International Journal of Engineering Education Vol. 32, No. 3(A), pp. 1107–1116, 2016
Printed in Great Britain

0949-149X/91 $3.00+0.00
© 2016 TEMPUS Publications.

Exploration of Computational Thinking of Software Engineering Novice Students Based on Solving Computer Science Tasks*

VLADIMIRAS DOLGOPOLOVAS
Vilnius University Institute of Mathematics and Informatics, 4 Akademijos Street, Vilnius LT-08663, Lithuania and University of Applied Sciences, Faculty of Electronics and Informatics, 15 J. Jasinskio Street, LT-01111, Vilnius, Lithuania.
E-mail: vladimiras.dolgopolovas@mii.vu.lt

TATJANA JEVSIKOVA and VALENTINA DAGIENĖ
Vilnius University Institute of Mathematics and Informatics, 4 Akademijos Street, Vilnius LT-08663, Lithuania.
E-mail: tatjana.jevsikova@mii.vu.lt, valentina.dagiene@mii.vu.lt

LORETA SAVULIONIENĖ
University of Applied Sciences, Faculty of Electronics and Informatics, 15 J. Jasinskio Street, LT-01111, Vilnius, Lithuania.
E-mail: l.savulioniene@eif.viko.lt

During the recent years computational thinking has been actively promoted through the K-12 curriculum, higher education, contests, and many other initiatives. Computational thinking skills are important for a further students' educational and professional career. Our focus is on computational thinking for software engineering novice students, a term meant to encompass a set of concepts and thought processes that are helpful in formulating problems and their solutions. Annually organized international challenge on Informatics and Computational Thinking "Bebras" has developed many tasks to promote deep thinking skills in this area. It is important to motivate students to solve various informatics or computer science tasks and evaluate their computational thinking abilities. The paper presents a study conducted among first-year students of software engineering, studying the structured programming course. As an instrument to measure computational thinking, a test of internationally approved and well-preselected tasks from the "Bebras" challenge has been suggested and validated. The correlation between the students' test results and the structured programming course results has been investigated. We conclude with a discussion and future directions to enhance computational thinking skills of novice software engineering students.

Keywords: computational thinking; Bebras challenge; computer science concepts; computer engineering education; contest; novice programming students; novice software engineering students

1. Introduction

During the past years, computational thinking (CT) has been actively promoted through the K-12 curriculum as a part of computer science (CS) subject or in an integrated way, paying more and more attention to programming and fundamental computer science concepts (e.g. [1, 2]).

Since programming is fundamental to computer science education, computer scientists tend to think like programmers. They look for algorithmic solutions to problems, in terms of data manipulation and process control. Computer scientists have a toolbox of methods for matching problem situations to the standard types of solution, drawn from various parts of the computer science curriculum, and, perhaps just as important, a standard terminology to describe these abstract problem solution patterns [3]. Unless originated from computer science, CT is a term encompassing a set of CS concepts and thought processes that aid in formulating problems and their solutions in different fields of life. As Jeannette Wing has defined, "computational thinking represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" [4, p. 33]. So, CT should be educated not only through the school curriculum, higher education, but also through the initiatives, accessible for a wide area of participants.

One of increasingly popular activities in promoting CS and CT is an international challenge on Informatics and Computational Thinking called Bebras (Lithuanian word for beaver), originated in Lithuania [5]. The main goals of the challenge are to raise students' awareness of CS and CT and evoke interest in the field through a set of inspiring tasks.

This paper presents a study conducted among the first-year students of software engineering, studying the structured programming course at the applied science university. The study is based on authors' experience in developing teaching methods and practical teaching SP for novice software

* Accepted 12 February 2016.

engineering students, as well as in authors' experience in developing CS contests and related educational methods [5–8]. As we have already mentioned, CT is essential for all, but it is obviously one of the fundamental skills for software engineers and programmers.

The structured programming course is problem solving-oriented and based mainly on the Python programming language. Python as a language is widely promoted as an alternative to other languages and as a first language to study programming. The reason is that Python is positioned as a tool for a rapid application development and, at the same time, as an educational tool enabling problem solving programming courses to be developed [9, 10]. During such a course, software engineering students develop their knowledge of CS basics and concepts and study the language syntax in parallel. In addition, the use of game-like tasks or studying how computer games work increases students' motivation and is one of the promising ways to learn new CS concepts [11].

The aim of this paper is to investigate how CT skills of software engineering novice students can be evaluated and how the evaluation results correlate with the programming course results.

As an instrument to evaluate computational thinking, a test of internationally approved and well-preselected tasks of the Bebras challenge has been suggested and validated. The research is based on the authors' practical experience in teaching structured programming for the first semester software engineering students and as well as their solid practical and theoretical experience in developing contest-related educational methods, based on the authors' research.

The structure of the paper is as follows. First, we analyse the existing resources on computational thinking and the Bebras challenge, the specifics and deepness of its tasks for computer science. Next, we present the methodology of the research: we rise research questions, discuss task selection and test preparation issues, and describe the participants of the study. Then we discuss the results (the correlation between the test results and the structured programming course results has been investigated and test validity has been studied, using Item Response Theory). We conclude with a discussion and future directions how to enhance computational thinking skills of novice software engineering students.

