# Psychology of Programming Interest Group Annual Conference 2014

University of Sussex,
Old Ship Hotel, Brighton
25-27th June 2014

## Proceedings

Edited by

Benedict du Boulay & Judith Good

# Exploring Core Cognitive Skills of Computational Thinking

Ana Paula Ambrosio

*Computer Science Institute*
*Federal University of Goias*
*apaula@inf.ufg.br*

Leandro da Silva Almeida

*Institute of Education*
*Universidade do Minho*
*leandro@ie.uminho.pt*

Joaquim Macedo

*Department of Informatics*
*Universidade do Minho*
*macedo@di.uminho.pt*

Amanda Franco

*Institute of Education*
*Universidade do Minho*
*Amanda.hr.franco@gmail.com*

Keywords: POP-II.A. novices, POP-V.A. cognitive theories, POP-VI.F. exploratory

## Abstract

Although still innovative and not largely disseminated, Computational Thinking is being considered as a critical skill for students in the 21st century. It involves many skills, but programming abilities seem to be a core aspect since they foster the development of a new way of thinking that is key to the solution of problems that require a combination of human mental power and computing power capacity. This paper presents an exploratory study developed to select psychological assessment tests that can be used to identify and measure Computational Thinking cognitive processes, associated to the programming component, so that strategies can be developed to promote it. After the literature review, we identified four central cognitive processes implied in programming, therefore important to Computational Thinking, and accordingly selected a set of four tests that were administered to a sample of 12 introductory programming students. Our results suggest that spatial reasoning and general intelligence are crucial dimensions for introductory programming, being also correlated to the students' academic success in this area. However, arithmetic reasoning and attention to detail tests did not correlate. Based on these results, directions for future research have been defined in order to effectively identify and develop the core cognitive processes of programming, ergo, to help develop Computational Thinking.

## 1. Introduction

Computing pervades everyday life (Bundy, 2007) and drives innovation necessary to sustain economic competitiveness in a globalized environment (White, 2010). Moreover, a growing number of researchers believe information processing occurs naturally in nature and that computation is needed to understand and, eventually, control such processing (Denning, 2009; Furber, 2012). In this sense, "Computational Thinking" is considered a critical skill for students in the 21st century, and one of the four primary skills, along with reading, writing and arithmetic (CSTB, 2010; Qualls & Sherrell, 2010).

Computational Thinking (CT) can be defined as the ability to interpret the world as algorithmically controlled conversions of inputs to outputs (Denning, 2009). Cuny, Snyder and Wing have defined this construct as the "(…) thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (n.d. as cited in Wing, 2011, p. 1). In other words, CT involves learning to think about, represent and solve problems that require a combination of human cognitive power and computing capacity (CSTB, 2010).

The main reason for the increasing interest in CT is the understanding that scientists will need to be computationally literate, and in the future, it will not be possible to do science without such literacy (Emmott, 2006). Computation is key to the solution of problems in all areas of expertise, and rather than playing merely a supporting role, computer science not only has introduced new, important ways to view and understand the world in which we live, but is transforming all other disciplines. This way, students that do not know how to reason computationally are highly undermined in their abilities as problem solvers (Emmott, 2006). Several skills and cognitive functions have been associated to CT, many of them related to problem solving. Moreover, this construct includes the ability to think logically, algorithmically and recursively, as well as creativity, ability to explain and ability to work in teams (CS4FN, n.d.). As to specific CT techniques employed by students, these include: problem decomposition, pattern recognition, pattern generalization to define abstractions or models, algorithm design, and data analysis and visualization, i.e., being able to collect, analyze and represent data in meaningful ways (Google, n.d.).

