Research on the Role of Abstraction in Computer Programming

Alexandre Lepage[1], Lucie DeBlois[1], Margarida Romero[2]
[1] *Université Laval, Canada*
[2] *Université Nice Sophia Antipolis, France*

Abstract

Nowadays, a growing number of schools are trying to introduce computer programming into their learning activities. It is presented as a mean of engagement throughout the learning process. This research is an ongoing attempt to better understand what characterizes computer programming, considered as a human activity, with a special focus on the role it may or may not play in the development of abstraction. A total of 19 subjects, each with a minimum of five years of cumulated programming experience, have accepted to take part in an individual 45-minute semi-structured interview which includes three themes: their background related to computer programming, their perception on the cognitive and affective components of this activity, and their perception on computer programming lessons for everybody from childhood. Here we propose a preliminary analysis of one of these interviews. In the end this research will help clarifying cognitive and affective components involved in computer programming.

1. Computer programming

In the 1960's, computer programming was mostly defined following its relationship with computer science. Hoare defined computer programming as an "exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning" [3].

Martin-Löf considers that programming is very close to constructive mathematics. The existence of any object – i.e. a computer program – must come along with a mean of creating it [9]. Computer programming is a way of creating objects. He also made a clear distinction between low-level programming languages such as assembly and high-level programming languages such as Fortran, saying that the shift from low-level to high-level programming results in different ways of expressing thoughts in code. High-level programming is a form of computer programming closer to human thinking: "[a high-level language is] a language in which the thought of the programmer can be expressed without too much distortion and understood by someone who knows very little about the structure of the hardware, but does know some English and mathematics" [9](p. 501). Hoc, Green, Samurçay, and Gilmore suggested that computer programs are more than just sets of instructions written for a compiler and that they are also "vehicles for expressing our own thought to ourselves" [4](p. 42).

Weinberg defined primarily a computer program by its ability to produce a correct output for a set of possible inputs [16]. We consider computer programming as a (co)creative activity in which the analysis of the initial situation is performed before engaging in an iterative approach of creative problem solving requiring an amount of computer literacy [13]. Within this approach, we situate computer programming as a social, cultural and technological process which requires both specific knowledge and transversal competencies to be deployed.

2. Computer programming concept in this study

For this exploratory research aiming to identify the participants' conceptions of computer programming, we reject none of the previous definitions. However, we find of critical importance to put some restrictions to the scope of computer programming.

First, it must involve the expression of a form of code. It may be by typing it on a keyboard, or by assembling blocks (i.e. visual programming). Second, computer programming requires a computer or an electronic device sufficiently advanced to allow the interpretation or compilation of code. Unplugged computer science activities, even if they may surely find their place in the learning process, are not considered as computer programming because they do not involve the constraints given by a machine. Finally, computer programming must involve the expression of code in a way that delays its execution or conditions its output. Following these two last criteria, turning on a light with a switch is not considered as computer programming because the result of the action is immediate. However, writing a line of code that turns a light on 10 seconds later would be considered as computer programming (event if at the edge).

3. From computer programming to computational thinking

Computational thinking is a concept brought by Wing [17]. To her the core of computational thinking is abstraction. She developed the idea by advocating that a computer scientist is someone who must handle multiple layers of abstraction simultaneously [18]. This paper has been widely referred to, and computational thinking is a concept used in many researches. However there is no consensus on what are the boundaries of such a competency [14]. For some it is strictly restricted to algorithmic skills whereas others include in computational thinking components such as analysis, design thinking, and metacognition [5][6][13].

4. Computer programming and abstraction

Abstraction is to Piaget a major component of what he called formal reasoning. His first work considered formal reasoning as the ability to build hypothesis and develop thinking upon possibilities that are not immediately linked to objects from the physical environment [12]. It is also related to the conceptual thinking of Vygotsky [15]. Even if both theories are close on the nature of reasoning, they partially agree on the way it is acquired. For Piaget, it is acquired through interaction between a subject and various objects upon which the subject builds abstraction whereas for Vygotsky it also relies on the social context.

