10th International Conference on Higher Education Advances (HEAd'24)
Universitat Politècnica de València, València, 2024
DOI: https://doi.org/10.4995/HEAd24.2024.17110

Practicing Abstraction through a Top-Down Problem-Solving Framework in a CS1 Course

Géraldine Brieven, Benoit Donnet
Institut Montefiore, Université de Liège, Belgium.

How to cite: Brieven, G.; Donnet, B. 2024. Practicing Abstraction through a Top-Down Problem-Solving Framework in a CS1 Course. In: 10th International Conference on Higher Education Advances (HEAd'24). Valencia, 18-21 June 2024. https://doi.org/10.4995/HEAd24.2024.17110

Abstract

Mastering abstraction skills is a crucial learning outcome for computer science students. However, students often struggle with these skills and can quickly feel overwhelmed when required to apply them. In our Introduction to Programming (CS1) course, we explicitly teach abstraction by providing a framework where students must solve problems from higher to lower levels of abstraction. This paper presents this framework and demonstrates its application through a collaborative activity that simulates large-scale development projects. This paper also examines students' perception and performance at each level of abstraction over three sessions of the activity. The findings highlight the concepts that should be prioritized to better prepare students for each session. They also emphasize the importance of illustrating to students the function of abstraction in problem-solving and its relevance to their future work life, as a complement to traditional academic activities.

Keywords: *Abstraction; Team-based learning; CDB; CS1; Problem Solving.*

1. Introduction

Problem-solving is a crucial skill in the programming learning process (Medeiros et al., 2019). It relies on abstraction to capture all dimensions of a problem, represent them, and model a corresponding solution (Beecher, 2017). In practice, students entering the Computer Science (CS) program often lack mathematical skills. This can lead to poor abstraction abilities, which can hinder students' success in the CS curriculum (Zehetmeier et al., 2020). Previous studies have shown that, when seeking solutions, students tend to focus on the code rather than exploring higher-level abstractions (Ginat & Blau, 2017). Additionally, students may not be aware of abstraction (Böttcher et al., 2016). As a result, most introduction to programming courses (abbreviated as CS1) focus on the syntax and semantics of the programming language (Malik, 2018).

In our CS1 course, however, we cover both programming language concepts (C language) and abstraction. To develop students' abstraction skills, we teach them a Top-down framework, which involves solving problems at different levels of abstraction. The theoretical and exercise classes provide clear details on these levels. For additional practice, students can participate in the *Collaborative Design and Build activity* (abbreviated here as "CDB") (Brieven et al., 2022). The activity aims to give students co-ownership and emphasize the purpose of each level of abstraction. In CDB, problem-solving is conducted through an assembly-line process where the solution needs to be designed (*Design Phase*) before getting implemented (*Building phase*). Teams of students are required to solve a problem at a given level of abstraction within a specific timeframe. The solution is then passed on to the next team, which takes over at a lower level of abstraction. This paper describes the Top-down framework and its compatibility with CDB.

It also examines students' ability to progress in the problem-solving flow. Previous studies (Brieven et al., 2022) have shown that CDB is motivating, but it also poses additional challenges such as time constraints, communication, and peer feedback. Many students cannot cope with these challenges, mainly due to a lack of knowledge about what is expected of them at the different steps (assimilated to levels of abstraction), leading to poor interactions. To mitigate this, this paper identifies the level of abstraction at which the problem-solving chain is the most fragile, in each CDB session. This allows for better preparation of students for future sessions.

2. Related Work

In Computer Science, Computational Thinking (CT) concepts are permanently mobilized in order to solve problems. These encompass problem decomposition, pattern recognition, algorithmic design, abstraction, data representation, algorithmic thinking, and generalization of patterns, simulation and evaluation (Rey et al., 2020). By proposing the Top-down framework, we frame problem solving in programming as a set of distinct stages, which promotes metacognitive scaffolding (Loksa et al., 2020). We also take up Wing (2006) argument that the essence of CT is abstraction and is of highest importance in a CS1 course (Sprague & Schahczenski, 2002).

Perrenet et al. (2005) define four levels of abstraction, from the highest to the lowest one: (1) *Problem* level; (2) *Object* level; (3) *Program* level; (4) *Execution* level. In this paper, we encapsulate the Problem and the Object levels in the Design phase and the Program and Execution levels in the Building phase of CDB. These Design and Building 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. (2020). They can be also respectively associated to the "design" and "code" levels of abstraction, introduced by Waite (2018). Further, they match with two of the stages of Loksa et al. (2016), where students must translate some abstract representations of a solution into code.