What is now being defined as CT has received, over time, other designations, such as “Algorithmic Reasoning”, “Algorithmic Thinking” or even “Process Thinking”, all closely related to Computer Programming. Thus, the development of CT has much in common with learning to program. During a workshop on “The Scope and Nature of Computational Thinking” (CSTB, 2010), several researchers highlighted the importance of programming within CT. However, teaching algorithms and computer programming presents challenges that persist to this day. Learning to program is a particularly difficult task, involving diverse knowledge and skills (Robins, Rountree & Rountree, 2003; Jenkins, 2002). Introductory programming, more specifically, is a highly complex activity, involving sub-tasks related to different knowledge domains and a variety of cognitive processes (Pea & Kurland, 1984). Here, a set of skills are valued: reading comprehension; critical thinking and systemic thinking; cognitive meta-components identification; planning and problem solving; creativity and intellectual curiosity; mathematical skills and conditional reasoning; procedural thinking and temporal reasoning; analytical and quantitative reasoning; as well as analogical, syllogistic and combinatorial reasoning.

Our exploratory study is part of a wider project aiming the conceptualization, assessment and promotion of CT. One of its primary purposes it to identify the skills associated to programming, ergo, central in CT – in fact, one of the assumptions of our study is that this construct and programming are closely related, so many of the cognitive processes used in one are also associated to the other. With this in mind, we selected a set of cognitive tests that could be used as assessment tools in signalling the cognitive processes associated to CT, and which we will call “computational thinking skills”. More specifically, we focused on psychological tests assessing different cognitive functions such as reasoning, numeric, spatial and attention skills. These four cognitive tests were applied to a sample of introductory programming students, in order to identify those skills that seem to be more relevant in programming. The rationale behind the choice of programming students to constitute the sample of our study is that novice programmers have to develop computational thinking skills to succeed, and this usually happens only when they begin their college education.

## 2. Computer programming and cognitive processes

One of the main cognitive variables considered in relation to programming refers to schemas or mental models (Cañas, Bajo, & Gonzalvo, 1994; Pennington, 1987; Soloway & Ehrlich, 1984; Wiedenbeck, LaBelle, & Kain, 2004). The importance attached to mental models seems to be associated to conceptual clarity. In this sense, it is believed that the existence of appropriate mental models is crucial in designing programs that are a direct representation of the first. In "Programming as theory building", Naur (1985) discusses the need for the programmer to have a "theory" about how the program clearly solves the problem in hands, as opposed to programming as a production process. This deep understanding of the problem and of the solution allows the programmer to not only implement the solution, but to be able to support what he is doing with explanations, justifications and answers to queries about it. The quality of the program is thus directly determined by the quality of the theory built by the programmer.

Mental models have been studied in relation to different aspects of programming and used in diverse ways to evaluate student's computing ability. Several authors analyzed the mental model of students regarding computer operation (Mayer, 1989; Perkins, Schwartz, & Simmons, 1988). For instance, Mayer (1989) claims that students without a fit mental model of how the computer processes variables and data in memory present greater difficulty in understanding programming language commands. Dehnadi (2006) suggests that students who succeed in programming courses are those that use a certain mental model, independently of its specificities, for the allocation and manipulation of variables in a consistent manner.

Authors have also been concerned with the differences between experts and novices regarding their mental models. According to Cañas et al. (1994), people have different mental representations of computer programs: these can be based on syntactic or semantic aspects, and the mental representations that are centered on semantic aspects are closer to experts' mental models. Ackermann and Stelovsky (1987) state that while novices perceive programs as sequences of commands, experts tend to group commands in schemes that represent given functionalities. To analyze the representation or mental model of a subject, tasks such as recall, categorization by similarity or association can be used, and seem to be sensitive to changes that occur in mental representation as students learn to program. Curiously enough, and contrarily to prior expectations, the students' mental model is not affected by previous programming experience; however, developing mental models leads to greater self-efficacy and better results (Wiedenbeck et al., 2004).

Despite several studies, or perhaps as a result of them, perception remains that programming is problem solving. Programmers learn to program by building programs, hence, by problem solving. Moreover, its main objective is to implement programs that solve computational problems. In fact, the resolution of problems is an important part of their work; however, in addition to restrictions that are normally found in problem solving, programming requires that these solutions are computationally executable. This imposes additional restrictions on the problems that must be implemented by general, effective and finite algorithms.

