Submitted for: WiPSCE 2012, Hamburg, Germany, November 8-9

Promoting Computational Thinking with Programming

Cynthia C. Selby
University of Southampton
Highfield
Southampton UK
44 (0) 2380 593475
C.Selby@soton.ac.uk

ABSTRACT

The term computational thinking has received some discussion in the field of computer science education research. The term is defined as the concept of thinking about problems in a way that can be implemented in a computing device. Of course, after having thought about a problem using computational thinking skills, the next step should be to use programming skills to implement the solution. This work in progress is exploring ways in which programming can be employed as a tool to teach computational thinking and problem solving. Data is collected from teachers, academics, and professionals from various industries. They are purposively selected because of their knowledge of or interest in the topics of problem solving, computational thinking, and the teaching of programming. This data is analyzed within the paradigm of the grounded theory approach. The results of an initial analysis imply an ordering of complexity associated with computational thinking skills, imply connections between computational thinking skills and programming activities, and imply a relationship between computational thinking skills and other taxonomies of learning.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computers and Education, Curriculum

General Terms

Design, Experimentation, Theory

Keywords

Computational thinking, pedagogy of programming, problem solving

1. INTRODUCTION

Problem solving skills are used in developing or implementing strategies to solve problems in many domains. These skills are often expressed as heuristics [12], appropriate and plausible approaches to a problem. Effective problem solving has been promoted by the use of strategies including means-ends analysis, schema acquisition, algorithmic approaches, and targeted frameworks [19, 12, 9].

However, in the domain of computer science, some research [11] has found that learners do not naturally solve problems in ways that can be translated to computing devices by the use of programming. This disparity is highlighted in the Lister study [6], where it is suggested that ineffective problem solving skills, including the ability to work through lines of logic, may be the cause of ineffective programming skills. Additional studies [15, 13] suggest that problems in learning to program are exacerbated by a lack of strategic tools.

The strategic tools, identified as useful for those attempting to solve problems with the aid of computational devices in various domains, include, but are not limited to, decomposition, abstraction, simulation, and generalization [10]. The name given to these specialized mental skills, resulting in solutions to problems directly translatable to a computing device, is computational thinking [21]. Actually implementing these solutions requires a different set of skills.

Programming skills are the specific technical skills needed to produce specific solutions using a set of defined digital tools, often associated with a programming language [9]. Research often reports that learners struggle with programming skills such as tracing [3, 6] and understanding a model of the machine [2, 9]. Having recognized this issue, other researchers [7, 20] highlight the need for a defined hierarchy of programming skills. One study [16] attempted to provide such a hierarchy for object-oriented programming. The researchers found that teachers interpreted the hierarchy as a capability hierarchy.

Given that a hierarchy of generic programming skills could be developed and interpreted as capability, the levels could be aligned with existing hierarchies, such as the cognitive domain of Bloom's Taxonomy. These same programming skills could be mapped to the higher-level computational thinking skills that they evidence, thereby defining a hierarchy of computational thinking skills. This setting provides the context for an ongoing investigation into the relationship between the teaching of programming and its effect on the acquisition of computational thinking skills by learners.

2. STUDY METHOD

This study is based on a grounded theory approach employing qualitative data collection methods and qualitative data analysis techniques. The first activity is the administration of an Internet based questionnaire. The second activity is the collection of data from an Internet based community of practice forum. The third activity is the administration of a face-to-face, audio recorded, semi-structured interview schedule for respondents previously identified by an analysis of the questionnaire results and community of practice discussions. All data is iteratively augmented and analyzed guided by Strauss and Corbin's [18] grounded theory procedures and techniques until theoretical saturation. It is anticipated that a product of the theory generation may be a model of relationships between problem solving skills, computational thinking skills, and programming skills.

2.1 Participants and Sampling

The participants in this research all have some interest in the teaching of programming, computational thinking, problem solving, or any combination of the three. Not all participants are teachers. Participants may be employed in industries where computational thinking skills and programming skills are useful or required. Other participants may be members of professional communities of practice, representing industry, academia, or education. They are still perceived to have an interest in and appropriate knowledge of the research context.

An individual participant may not engage with every data collection instrument. Participants are matched to instruments. In the case of the first instrument, an online questionnaire, the targeted sample consists of members of organizations whose ideologies promote the teaching of programming or computational thinking skills. In the case of the second, the online community of practice, conversation threads are chosen purposively for their applicability to the context of this research, without regard to the identity of the poster. From the questionnaire responses and the community of practice conversations, a further purposive selection is made to identify participants for the interviews. This purposive sampling is supported by Strauss and Corbin who affirm that theoretical sampling is a foundation stone of grounded theory which, "… enables the researcher to choose those avenues of sampling that can bring about the greatest theoretical return" ([18], p. 202).