3. Top-down problem-solving framework

Although there is a strong motivation to teach abstraction, it does not come with a universal definition (Mirolo et al., 2021; Zehetmeier et al., 2019). Similarly, there is no standard method for teaching it (Zehetmeier et al., 2019).

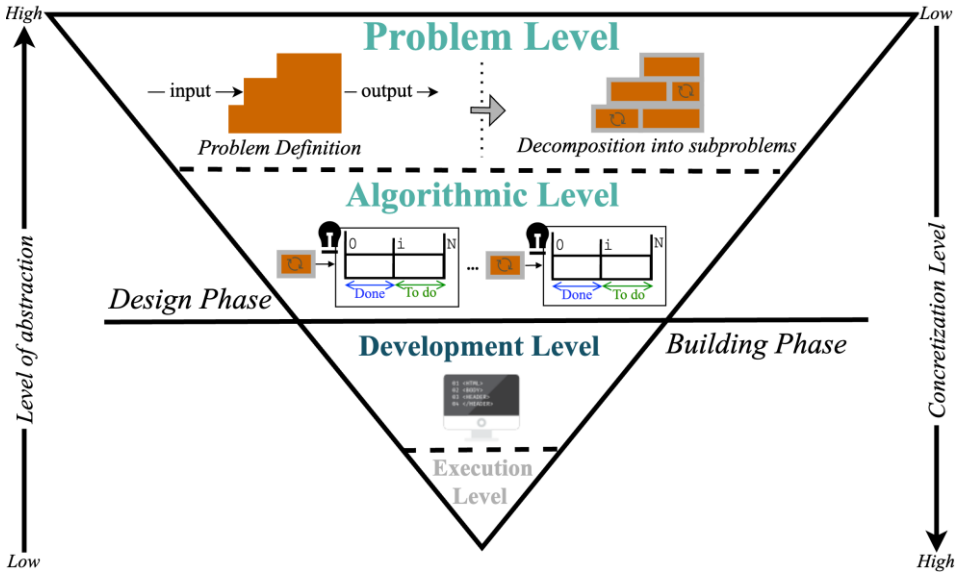

Figure 1. Top-down framework to solve problem.

In our CS1 course, we define the Top-down framework. It consists of approaching a problem from a high perspective (to avoid being overwhelmed by specific problem instances) and approaching the solution step by step, making it more and more concrete. In our course, we rely on four stages, illustrated in Fig. 1, to solve a given problem:

1) *Problem Level.* Students must define the problem by identifying its inputs, its outputs, and the relationships between them by reading a statement. They must then identify the key subproblems and show how they should be articulated.
2) *Algorithmic Level.* For each subproblem whose solution relies on an iterative process, a Graphical Loop Invariant (Brieven et al., 2023) should be drawn. It consists of using drawings to show important relationships between the variables involved in an iterative process while ignoring unnecessary aspects of the problem (Seel, 2011).
3) *Development Level.* Students must write the code (in C programming language) based on the two previous steps.
4) *Execution Level.* The code is tested with respect to the initial problem.

4. Collaborative Design and Build activity (CDB)

The CDB activity (Brieven et al., 2022) is made up of two phases: the *Design* phase and the *Building* phase. Fig. 2 shows how CDB is setup.