Once the solution to a problem has been found, the path can be reused to solve similar problems. Through multiple episodes of analog problem solving, individuals can induce abstract schematic representations of problems and their solutions, which can be retrieved and applied as solution plans when structurally similar problems are presented (Gick & Holyoak, 1980, 1983). So a schema can be defined as a cognitive structure that allows a problem solver to recognize a particular class of problems as belonging to a given state that normally requires particular moves or strategies. In other words, they create mental models.

Perhaps because of this relationship between programming and problem solving, evaluations of programming skills have been historically based on tests related to intelligence and deductive logic. In the early ages of computer programming, tests began to be developed in order to identify those who had "aptitude" for programming, a measure of the applicant's natural ability to understand and learn new things and perform new tasks. It was understood that the ability to program was innate and that not all could be programmers. For economic reasons, it was then necessary to identify those individuals who would succeed in learning programming. In the 60's, these aptitude tests were used by 68% of computer companies surveyed in the U.S.A. and 73% in Canada (Dickman & Lockwood, 1966 as cited in Robins, 2010). Many of these tests continue to be used by companies to select developers, but less intensely and as part of a broader process of selection.

The first of these tests to succeed was the *Programmer Aptitude Test* (PAT), developed by *IBM* in the 50's. The test consisted of three parts. First, the subject had to identify the next number in a series. Second, they had to identify analogies represented in figures, and in the third part had to do arithmetic problems. IBM currently uses the IPATO, an online test developed by IBM to test logical reasoning and ability to process information quickly.

A battery that obtained great success, still used today, is the *Aptitude Assessment Battery Programming* (AABP), developed in 1968 by Jack M. Wolfe . This battery is based on the traits and skills identified by Wolfe and simulates daily work tasks, assessing logic reasoning, ability to

interpret complex specifications, documentation and annotation skills, problem-solving skills, accuracy, attention to detail, speed, concentration and ability to follow instructions accurately.

The *Computer Programmer Aptitude Battery* (CPAB) is a battery of tests that determines individual aptitude for computer programmer or system analyst. Published by Vangent, Inc., is composed of five subtests that assess the subject's ability to understand the vocabulary used in mathematics, management and systems engineering literature; reasoning identified by the ability to translate ideas and operations in textual notation into mathematical notation; the ability to use abstract reasoning to find patterns in given letter series; the numerical ability translated by the ability to quickly estimate reasonable answers to calculations; and the ability to analyze problems and find solutions to a logical sequence using diagrams. There is a short and a long version. The short contains only those parts related to Reasoning and Diagramming that make up the long version, and which have been shown to be those that best predict the performance of programmers. Both tests are comparable and can be used for retest.

Another widely used battery is the *Berger battery*, a set of proficiency and aptitude tests marketed by Psychometrics, Inc., Henderson, NV (www.psy-test.com). Apart from tests on several programming languages, there is the B-APT for people with no programming experience which aims to identify candidates for training. It consists of 30 questions that must be answered in 1h15min and uses a hypothetical language that candidates must use to write small programs.

These tests are mainly used for recruiting programmers, and are more suited to experienced adult programmers, where we are interested in identifying core cognitive skills important to programming, especially to help those students that have difficulty in learning to program. Furthermore, the correlation between the results obtained in testing and the subjects' programming skills has been consistently low (Mayer & Stalnaker, 1968; Pea & Kurland, 1984). Experiments have shown little or no relationship between the final ranking of students in programming and their evaluated "aptitude" (Davy & Jenkins, 1999; Mazlack, 1980). In his article, Wolfe (1971) discusses the limitations of programming aptitude tests, including the use of multiple-choice questions, the undergraduate students' "tricks" in the testing and the inclusion of mathematical issues that reduce the test's effectiveness for commercial programming.

