Teaching CS1 students to adopt a structural view to solve problems - Motivating that abstraction via a Collaborative Activity and Training it via Automated Feedback

Géraldine Brieven
University of Liege
Liege, Belgium
gbrieven@uliege.be

CCS Concepts

• **Applied computing** → *Collaborative learning*; *Computer-assisted instruction*; • **Theory of computation** → **Control primitives**.

Keywords

Abstraction, CS1, Diagrammatic Reasoning, Graphical Loop Invariant, Collaborative Learning, Automated Feedback, Problem Solving

ACM Reference Format:
Géraldine Brieven. 2025. Teaching CS1 students to adopt a structural view to solve problems - Motivating that abstraction via a Collaborative Activity and Training it via Automated Feedback. In *25th Koli Calling International Conference on Computing Education Research (Koli Calling '25), November 11–16, 2025, Koli, Finland*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3769994.3770051

My research is summarized in Figure 1 and then described in detail.

1 Context and Problematic

Developing first-year students' minds to solve problems at different levels of abstraction is both important and challenging. However, students may quickly feel overwhelmed when asked to abstract their view. They prefer focusing on the code that appears more concrete. The development of abstraction skills requires students to see the relevance of these skills, which motivates them to consistently practice designing abstract representations and actively engage with feedback. This is illustrated as "Objectives" in Figure 1.

On the right, the figure also shows that, in our Introduction to Programming (CS1) course, we explicitly teach abstraction by providing a top-down framework where students must solve problems from higher to lower levels of abstraction (see the inverted pyramid in red). This framework is inspired by the PGK hierarchy. Our contribution comes when students must implement a loop to solve a problem. In that case, we added an abstraction level called *"From problem to solution model"* to better bridge the transition from problem (*structural views*) to algorithmic and code levels (*operational views*). At this intermediate level, we ask students to graphically represent the output of the problem and its dependency on the input (i.e., the postcondition). Then, they should generalize it to the solution **under construction**, thus forming a Graphical Loop Invariant (GLI)[1]. It encourages students to visually model the objects and variables that they will manipulate in their loop. Based on this representation, students should determine the steps needed to advance toward the final solution and translate them in code.

2 Solution

The central portion of Figure 1 shows how this top-down approach, based on the Graphical Loop Invariant, can be taught to fulfill the key objectives depicted at the heart of the figure. To *motivate* students to reflect thoughtfully at each level of abstraction (instead of jumping to the code), I created the Collaborative Design and Build (CDB) activity. It simulates professional team dynamics, thereby triggering student's interest. The activity dedicates specific time periods for working at each level of abstraction. The success of the activity hinges on students working together in a structured chain, where each team builds upon and contributes to the success of the others. This fosters student engagement as they realize the impact of their actions on the entire chain. I also created a generic framework highlighting the general components and mechanisms of this activity to facilitate its transposition in other courses. Besides this classroom activity, to enable students to *regularly practice* the Graphical Loop Invariant design and translation into C-instructions, we developed a learning tool called Café 2.0. It supports a semester-long activity in which students solve problems by submitting both a graphical representation of their solution and its implementation. In addition 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. The cost of this automated feedback is that students' solutions are constrained (using a fill-in-the-blank diagram) so that it can be automatically processed.

3 Method and results

My research explored several research questions. Specifically, it addressed:

RQ 1: Does the CDB activity actually motivate students on this top-down approach?
RQ 2: Does CAFÉ make students improve on this approach via its Automated Feedback?
RQ 3: Does the Graphical Loop Invariant help students to find solutions and write code correctly?

To answer these questions, data was collected in our CS1 course where about 100 students are registered every year. I relied on three

[1]The Graphical Loop Invariant is explained in this paper: https://orbi.uliege.be/handle/2268/298149

This work is licensed under a Creative Commons Attribution 4.0 International License.
Koli Calling '25, Koli, Finland
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1599-0/25/11
https://doi.org/10.1145/3769994.3770051

