Conditions for successful learning of programming skills

Jaana Holvikivi

Conditions for successful learning of programming skills

Jaana Holvikivi

Helsinki Metropolia University of Applied Sciences
jaana.holvikivi@metropolia.fi

Abstract.

First programming courses often fail to motivate students to continue their software studies. Students find it hard to acquire the logic of computer programming. Especially students in multicultural, heterogeneous student groups are unable to apply logical thinking consistently or to follow instructions in a systematic fashion. Transfer of thinking skills from mathematics to programming does not take place as expected. Efforts to describe the thinking process in program authoring have failed, and process of problem solving in program design remains as evasive as heuristic processes in general. Evidently, it is based on accumulated expert knowledge that is not easily describable. Programming is an independent domain of expert knowledge that requires systematic practice and self-monitoring in construction of appropriate mental patterns.

Keywords: programming skills, computer science education, expert knowledge, mental patterns

1 Introduction

Learning fundamental programming skills has been found to be difficult and the inherent difficulties are hard to analyze [1], [2]. Research in computer science education has attempted to find efficient ways to teach programming, thus far with limited success. Undeniably, several good programming languages for beginners, as well as software and methods to assist in classroom practice have been developed. Nevertheless, none has claimed universal acceptance and the outcome of first programming courses often remains disappointing [3], [4], [5].

A growing body of research has attempted to investigate the learning process of an individual student in programming courses in order to trace common obstacles and misconceptions. Studies on individual learners, on one hand [2], and comparisons of large student populations [6], on the other hand, have been conducted. The results are, however, inconclusive. The present study attempts to shed light on the underlying conditions in learning programming. First, the current understanding in programming

education research is presented, enhanced by insights from cognitive science. Second, a study on student analytical thinking abilities and programming is described, and its findings are compared with ethnographic data on classroom practice. Finally, the results are discussed, and a comprehension of how programming thinking is learnt based on cognitive abilities and accumulation of expert knowledge is outlined.

2 Programming instruction and thinking skills

There is a considerable agreement in the literature that students experience many difficulties in learning to program [4], [5], [6]. For instance, an ACM/ICER working group 2001 reported on beginning computing students' inability to program [7]. The authors tested first year students in several universities in the US and other countries on a common set of programming problems, and the majority of students performed more poorly than expected. The research did not reveal what the principal reasons behind failures were: whether they were caused by fragile knowledge and skills or by poor problem-solving skills. An extensive follow-up research [1] aimed at determining what strategies are likely to lead to success versus those that are likely to lead to failure in program code tracing. The findings indicate that successful and failing students do not actually employ different kinds of approaches, or higher or lower order thinking skills. Most students used multiple strategies in working on a given question, different strategies were used for different questions, and many strategies that were used successfully by certain students would be used incorrectly or unsuccessfully by other students. Success was determined not just by which strategies were employed but rather by how well the particular strategies were employed.

Computer science education can be improved by various means, including development of learning tools, pedagogical methods, and the curriculum. One of the central questions concerns the choice of the initial programming language for the first CS course, which is commonly believed to be crucial for later learning and motivation in software studies [5]. Whether to start with algorithmic or object-oriented programming, or other tools and methods is eagerly debated. Accordingly, many different languages have their supporters: Ada, Pascal, C, Java, PHP, JavaScript, and Python [3], as well as visual aids and building-block kind of tools [5], [6], to mention some. One view that is strongly supported by students favors development environments and powerful languages that are used commercially, whereas an opposing view stresses development of thinking skills, particularly algorithmic thinking. Educators agree, however, that students need to learn a programming way of thinking, which is more than just knowing the syntax and semantics of a language. Moreover, other subjects in computer science such as databases and networks are complex and conceptually demanding, and understanding them requires abstract thinking and good problem-solving skills.

Many innovative approaches have been developed to aid beginner programmers. Stein [8] has introduced a view of "computing as interaction": event-based and object-oriented thinking. Her method uses Java in the introductory course, emphasizing the user-computer interaction that is central in modern networked environments. Radenski [3] has developed a "Python first, Java second" program that according to

his surveys has gained popularity among students. Python comes first because its relative simplicity and Java second to offer features that are more powerful. The program includes a considerable amount of actual practice and supporting materials. In addition, several projects have utilized visualization of program elements [5], [6].