According to Weinberg (1998), being an area where there are basically no known factors that influence the behavior of programmers, much of the psychology of programming research attempts to identify what is important to be measured. So the big question is not "how to measure", but "what to measure". In this sense, and according to the same author, the use of aptitude tests is much more an analysis of tools than of subjects, so we first must define "what to measure". In this sense, and based on the review of literature, we have identified a set of cognitive abilities that seem to be correlated to programming success and that we considered in our study.

## 3. Materials and Methods

### 3.1 Participants

The study was undertaken with 12 freshmen students from the University of Minho in Portugal: five students from the BSc in Informatics Engineering (LEI) course, a 1st Cycle 3 year course that results from a transition to the Bologna Process of the old BSc in Informatics and Systems Engineering, and seven students from the BSc in Computer Science (LCC) course, a 1st Cycle 3 year course that results from a transition to the Bologna Process of the old BSc in Mathematics and Computer Science. Half of the participants were female and half male, with ages between 18 and 29 years old ($M$ = 20.6; $SD$ = 3.55). The participants were selected from two contrasting groups: students with good academic results and students with poor academic results. Both groups were selected by their teachers, based on their observations, since tests were applied in the beginning of the course and that grades were not yet available.

### 3.2 Instruments

To identify basic cognitive skills involved in CT, we used validated cognitive tests. Four tests were chosen based on perceived associations of the basic skills assessed by the tests and CT (or programming) found in the literature: a general intelligence test, a spatial reasoning test, a mathematical reasoning test, and an attention to details test. Very briefly, the intelligence test was chosen to verify the general idea that programming ability is innate and closely related to intelligence level, mainly from an abstraction and inductive reasoning point of view; the spatial reasoning test was proposed to verify the subjects' ability to "see" the problem and work with it mentally, reasoning with figures and patterns; the mathematical reasoning test aimed to evaluate the subjects' problem solving abilities and the role of mathematical knowledge in programming; finally, the attention to details test was used to evaluate the subjects' capacity to identify pertinent details and pattern recognition abilities.

*Test D.48* (Pichot, 1949) is a nonverbal test that measures the g factor of subjects based on their abstraction capacity, ability to understand relations, reasoning by analogy and ability to solve new problems (induction). It is composed of 44 items that must be completed in 25 minutes. Each item contains a series of dominos to which a new one must be added taking into account the series' configuration. To discover the missing domino, the subject must apply reasoning, not only attending to the number of dots in the domino (discovering, for example, ascending values), but also to the configuration (discovering, for example, a particular symmetry) and analogy (discovering what B is to A as to discover what D is to C). This test has shown high correlation to academic achievement.

The *Spatial Reasoning test* assesses two components associated to spatial factor – the ability to recognize and view figures, and the ability to rotate or follow the movements of figures. It is a subtest of the *Differential Reasoning Tests* (*Bateria de Provas de Raciocínio, BPR-5*), composed of five subtests evaluating reasoning (common cognitive processes) with different content items (abstract, verbal, numerical, spatial and mechanical). The *Spatial Reasoning* subtest is composed of 20 items containing series of 3D cubes to be completed in 8 minutes. The series are obtained through the rotation of the cubes by movements that can be constant (for example, from left to right) or alternate (for example, left and up). By identifying the movements that took place, the subject must choose, amongst the alternatives, the cube that represents the next cube in the series, obtained by applying the identified movements (Almeida & Primi, 1996 as cited in Almeida, Antunes, Martins, & Primi, 1997).