2. Background of the research

One of the drivers of this research was high drop-out rates of software engineering students. This problem is addressed in many studies (e.g. [12] and suggestions are made, e.g. introducing new solutions for e-learning courses [13], using and analysing students' profiles [14], introducing educational games to raise students' motivation [15], using short educational videos [16], electronic interactive tests [17], and other decisions). CT skills as well as the programming course and methodology are essential in this problem.

2.1 Computational thinking

Later on, J. Wing gave a more concrete definition, stating that CT can be understood as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent" [18]. The latter definition is based on the idea that there are layers of abstraction of both data and processes involved in CT. Wing also distinguishes between what CT means for everyone and what it means for scientists, engineers, and professionals.

A study of the existing research on computational thinking [19] uses two main definitions of CT as a starting point: operational definition of Computer Science Teachers Association [20] and Google's characteristics of CT [21].

The operational definition of CT [20] suggests that CT is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data.
- Representing data through abstractions such as models and simulations.
- Automating solutions through algorithmic thinking (a series of ordered steps).
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
- Generalizing and transferring this problem solving process to a wide variety of problems.

CSTA goes on to add that these skills are supported by a set of broader attitudes: ability to deal with complexity and open ended problems, tolerance for ambiguity, and ability to work with others to achieve a common goal. This definition underwent a review process that included a survey of over 700 experts including computer science teachers, researchers, and practitioners. The vast majority of respondents (n = 697, about 82%) indicated their agreement or strong agreement when asked if CSTA's definition captured the fundamental elements of computational thinking, and a further 9% indicated that the definition was sufficient to reach a consensus in the computer science education community [19].

The definition, offered by the Google's Exploring Computational Thinking initiative (the first large scale program to provide an operational definition, disseminate resources, and promote discussions among K-12 educators about computational thinking), describes CT as a process that includes four techniques: decomposition, pattern recognition, pattern generalization and abstraction, and algorithm design [21].

In order to measure computational thinking of the students, there should be defined core concepts, operations or skills, encompassed by CT. Such operations are reflected in the CSTA and Google definitions cited above. The CSTA's operational definition of CT includes the initial step that describes formulating problems so that the technology can be used to help solve them is similar to the Google's concept of decomposition, which involves taking a large, complex problem and breaking it into smaller and easier/more manageable ones. Next, CSTA describes the logical organization and analysis of data, which have parallels to Google's pattern recognition, or the ability to find similarities or differences that help "make predictions or lead to shortcuts". Representing data through abstractions is similar to pattern generalization and abstraction, or the ability to remove the details of a problem in order to find a solution that is able to solve similar problems. This involves filtering out unnecessary details and designing a solution that can be used to solve similar problems. The CSTA's next two steps of using algorithmic thinking to automate solutions and analyze possibilities in order to find an efficient and effective solution are similar to the Google's algorithm design, which is described as development of a step-by-step strategy or a set of instructions for completing or solving a similar problem. The CSTA definition extends that one step beyond to include the idea of generalizing and transferring this process to diverse problems [19].

Other initiatives and resources define their own concepts or skills that are varying from author to author, but anyway have core commonalities. For example, Computing at School initiative in the UK has included computational thinking concepts into the K-12 curriculum. These concepts include Abstraction (AB), Decomposition (DE), Algorithmic thinking (AL), Evaluation (EV), and Generalisation (GE) [22] that can be easily mapped to the previously cited operations.

A systematic review for CT literature, conducted in 2011 and including 3465 articles in the initial phase [19], has shown that 46% of articles address undergraduate students, 2% address postgraduate students, 42% address K-12 students, and 19% computing education community.

Most students today feel comfortable using computers as they begin their studies, but few are comfortable applying them to solve engineering problems. To close this gap, computational thinking and the development of associated skills must be integrated throughout the engineering curriculum [23]. There are some examples of classes and projects aimed at enabling engineering students to develop and increase CT through systematic introduction of computational tools, e.g. the creative use of MATLAB and LEGO Mindstorms integration [24]. Some articles concern the curriculum development in higher engineering and computer science education [25]. In his paper, Hu [26] presents a deep analysis of the computational thinking notion related to mathematical and other ways of thinking and one of the parts of the suggested extensive definition of CT states that CT is "engineering-oriented: to design the models and representations against known constraints and practical concerns, and to plan, execute, manage and evaluate the process of computation in order to improve our capability and maturity level".

To encourage and stimulate student's computational thinking abilities, we should focus on evaluation tools and methods. Our starting point is the observation that solving tasks can be one of the most useful tools for evaluating student's ability to think "computationally". We have a long-time experience in developing various tasks in computer science, as well as producing thousands of tasks for national, regional (Baltic countries) and international olympiads in informatics, contests, and other quizzes for secondary schools. One of the most popular outreach activities is a *Bebras* challenge on Informatics and Computational Thinking [27].

2.2 International challenge on informatics and computational thinking

The *Bebras* challenge is arranged annually in local languages over the world [28]. Actually, it is a contest but not only, because it focuses on more activities beyond the contest. The main contest consists of a set of tasks in a form of short questions (problems) or interactive tasks. Each task can both demonstrate the aspect of CS and test the aspects of CT of the participant. These tasks can be solved without prior knowledge about CS or computing, but are clearly related to fundamental CS concepts and aimed to develop CT of students. The tasks are developed collaboratively during an annual workshop, after which each country selects their own task set to translate and use in the local contest (all tasks for the *Bebras* challenge have been developed under the Creative Common BY-NC-SA licence). The focus is put on problem-solving activities that do not require any previous knowledge, and each task is categorized as belonging to one or several of the age and topic groups [28]. All tasks are accompanied with an explanation for how a given task was to be solved and a part called "It's informatics", which provides both teachers and students with some additional information on how a given task is related to CS or CT. The requirements for quality of the tasks have been already discussed [6, 29].

