Learning Computer Programming around a CAFÉ

Simon Liénardy
Université de Liège, Montefiore Institute – Belgium
simon.lienardy@uliege.be

CCS CONCEPTS
• Social and professional topics → CS1. • Theory of computation → Algorithm design techniques.

1 MOTIVATION THAT DRIVES THE DISSERTATION RESEARCH

At the University of Liège, the CS1 course has been using, for a couple of years, a programming methodology that consists in determining an informal Loop Invariant prior to any code writing and to use it to deduce the code instructions. This informal Loop Invariant is a graphical representation that depicts key information that will eventually be used to actually write the code. As such, the Graphical Loop Invariant represents a strategy to solve the problem and is used to support thoughts on the code. Although being informal, this drawing must at least detail variables, constant(s), and data structures manipulated by the program; the constrains on them; the relationships they may share, and that are conserved all over the iterations. It should also express, in a general way, what has been already computed by the program after a certain number of loop iterations. While this methodology has been used for a long time, it was never properly assessed, especially in the context of a CS1 course. The main goal of the research is, on one hand, to study the efficiency of the methodology as a way to teach programming in a CS1 course and, on the other hand, to develop tools that will support the teaching of the Graphical Loop Invariant based programming by complementing the other course materials.

Until now, we have already developed a program (called CAFÉ) to automatically test students’ programs. CAFÉ takes into account the Graphical Loop Invariant that was used to build the program code. It also provides student feedback and feedforward information. We also developed a web application (called GLI) that enable students to draw easily Graphical Loop Invariant and obtain feedback about their drawing. We collected various data from both tools usage.

2 LITERATURE REVIEW

Loop Invariant. While there is an abundant literature on Loop Invariants for code correctness and on automatic generation of Loop Invariants (e.g., [6, 7, 12, 16, 17, 30–32]), their usage for building the code has attracted little attention from the research community.

With respect to Loop Invariant based programming (i.e., the Loop Invariant applied in a constructive approach), the seminal work has been proposed by Dijkstra [10], followed by Meyer [23], Gries [15], and Morgan [24]. As such, the program construction becomes a form of problem-solving, and the various control structures are problem-solving techniques. Those works proposed Loop Invariant as logical assertion.

Tam [34] suggests to introduce students to Loop Invariant as early as possible in their courses. Tam describes several examples of code construction based on informal Loop Invariants expressed in natural language. Astrachan [1] is probably the closest to our work as he suggests the use of Graphical Loop Invariants in the context of CS1/CS2 courses. However, his approach is incomplete as the suggested drawing lack of completeness (e.g., objects manipulated, such as arrays, are not named according to code variables), might lead to confusion (e.g., variables positions in the drawing are somewhat unclear), and the drawing is not explicitly manipulated to derive particular situations (e.g., code prior and after the loop). Finally, Back [3, 4] proposed nested diagrams (a kind of state charts) representing, at the same time, the Loop Invariant and the code. However, in such a situation, Loop Invariants are expressed as logical assertions, possibly leading to difficulties to students with a low mathematical and abstraction background. To the best of our knowledge, none of these works evaluate the reception, by students, of a programming methodology based on Loop Invariant.

Following Furia et al. [13] classification, the Graphical Loop Invariant falls within the scope of essential (i.e., a Loop Invariant defining what has already been achieved so far) and bounding Loop Invariant (i.e., variables are bounded by, e.g., an array limits).

More generally speaking, Graphical Loop Invariant based programming falls within the scope of metacognition [22], as it provides a problem-solving strategy and self-reflection on where one is in the problem-solving process. As such, Graphical Loop Invariant based programming can be related to three problem-solving stages introduced by Loksa et al. [21], i.e., search for solutions, evaluate a potential solution, and implement a solution. Also, writing a Graphical Loop Invariant prior to coding should help students in understanding the problem to be solved [8]. This actual impact of the Graphical Loop Invariant on problem understanding will be studied in future works.

Automatic Programs Assessing Tools. Many automated system for providing feedback to programming exercises were already proposed (e.g., [5, 9, 11, 18, 20, 26, 27]). Most of them apply test-based feedback, i.e., student’s code is corrected through unit testing (except UNLOCK [5] that tackles the problem solving skills in general, not just coding skills). WebCAT [11] even makes students write their own tests too. Kumar’s Problets [18] enables step by step code execution as part of feedback. More advanced automatic feedback has been proposed by Singh et al. [33] by providing, to students, a numerical value (the number of required changes) and the suggestion(s) on how to correct the mistake(s).
With respect to metacognition, CAFÉ is an automated assessment tool increasing metacognitive awareness [29], as it relies on Graphical Loop Invariant for building the code to solve programming activities. However, future work should reveal to what extent CAFÉ really helps in improving students’ performance.

3 HYPOTHESIS AND KEY IDEAS

As the Graphical Loop Invariant is concerned, we believe that a graphical approach is simpler to understand for CS1 students [14, 25, 28]. Moreover, a Graphical Loop Invariant can be used to derive the code of a program, like a formal Loop Invariant [10], but it only needs graphical transformations of the Graphical Loop Invariant to deduce, e.g., variables initialization, Loop Condition, etc. This methodology needs a regular exercises [2] to be mastered. Due to human resources constrains, these exercises have to be automatically corrected, suggesting to develop tools to ease the teaching of the Graphical Loop Invariant based programming.

4 RESEARCH APPROACH AND METHOD

CAFÉ. CAFÉ [19] stands for “Automatic Correction and Feedback for Students” (the acronym in French means “Coffee”). This is a system for assessing student’s programs providing them feedback and feed-forward (what could be done to improve their solution) information. CAFÉ differs from previous programs automatic assessing system by allowing students to submit both their code and the Graphical Loop Invariant upon which it was written. CAFÉ performs tests to detect if the Graphical Loop Invariant and the code are matching.

Gli. Gli (standing for Graphical Loop Invariant) is a web application that enables to draw Graphical Loop Invariant easily. The application is able to check whether there are missing elements in the drawing and to precisely indicate what is missing. Of course, it cannot check if a drawing is suited for solving a particular problem but it can give students early feedback on their drawing and help them to enhance their Graphical Loop Invariants quality, hence preventing eventually code conception mistakes. The drawing can also be manipulated to derive particular situations, easing so the code construction.

Data Collection Methodology. Our studies are conducted in accordance to the 3 P’s framework [35] that recommends to consistently assess learners’ experience by gathering and meshing three types of data: Participation data, performance data and perception data. We collected data over the last seven Academic Years.

REFERENCES