Eckerdal & Berglund [9] concentrate in object-oriented programming thinking in their study because they found that their students had difficulties in it and in describing it when interviewed. The object-oriented paradigm is built on a set of abstract concepts. Students need to reach a certain level in their understanding in order to be able to use the concepts for analysis and design in object-oriented programming tasks. However, there are ways for educators to assist students to reach such understanding. The researchers suggest that these may include problem-solving skills and a systematic way of thinking, as well as thinking about learning, that is, metacognitive skills.

The supposition that higher level thinking skills influence programming ability is based on the idea of transfer of skills from one cognitive domain to another. However, how far this is applicable and whether transfer exists is somewhat debatable. Recent neuroscientific research has tackled the question of transference of learnt skills in the brain. The results have thus far been less promising for educators: transfer of simple cognitive skills does not seem to take place, on the contrary, each task needs a new training [10], [11], [12]. However, this result has thus far been found on simple tasks only; higher level skills are still beyond the means of cognitive science and brain scanning, in particular. The common methodology in cognitive psychology is to isolate cognitive mechanisms. In order to study them in laboratory setting and to measure the results, cognitive processes have to be split into tiny parts. In reality, however, learning complex ideas depends on recruiting multiple cognitive, as well as social and motivational mechanisms and resources [10].

Vainio & Sajaniemi [2] have noticed that novice programmers have poor program tracing skills. They describe programming as a process that includes program authoring, design, program comprehension, debugging, etc. These tasks occur often in a highly overlapping fashion, for instance, program debugging requires program comprehension and detailed tracing, specifically, mental simulation of program execution. Vainio conducted exploratory interviews that included program comprehension tasks with a group of novice students. He then analyzed comprehension protocols to identify specific difficulties affecting novices' ability to trace programs. The study found that students had four kinds of specific difficulties with program tracing: single value tracing, confusing function and structure, inability to use external representations, and inability to raise the level of abstraction.

Programmers are able to hold programs only partially in their memory because of the very limited capacity of human working memory [2]. The paradox in the functioning of the brain is the contrast between the enormous capacity of long-term memory and the narrow pipeline of executive functions. The organization of the brain is far from the organization of disk storage, or any kind of modular storage of information rather, it resembles a complex and dynamic network where many modalities collaborate [13], [14]. The brain is highly analogical and parallel in its activities. For instance, much of language processing takes place in a so-called Broca's area, which together with the premotor cortex links language processing with visual perception and motor activity [14]. Writing is in itself a complex thinking

process, and the same applies to coding. The physical writing activity, which is a motor process, enhances learning results.

Conscious activity is severely limited by the executive function that controls working memory. Conscious processing has a narrow pipeline where the processing is serial, which prevents multitasking. The executive function is vulnerable to distraction, as well. Because of the limited capacity of working memory, we cannot hold long, complex ideas or code fragments in our heads as long as we understand them as separate data. As soon as we learn patterns and algorithms and create mental models of them, we can tackle tasks that are more complex. Mental models or external tools help us structure ideas [15], [16].

Moreover, humans are not able to accurately report their thinking processes [17], [18]. Many studies show that thinking process actually changes when a person tries to report her thinking. Particularly heuristic problem-solving evades awareness. The evidence of many studies strongly suggests that people frequently generate a solution without knowing it [19]. Similarly, when they learn a new strategy to solve problems, it seemingly occurs outside awareness. Insight problems are solved nonconsciously, sometimes during sleep or apparently idle moments [20].