Papert came to consider computer programming as something not only formal saying that it may relies on both abstract and concrete thinking [10]. Current researches tend to go in the same way. In a study about copy and paste practices, Kim, Bergman, Lau, and Notkin found out that programmers tend to work in a very concrete way in some contexts (i.e. creating a new class by copying a previous one that does about the same thing and then trying to adapt it to fit the new case – instead of doing an analysis before starting to write a new code) [7]. Some of their participants have said that the process of programming acts as a companion throughout the formal analysis task that the programmer must accomplish. Some of them have stated that they discover the appropriate level of abstraction as they program [7]. Moreover, to be more efficient, some programmers tend to avoid whenever possible to develop a formal understanding of the code they are working on [8]. To understand the process of debugging, Pea, Soloway and Spohrer found that trials and errors are a widely used strategy by programmers [11]. Such findings tend to confirm the idea of Papert that programming is not only a cognitive activity. We could hypothesis that these trials and errors are related to socio-affective components. But then the question seems to remain open: what is computer programming?

This project aims at better understanding the nature of computer programming by investigating the potential competency development that may occur when practicing it. The research question is about the boundaries of such a competency and the possibility of it to be deployed in contexts other than those involving computer programming, with a focus on the role of abstraction in it. The research question goes as follow: what could be the boundaries of a competency trained through computer programming? Based on our researches, we came to classify these boundaries under three categories. The first is about the cognitive work involved in computer programming. Under this theme will fit anything related to abstraction, mathematics, logical thinking, algorithmic, and problem solving. The second category is about emotional or social involvements of computer programming. That may comprise anything related to professional relationships, artistic aspect of code, or self-motivation. Finally, the third category is about the different contexts in which computer programming occurs. To explore these boundaries, we asked computer programmers how they perceive their competencies by conducting semi-structured interviews.

5. Method

This study consisted in interviews with experienced programmers. Here we detail who were the participants, what was the procedure, and how is the analysis performed.

5.1 Participants

Nineteen participants were recruited through the mailing list of Université Laval in Quebec City (both from the students list and the employees list). All of them were over 18 years of age and cumulated at least five years of experience in computer programming.

5.2 Procedure

Participants were asked to participate in an audio-recorded 45-minute semi-structured interview. Then, interviews have been transcribed.

The interview is built upon three themes. The first theme is about the background in computer programming with three subquestions where participants are asked about (1) their first steps in computer programming, (2) their motivations to do computer programming – and changes in time, if any –, and (3) the nature of what they do (or have done) with a technical perspective (ie. programming languages they use). The first theme is mostly an open discussion in which the participant describes his background. As the discussion goes on, the interviewer asks clarification when required, and tries to propose synthesis of the participant speech to validate the mutual understanding of the speech.

The respective roles of the participant and the interviewer are about the same in the second and third themes, but in these cases the participant is asked to react to statements proposed by the interviewer. The participant cannot see statements in advance. All the participants receive statements in the same order. The procedure goes as follow: the interviewer reads the statement, gives a printed version to the participant, and then the participant starts discussing about it. He may or may not agree with the entire statement, may or may not choose to divide the statement to react to parts of it. Statements were selected prior to the interviews and organized with the intention to bring the participant to clarify his speech by anticipating possible contradiction based on existing literature. For example, in the second theme, the first statement is “Everybody can do computer programming”. The third statement goes as follow: “It takes some intellectual capacities of analyze to be a good programmer. One must not be afraid of mathematics and logic, and be ready to face frustrations. Not everybody has these skills or has the patience required to developed them, so computer programming is not for everybody.”. Both statements are trying to investigate the same generic idea about the ability of anybody to program. But in an exploratory perspective, the first statement provides the participant with the chance to bring by himself any nuance he wants as he is free of influences. The third statement may or may not fit his speech about the first statement, but in any case, it brings the participant to clarify his thinking and thus provide the interviewer with a richer understanding of the participant’s speech.

5.3 Analysis

At this stage the analysis is still going on. We will present in this paper the qualitative analysis of the first participant to show the kind of information it may bring and how we intend to relate it to the research question. The qualitative analysis is performed through the software QDA Miner.