2.2 Data Collection

The questionnaire and interview schedule have been designed specifically to elicit responses applicable to the topics of problem solving, computational thinking, and the teaching of programming. To ensure the same level of appropriateness of response, a set of keyword criteria has been developed on which the community of practice messages are searched.

2.2.1 Online Questionnaires

The questionnaire makes use of some closed questions but the majority of questions are open-ended to allow participants to respond as they wish. The ordering of the questions is from general to specific, divided into major sections. Results are submitted one screen or page at a time. In this way, even the results of abandoned questionnaires have the potential to be used. Personal information is requested early in the response process to identify participants. This provides a mechanism for contacting the participant, should he or she be selected for an interview. The design of the resulting questionnaire aims to be as open as possible to facilitate depth of response, while controlling for researcher and question bias.

2.2.2 Community of Practice

The community of practice, whose discussions and opinions are of interest in this study, is computer-mediated. Simply by contributing, the members signify some interest in the topics that overlap with this study. Although computer-mediated, some individuals share collaborative practices in the classroom. There are also face-to-face meetings, of varying scale, held throughout the year.

In order to identify the most appropriate threads for inclusion in the dataset, discussions are keyword searched. The keywords have been chosen to correspond to the terminology used in the initial research literature and early questionnaire responses. These terms include computational thinking, abstraction, decomposition, algorithm, and problem solving. Discussions, composed of individual and related messages, are considered as a whole [17]. Regardless of the age of a discussion, once it has been identified as pertinent, every individual message in that discussion is read and coded, in line with the questionnaire and interview data.

2.2.3 Interviews

The design of the interview schedule used in this research is based on a semi-structured approach, as defined by Cohen, Manion, and Morrison [1]. In particular, the question wording and sequences are specified in advance of the interview. The interviewer is granted the flexibility to provide additional questions in order to elicit greater depth in the responses. The interviewer is also granted the flexibility to record non-verbal indicators, such as body language or gestures. This semi-structured approach should provide sufficient control to ensure comparability of results, sufficient flexibility to ensure depth of responses, and sufficient consistency to support the simultaneous collection and analysis of data indicated by the grounded theory paradigm.

3. FINDINGS

The current, non-saturated dataset is being analyzed in line with grounded theory, first as conceptual free nodes, then as categories. These categories and concepts may change, as more data is added and processed. Three of the categories presented here, problem solving skills, computational thinking skills, and programming skills have been introduced above.

3.1 Problem Solving Skills

In the context of this study, problem solving skills are not specific to programming, but are a wider skill set applicable in many domains. Participants have highlighted problem understanding and persistence as important concepts in this category. Analysis of the data indicates that a common key first step in both learning to solve problems and learning to program is being able to understand the problem and its constraints. This observation agrees with Pólya's problem solving approach [12]. The theme of persistence is often linked with the idea of "not giving up". Puzzles and games are named as appropriate activities to promote persistence. They are identified as providing sustained and lengthy problem solving with discrimination of useful data, back tracking, and constant evaluation. Many participants recognize that the opportunity to problem solve is relevant in many different contexts.

3.2 Computational Thinking Skills

Participants in this study identified explicit examples of computational thinking skills, as defined by the National Research Council [10], and related them specifically to problem solving. Recognizable computational thinking skills such as decomposition, modeling, and algorithm design are found in the responses, along with other skills such as planning, justifying, and evaluating.

Decomposition, the skill to break problems down, is viewed as being taken for granted. However, for some students, this is reported as being very difficult and requiring explicit teaching. Modeling is identified in the sense of high-level systems that are decomposed into smaller parts, with each individual part modeling behavior of a subsystem. The act of planning an algorithm or a product is viewed as a high-level computational thinking skill. Algorithm design is expressly tied to problem solving by the participants. It is defining the steps, using some accepted convention, to solve a problem. This is viewed differently to program design, which is seen as the translation of an algorithm into automation understandable by a computing device. Unexpectedly, participants also included learning to ask questions about alternatives, identifying trade-offs, justifying decisions, identifying limitations, refining solutions, and evaluating results. These are frequently complemented by the term analytical thinking, which is perceived to involve comparing alternatives, precisely describing, explaining how, and criticizing weaknesses.