3 Methods

The preliminary data for the present analysis was gathered through a set of surveys among information technology students at a university of applied sciences in the Helsinki area in Finland. The student population consists of over 30 nationalities; predominantly male students. The surveys included questions on study habits, experiences, and difficulties in learning computer science. 124 students participated in the surveys, out of whom 62 responded to a paper-based questionnaire in the classroom and 62 answered to an on-line questionnaire. This data was subsequently compared to course results (examinations and assignment work) of introductory computer science courses. Further explanations were sought through ethnographic observation data from six years' field work. The ethnographic data included field notes on classroom activity, particular emphasis being in unexpected incidents and difficulties. As course work was replicated yearly with new student populations, certain patterns started to emerge. Additionally, students produced learning logs and team work reports, which were analyzed as regards their own perception of their learning. The research process and data are described in detail in the doctoral thesis by Holvikivi [21].

Additionally, two specific studies were conducted to test logical thinking, which was considered an essential component in programming. The first questionnaire tested verbal logic and mathematical reasoning (135 respondents), and the second test compared logical thinking skills with knowledge of Boolean logic acquired in digital circuits courses. The findings of the first study are presented in detail in Holvikivi [22]. The second test was a short questionnaire that was given out in the classroom, and included Boolean operations, four logical syllogisms, and two other reasoning tasks. 45 students of 16 nationalities answered the questions. The results of this test are described below.

4 Findings

The transfer of learning from one course to another turned out to be extremely low in this study. Especially mathematical skills were not fully applied in other contexts. Even arithmetic calculations by students who had passed basic mathematics courses failed in the context of other testing situations (overall 50 % success rate). Failure in number system conversions was even more dramatic, with a success rate of 37 %. The connection of mathematical operations and calculation to problem-solving seemed to be lost for half of the students, which could be explained by mechanical learning and lack of deep understanding [23].

4.1 Logical reasoning

The results of applying logical thinking in syllogisms and reasoning tasks in tests indicated that logical skills are by no means transferable between different types of reasoning [22]. For instance, training in Boolean logic did not help in solving syllogisms. In fact, the group that had not received training in binary logic performed best. The test included Boolean AND and OR operations, four logical syllogisms, and one other reasoning task. The results (Table 1) show that 69% of respondents succeeded in Boolean operations, but only 37% gave correct answers to the verbal tasks, syllogisms. Thus, about half of those who mastered basic mathematical logic failed in verbal reasoning.

Table 1. Number of correct answers in the second logic test

Total respondents 45 (Men 40, Women 5)		
Syllogisms: 4 correct	6	13%
3 correct	11	24%
0, 1 or 2 correct	28	62%
Boolean AND and OR correct	31	69%

A closer look at reasoning studies in general points to the same conclusion: a reasoning task often produces results that are task-dependent or context-dependent, and predicting the outcome from previously performed tasks seldom succeeds amongst untrained subjects [12].

4.2 Working with instructions

Additionally, coding success is not only dependent on abstract thinking but also on systematic working patterns. Coding requires an ability to follow instructions accurately. Observations at our university indicate that when inexperienced students work with laboratory assignments they might follow instructions in somewhat random

order instead of from beginning to end. Assignments, such as creating Excel sheets or typing Linux command line instructions in order to create and copy files, always proceed stepwise. A command line interface requires discipline from the student, and the task leads to chaos when the order of commands is changed, or the student skips some steps or starts from the middle of the exercise. Students who type in incorrect order, and do not follow on the screen what happens, cannot comprehend whether they get an approval (which is seldom indicated by the system) or an error message. The communication process with the computer fails in these cases because a human person is not accustomed to purely formal and very sparse communication by the computer system, and might not even feel that it is a communication situation. [21]

The surveys that we conducted among students revealed that especially immigrant students feel shortcomings in communication in general: more than half of the foreign respondents reported that they found directions unclear whereas home national students had no problems of this kind. This difficulty was particularly connected to laboratory assignments, as up to 65 % of the foreign respondents stated that they often failed to understand instructions. However, the failure to understand instructions did not directly correspond with the English proficiency of the student, even though 20% of students reported language difficulties as well. The failure to understand seems to be connected to the procedural style of the instruction, and perhaps to the unwritten assumptions of what a student is expected to understand without being told [21].