6. Preliminary analysis

We propose the descriptive analysis of the first participant through five major ideas from the three categories presented earlier: contexts for computer programming, cognitive work, and socio-affective components. Because the interview was conducted in French, every quote from the participant’s speech was translated to English for this paper.

6.1. Contexts in which computer programming occurs

6.1.1 First contact with computer programming. Participant #1 started programming in the late 1970's while he was between 16 and 18 years old. His first memory of computer programming involved a TRS-80 computer that he described as follow: "It was a TRS-80... color... 4K of memory... it was in VC, programming language, with small tapes on which we saved... That was the technology of that time, and then I had a computer with 256K of RAM, 2 floppy drives 5 ¼, it was... worthed $3000 in 1994..." (from participant #1). He related a TV channel on which it was possible to play games like the hangman game. He remembered having created a hangman game using the TRS-80 for his mother and other games for his friends. These experiences were rewarding to him: "It was more for my friends than for me! My fun was really to program and I enjoyed seeing others using my... it was self-rewarding to see others using my programs" (participant #1). It was his first reason for programming. He also related as a reason to program the fun he had to control an automat: "Hey, I can also enter lines of code to make a system react, an automat, it's amazing" (participant #1). The participant then completed a bachelor's degree in computer science in which he discovered analysis, which will be discussed later.

6.1.2 Computer programming for everybody from elementary school. The participant agrees on the idea of introducing computer programming in elementary school in order to allow everybody to explore and be aware of that field: "In first grade [the child] may not understand all the consequences of not doing computer programming but at least to make them understand that it exists, it may open more doors if they are aware of it" (participant #1). The participant supported this idea not only for career-related purposes, but also for personal development purposes. He says that computer programming may empower a person to act on its environment: "Because programming gives a certain control over tools. So if you are provided with a tool and you like it, you can spend your life without programming. But if you are aware of computer programming, someday you can say 'yes, I could change that tool by myself to better fit my needs" (participant #1). The participant said that even if everybody is capable of computer programming, it may happen that some people do not perceive themselves as so: "Again I repeat what I said earlier, we build ourselves a universe, and so if someone thinks 'I am not capable of being a programmer', if he his convinced of that, I couldn't change him" (participant #1).

In conclusion, these results about the context in which computer programming occurs suggest the following boundary: the context in which programming is done must allow to experience with the control of an automat and must empower the programmer over its environment.

6.2. Cognitive work

6.2.1. Abstraction. Participant #1 related that abstraction is necessary in many ways in computer programming. He often related the need for abstraction when exchanging with a customer while specifying that the term *customer* is not be exclusively understood as a business customer. It may be understood as "somebody else" (participant #1). Even though the participant referred to it as abstraction, we consider it more like an object of communication than a real abstraction. These objects of communication acts as tools to allow a mutual understanding between this customer and the programmer: "It gives a level at which you can exchange with the customer" (participant #1). This abstraction built upon the needs for communication does not exist when programming for yourself according to the participant.

The participant recognizes that the term *abstraction* may have a particular meaning in computer programming, and he named object-oriented programming. He gave as example the modeling of an object: "A student comes to ask help: what do you need to know about him? Oh, all his email addresses, his phone numbers. So when you design an entity 'Person' with properties requested by the person, you design a universe like so" (participant #1). The participant related that the levels of abstraction used to create that universe of objects are essential to perceive some relations between objects: "There is a level of abstraction that you must develop. The more you understand relations, that your object inherits another, so the more you can organize your universe. You win on the long term" (participant #1). Participant #1 said that he was not capable of that level of abstraction when he started computer programming.

We retain of his conception of abstraction that abstraction is a cognitive process with two purposes in computer programming: developing a mutual understanding between the programmer and somebody else, and helping the construction of an object like a complex computer program. These complex programs involve the manipulation of relations between entities that only appear on upper abstractions of these entities.