The tasks of the *Bebras* challenge involve students into CT operations. As we can see in the next sections of this paper, each task we have analysed has at least some components of CT: Abstraction (AB), Decomposition (DE), Algorithmic thinking (AL), Evaluation (EV), and Generalisation (GE) (CAS [22] concepts of CT have been used here). To solve these tasks, students are required to think in and about computer science, discrete structures, computation, data processing, data visualisation, but they also must use algorithmic as well as programming concepts.

The prior related research has studied the differences in task solving results of boys and girls and classification of the contest tasks using the Bloom taxonomy [30]. Within the scope of this research, we would like to study the CS concepts behind the CT tasks of the *Bebras* contest and the ability of novice programmer students to use them in the real-life context, modelled by the task.

3. Research methodology

In this research, we would like to bring up a hypothesis: "Well developed structured programming course, focused on the problem solving, develops computational thinking skills as well". A relevant measurement tool is needed, taking into account that CT is a latent trait and could be implemented in the problem solving process. To test the hypothesis, the programming language independent test has been used. The aim of the test is to evaluate students' CT skills and to compare the test results with the structured programming course results. The test process had several steps and actually used a combination of quantitative and qualitative testing approaches. The first step of the test process has been developed as a homogenous dichotomous test using the tasks of the *Bebras* contest as the test questions. When solving the presented tasks, students should employ CT skills, which we intend to measure as a latent trait. The next steps of the test process include questions requiring that students identify the CS concepts presented in the first step test tasks; solving CT problems by finding coding solutions to the presented tasks. The last two steps of the test process still need to be evaluated and are positioned as further work.

So the main *Research Questions* of this paper are:

- *RQ1*: How can the computational thinking skills of novice software engineering students be evaluated in the way independent of programming language?
- *RQ2*: What is the relation between novice software engineering students' computational thinking skills and programming course results?

Why these questions are of primary importance for us? We consider computational thinking skills as very important for further students' educational and professional career. Such skills form a basis for students' better understanding of further computer science knowledge and could sufficiently reduce the students' failure and drop-out rate. In order to answer the above research questions, we conducted a case study where the novice software engineering students, who studied structured programming, had to solve the test of preselected tasks.

The preparation phase of the research consisted of several steps. First, the appropriate tasks from the *Bebras* contest were selected. The main selection criteria were:

- *Computational thinking concepts*. Each task should have at least one well-expressed CT concept.
- Focus on *algorithmical thinking*. The test is aimed at software engineering students, therefore all the tasks are related to data structures, algorithms and their methods.
- *Difficulty level of problem solving*. Since the *Bebras* contest is mainly addressed to school pupils, and we are going to use them for the first-year higher education students studying software engineering, we selected the tasks that were considered as difficult with regard to the international experts' evaluations and the contest results.

The second phase included the analysis of the selected tasks, marking the main CS concepts, used in them. During the third step, an online quiz for students has been designed. The quiz included 10 selected tasks with a deep focus on CT. We used quantitative research methods to analyse the data. Test validity has been studied using the Item Response Theory.

4. Participants

Sixty-five first year (first semester) software engineering students, studying the structured programming course, took part in the experiment. The experiment was done with four groups of students, consisting of 16, 10, 20 and 19 students, respectively. A more detailed structure, including percentage of

Table 1. Basic characteristics of the participants of the experiment

			Maths maturity exam scores		IT maturity exam scores	
Total number	**Male**	**Female**	**Score**	**Students**	**Score**	**Students**
65	95%	5%	80–100	6.2%	80–100	10.8%
			60–80	12.3%	60–80	16.9%
			40–60	32.3%	40–60	27.7%
			20–40	36.9%	20–40	33.8%
			<20	9.2%	<20	10.8%

male and female students, scores in Maths and Information Technologies (IT) maturity exams, is presented in Table 1. We would like to notify that the IT maturity exam in Lithuania includes programming as well (50% of exam tasks require coding skills).

5. Results

The goal of the students participating in the study was to choose the correct answer to the presented task set. 10 tasks were selected with a deep focus on CT and followed by a set of CS concepts inside them prepared for the study.

The average overall result of the test was 54.2% of correct solutions. A more detailed structure of the correct answers per task is presented in Table 2.

Table 2. General results of the test per task

No.	CT task name	Correct answers
1	Beaverrail	38.5%
2	Beaver's log factory	33.8%
3	Bob's Best strategy	56.9%
4	0X	41.5%
5	Water supply	63.1%
6	Collecting candies	73.8%
7	Beaver in his canoe	86.2%
8	Constructive Beaver	21.5%
9	Sorting the Sticks	90.8%
10	Bebras-city streets	35.4%

5.1 Test structure

The main concepts, "encoded" inside the gamified tasks, were identified. They include main concepts of data structures, algorithms, methods, logical operations, and control structures (Table 3). The concept name is followed by the number that corresponds to the computational thinking task number, already referred to above in Table 2.

