, and insight," *Pragmatics and Cognition*, vol. 22, no. 2, pp. 264–283, January 2014.

[5] P. Osztián, Z. Katai, and O. Erika, "On the computational thinking and diagrammatic reasoning of first-year computer science and engineering students," *Frontiers in Education*, vol. 7, September 2022.

[6] A. Bakker and M. Hoffmann, "Diagrammatic reasoning as the basis for developing concepts: A semiotic analysis of students' learning about statistical distribution," *Educational Studies in Mathematics*, vol. 60, no. 3, pp. 333–358, November 2005.

[7] E. W. Dijkstra, *A Discipline of Programming*. Prentice-Hall, Inc., 1976.

[8] G. Brieven, S. Liénardy, L. Malcev, and B. Donnet, "Graphical loop invariant based programming," in *Proc. Formal Method Teaching Workshop (FMTea)*, March 2023.

[9] H. M. Walker, "Curricular syncopations: Where/how do loop invariants fit?" *ACM Inroads*, vol. 14, no. 1, p. 6–13, February 2023.

[10] S. H. Edwards and M. A. Perez-Quinones, "Web-CAT: Automatically grading programming assignments," in *Proc. Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, June/July 2008.

[11] Pearson, "My lab programming," [Last Accessed: October 12th, 2023]. [Online]. Available: https://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/

[12] R. Lobb and J. Harlow, "Coderunner: a tool for assessing computer programming skills," *ACM Inroads*, vol. 7, no. 1, pp. 47–51, March 2016.

[13] C. Van Petegem, R. Maertens, N. Strijbol, J. Van Renterghem, F. Van der Jeugt, B. De Wever, P. Dawyndt, and B. Mesuere, "Dodona: Learn to code with a virtual co-teacher that supports active learning," arXiv, cs.CY 2210.10719, October 2022.

[14] D. Pawelczak and A. Baumann, "Virtual-C – a programming environment for teaching C in undergraduate programming courses," in *Proc. IEEE Global Engineering Education Conference (EDUCON)*, April 2014.

[15] S. Liénardy, L. Leduc, D. Verpoorten, and B. Donnet, "Challenges, multiple attempts, and trump cards – a practice report of student's exposure to an automated correction system for a programming challenges activity," *International Journal of Technologies in Higher Education (IJTHE)*, vol. 18, no. 2, pp. 45–60, June 2021.

[16] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, March 2006.

[17] A. Darwish, "The abstract thinking levels of the science-education students in Gaza universities," *Asia-Pacific Forum on Science Learning and Teaching*, vol. 15, no. 2, December 2014.

[18] D. Zehetmeier, A. Böttcher, A. Brüggemann-Klein, and V. Thurner, "Defining the competence of abstract thinking and evaluating CS-students' level of abstraction," in *Proc. Hawaii International Conference on System Sciences (HICSS)*, January 2019.

[19] N. M. Seel, Ed., *Encyclopedia of the Sciences of Learning*. Springer Verlag, 2011.

[20] M. Anderson, B. Meyer, and P. Olivier, Eds., *Diagrammatic Representation and Reasoning*. Springer London, 2002.

[21] G. Pólya, *How to Solve It*. Princeton University Press, 1945.

[22] N. Kiernan, A. Manches, and M. Seery, "The role of visuospatial thinking in students' predictions of molecular geometry," *Chemistry Education Research and Practice*, vol. 22, no. 3, pp. 626–639, January 2021.

[23] L. B. Nilson, *The Graphic Syllabus and the Outcomes Map*. Jossey-Bass, 2007.

[24] J. Parkinson and Q. Cutts, "Investigating the relationship between spatial skills and computer science," in *Proc. ACM Conference on International Computing Education Research (ICER)*, August 2018.

[25] M. Cooper, C. Shaffer, S. Edwards, and S. Ponce, "Open source software and the algorithm visualization community," *Science of Computer Programming*, vol. 88, pp. 82–91, August 2014.

[26] D. Zehetmeier, A. Böttcher, V. Thurner, and A. Brüggemann-Klein, "A concept for addressing abstract thinking competence while teaching software development," in *Proc. IEEE Global Engineering Education Conference (EDUCON)*, April 2020.

[27] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, problem solving, and self-awareness: Effects of explicit guidance," in *Proc. Conference on Human Factors in Computing Systems (CHI)*, 2016.

[28] J. Sweller, *Instructional Design in Technical Areas*. Australian Council for Educational Research, April 1999.

[29] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercices," *ACM Transactions on Computing Education (TOCE)*, vol. 19, no. 1, pp. 1–43, September 2018.