Moreover, some immigrant students come from cultures with a strong oral tradition such as the Somali culture [24]. They prefer personal communication, which can be observed in regular visits to teachers' offices. Students who find it difficult to work based on written instructions would probably benefit from an apprenticeship mode of learning. The instruction at Finnish universities relies strongly on mediating artifacts, such as written or on-line materials. Finnish and other Western students are accustomed to a textual learning environment, which could be described as a "paper-trail" way of working (discussed by Teräs [25]). It pertains to their behavioral patterns, and papers (or documents on computer) are essentially integrated in the work.

4.3 Code tracing and programming

Students often attempt to find a solution intuitively in code tracing tasks, in spite of being told to emulate a computer (this phenomenon was also observed in [1] and [2]). They apply knowledge of natural language that involves concluding from partial information and filling in gaps. Moreover, failure in code tracing tasks indicates a lack of student capability in monitoring one's thinking process.

Additionally, code tracing and error detecting is more difficult for students who have spelling problems. Students who did not study alphabetic languages as their first language, such as Amharic or Nepalese speakers, are particularly afflicted by this impediment [21]. They are unable to "see errors" in punctuation and spelling when proofreading their texts or codes.

In addition to programming concepts that have been discussed above, such as procedural, object-oriented and event-based thinking, we have noticed difficulties in grasping many other fundamental concepts in information technology: the

organization of file systems, local and global networks, relational and hierarchical data base structures, client-server architecture, and so on.

The previous schooling experience of immigrant students may consist mainly of teacher-led lectures that require memorizing of presented material. The laboratory setting might be a completely new kind of classroom environment, and students lack a model for controlling their work in it. If the student's schoolwork schema consists only of teacher speaking and student listening, she may have adopted a recipe-type learning pattern: do as told and repeat it without reflection. However, creative thinking is needed even in simple programming, which is displayed by the variety of solutions to most trivial tasks. A short piece of programming code, a tiny database table or XML structure – whatever it is, each student brings a unique solution! The ability to create new solutions is connected to the ability to apply theory to practice. It is a skill that has to be explicitly taught, and evidently, different educational systems produce it differently. As any other skill, it is developed by constant practice.

On the other hand, the application of object-oriented thinking has proven to be less challenging than in earlier studies. We have applied it in naturally occurring settings, namely manipulating parts of CSS, HTML or XML documents. The Document Object Model that is used in these languages has proven to be intuitive and the results of applying it are immediately visible on the HTML page. Therefore, teaching JavaScript as the first programming language has certain advantages in this respect, in addition to being motivating because of its easy syntax and applicability on Web. Our course program is partially based on Snyder's textbook [26]. Flowcharts in simple program design act as thinking tools and help in code tracing practice. Alternatively, XML structures and XSLT programming are also relatively easy to learn, if they are approached step-by-step proceeding from simple to more complex structures. XSLT is such a high level language that it allows concentration on structures without a need to worry about syntax details. Unfortunately, it is too specific to be used for a general introduction to programming.

5 Discussion

The variety of difficulties in programming thinking is disturbing. Success in learning programming seems to depend on general problem-solving abilities, student cognitive capacity, and analytical intelligence. If the fact is that good students are successful, what remains to be done by educators? Obviously, it is more difficult to improve the outcome, if no particular recipe or optimal working strategy can be identified. An important research finding regarding higher-level expertise has been that an individual's ability to develop his or her content-specific knowledge and apply it in varying situations often co-evolves with the development of general thinking skills and metacognitive strategies [19], [27]. Thus, to have better students, we need to improve their thinking and learning skills, and self-regulation.

Self-reflection and conscious monitoring of progress are important in improving practice. The development of expertise takes place through the deepening mastery of problem-solving processes. Studies on chess masters have shown that they have mental models of a huge number of chess patterns and remember locations of pieces

in them effortlessly [15]. Similarly, jazz pianists have developed a large repertoire of tunes. Programmers have a set of solutions and structures in their mind, as well [2]. The central role of practice and ability to apply theoretical knowledge has also been observed in this study. Students need time and sufficient practice in various problems to build mental patterns of software structures.