We present here an example of one of 10 tasks selected for a detailed study (Table 4).

We can identify the main components of computational thinking in this example of the task, as it was already mentioned in the Introduction and Background of the research sections of this paper:

- Abstraction (AB): from real objects (lakes, rivers) to abstract objects like a binary tree.
- Decomposition (DE): checking the rule, application of the rule to the parts of the tree.
- Algorithmical thinking (AL): the task itself provides an algorithm that should be understood and applied. This task, however, is not an example where students should develop their own algorithm to select the correct answer.
- Evaluation (EV): evaluation of all the correct decisions, evaluation of a set of wrong answers.
- Generalisation (GE): applying the algorithm rule to the whole tree, analysing the result in general.

The main CS concepts included in this task are

Table 3. The main concepts inside the computational thinking tasks

Data structures	Algorithms	Methods	Logics	Control structures
Binary tree (3)	Algorithm (1)	Dynamic programming (6)	Binary logical operation (5)	Loop (1, 4)
Graph (6)	Maximum element search (9)	Binary tree modelling (7)	Disjunction (5)	Function (8)
Logical data (4)	Modified sorting algorithm (9)	Tree traversal (3)	Equivalence (5)	Conditional sentence (1)
Array (6)	Depth-first search (3, 7)	Operation abstraction (8)	Inversion (5)	Conditional loop (9)
List structure (4)	Breadth-first search (7)	Optimization (10)	Conjunction (5)	
Directed graph (2)	Greedy algorithm (10)	Parameterization (8)		
Weighted graph (3)		Automation (7)		

Table 4. Task 7, "Beaver in his canoe"

Beaver paddles in his canoe on a river. The river has a number of little lakes (Fig. 1).

Beaver likes all lakes of the river and has thought of an algorithm to make sure that he reaches every lake.

He knows that at each lake there is a maximum of two rivers that he has not yet seen.

If beaver arrives at a lake he decides which river to take with the following rules:

- If there are two rivers he has not yet seen, he takes the river on his left hand side.
- If there is one river which beaver has not yet seen, beaver takes this river.
- If beaver has seen all the rivers from a little lake, he paddles his canoe one lake back towards the previous lake.

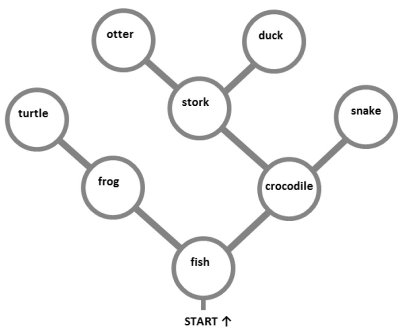


Fig. 1. Task "Beaver in his canoe"

Beaver stops his day of canoeing if he has seen everything and has come back to the start point. In Fig. 1 you can see the river and the little lakes where beaver paddles his canoe.

In each little lake beaver sees a different animal. Beaver writes down the animal name when he sees an animal for the first time.
In which order will beaver write down the animals?

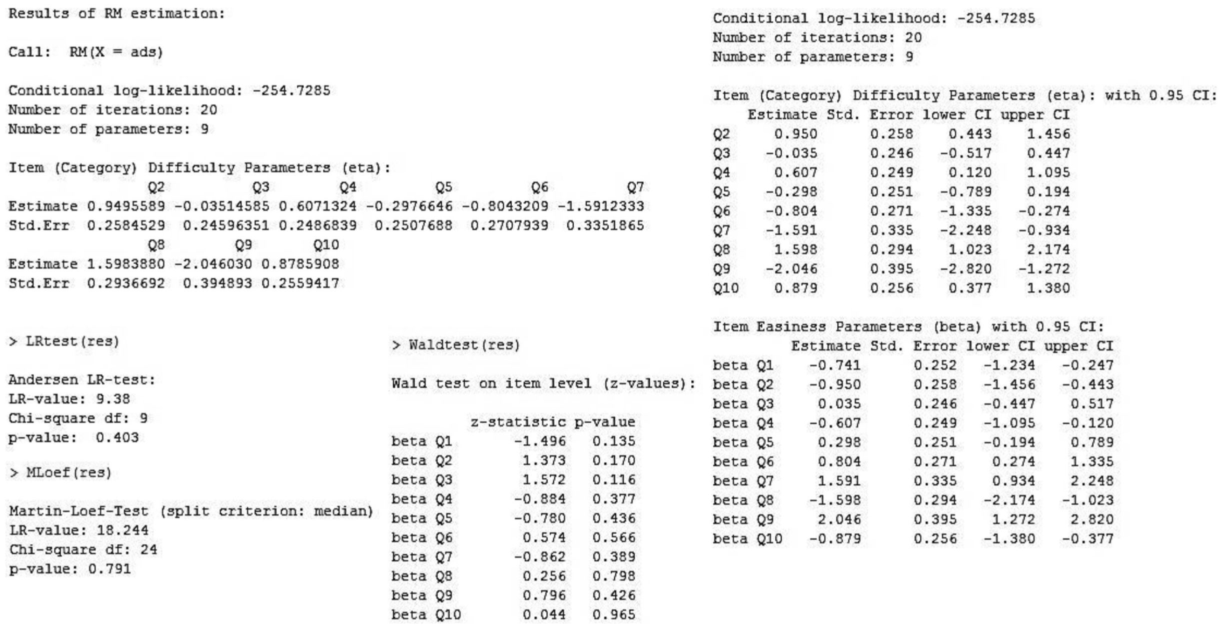
Answer (the correct answer is written in bold)
a. fish, frog, crocodile, turtle, stork, snake, otter, duck
b. fish, crocodile, snake, stork, duck, otter, frog, turtle
c. **fish, frog, turtle, crocodile, stork, otter, duck, snake**
d. fish, frog, turtle

binary tree modelling, *depth-first search*, and *automation*.