6.2.2. Analysis and programming. The words of participant #1 about abstraction were related to the ones he had about analysis in computer science. To him, computer programming and analysis are two separate tasks. He gave as examples some places where he worked in which these two tasks were distinct: "It happens often that I will be the analyst and I won't program at all, I will do all the plans, the design, I will work with a designer for user interfaces, we will give it to the programmer and... he programs... so yes, the programmer has a certain flexibility, he can invent stuff but he must not go out of the universe that we built... or if he wants to, he must ask us before, because he often does not have the global view with all other systems because yes we want to let him... we want to involve him in the creation but..." (participant #1). Through these words the participant opened the door to an imperfect frontier between programming and analysis, while stating that they are two separate tasks. He related having filled the two roles simultaneously in some occasions. He gave as example a personal experience where he had to do analysis, programming and project management.

In conclusion, these results about the cognitive work suggest the following boundary: computer programming may be a way mean of creating objects used for communication between people who may or may not be programmers.

6.3. Socio-affective category

6.3.1 Trial and error. Participant #1 recognized a place for trial and error in computer programming. He said that analysis is sometimes less efficient than trial and error depending on the problem to solve: "But I agree that trial and error, if it is faster. Sometimes we get lost in analysis while... you could do only two or three quick tests and the problem is solved. I completely agree on that" (participant #1). The participant draws a comparison between trial and error in computer programming and woodworking. He gave the example of a carpenter who wants to build a table, saying that he would not spend a long time doing the plan. He admitted having used *bricolage* (by an approximate translation we define it as a set of concrete methods sometimes used by amateurs but not only, for example collapsing two programs together to make one, approximate a value instead of thinking to a formula, ...) when he started computer programming and then discovered analysis. It appeared to him as a way to put a distance between himself and his programs: "Yes, when I build a system for somebody else, if I *bricole* [use concrete methods], for sure I would become part of the system like in the expression “having an arm stucked in the machine”, and they will have to keep me because it won't be a solution independant from the person who made it" (participant #1).

In conclusion, these results about the socio-affective category suggest the following boundary: the analysis task may be used consciously by a programmer to distance him from his creation. It causes the programmer to develop a confidence about the place of computer programming in society.

7. Discussion

The analysis of the first interview makes us confident that the different themes give place to a meaningful discussion about the research question that goes as follow: what could be the boundaries of a competency trained through computer programming? Existing literature brought us to focus on the role of abstraction to answer the question, but also to consider the socio-affective components or the contexts in which programming occurs.

Participant #1's speech offered a view on the influence of contexts. In these various contexts, programming seems to be a way for him to act consciously on its environment. In that it may be an empowering mean of action. The participant gave examples from both his personal life and professional life. That leads us to think that programming is more than a career. Some properties of programming seem to be specific to certain contexts and in that it may be appropriate to go further in the distinction we make between professional and personal programming. For example, in a professional setting, the participant had sometimes completely different roles in the organization. Sometimes these roles were really close to his personal interests; sometimes they were more distant from his personal interests. The engagement was so depending from the context in which computer programming is performed. That difference in personal engagement may be better explained by motivational theories.

The speech of Participant #1 about cognitive work involved in computer programming shows a consciousness of how and when abstraction is performed. Abstraction appears sometimes as a choice, suggesting that it is one of the many strategies that may be deployed to solve a problem. In addition, the participant said that he developed new capacities of abstraction when at university, years after his first contact with computer programming. In addition, the pleasure of creating an object to use as a mean of communication is related with the work of Douady in mathematics education about the process of mathematic concept as an object or a tool [2]. These ideas make us consider that more investigation is required to understand the role of abstraction in computer programming. It suggests that abstraction is maybe not the only component at the core of a competency developped by computer programming as stated by Wing [18]. It could be more related with the socio-affective category.

The words of participant #1 about trial and error tend to fit the ideas of Papert about concrete methods [10] and other findings about the role of analysis in computer programming [7][11].

We summarize the position of the participant about computer programming from elementary school by saying that he is in favor of it in an exploratory perspective. He also underlined the importance of the self-perception of an individual to explain the capacity or incapacity to do computer programming.