Moreover, a certain amount of functional understanding of processes, in addition to technical skills, is a prerequisite for successful software engineering work [28]. Visual aids, graphs and animations help in modeling processes and structures. Additionally, understanding abstract concepts and capability in algorithmic thinking are required. Understanding computers requires an understanding of formal systems. In formal systems, a set of phenomena is encoded as symbols, and the symbols are manipulated by reference to their form only. The meanings of symbols are not interpreted while they are being manipulated [29].

The difficulties in logical reasoning that were noticed in this study can be explained with the organization of cognitive functions in the brain. Reasoning probably consists of several skills that are located separately in the brain. Recent fMRI studies indicate that brain uses different regions and networks for mathematical functions and verbal functions, different regions for enumeration and calculation, for economical decision-making and social decision-making, and so forth [30], [31], [32]. Whether programming logic is processed in the brain predominantly as a mathematical problem-solving task, as a verbal task, or even as a visuo-spatial task, is not known, but would certainly be worth studying.

6 Conclusions

When we learn, we build on existing patterns and schemas in our minds. Learning is easier when we have relevant experience and schemas that are applicable in the particular context. Accumulation of programming expertise is a continuous process of constructing new, useful patterns in the mind to be called on demand.

The traditional knowledge of introducing an easy language first [3] has once again found support from the present research findings. Students need to be allowed to concentrate on the essential patterns, and all additional details should be kept to a minimum, as student capacity to learn is always severely constrained by limitations of the brain. With a simple syntax language, students are allowed to concentrate on the problem-solving aspect, instead of being forced to struggle with a layer of initializations and definitions. Moreover, easy languages often offer fun and enjoyment of learning that motivates students.

Students must have certain basic abilities when they start learning programming. These include mental schemas in regard to procedural thinking and action, functional understanding of processes, and a comfortable relationship to technological artifacts. Knowledge construction takes place through a self-monitoring, intelligent practice that accumulates skills in small increments. In sum, learning programming follows the same patterns as any other domain of expert knowledge. The skill is built on previous experiences and abilities, and requires substantial practice.

References:

1. Fitzgerald, S., Simon, B., Thomas, L.: Strategies that Students Use to Trace Code: an Analysis Based in Grounded Theory. In Proc of ICER'05, ACM Press, New York, pp. 69-80 (2005)
2. Vainio, V., Sajaniemi, J.: Factors in Novice Programmers' Poor Tracing Skills. In Proc of ITiCSE'07, ACM (2007)
3. Radenski, A.: "Python first": a Lab-Based Digital Introduction to Computer Science. In Proc of 11th ITICSE (2006)
4. Webber, C.G., Possamai, R.: An Immune-based Approach to Evaluate Programming Learning. In 9th IFIP World Conference on Computers in Education (2009)
5. Koscianski, A., Bini, E.: Tackling Barriers in the Learning of Computer Programming. In 9th IFIP World Conference on Computers in Education (2009)
6. Lahtinen, E., Ala-Mutka, K., Järvinen, H-M.: 2005 A Study of the Difficulties of Novice Programmers. Proc of ITiCSE'05, ACM (2005)
7. McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I., Wilusz, T.: A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. ACM SIGCSE Bulletin, 33, 4 pp. , 125-140 (2001)
8. Stein, L.A.: The Rethinking CS101 Project, http://www.cs101.org/
9. Eckerdal, A., Berglund, A.: What Does it Take to Learn 'Programming Thinking'? In Proc of ICER'05, pp. 135-142., ACM Press, New York (2005)
10. Schwartz, D. L., Martin, T., Nasir, N.: Designs for Knowledge Evolution: Towards a Prescriptive Theory for Integrating First- and Second-Hand Knowledge**.** In: Gärdenfors, P. & Johansson, P. (eds). Cognition, Education, and Communication Technology. pp. 21-54. Lawrence Erlbaum Associates, Mahwah, NJ (2005).
11. Dos Santos Sequiera, S., Specht, K., Hämäläinen, H., Hugdahl, K.: The Effects of Background Noise on Dichotic Listening to Consonant-vowel Syllables. - Brain and Language 107: 11-15.[C] (2008)
12. Stanovich, K.: The fundamental computational biases in human cognition. In: Davidson, J.E., Sternberg, R. (eds.) The psychology of problem solving. pp. 291-342. Cambridge University Press, Cambridge. (2003)
13. Kandel, E.: The new science of mind. Scientific American Mind, 17(2), pp. 62-69. (2006).
14. Kalat, J. W.: Biological Psychology. 8th Ed. Thomson Wadsworth, Belmont, CA (2004).
15. Ericsson, K.A.: The Acquisition of Expert Performance as Problem Solving. Construction and Modification of Mediating Mechanisms through Deliberate Practice. In: Davidson, J.E, Sternberg, R. (eds.) The psychology of problem solving. pp. 31-83. Cambridge University Press, Cambridge. (2003)
16. Neuroscience and Education. Issues and Opportunities. A Commentary by the Teaching and Learning Research Programme. University of London. (2007)
17. Scherer, K.S.: Feelings Integrate the Central Representation of Appraisal-Driven Response Organization in Emotion. In: Manstead, A.S.R.., Frijda, N., Fischer, A. (eds.) Feelings and Emotions. The Amsterdam Symposium, pp. 136-157. Cambridge University Press, Cambridge. (2004)
18. Wittgenstein, L.: Philosophical Investigations: The German Text with a Revised English Translation: German Text, with a Revised English Translation. Blackwell; 3Rev e. edition (Jan 2002) (1953)
19. Davidson, J.E., Sternberg, R. (Eds.): The Psychology of Problem Solving. Cambridge University Press, Cambridge. (2003)
20. Frith, C.: Making up the Mind: How the Brain Creates our Mental World. Blackwell Publishing, Oxford. (2007)