5.2 Test validity

Evaluating the test statistics we implement several basic considerations. First, we consider the test as a whole as a homogenous dichotomous test for evaluating students' implicit ability of computational thinking in the context of the process of solving gamified tasks that implicitly involve cognitive procedures of computational thinking. Next, we should make a note on the design and implementation of the test. We use a dichotomous scale for measuring the latent trait, scoring positively for the right solution.

We implement the Rash model for the evaluation and use environment R and eRm package to test model [31–34]. The result of estimation of the dichotomous model is presented in Figs. 2 and 3. The Wald test plot is presented in Fig. 4.



```
Results of RM estimation:

Call:  RM(X = ads)

Conditional log-likelihood: -254.7285
Number of iterations: 20
Number of parameters: 9

Item (Category) Difficulty Parameters (eta):
                Q2          Q3        Q4         Q5         Q6         Q7
Estimate 0.9495589 -0.03514585 0.6071324 -0.2976646 -0.8043209 -1.5912333
Std.Err  0.2584529  0.24596351 0.2486839  0.2507688  0.2707939  0.3351865
                Q8        Q9       Q10
Estimate 1.5983880 -2.046030 0.8785908
Std.Err  0.2936692  0.394893 0.2559417
```

```
> LRtest(res)

Andersen LR-test:
LR-value: 9.38
Chi-square df: 9
p-value:  0.403

> MLoef(res)

Martin-Loef-Test (split criterion: median)
LR-value: 18.244
Chi-square df: 24
p-value: 0.791
```

```
> Waldtest(res)

Wald test on item level (z-values):

          z-statistic p-value
beta Q1        -1.496   0.135
beta Q2         1.373   0.170
beta Q3         1.572   0.116
beta Q4        -0.884   0.377
beta Q5        -0.780   0.436
beta Q6         0.574   0.566
beta Q7        -0.862   0.389
beta Q8         0.256   0.798
beta Q9         0.796   0.426
beta Q10        0.044   0.965
```

```
Conditional log-likelihood: -254.7285
Number of iterations: 20
Number of parameters: 9

Item (Category) Difficulty Parameters (eta): with 0.95 CI:
    Estimate Std. Error lower CI upper CI
Q2     0.950      0.258    0.443    1.456
Q3    -0.035      0.246   -0.517    0.447
Q4     0.607      0.249    0.120    1.095
Q5    -0.298      0.251   -0.789    0.194
Q6    -0.804      0.271   -1.335   -0.274
Q7    -1.591      0.335   -2.248   -0.934
Q8     1.598      0.294    1.023    2.174
Q9    -2.046      0.395   -2.820   -1.272
Q10    0.879      0.256    0.377    1.380

Item Easiness Parameters (beta) with 0.95 CI:
         Estimate Std. Error lower CI upper CI
beta Q1    -0.741      0.252   -1.234   -0.247
beta Q2    -0.950      0.258   -1.456   -0.443
beta Q3     0.035      0.246   -0.447    0.517
beta Q4    -0.607      0.249   -1.095   -0.120
beta Q5     0.298      0.251   -0.194    0.789
beta Q6     0.804      0.271    0.274    1.335
beta Q7     1.591      0.335    0.934    2.248
beta Q8    -1.598      0.294   -2.174   -1.023
beta Q9     2.046      0.395    1.272    2.820
beta Q10   -0.879      0.256   -1.380   -0.377
```

Fig. 2. Dichotomous model. Results of the Rash model estimation, Andersen's Likelihood Ration and Martin-Loef tests. Summary of the Rash model estimation.

