ALGORITHMS, PROGRAMMING, FLOWCHARTS AND FLOWGORITHM

R. Robert Gajewski
Warsaw University of Technology, Faculty of Civil Engineering
rg@il.pw.edu.pl

Abstract. *The paper tries to answer the question – can the basics of algorithms and programming at faculties other than computer science (informatics) be taught more effectively using spreadsheets, computer algebra systems and e-Learning tools and materials like e-Books, software animations and specialized flowchart software. The first part of the paper gives a critical review of the literature of the subject. In the second part of the paper the programme of an applied computer science course devoted to algorithms programming is presented. The third part shows results of two surveys.*

Keywords: computational thinking, software animations, flowcharts.

INTRODUCTION

How to teach algorithms and programming as part of computational thinking (Wing, 2006) is still an open question (Wolfram, 2016). Sleeman (Sleeman, 1986) described programming as the new Latin of the school syllabus. Even there are developments in ITC programming is still causing problems (Gomes & Mendes, 2007) perhaps because of the fact that it includes knowledge of appropriate tools and languages, problem-solving skills and strategies for program design and implementation.

1. LITERATURE REVIEW

One of the first articles on experimental investigations of the utility of detailed flowcharts in programming was written in 1977 (Shneiderman, Mayer, McKay, & Heller, 1977). Later there were theses prepared on design and implementation of a tool for teaching programming (Goktepe, 1988) and about visual programming (Nickerson, 1994). There is also a whole book written on software visualization (Diehl, 2002). Baldwin and Kuljis presented in Balwdin & Kuljis (2001) the way of learning programming using program visualization

techniques. Books written by Gaddis (Gaddis, 2015) and Venit (Venit & Drake, 2014) give an excellent framework for programming course on any level. A review and discussion of problems in learning and teaching programming is created by Robins (Robins, Rountree, & Rountree, 2003).

1.1 Choice of the flowchart tool

There are many flowchart-based programming environments for improving comprehension and problem-solving skills of novice programmers (Hooshyar, Ahmad, Nasir, Shamshirband, & Horng, 2015). Three of them were tested during the last few years:

- LARP - Logic of Algorithms for Resolution of Problems created by Marco Lavoie (the last version is from 2008)
- RAPTOR – Rapid Algorithmic Prototyping Tool for Ordered Reasoning created by Martin Carlisle and described in many articles (Carlisle, Wilson, Humphries, & Hadfield, 2005, Carlisle, 2009 and Thompson, 2012) (the last version is from April 2015)
- FLOWGORITHM – created by Devin Cook (the last version 2.18.3 is from November 2018).

The third one, Flowgorithm, was chosen mainly for three reasons. This was students' favourite code, it is still being developed and it was possible to create its localization (translation). The main Flowgorithm features are as follows: easy to understand output, graphical variable watch window, interactively generated code (for 12+ languages), safe recursion, loops, arrays, and flexible expressions and multilingual support. Moreover, there is an e-book created by Roberto Atzori with more than 250 flowcharts.

To some extent ALVIS Live! (ALgorithm VIsualization Storyboarder) represents a similar idea. It is the part of the VEUPL project (Visualization and End User Programming Lab), whose leader was Chris Hundhausen. The program, of which the last version is from September 2006, was described in many papers, e.g. (Hundhausen & Douglas, 2002) and (Hundhausen & Brown, 2005). More information about the flowchart-based programming environments for improving comprehension and problem-solving skills of novice programmers can be found in (Hooshyar et al., 2015). The use of a flowchart interpreter for the introductory programming course was presented by Crews and Ziegler in Crews & Ziegler (1998). Kuen (Kuen, 2011) described the learning programming concepts using flowcharting software. A similar problem – an animated flowchart with an example to teach the algorithm based courses in engineering was published by Dol (Dol, 2015).