21. Holvikivi, J.: Culture and Cognition in Information Technology Education. Helsinki University of Technology. SimLab Publications. Dissertation series: 5. Espoo (2009)
22. Holvikivi, J.: Logical Reasoning Ability in Engineering Students: A Case Study. IEEE Trans Educ., 50(4), pp. 367-372. (2007).
23. Hannula, M. S.: Affect in Mathematical Thinking and Learning. Annales Universitatis Turkuensis B 273, Turku, Finland. (2004).
24. Alitolppa-Niitamo, A.: The Icebreakers. Somali-speaking Youth in Metropolitan Helsinki with a Focus on the Context of Formal Education. The Family Federation of Finland, The Population Research Institute. D42/2004 (2004)
25. Teräs, M.: Intercultural Learning and Hybridity in the Culture Laboratory. Dissertation. University of Helsinki, Department of Education. (2007)
26. Snyder, L.: Fluency with Information Technology. Skills, Concepts, Capabilities. Pearson (2006).
27. Kirsch, D.: Metacognition, Distributed Cognition, and Visual Design. In: Gärdenfors, P., Johansson (Eds). Cognition, Education, and Communication Technology. pp. 147-179. Lawrence Erlbaum Associates, Mahwah, NJ. (2005).
28. Moss, J., Kotovsky, K., Cagan, J.: The Role of Functionality in The Mental Representations of Engineering Students: Some Differences in the Early Stages of Expertise. Cognitive Science, 30, pp. 65-93. (2006)
29. Hutchins, E.: Cognition in the Wild. MIT Press, Cambridge, MA. (1995)
30. Polk, T.A., Farah, M.J.: The Neural Development and Organization of Letter Recognition: Evidence from Functional Neuroimaging, Computational Modeling, and Behavioral Studies. Proc Natl Acad Sci U S A., 95(3), pp. 847–852. (1998)
31. Masataka, N., Ohnishi, T., Imabayashi, E., Hirakata, M., Matsuda, H.: Neural Correlates for Learning to Read Roman Numerals. Brain and Language 100, pp. 276–282. (2007)
32. Tang, Y., Zhang, W., Chen, K., Feng, S., Ji, Y., Shen, J., Reiman, E., Liu, Y.: Arithmetic Processing in the Brain Shaped by Cultures. Proc Natl Acad Sci U S A. 2006 July 11; 103(28): pp. 10775–10780. (2006)