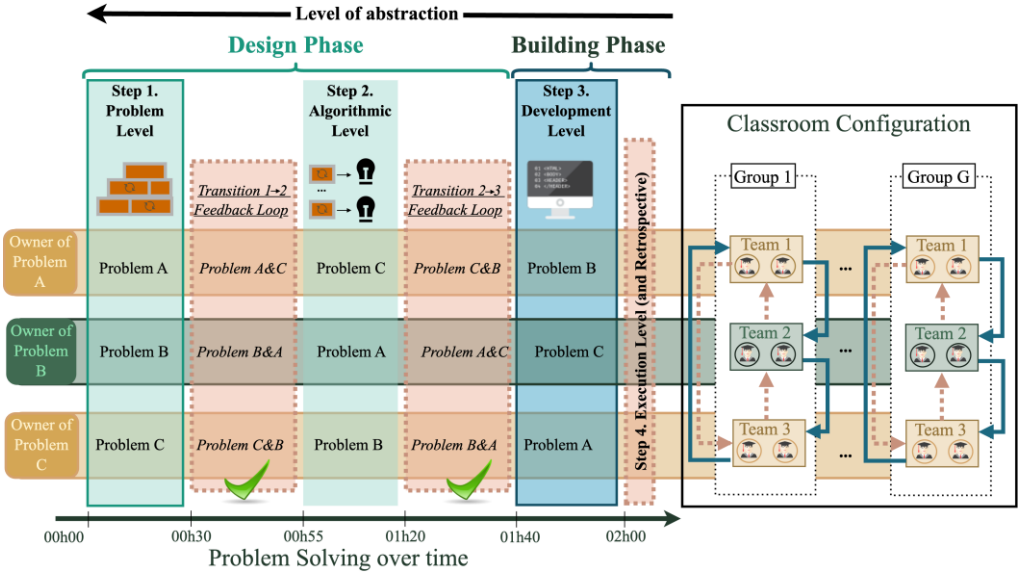

Figure 2. CDB set-up (adapted from (Brieven et al. 2022)). G groups are defined, where G=N/6 and N is the number of participants.

The right side of Figure 2 ("Classroom Configuration") illustrates that the CDB activity is based on G groups of students, each group being divided into three teams (each team comprising S students, where $S \geq 2$). The goal of each group is to solve three problems in a limited amount of time. The left part of Fig. 2 shows how the three problems are solved step by step, in parallel, by standing at one level of abstraction at a time. This approach is inspired by real professional life, as in large development projects, the problem and its solution are handled at different levels of abstraction by different teams. Here, each level of abstraction belongs to the Top-down framework (Fig. 1). The first step is to define and decompose the problem (Problem Level). During the second step, students must model the solution (Algorithmic Level) before moving on to the third step, where the solution is implemented. Once this development step is completed, each team verifies that the final code related to the problem they own correctly answers the initial question. Then, the three teams join together and share their conclusions.

Further, a transition period (Brieven et al., 2023) is dedicated to allow each team to provide feedback on the previous team's work. The objective is to reduce the impact of a "bad work" on subsequent productions that rely on it. Feedback is given using a rubric checklist with the same criteria used to evaluate the midterm and the final exam.

5. Method

During the academic year 2023-2024, 101 students enrolled in our CS1 course. Of these students, 47% had more than four hours per week of Maths in Secondary school. Additionally, 18% of students had some prior programming experience. From a content perspective, our course emphasizes the Top-down framework. Fifteen hours of theoretical and traditional exercise sessions are dedicated to it (roughly half of the course schedule).

During the semester, three CDB sessions were organized, with an increasing level of complexity from session to session. Participation was not mandatory and a team consisted of two or three students. In the first session (*N=62, G=10*), students were tasked with drawing a geometric figure. In the second session (*N=43, G=7*), they were required to print the numbers that met a specific property. In the last session (*N=32, G=5*), they were presented with problems related to managing a store selling different products. The decreasing number of participants (from 62 to 32) is mainly due to the students feeling discouraged after the midterm (which follows the first session). From that point on, they stop taking part in any academic activity. Data was collected during each session. First, a survey was addressed to the students at the end of each session. Then, all the students' productions were collected and their quality was evaluated by a supervisor. The grades vary from 0 to 5, where 0 means that the team has done a poor job, while 5 reflects a completely correct production. The criteria used to assess the productions are those of the checklists that support the transition phases of CDB (Brieven et al., 2023). Moreover, if the previous productions were too poor, only the syntax errors are considered.

6. Results

This section answers the question: "*At which level of abstraction is the problem-solving chain most fragile (for each CDB session)?"***.** To answer this, on the one hand, the quality of what students produced at each level is illustrated (Figures 3 to 5). On the other hand, Figure 6 shows which level of abstraction students found the most difficult at the end of each session.

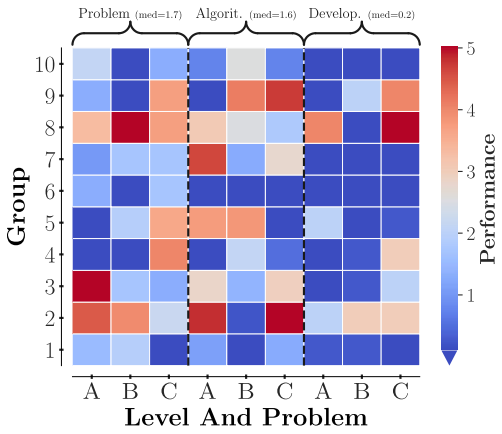

Fig 3. Students' performance during session 1.