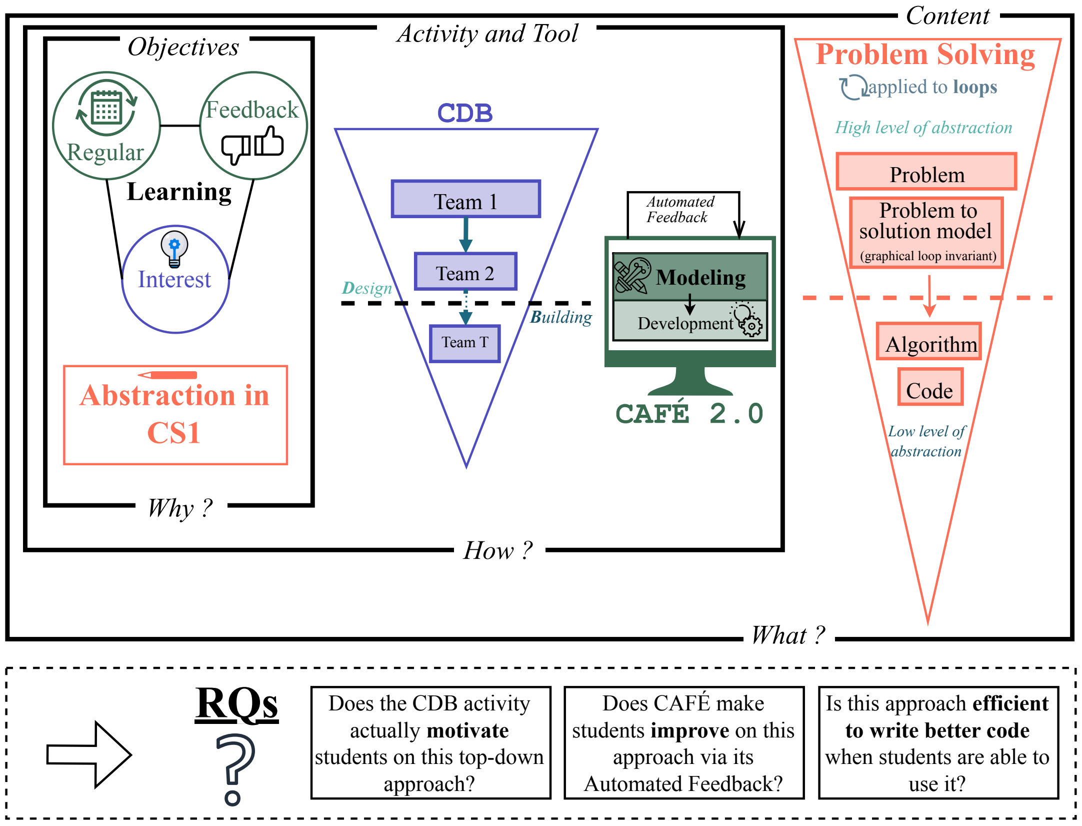
Objectives
Regular
Feedback
Learning
Interest
Abstraction in CS1
Why ?
Activity and Tool
CDB
Team 1
Team 2
Team T
Design
Building
Automated Feedback
Modeling
Development
CAFÉ 2.0
How ?
Content
Problem Solving
applied to loops
High level of abstraction
Problem
Problem to solution model
(graphical loop invariant)
Algorithm
Code
Low level of abstraction
What ?
RQs
Does the CDB activity actually motivate students on this top-down approach?
Does CAFÉ make students improve on this approach via its Automated Feedback?
Is this approach efficient to write better code when students are able to use it?

Figure 1: Connecting Teaching Objectives to CS1 Course Content and Innovative Pedagogical Approaches.

data collection methods: surveys (primarily composed of five-point Likert scale questions), analysis of students' work (GLI and code), namely from an A/B testing setup, and learning analytics (LA) collected through Café 2.0. Student survey responses captured their perceptions, manual analysis of student work measured their performance and revealed students' errors, and LA data provided insights into students' problem-solving behaviors and their processing of automated feedback.

During the CDB activity, I could observe that students actively engage in reflecting on a problem and its solution at each level of abstraction. However, especially in the first session, the quality of students' work is quite limited, especially in the lower levels, due to poor foundations from the previous teams.

In Café 2.0, students tend to oscillate between modeling (GLI) and development. I also noticed that their level of freedom to form their GLI seems too constrained to really prepare them to create GLI from scratch in the exam. Their GLI often lack accuracy or even consistency. In general, many students struggled to understand or represent GLI, especially when they need to textually describe the relationship between the variables. Some students are overwhelmed by the design of the GLI, while others can still extract useful information from their GLI to guide their coding (such as defining their loop guards, etc.).

4 Implication and recommendation

From our results, two recommendations emerged to (better) teach the GLI. We should first only train students to code using provided GLI, focusing solely on translating diagrams into code. Then, we should teach students to sketch their own models by recognizing similarities to previously solved problems. That would be a good preparation to the CDB activity and the exam.

Regarding Café 2.0, we should provide students with greater freedom in creating diagrams while maintaining relevant feedback. To achieve this, we plan to integrate Large Language Models to translate students' natural language input into a constrained format (predicates or sentences with predefined words) that can be processed by our current rule-based system. This approach allows us to maintain control over the feedback provided to students, thus limiting the uncertainty associated with AI usage.