The *General Aptitude Test Battery* (GATB) (U.S. Department of Labor, 1983), consists of 12 separately timed subtests which make up nine aptitude scores (general learning ability, verbal aptitude, numerical aptitude, spatial aptitude, form perception, clerical perception, motor coordination, finger dexterity and manual dexterity). Two subtests from the Portuguese version of GATB were used: the arithmetic reasoning subtest and the tool matching subtest. The *Arithmetic Reasoning subtest* assesses the ability to verbally solve expressed arithmetic problems and the ability to deal intelligently with numbers. It is a measure of general intelligence and numerical aptitude. The problems require basic mathematical skills like addition, subtraction, multiplication or division. The test includes operations with whole numbers, rational numbers, ratio and proportion, interest and percentage, and measurement. It contains 25 word problems with five alterative answers, to be answered within a seven minute time limit. Finally, the *Tool Matching subtest* assesses the ability to understand pertinent details in objects in pictorial or graphic material, as well as the ability to make visual comparisons or discrimination of detail. It consists of a series of exercises containing a stimulus drawing and four black-and-white drawings of simple shop tools. The four drawings have different parts of the tools painted in black and white. The subject must identify which of the four options exactly matches the stimulus drawing. There are 49 items, measuring form perception, with a time limit of five minutes.

### 3.3 Procedures

Participation was voluntary and in the students' free time. For this reason, the tests were administered in three different sessions to comply with the students' schedule. Preceding the application of the tests, we presented the goals of our study and assured the confidentiality of the individuals' results.

Students received one test at a time, and the instructions were read out loud previously. Since each test had a specific time limit, when it was up, the test was collected and another one given. The application of the four tests took approximately one hour. The order in which the tests were given to the students varied between sessions.

A raw score was obtained for each test, adding the number of correct items completed by the students. Thus, even though other formulas may be used to calculate global factors, as the aptitude scores in the GATB, these were not used in our analysis.

The quantitative analysis was undertaken. The results were correlated to the student's final exam result and final grade in their second semester programming course, and with a performance evaluation done by the teacher, that attributed a classification from 1 (min) to 5 (max) according to his perception of the student's aptitude and facility in learning programming, disregarding the student's effort and motivation. This evaluation was important to identify those students that showed good capacity for programming, but had poor academic results due to other factors, such as not doing homework assignments and missing the final test, as well as those students that were able to pass the class mainly due to their motivation and hard work, despite a blatant lack of skill.

## 4. Results and Discussion

In Table 1, we present the descriptive data (number of students that took the test, maximum possible score in the test, minimum and maximum scores obtained by the students, mean and standard deviation), regarding the cognitive tests undertaken by our sample as well as the students' academic achievement. Values have been standardized to maximum score 10 in order to facilitate reading and comparison of results. This exploratory study has shown interesting results: tests varied significantly, not only when comparing the different test scores for each student, but also when comparing the results our sample achieved in each test. This means that no student was consistently good in all tests, and in a given test there was a significant variation between the minimum and maximum number of correct answers. Furthermore, the students' academic results also varied: eight students were approved in the discipline and three failed, and for one the grades were not available; moreover, one of the students was approved after retaking the exam, obtaining a significantly better result (almost doubled).

| *Performance indicators* | *N* | *Maximum possible score* | *Min.* | *Max.* | *M* | *SD* |
|---|---|---|---|---|---|---|
| D48 | 12 | 44 | 6.0 | 9.0 | 7.27 | 1.11 |
| Spatial Reasoning | 12 | 20 | 2.5 | 9.0 | 6.13 | 1.93 |
| Arithmetic Reasoning | 12 | 20 | 1.5 | 7.5 | 4.67 | 1.87 |
| Matching Tools | 12 | 49 | 7.0 | 9.0 | 7.55 | .79 |
| Final grade | 11 | 20 | 4.5 | 8.5 | 6.18 | 1.38 |
| Exam | 11 | 20 | 2.0 | 7.8 | 4.40 | 1.93 |
| Teacher evaluation | 11 | 5 | 5.0 | 10.0 | 8.18 | 1.94 |

*Table 1. Standardized students' results according to each cognitive test*

In Table 2, the correlation coefficients between psychological tests and two academic achievement measures are presented. Analysing the correlation between the students' evaluation factors, we verified a very strong correlation between the grade obtained in the exam and the final grade ($r$ = .898, $p < .001$). The teacher's evaluation correlated with the exam grade ($r$ = .614, $p < .05$) but not with the final grade ($r$ = .489, *n.s.*). This could be expected since the exam grade depends less on motivation and dedication than the final grade, which takes into account other factors such as home assignments and class work. Spearman's rank correlation coefficient was also tested, but did not yield better coefficient and significance values.