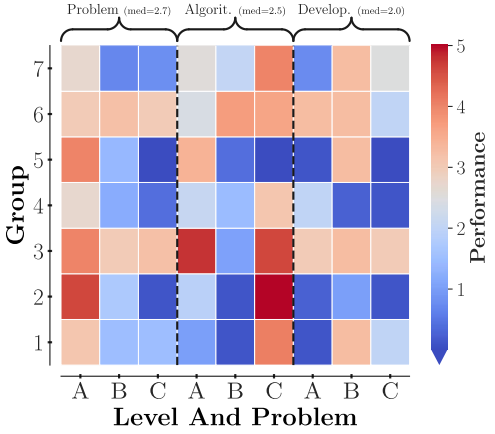

Fig 4. Students' performance during session 2.

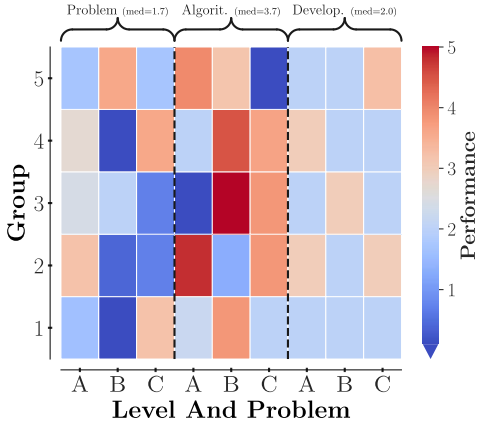

Fig 5. Students' performance during session 3.

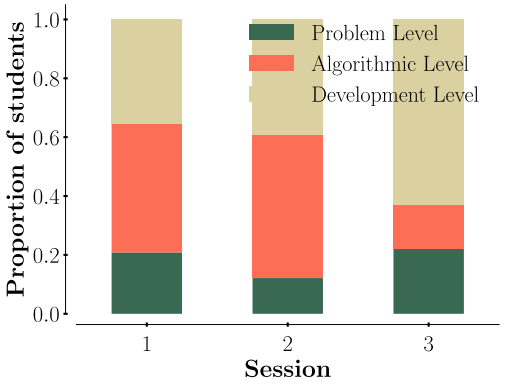

Fig 6. Most difficult level of abstraction for students.

Overall, Figures 3 and 4 show that performance deteriorates across the levels of abstraction (see the median indicated at the top of the figures for a higher level vision). This makes sense because problem solving is cumulative here. This is not the case in the third session (Figure 5), where students were able to design a solution despite a low quality problem analysis. In this last session, students were actually asked to formally define the problem and its subproblems, which was quite new to them. This probably led to a lower performance, but it did not block the next levels because the next students could understand what to do, despite the lack of formalization.

Next, performance improves slightly across sessions (the red color is spreads to the right when we compare Figures 3, 4 (and to a lesser extent, 5)). This confirms previous suggestions (Brieven et al., 2022) that emphasize the importance of running the activity several times to familiarize students with it. Again, the only exception is the last session, where the difficulty of

the statements was increased not only at the problem level, but also at the development level, where students had to use new functions and syntax. This is subtantiated in Figure 6, which shows that this is where the students felt most uncomfortable, while in the first two sessions, the difficulty was upstream (at the Problem and Algorithmic Levels). Finally, contrary to what the performance results may suggest, Figure 6 illustrates that the Algorithmic Level is the one where students struggle the most compared to the Problem Level. This is likely due to the additional difficulty of having to understand and adopt a partial solution at the Algorithmic Level, whereas initiating it at the Problem Level gives more freedom with a clear start.

7. Perspective and Conclusion

In conclusion, this paper presents the Top-down framework, which involves solving a problem from a high to a low level of abstraction. CDB is a great opportunity to practice it because it requires solving a problem step by step to obtain to a final concrete solution. In CDB, the levels of abstraction are defined as sequential steps that form an assembly line. At each step, students must take the solution under construction and think it further only within the level for which they are responsible, abstracting from where it comes from and what the final results should be.