8. Conclusion

The preliminary analysis of one participant brought some ideas that may contribute to answer the research question. The research question was about the boundaries of a competency trained through computer programming. We divided our analysis in three categories: contexts for computer programming, cognitive work, and socio-affective components. It opens a frame of analysis for computer programming that is not only based on the immediate task of writing code. Results suggest that the task of programming and the way it is performed is influenced by the creativity of the programmer and his relationship with knowledge [1]. By investigating the socio-affective components and the contexts for computer programming in addition to the cognitive work, we think we may offer a wider view of what is computer programming today and how it may be integrated in schools.

9. References

[1] Charlot, B. (2003). “La problématique du rapport au savoir” In Rapport au savoir et didactiques, S. Maury & M. Caillot (eds), Éditions Fabert, Paris, 2003, pp. 33-50.

[2] Douady, R. “Jeux de cadre et dialectique outil-objet dans l'enseignement des mathématiques - une réalisation dans tout le cursus primaire”, Thèse d'État, Université Paris 7, 1984.

[3] Hoare, C. A. R. “An axiomatic basis for computer programming”. Communications of the ACM, 12(10), https://doi.org/10.1145/363235.363259, 1969, pp. 576–580.

[4] Hoc, J.-M., Green, T. R. G., Samurçay, R., & Gilmore, J. D. Psychology of Programming, Academic Press Inc., San Diego, 1990.

[5] Howland, K., Good, J., & Nicholson, K. “Concrete Thoughts on Abstraction”, Psychology of Programming Workshop (PPIG 2009), 2009, pp. 715–728.

[6] Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. "Computational Thinking Patterns”, 2011 Annual Meeting of the American Educational Research Association (AERA), 2, https://doi.org/10.1098/rsta.2008.0118, 2011.

[7] Kim, M., Bergman, L., Lau, T., & Notkin, D. “An ethnographic study of copy and paste programming practices in OOPL”, Proceedings of the 2004 International Symposium on Empirical Software Engineering, https://doi.org/10.1109/ISESE.2004. 1334896, 2004, pp. 83-92.

[8] Maalej, W., Tiarks, R., Roehm, T., & Koschke, R. “On the Comprehension of Program Comprehension”, ACM Transactions on Software Engineering and Methodology, 23(4), https://doi.org/10.1145/2622669, 2014, pp. 31:1-31:38.

[9] Martin-Lof, P., & Lozinski, Z. A. “Constructive Mathematics and Computer Programming”, Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences, 312(1522), https://doi.org/10.1098/rsta.2005.1579, 1984, pp. 501–518.

[10] Papert, S. (1992). The Children’s Machine. BasicBooks, New York, 1992.

[11] Pea, R. D., Soloway, E., & Spohrer, J. C. “The buggy path to the development of programming expertise”, Focus on Learning Problems in Mathematics, 9(1), 1987, pp. 5–30.

[12] Piaget, J. La psychologie de l’intelligence. http://dx.doi.org/10.4324/9780203278895, 1947.

[13] Romero, M., Lepage, A., & Lille, B. “Computational thinking development through creative programming in higher education”, International Journal of Educational Technology in Higher Education, 14(1). https://doi.org/10.1186/s41239-017-0080-z, 2017.

[14] Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. “Computational thinking in compulsory education: Towards an agenda for research and practice”, Education and Information Technologies, 20, https://doi.org/10.1007/s10639-015-9412-6, 2015, pp. 715–728.

[15] Vygotsky, L. Thought and Language, The MIT Press, Cambridge, 1986.

[16] Weinberg, G. M. “What makes a good program?” In The Psychology of Computer Programming, Dorset House Publishing, New York, 1998, pp. 15–26.

[17] Wing, J. M. Computational Thinking. Communications of the ACM, 49(3), https://doi.org/10.1145/1118178.1118215, 2006, pp. 33–35.

[18] Wing, J. M. “Computational thinking and thinking about computing”, Philosophical Transactions of the Royal Society A – Mathematical Physical and Engineering Sciences, 366(July), https://doi.org/ 10.1098/rsta.2008.0118, 2008, pp. 3717–3725.