| | | D48 | Spatial Reasoning | Arithmetic Reasoning | Matching Tools |
|---|---|---|---|---|---|
| FinalGrade | Pearson Correlation | .491 | .579 | .111 | .310 |
| | Sig. (2-tailed) | .125 | .062 | .746 | .353 |
| Exam | Pearson Correlation | .378 | .427 | -.069 | .273 |
| | Sig. (2-tailed) | .252 | .191 | .840 | .416 |

*Table 2. Correlation between performance indicators and results from cognitive tests (n=11)*

As our main objective was to cross cognitive abilities (as measured by the four presented tests) and the students' performance in introductory programming courses, the correlation between the cognitive tests and the academic results was undertaken. Due to the small number of subjects, we will consider the correlation coefficient to verify the effect size since it gives us a perception of the strength of relationship between two variables (Field, 2009) without taking into account the significance – since the significance depends on the correlation coefficient and the degrees of freedom, and the latter is under the influence of the size of the sample (in our case small), which has an impact on the significance.

The correlations between psychological tests and academic classification in the domain of computer science point to two distinct patterns of correlation with regard to their effect size. On one hand, the D48 test (g factor) and the spatial reasoning test (rotation of figures in space and mental retention of these positions to continue the task) showed a large effect when correlated to the students' academic results, as opposed to the attention to detail test and the arithmetic reasoning test, which showed small and medium effects respectively. Thus, the students' computational skills, at the academic learning level, seem to require more from their logic-deductive reasoning skills (infer and apply or generalize relations) and a holistic or simultaneous organization of the information (spatial organization). Simple attention or calculus tasks do not seem to be relevant in differentiating performance in computer science students; this possibly means that, being very basic skills, with no inherent significant difficulty, all students attain a very similar level of performance, and therefore, they are not relevant to the differentiating goal of our study. It should be noted that the lack of correlation concerning the *Arithmetic Reasoning test* supports previous research suggesting that mathematical knowledge is not correlated to programming ability. However, the test was composed of word problems, which could imply the need for problem solving abilities, and may account for the medium size effect encountered. In this sense, further investigation may be needed to verify the correlation of problem solving abilities and computational thinking.

The data obtained in our study agrees with results obtained in other tests. The PISA 2003 test (OECD, 2003), in addition to items related to the domains of reading, mathematics and scientific literacy, included items related to problem-solving that evaluated processes very similar to those necessary in programming. According to its results, less than 20% of the students were able to solve difficult problems (level 3). In such problems, students should be able to analyze a situation and make decisions, handle multiple conditions simultaneously, think about relationships underlying a problem, solve it in a systematic way, evaluate their work and communicate results. About half of the students were able to answer level 2 questions, which imply good reasoning, being able to confront unfamiliar problems and being capable of coping with a certain degree of complexity. Data analysis showed a high correlation between problem solving and other areas, especially mathematics. However, the performance in solving mathematical problems that involve simple calculations, without further inferences, has a low correlation with performance in problem solving. These findings align with the evaluation of programming teachers that mathematical knowledge does not determine success in programming. They believe that the correlation between mathematics and programming is linked to the development of mental processes common to both areas, such as inference.

## 5. Conclusions

In our study, we aimed to identify the cognitive processes that are central in programming, therefore implied in CT, in order to understand which skills students must develop in order to succeed in learning introductory programming. By identifying which are these skills, and how they relate to academic success, it is viable to apply fitter pedagogic methodologies that will foster this essential type of thinking. This matter is most important, given the transversal pertinence of CT for every individual, and that the problems faced in teaching computer programming courses pass from the computer science realm to a global setting, justifying further study of the cognitive processes associated with Programming and CT, as to stimulate and promote it in all levels of education.