In general, participants in this study agree with the National Research Council [10] and Wing [21] concerning the broad definition of computational thinking and none limited the use of the term or the skill set identified by the use of the term only to the domain of computer science.

3.3 Teaching Programming Skills

The concepts in this category represent high-level concerns for the participants. Included here are the concepts of logical thinking, programming as a tool, and collaboration as a pedagogic strategy.

The term logical thinking occurs prolifically in the dataset and appears to be associated closely with programming constructs such as sequence, selection, and iteration. This association is anticipated and parallels that of Saeli [9] who reports that the most identified big idea of programming is control structures. The idea of programming as a vehicle for teaching computational thinking crosses boundaries between respondents, encompassing academics, teachers, and industry professionals. The components of computational thinking, such as decomposition and generalization, are also reported by Saeli [9] to be a big idea of programming. Collaboration is identified, by participants, as an effective strategy for teaching computational thinking. This is usually described as paired or group work, most commonly involving discussions at the analysis or design phases of software development. Notably, there are currently no responses indicating provision for group implementation or paired programming.

While it is not surprising that participants associate the teaching of programming constructs, decomposition, and generalization with computational thinking, it is surprising that an established pedagogic technique, collaboration, has not been extended to opportunities for paired programming.

4. CONCLUSION

4.1 Preliminary Model

Although the dataset has not yet been shown to be saturated, as proscribed by grounded theory, it can form the basis for preliminary theory generation. As indicated in the introduction, participants' responses are used directly to derive a model of the relationships between computational thinking skills, programming skills, and the cognitive domain of Bloom's Taxonomy. Figure 1, a preliminary model, has been derived to illustrate some of these relationships.

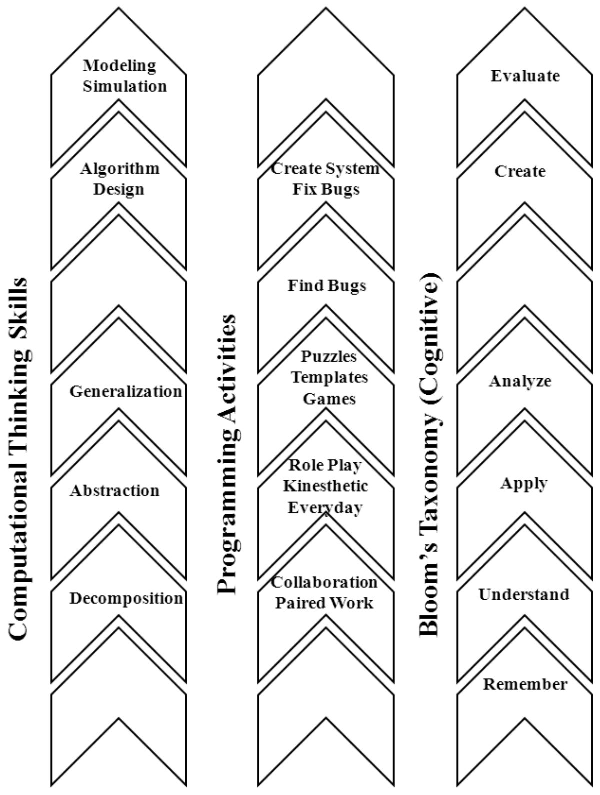

Figure 1: Preliminary Model

The computational thinking skills, reflecting the terminology [10, 21] introduced previously, are represented by an increasing level of complexity. This hierarchy can be discerned from the participants' responses and the reported order of introduction in the classroom. For example, breaking problems down, decomposition, is one technique introduced early in the teaching of both problem solving and programming. The programming activities column represents those activities that participants view as promoting computational thinking. For example, the collaborative work, reported by participants and described in 3.3, usually takes place during the analysis or design phase where a problem is broken down into subcomponents. Interestingly, the terminology used in Bloom's Taxonomy, the last column, is represented directly in the participants' responses. The terms analyze and understand are also used to describe activities found in the initial stages phases of problem solving or programming task. Although the dataset on which the model is based will change and grow, possible relationships can already be discerned

4.2 Implications

This study assumes, in line with Isbell and colleagues [5], that computational thinking skills are a requirement of 21st century society and that these skills must be taught. This research contributes to the body of knowledge that may be used to inform the issue of effective teaching strategies for both programming and computational thinking. By more explicitly defining the relationship between computational thinking and programming, educators may be motivated to move the focus of activities from the production of an artifact to the acquisition of computational thinking skills. In the context of the current educational requirements to include more computer science at all key stages, the results of this study could influence the design of curriculums aiming to incorporate the development of computational thinking skills. In addition, this research responds directly to Guzdial's call [4] for more research into how to teach computing in a way that enforces computational thinking.

