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 with 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 cursus. 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 exercices 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).

ICER '20, August 10–12, 2020, Virtual Event, New Zealand

ACM ISBN 978-1-4503-7092-9/20/08.
https://doi.org/10.1145/3372782.3407119

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 extend 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 exercices [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 feedforward (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

[1] Owen Astrachan. 1991. Pictures as invariants. In *Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*. 112–118.

[2] Thomas Austin, Cay Horstmann, and Hien Vu. 2018. Explicit short program practice in a programming languages course. *Journal of Computing Sciences in Colleges* 33, 4 (2018), 114–122.

[3] Ralph-Johan Back. 2006. Invariant Based Programming Revisited. In *Proc. International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (Petri Nets)*.

[4] Ralph-Johan Back. 2009. Invariant based programming: basic approach and teaching experiences. *Formal Aspects of Computing* 21, 3 (2009), 227–244.

[5] Theresa Beaubouef, Richard Lucas, and James Howatt. 2001. The UNLOCK system: enhancing problem solving skills in CS-1 students. *ACM SIGCSE Bulletin* 33, 2 (2001), 43–46.

[6] Aaron R Bradley and Zohar Manna. 2007. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[8] Michelle Craig, Andrew Petersen, and Jennifer Campbell. 2019. Answering the Correct Question. In *Proceedings of the ACM Conference on Global Computing Education*. 72–77.

[9] Guillaume Derval, Anthony Gego, Pierre Reinbold, Benjamin Frantzen, and Peter Van Roy. 2015. Automatic grading of programming exercises in a MOOC using the INGInious platform. *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCSâĂŽ15)*, 86–91.

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

[11] Stephen H Edwards and Manuel A Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*. 328–328.

[12] Jicheng Fu, Farokh B Bastani, and I-Ling Yen. 2008. Automated discovery of loop invariants for high-assurance programs synthesized using AI planning techniques. In *2008 11th IEEE High Assurance Systems Engineering Symposium*. IEEE, 333–342.

[13] Carlo A Furia, Bertrand Meyer, and Sergey Velder. 2014. Loop invariants: Analysis, classification, and examples. *ACM Computing Surveys (CSUR)* 46, 3 (2014), 1–51.

[14] David Ginat. 2004. On novice loop boundaries and range conceptions. *Computer Science Education* 14, 3 (2004), 165–181.

[15] David Gries. 1987. *The Science of Programming*. Springer.

[16] L Kovacs and Tudor Jebelean. 2004. Automated generation of loop invariants by recurrence solving in theorema. In *Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC04)*. Mirton Publisher Timisoara, Romania, 451–464.

[17] Laura Ildiko Kovacs and Tudor Jebelean. 2005. An algorithm for automated generation of invariants for loops with conditionals. In *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*. IEEE, 5–pp.

[18] Amruth N Kumar. 2013. Using problets for problem-solving exercises in introductory C++/Java/C# courses. In *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE, 9–10.

[19] Simon Liénardy, Laurent Leduc, Dominique Verpoorten, and Benoit Donnet. 2020. Café: Automatic Correction and Feedback of Programming Challenges for a CS1 Course. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*. 95–104.

[20] Richard Lobb and Jenny Harlow. 2016. Coderunner: A tool for assessing computer programming skills. *ACM Inroads* 7, 1 (2016), 47–51.

[21] Dastyni Loksa, Andrew 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*. 1449–1461.

[22] Janet Metcalfe, Arthur P Shimamura, et al. 1994. *Metacognition: Knowing about knowing*. MIT press.

[23] Bertrand Meyer. 1980. A Basis for the Constructive Approach to Programming.. In *IFIP Congress*. Citeseer, 293–298.

[24] Carroll Morgan. 1990. *Programming from specifications*. Prentice-Hall.

[25] Linda B Nilson. 2009. *The graphic syllabus and the outcomes map: Communicating your course*. Vol. 137. John Wiley & Sons.

[26] Nick Parlante. 2011. CodingBat: Code Practice. https://codingbat.com [Online; accessed: 30 March 2019].

[27] Pearson. [n. d.]. My Lab Programming. https://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/ [Online; accessed: 30 March 2019].

[28] George Pólya. 1945. *How to Solve It*. Princeton University Press.

[29] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 531–537.

[30] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. Program verification using automatic generation of invariants. In *International Colloquium on Theoretical Aspects of Computing*. Springer, 325–340.

[31] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. 2004. Non-linear loop invariant generation using Gröbner bases. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 318–329.

[32] Peter H Schmitt and Benjamin Weiß. 2007. Inferring Invariants by Symbolic Execution. *VERIFY* 259, 195–210.

[33] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*. 15–26.

[34] Wing C Tam. 1992. Teaching loop invariants to beginners by examples. In *Proceedings of the twenty-third SIGCSE technical symposium on Computer science education*. 92–96.

[35] Dominique Verpoorten, Emmanuelle Parlascino, Marine André, Patricia Schillings, Julie Devyver, Olivier Borsu, Jean-François Van de Poël, and Françoise Jérôme. 2017. *Blended learning-Pedagogical success factors and development methodology*. Report. IFRES-Université de Liège.