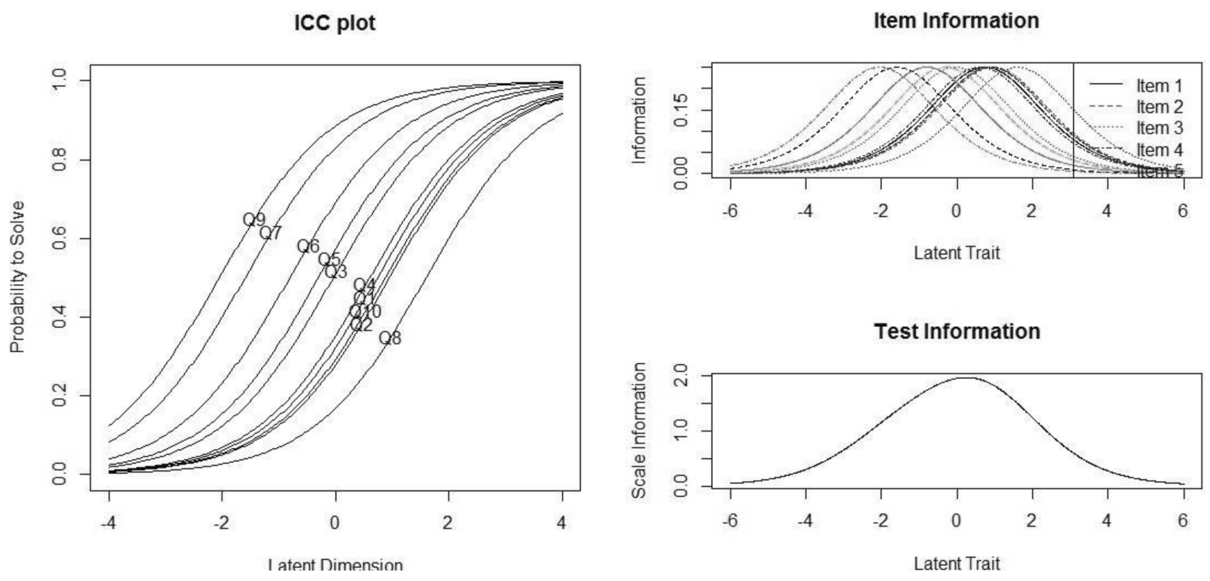


Fig. 3. Dichotomous model. Item characteristics curves. Curves of items and test information.

Figure 2 shows easiness of parameter estimation of the Rash model item for 10 test items, standard errors of items, and 95% confidence intervals of items. For this study eight eta parameters are significantly different from zero ($p < 0.05$) with five negative (Q1, Q2, Q4, Q8, Q10) and three positive (Q6, Q7, Q9) values. The confidence intervals of the other two eta parameters (Q3, Q5) include zero.

We check fitness of the Rash model according to Andersen's likelihood ratio test. The mean of raw scores was chosen as the partitioning criterion. The significance level we specify is equal to 0.05. The computed p-value ($p = 0.403$) shows that the likelihood ratio test results are non-significant and, therefore, the Rash model holds for the data.

Martin-Loef's test evaluates unidimentionality of two sets of items. The calculated p-value for the Martin-Loef's test is 0.791. As well as Andersen's test, Martin-Loef's test confirms the Rash model data. Wald's test shows on non-significant difference for all the test items as seen from the plot (Fig. 4). Item characteristics curves show closeness to the uniform distribution of test items (Fig. 3).

The test statistics confirms the general validity of the test.

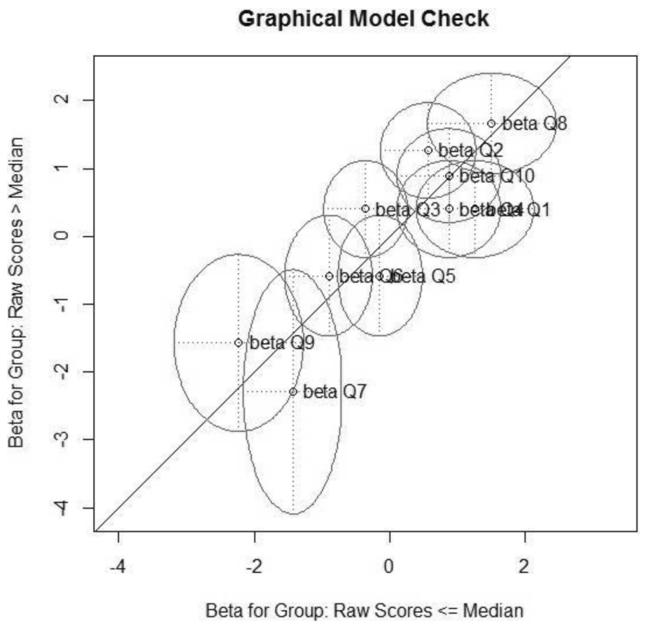


Fig. 4. Dichotomous model. The Wald test plot.

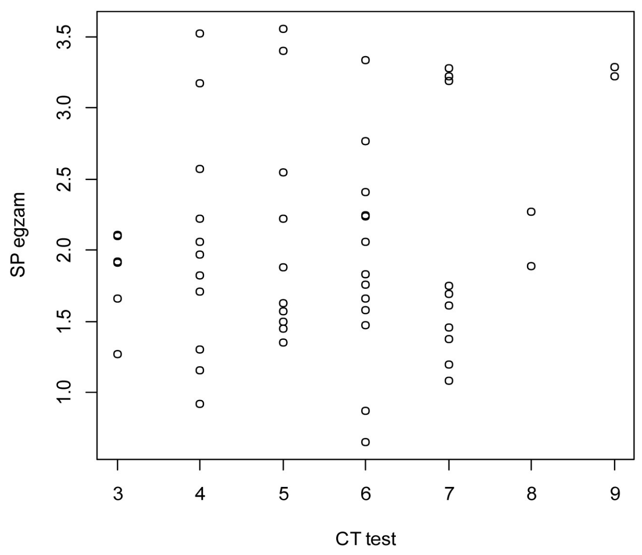


Fig. 5. The plot of the CT test vs. the structured programming course exam.

5.3 Correlation results

In order to answer our second research question we measure and compare the measured latent abilities with the results of the structured programming course. The positive correlation would indicate a success of the structured programming course. The correlation plot of the test results and the structured programming course examination results is presented in Fig. 5. The plot shows no correlations.