Our exploratory study emphasizes the complexity of CT as a mental process, for which is necessary to discriminate the underlying cognitive functions that comprise this construct. To do so, further investigation is needed, using a larger sample. In future research, our proposal regarding the cognitive processes that are core to CT needs refinement. First, to confirm the correlation between academic success in programming, and intelligence and spatial reasoning, other tests may be included, such as *Raven Progressive Matrices* and other spatial tests involving 3D figures from the *Differential Reasoning Tests*. It may also be important to include working memory tests, in light of recent research that identifies these executive processes as crucial for intellectual functioning. Second, to confirm the lack of correlation between programming and attention to details, a different perception test should replace the *Tool Matching* test, which may not have shown correlation due to the test format and lack of face validity. Similarly, the arithmetic reasoning test should be replaced by two distinct tests: one that measures problem solving abilities and another that measures math abilities, as to confirm the correlation between programming and problem solving that has been verified in related works, as well as the lack of correlation with pure mathematical abilities. Nevertheless, selected tests must present a higher level of difficulty as the results obtained seem to indicate low degree of student differentiation in the tests that were applied, making it impossible to estimate if these cognitive processes are really important to explain the academic result differences presented by the introductory programming students. Finally, future studies must combine qualitative and quantitative methodologies. For this, a larger number of subjects must be considered, always combining two levels of academic achievement in introductory programming, or, adopting a logic of expert versus novice, select students with different skill levels in this domain. Besides applying the tests, these two groups can be interviewed to interpret their performance in the tests. Content analysis of the answers may yield leads that approximate the learning that occurs and the different levels of academic performance of these students, with the inherent processes of the test items in which they had greater or lesser difficulty in solving. This might shed some light in understanding to what extent programming aptitude is “innate” and depending on intelligence, or developed, or even in analyzing the students’ schemas and mental models, and trying to comprehend how they develop.

## 6. References

Ackermann, D., & Stelovsky, J. (1987). The role of mental models in programming: From experiments to requirements for an interactive system. Lecture Notes in Computer Science, 282, 53-69.

Almeida, L. S., Antunes, A. M., Martins, T. B. O., & Primi, R. (1997). Bateria de Provas de Raciocínio (BPR-5): Apresentação e procedimentos na sua construção. Actas do I Congresso Luso-Espanhol de Psicologia da Educação (pp.295-298). Coimbra: Associação dos Psicólogos Portugueses.

Almeida, L. S., Candeias, A., Primi, R., Ramos, C., Gonçalves, A. P., Coelho, H., Dias, J., Miranda, L., & Oliveira, E. P. (2003). Bateria de Provas de Raciocínio (BPR5-6): Estudo nacional de validação e aferição. Revista Psicologia e Educação, 2 (1), 5-15.

Bundy, A. (2007). Computational thinking is pervasive. Journal of Scientific and Practical Computing, 1, 67-69.

Cañas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). Mental models and computer programming. International Journal of Human-Computer Studies, 40 (5), 795-811.

CS4FN, Computer Science for Fun, (n.d.). Retrieved February 16, 2012, from http://www.cs4fn.org/computationalthinking/

CSTB, Computer Science Telecommunications Board, (2010). Report of a workshop on the scope and nature of computational thinking. Washington, D.C.: The National Academies Press. Retrieved February 5, 2012, from http://www.nap.edu/openbook.php?record_id=12840&page=R1

Davy, J., & Jenkins, T. (1999). Research-led innovation in teaching and learning programming. Proceedings of ITiCSE '99 (pp.5-8). Cracow, Poland.

Dehnadi, S. (2006). Testing programming aptitude. In P. Romero, J. Good, E. A. Chaparro, & S. Bryant (Eds.), Proceedings of the 18th Workshop of the Psychology of Programming Interest Group (pp.22-37). University of Sussex.

Denning, P. (2009). The profession of IT- Beyond computational thinking. Communications of the ACM, 52 (6), 28-30.