The results show that CDB enhances problem solving through the Top-down approach, as performance increases across sessions. However, the performance remains limited. To mitigate this, it is essential to enhance students' preparation for CDB by revisiting our traditional exercise sessions. Currently, during these sessions, students are provided with a brief theoretical reminder to assist them in solving a series of exercises, with some guidance from a supervisor if they request it. In practice, many students do not engage in these sessions due to a lack of interest in the framework, until they attend the CDB session and realize why abstract reasoning is relevant by facing more complex problems and feeling the group effect. Our new challenge is to create exercise sessions that are closer to CDB. We should propose easier tasks, but still encapsulate them in time and maintain peer feedback for social motivation. In particular, our results show that students should be more familiar with the Problem and Algorithmic levels before the first session, while we should emphasize more the Algorithmic level before the second session. Finally, before the last session, students should practice more the problem formalization as well as the new function calls they are expected to make.

References

Beecher, K. (2017). *Computational Thinking: A Beginner's Guide to Problem-Solving and Programming*. BCS, The Chartered Institute for IT.

Böttcher, A., Schlierkamp, K., Veronika, T., & Zehetmeier, D. (2016). Teaching Abstraction. In *Proc. International Conference on Higher Education Advances (HEAd)*. doi: 10.4995/HEAD16.2016.2770.

Brieven, G., Liénardy, S., Malcev, L., & Donnet, B. (2023). Graphical Loop Invariant Based Programming. In *Proceedings of Formal Methods Teaching Workshop (FMTea)*. Springer.

Brieven, G., Leduc, L., and Donnet, B. (2022). Collaborative Design and Build Activity in a CS1 Course: A Practical Experience Report. In *8th International Conference on Higher Education Advances (HEAd).* doi: 10.4995/HEAd22.2022.14386.

Brieven, G., Leduc, L., and Donnet, B. (2023). How Students Manage Peer Feedback through a Collaborative Activity in a CS1 Course. In *9th International Conference on Higher Education Advances (HEAd)* . doi: 10.4995/ HEAd23.2023.16142.

Ginat, D., & Blau, Y. (2017). Multiple Levels of Abstraction in Algorithmic Problem Solving. In *Proc. ACM Technical Symposium on Computer Science Education (SIGCSE)*. doi: 10.1145/3017680.3017801.

Loksa, D., Ko, A., Jernigan, W., Oleson, A., Mendez, C., & Burnett, M. (2016). Programming, Problem Solving, and Self-Awareness: effects of Explicit Guidance. In *Proc. CHI Conference on Human Factors in Computing Systems*. doi: 10.1145/2858036.2858252.

Malik, S. (2018). Improvements in Introductory Programming Course: Action Research Insights and Outcomes. *Systemic Practice and Action Research, 31(6), 637–656.*

Medeiros, R., Ramalho, G., & Falcao, T. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education, 62(2), 77–90.* doi: 10.1109/TE.2018.2864133.

Mirolo, C., Izu, C., Lonati, V., & Scapin, E. (2021). Abstraction in Computer Science Education: An Overview. *Informatics in Education, 20(4), 615-639.* doi: 10.15388/infedu.2021.27.

Perrenet, J., Groote, J., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin, 3(37), 64–68*. doi: 10.1145/1151954.1067467.

Rey, Y., Nissandra Cawanga Cambinda, I., Deco, C., Bender, C., Avello-Martinez, R., & Villalba-Condori, K. (2020). Developing Computational Thinking with a Module of Solved Problems. *Computer Applications in Engineering Education, 29(3), 506–516.* doi: 10.1002/cae.22214.

Seel, N. (2011). *Encyclopedia of the Sciences of Learning*. Springer Verlag.

Sprague, P., & Schahczenski, C. (2002). Abstraction the key to CS1. *Journal of Computing Sciences in Colleges, 17(3), 211–218.* doi: 10.5555/772636.772671.

Waite, J., Curzon, P., Marsh, W., Sentance, S., & Hadwen-Bennett, A. (2018). Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools (IJCSES), 2(1), 14–40.* doi: 10.21585/ijcses.v2i1.

Wing, J. (2006). Computational Thinking. *Communications of the ACM, 49(3), 33-35.*

Zehetmeier, D., Böttcher, A., Brüggemann-Klein, A., & Thurner, V. (2019). Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction. In *Proc. Hawaii International Conference on System Sciences (HICSS)*. http://hdl.handle.net/10125/60202.

Zehetmeier, D., Böttcher, A., Thurner, V., & Brüggemann-Klein, A. (2020). A Concept for Addressing Abstract Thinking Competence While Teaching Software Development. In *Proc. IEEE Global Engineering Education Conference (EDUCON)*. doi : 10.1109/EDUCON45650.2020.9125128.