One of the reasons of correlation absence between the test results and course results may be that the exam of the course is still procedural, logical thinking oriented and should be re-developed to address more problem solving skills.

6. Discussion and future work

The case study, presented in this paper, was aimed to investigate how the problem solving oriented structured programming course enhances computational thinking skills of novice software engineering students. The type of students' activity of the during this study (game-like computer science task with the concepts "encoded" into it) was not only to test students, but also can be useful as a learning activity for the computer science/engineering students. The activity itself may be highly motivational to learn new CS concepts and develop CT skills.

The case study has given some controversial results. This case study may be considered as a first step in a series of research activities. The next steps of the research would be to analyse, how novice software engineering students explicitly identify the presence of CS concepts in computational thinking tasks, create game-like computer programs, based on the computational thinking tasks, and use the main concepts in their programming activities. This fact would possibly bring more light on the reasons of the problems and limitations identified in this case study.

The results have also pointed out the problems of the structured programming course itself that should be improved. Using the programming language-independent test, presented in this paper, the influence of different didactic approaches on the improvement of CT skills could be studied and compared.

7. Conclusion

1. CT tasks used in this study are designed for secondary school students. Unless the tasks were designed to use in a contest setting, it was quite surprising there were only 54.2% of correct answers in general from the first year software engineering students. Possibly, the reason for that is a diverse and insufficient preparation for computer science on a school level, not enough addressing the problem solving and computational thinking skills.
2. The statistical evaluation of the test used in this study, has shown the validity of the test as an instrument to evaluate computational thinking.
3. In spite of our expectations, the test did not present any correlation between CT skills and the structured programming course. It means that the course has to be improved. At this point of our research, we cannot either confirm or reject the hypothesis we have raised at the beginning of this paper. The course improvement strategy could be based on the analysis of the structure of test tasks, studying CS concepts

and shifting the course and its exam structure to the test shown most difficult to solve problems.