Emmott, S. (2006). (Ed.). Towards 2020 Science. Microsoft Research. UK: Cambridge.

Field, A. (2009) Discovering Statistics using SPSS, 3rd ed. Sage Publications, London UK.

Furber, S. (2012) Shut down or restart? The way forward for computing in UK schools. London, UK: The Royal Society. Retrieved February 5, 2012, from http://royalsociety.org/education/policy/computing-in-schools/report/

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. Cognitive Psychology, 12, 306-355.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. Cognitive Psychology, 15, 1-38.

Google. (n.d.). Exploring computational thinking. Retrieved February 5, 2012, from http://www.google.com/edu/computational-thinking/what-is-ct.html

Jenkins, T. (2002). On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences (53-58). Loughborough: University United Kingdom.

Mayer, R. E. (1989). The psychology of how novices learn computer programming. In E. Soloway, & J. C. Spohrer (Eds.), Studying the novice programmer. Hillsdale, NJ: Lawrence Erlbaum Associates.

Mayer, D. & Stalnaker, A. (1968) Selection and evaluation of computer personnel - the research history of SIG/CPR, ACM '68 Proceedings of the 1968 23rd ACM national conference, New York, NY

Mazlack, L.J. (1980) Identifying Potential to Acquire Programming Skill. Comm. ACM, Vol. 23, pp 14-17.

Naur, P. (1985). Programming as theory building. Journal of Systems Architecture: The Euromicro Journal, 15 (5), 253-267.

OECD, Organization for Economic and Co-operation and Development. (2003). PISA 2003: First results from Pisa 2003 - Executive Summary. Retrieved February 5, 2012, from http://www.oecd.org/dataoecd/1/63/34002454.pdf.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive prerequisites of learning computer programming (Technical Report No.18). New York: Bank Street College of Education, Center for Children and Technology.

Pennington, N. (1987). Comprehension strategies in programming. In E. Soloway, & S. Iyengar (Eds.), Empirical studies of programmers: Second workshop (pp.100-113). Norwood, NJ: Ablex.

Perkins, D. N., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), Teaching and learning computer programming (pp.153-178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pichot, P. (1949). Les test mentaux en Psychiatrie: Tome premier; instruments et methods. Paris: Presses Universitaries de France.

Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. Journal of Computer Science in Colleges, 25, 66-71.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS. Computer Science Education, 20 (1), 37-71.

Robins A., Rountree J., Rountree N. (2003) Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2): 137-172

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programmer knowledge. IEEE Transactions of Software Engineering, SE-10, (5), 595-609.

U.S. Department of Labor. (1983). The dimensionality of the General Aptitude Test Battery (GATB) and the dominance of general factors over specific factors in the prediction of job performance for the U.S. Employment Service. (USES Test Research Report No. 44). Washington, D.C.: U.S. Government Printing Office.

Weinberg, G. M. (1998). The psychology of computer programming (Silver Anniversary Edition) (2nd ed.). New York: Dorset House Publishing.

White, J. (2010). The state of computer science education (Special Report). Retrieved February 5, 2012, from http://www.cioupdate.com/reports/article.php/3915826/Special-Report---The-State-of-Computer-Science-Education.htm

Wiedenbeck, S., LaBelle, D., & Kain, V. (2004). Factors affecting course outcomes in introductory programming. In E. Dunican, & T. R. Green (Eds.), Proceedings of the 16th Workshop of the Psychology of Programming Interest Group (pp.97-110). Carlow, Ireland: Institute of Technology Carlow.

Wing, J. (2011) Research notebook: Computational thinking - What and why? Retrieved February 5, 2012, from http://link.cs.cmu.edu/article.php?a=600

Wolfe, J. M. (1971). Perspectives on testing for programming aptitude. ACM '71 Proceedings of the 1971 26th annual conference (pp.268-277). Chicago, Illinois.

Wolfe, J. M. (1968). Aptitude Assessment Battery Programming. Walden Personnel Testing & Consulting Inc.