4.3 Future Work

Although the current study has not yet reached its conclusion, areas for further study have already been exposed by analysis of the data. These include:

- How do learners move from the specifics of programming, such as language constructs or blocks, to more abstract concepts, such as sorting an array or finding an average, which aid higher-level problem solving?
- How could the explicit teaching of general problem solving skills and high-level problem solving strategies influence the development of computational thinking skills?

More work into the relationships between problem solving, computational thinking, and programming could lead to improved classroom lessons, improved curriculums, and an improved understanding of the skills required in 21st century society.

5. REFERENCES

[1] Cohen, L., Manion, L. & Morrision, K. 2007. *Research Methods in Education,* Abingdon, England, Routledge.

[2] Du Boulay, B. 1989. Some difficulties of learning to program. *In:* SOLOWAY, E. & SPOHRER, J. C. (eds.) *Studying the novice programmer.* Hillsdale, NJ: Lawrence Erlbaum.

[3] Fitzgerald, S., Simon, B. & Thomas, L. 2005. Strategies that students use to trace code: an analysis based in grounded theory. *Proceedings of the first international workshop on Computing education research.* Seattle, WA, USA: ACM.

[4] Guzdial, M. 2008. Education: Paving the way for computational thinking. *Commun. ACM,* 51**,** 25-27.

[5] Isbell, C. L., Stein, L. A., Cutler, R., Forbes, J., Fraser, L., Impagliazzo, J., Proulx, V., Russ, S., Thomas, R. & Xu, Y. 2010. (Re)defining computing curricula by (re)defining computing. *SIGCSE Bull.,* 41**,** 195-207.

[6] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. & Thomas, L. 2004. A multi-national study of reading and tracing skills in novice programmers. *Working group reports from ITiCSE on Innovation and technology in computer science education.* Leeds, United Kingdom: ACM.

[7] Lopez, M., Whalley, J., Robbins, P. & Lister, R. 2008. Relationships between reading, tracing and writing skills in introductory programming. *Proceeding of the Fourth international Workshop on Computing Education Research.* Sydney, Australia: ACM.

[8] Ma, L., Ferguson, J., Roper, M. & Wood, M. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education,* 21**,** 57 - 80.

[9] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working group reports from ITiCSE on Innovation and technology in computer science education.* Canterbury, UK: ACM.

[10] National Research Council. 2010. Report of a Workshop on the Scope and Nature of Computational Thinking. Available: http://www.nap.edu/catalog.php?record_id=12840 [Accessed 10-05-2011].

[11] Pane, J. F., Ratanamahatana, C. A. & MYERS, B. A. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies,* 54**,** 237-264.

[12] Pólya, G. 1985. *How To Solve It,* London, Penguin.

[13] Robins, A., Rountree, J. & Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education,* 13**,** 137 - 172.

[14] Saeli, M. 2012. *Teaching Programming for Secondary School: a Pedagogical Content Knowledge Based Approach*. PhD, Eindhoven University of Technology.

[15] Saknini, V. & Hazzan, O. 2008. Reducing Abstraction in High School Computer Science Education: The Case of Definition, Implementation, and Use of Abstract Data Types. *J. Educ. Resour. Comput.,* 8**,** 1-13.

[16] Schulte, C. & Bennedsen, J. 2006. What do teachers teach in introductory programming? *Proceedings of the second international workshop on Computing education research.* Canterbury, United Kingdom: ACM.

[17] Sixsmith, J. & Murray, C. 2001. Ethical issues in the documentary analysis of e-mail posts and archives. *Qualitative Health Research,* 11**,** 423-432.

[18] Strauss, A. & Corban, J. 1998. *Basics of qualitative research: Techniques and procedures for developing grounded theory,* London, Sage Publications Ltd.

[19] Sweller, J. 1988. Cognitive Load During Problem Solving: Effects on learning. *Cognitive Science,* 12**,** 257-285.

[20] Venables, A., Tan, G. & Lister, R. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the fifth international workshop on Computing education research workshop.* Berkeley, CA, USA: ACM.

[21] Wing, J. 2011. Research Notebook: Computational Thinking - What and Why? The Link. Pittsburgh, PA: Carneige Mellon.