References

1. Royal Society, *Shut down or restart: The way forward for computing in UK schools*, http://royalsociety.org, 2012. [Retrieved September 22, 2015]
2. V. Barr and C. Stephenson, Bringing computational thinking to k-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, **2**(1), 2011, pp. 48–54.
3. S. Easterbrook, From Computational Thinking to Systems Thinking: A conceptual toolkit for sustainability computing, In: *Proceedings of the 2nd international conference on information and communication technologies for sustainability (ICT4S'14), Stockholm, Sweden*, 2014.
4. J. M. Wing, Computational thinking, *Communications of the ACM*, **49**(3), 2006, pp. 33–35.
5. V. Dagienė, Information technology contests—introduction to computer science in an attractive way, *Informatics in Education*, **5**(1), 2006, pp 37–46.
6. V. Dagienė and G. Futschek, Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks, *LNCS* 5090, 2008, pp. 19–30.
7. V. Dolgopolovas, V. Dagienė, S. Minkevičius and L. Sakalauskas, Python for Scientific Computing Education: Modeling of Queueing Systems, *Scientific Programming*, IOS press, **22**(1), 2014, pp. 37–51.
8. V. Dolgopolovas, V. Dagienė, S. Minkevičius and L. Sakalauskas, Teaching Scientific Computing: A Model-Centered Approach to Pipeline and Parallel Programming with C, *Scientific Programming*, Article ID 820803, 18 p., 2015.
9. C. Dierbach, Python as a first programming language, *Journal of Computing Sciences in Colleges*, **29**(6), 2014, pp. 153–154.
10. W. Xinxiu, Z. Yang, Z. Wang and M. Zhao, Exploration on Cultivating Students' Abilities Based on Python Teaching Practice. In: *2nd International Conference on Teaching and Computational Science (ICTCS 2014),* Atlantis Press, 2014.
11. A. Theodoraki and S. Xinogalos, Studying Students' Attitudes on Using Examples of Game Source Code for Learning Programming, *Informatics in Education*, **13**(2), 2014, pp. 265–277.
12. P. Kinnunen and L. Malmi, Why students drop out CS1 course? *Proceedings of the second international workshop on computing education research*, September 9–10, 2006, Canterbury, United Kingdom.
13. L. De-La-Fuente-Valentín, A. Pardo and C. D. Kloos, Addressing drop-out and sustained effort issues with large practical groups using an automated delivery and assessment system, *Computers & Education*, **61**, 2013, pp. 33–42.
14. F. Araque, C. Roldán, A. Salguero, Factors influencing university drop-out rates, *Computers & Education*, **53**(3), 2009, pp. 563–574.
15. J. McKee-Scott and L. Patricia, Educational games for CS1: raised questions, *WCCCE '11 Proceedings of the 16th Western Canadian Conference on Computing Education*, ACM, 2012, pp. 46–46.
16. H. Kinnari-Korpela, Using Short Video Lectures to Enhance Mathematics Learning—Experiences on Differential and Integral Calculus Course for Engineering Students, *Informatics in Education*, **14**(1), 2015, pp. 69–83.
17. M. Magdin and M. Turčáni, A Few Observations and Remarks on Time Effectiveness of Interactive Electronic Testing, *Informatics in Education*, **14**(1), 2015, pp. 85–104.
18. J. M. Wing, *Computational Thinking: What and Why*, 2011, http://www.cs.cmu.edu/link/researchnotebook-computational-thinking-what-and-why, Accessed September 22, 2015.
19. A. E. Weingberg, *Computational thinking: an investigation of the existing scholarship and research*, Dissertation, Colorado State University, 2013.
20. ISTE&CSTA (International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA)), *Operational definition of computational thinking for K-12 education,* 2011.
21. Google, *Exploring Computational Thinking*, 2011. http://www.google.com/edu/computational-thinking/ [Retrieved September 4, 2015]
22. CAS (Computing at School), *CAS computational thinking guidance for teachers*, 2015, http://community.computingatschool.org.uk/resources/2324 [Retrieved September 22, 2015]
23. C. Mohtadi, M. Kim and J. Schlosser, Why integrate computational thinking into a 21st century engineering curriculum? *Proceedings of SEFI Annual Conference*, article no 62, Leuven, Belgium, 2013, http://www.sefi.be/conference-2013/authors.html [Retrieved April 17, 2016].
24. S. Gross, M. Kim, J. Schlosser, D. Lluch, C. Mohtadi and D. Schneider, Fostering computational thinking in engineering education: Challenges, examples, and best practices. In: *Global Engineering Education Conference (EDUCON)*, IEEE, 2014, pp. 450–459.
25. C. E. Vergara, M. Urban-Lurain, C. Dresen, T. Coxen, T. MacFarlane, K. Frazier, D. Briedis, N. Buch, A.-H. Esfahanian, L. Paquette, J. Sticklen, J. LaPrad and T. F. Wolff, Aligning Computing Education with engineering workforce computational needs: New curricular directions to improve computational thinking in engineering graduates, In: *39th IEEE Conference on Frontiers in Education 2009*, 2009, pp. 1–6.
26. C. Hu, Computational thinking: what it might mean and what we might do about it, In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, ITiCSE'11*, ACM, 2011, pp. 223–227.
27. Bebras.org, *International Challenge on Informatics and Computational Thinking*, http://www.bebras.org/ [Retrieved September 22, 2015]
28. V. Dagiene and G. Stupuriene, Bebras—a Sustainable Community Building Model for the Concept based Learning of Informatics and Computational Thinking, *Informatics in Education*, **15**(1), 2016, pp. 25–44.
29. J. Vanicek, Bebras Informatics Contest: Criteria for Good Tasks Revised, In: Gülbahar, Yasemin, Karataş, Erinç (Eds.) *Informatics in schools: teaching and learning perspectives: 7th international conference on informatics in schools: situation, evolution, and perspectives, ISSEP 2014, Istanbul, Turkey, September 22–25, 2014: proceedings*, *LNCS 8730*, 2014, pp. 17–28.
30. V. Dagienė and G. Stupurienė, Informatics Education based on Solving Attractive Tasks through a Contest, *Conference KEYCIT 2014*, 2014, pp. 51–62.
31. R. Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
32. R-forge, eRm Project, http://r-forge.r-project.org/projects/erm/ [Retrieved September 22, 2015]
33. P. Mair, R. Hatzinger and J. M. Maier, *Extended Rasch Modeling: The R Package eRm*, PDF-Dateianhang zum Programmpaket eRm, 2009.
34. F. De Battisti, G. Nicolini and S. Salini, The Rasch Model, *Modern Analysis of Customer Surveys: with Applications using R*, 2012, pp. 259–281.

Vladimiras Dolgopolovas received a degree in civil engineering from the Vilnius Gediminas Technical University and a second degree in mathematics and computer science teaching from Vilnius University. He has participated in research projects on embedded systems, software defined radio and industrial networks. He is a lecturer of Computer Science at Vilnius University of Applied Science and a member of research staff at Institute of Mathematics and Informatics of Vilnius University. His research interests include numerical analysis, operational research, and applications to teaching and learning.

Tatjana Jevsikova is a researcher at the Vilnius University Institute of Mathematics and Informatics as well as an associate professor at the Vilnius University Faculty of Mathematics and Informatics. She received her PhD in computer science. Her main research interests include informatics education, e-learning, teacher training, software localization, and cultural aspects of human-computer interaction. She is the author (or a co-author) of more than 20 scientific papers, several methodological books and dictionaries of computer science terms. She participated in several EU-funded R&D projects, as well as in a number of national research studies, connected with technology in education, as well as software localization.

Valentina Dagienė is a professor at Vilnius University, Lithuania (MS in Applied Mathematics, PhD in Computer Science, Dr. Habil in Education). Her research interests focus on informatics and informatics engineering education, teaching algorithms and programming, and localization of educational software. She has published over 200 scientific papers and methodological works, has written more than 50 textbooks in the field of Informatics and Information Technology for primary and secondary education. She works in various expert groups and work groups, organizing the olympiads and contests. She is Editor of international journals "Informatics in Education" and "Olympiads in Informatics". She has participated in several EU-funded R&D projects, as well as in a number of national research studies connected with information technology and education.

Loreta Savulionienė received a master's degree in mathematics and a PHD degree in informatics from Vilnius University. She is a vice dean of the Faculty of Electronics and Informatics and a lecturer at Vilnius University of Applied Sciences. In 2013, she was awarded a grant for best Lithuanian programming lecturers. Her research interests include computer science, programming education research, educational planning and organization of an educational process.