[30] L. Nieuwkoop, "Data-driven intervention: a minor tweak, a major revelation – correcting mathematic students' misconceptions, not mistakes," *Mathematics Educator*, vol. 23, no. 1, pp. 24–44, 2013.

[31] I. Sikurajapathi, K. Henderson, and R. Gwynllyw, "Using e-assessment to address mathematical misconceptions in engineering students," *International Journal of Information and Education Technology*, vol. 10, no. 5, pp. 356–361, May 2020.

[32] S. Bharuthram and M. Patel, "Co-constructing a rubric checklist with first year university students: A self-assessment tool," *Journal of Applied Language Studies (APPLE)*, vol. 11, no. 4, pp. 35–55, December 2017.

[33] D. Zehetmeier, A. Bottcher, A. Brüggemann, and V. Thurner, "Development of a classification scheme for errors observed in the process of computer programming education," in *Proc. Higher Education Advances (HEAd)*, June 2015.

[34] V. Almstrum, P. Henderson, V. Harvey, C. Heeren, W. Marion, C. Riedesel, L.-K. Soh, and A. Tew, "Concept inventories in computer science for the topic discrete mathematics," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR)*, June 2006.

[35] S. Narciss, *Feedback Strategies for Interactive Learning Tasks*. Springer, January 2008, pp. 413—-424.

[36] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2013.

[37] A. Bottcher, K. Schlierkamp, T. Veronika, and D. Zehetmeier, "Teaching abstraction," in *Proc. International Conference on Higher Education Advances (HEAd)*, June 2016.

[38] J. Perrenet and E. Kaasenbrood, "Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective," in *Proc. ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, June 2006.

[39] M. Friebroon, R. Levy, and M. Armoni, "Abstraction and problem solving in a CS curriculum for 4th grade," in *Proc. APSCE Conference on Computational Thinking and STEM Education (CTE-STEM)*, July 2023.

[40] F. E. V. Castro and K. Fisler, "Qualitative analyses of movements between task-level and code-level thinking of novice programmers," in *Proc. ACM Technical Symposium on Computer Science Education (SIGCSE)*, February 2020.

[41] A. Heck, "Using SOWISO to realize interactive mathematical documents for learning, practising, and assessing mathematics," *MSOR Connections*, vol. 15, no. 2, pp. 6–16, 2017, [Last Accessed: December 15th, 2023]. [Online]. Available: https://cloud.sowiso.nl

[42] R. Ziatdinov and J. Valles, "Synthesis of modeling, visualization, and programming in geogebra as an effective approach for teaching and learning STEM topics," *Mathematics*, vol. 10, no. 3, p. 398, January 2022.

[43] H. Keuning, J. Jeuring, and B. Heeren, "Towards a systematic review of automated feedback generation for programming exercise," in *Proc. ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, July 2016.

[44] M. Kölling and J. Rosenberg, "Objects first with Java and BlueJ (seminar session)," *ACM SIGCSE Bulletin*, vol. 32, no. 1, p. 429, March 2000, [Last Accessed: December 15th, 2023]. [Online]. Available: https://www.bluej.org/

[45] B. Goncalves, M. Moraes, B. Teixeira, and L. Leite, "CODEX – learning object to support loop structure teaching," in *Proc. Latin American Conference on Learning Technologies (LACLO)*, October 2018.

[46] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Learning computer science concepts with scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, September 2013.

[47] P. Plaza M., A. Peixoto, E. Sancristobal, M. Castro, M. Blázquez, A. Menacho, F. Loro, G. Carro Fernandez, J. Munoz, M. Munoz, A. Munoz, S. Munoz, C. Munoz, B. Quintana Galera, and A. Lopez-Rey, "Visual block programming languages and their use in educational robotics," in *Proc. IEEE Global Engineering Education Conference (EDUCON)*, April 2020.

[48] L. Begosso, L. R. Begosso, and N. Christ, "An analysis of block-based programming environments for CS1," in *Proc. IEEE Frontiers in Education Conference (FIE)*, October 2020.

[49] H. Nishino, "Ardestan: A visual programming language for arduino," in *Adjunct Proc. ACM Symposium on User Interface Software and Technology (UIST)*, October 2019.

[50] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Transactions on Computing Education*, vol. 24, no. 1, pp. 1–43, March 2024.

[51] M. Forisek, "On the suitability of programming tasks for automated evaluation," *Informatics in Education*, vol. 5, no. 1, pp. 63–76, January 2006.

[52] A. Böttcher and R. Grellner, "Analysing students' problem solving capabilities to support teaching in software development," in *Proc. IEEE Global Engineering Education Conference (EDUCON)*, March 2022.