2 FUNDAMENTALS OF COMPUTER SCIENCE

Fundamentals of the course in Computer Science at the Faculty of Civil Engineering at Warsaw University of Technology have been already described in many publications like Gajewski, Wlasak, & Jaczewski (2013) and Gajewski & Jaczewski (2014). Algorithms and programming are only a part of the course consisting of three hours of lectures and six hours of classes. The computer algebra system Mathcad Prime (Gajewski, 2014) is used for this course with some elements of blended learning. A similar approach was presented by Azemi in Azemi & Pauley (2008) and Asad Azemi, Bodek, & Chinn (2013). Basic and introductory programming courses frequently cause problems. Giannakos (Giannakos, Pappas, Jaccheri, & Sampson, 2016) tried to understand student retention in computer science education. Rahmat discussed (Rahmat et al., 2012) major problems in basic programming that influence students' performance. In another paper Zainal (Zainal et al., 2012) investigated students' perception and motivation towards programming. The answer to the question how to reduce the dropout rate in an introductory programming course (Yadin, 2011) is still open. More information about teaching and learning programming can be found in the review papers written by Ala-Mutka (Ala-Mutka, 2004) and Pears (Pears et al., 2007).

2.1 Basic Algorithmic Problems

During lectures three basic and classical algorithmic problems which do not require deep mathematical knowledge are presented. Their excellent description can be found also in Wikipedia.

Square root – Babylonian method. Algorithm is described precisely even in Wikipedia: "The basic idea is that if x is an overestimate to the square root of a non-negative real number S then S/x will be an underestimate and so the average of these two numbers may reasonably be expected to provide a better approximation"

Root of the function – bisection method is described in Wikipedia as follows. "At each step the method divides the interval in two by computing the midpoint c = (a+b) / 2 of the interval and the value of the function f(c) at that point. Unless c is itself a root (which is very unlikely, but possible) there are now only two possibilities: either f(a) and f(c) have opposite signs and bracket a root, or f(c) and f(b) have opposite signs and bracket a root. The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step."

Greatest common divisor – Euclidean algorithm. According to Wikipedia definition: "The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD

of the original two numbers." All these algorithms are discussed during lectures using Flowgorithm (see Fig. 1).

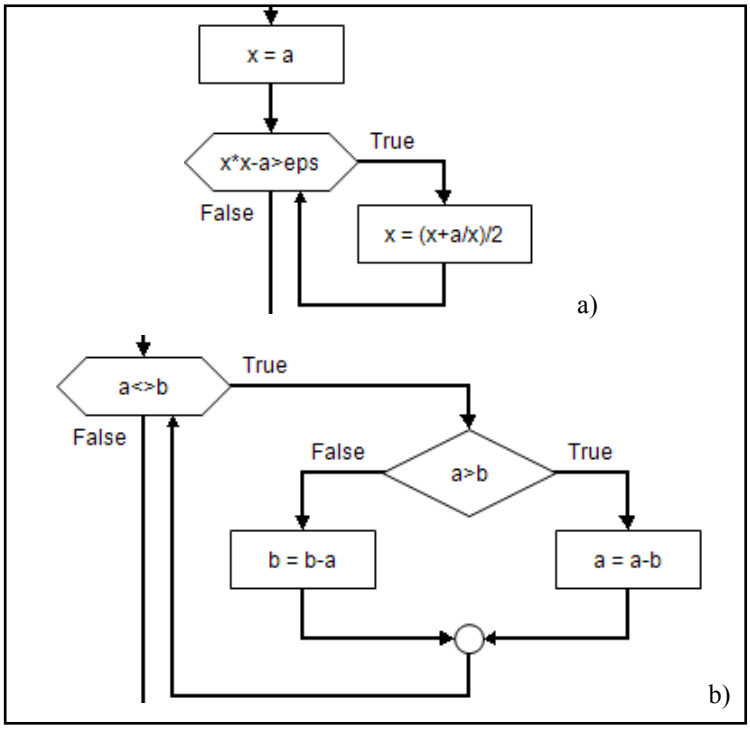

Figure 1. Flowcharts of the Babylonian method (a) and Euclidean algorithm (b)

Source: Own work

2.2 Branching

If a statement (branching) is for the first time introduced in a spreadsheet for simple problems like a function given by distinct formulas for different ranges of an argument. In the case of three intervals nested if is used (see Fig. 2).

$$f(x)=\begin{cases}-x & x<-1\\ 1 & x\in[-1,1]\\ x & x>1\end{cases}$$

```
=IF(A1<-1,-A1,IF(A1>1,A1,1))
```

Figure 2. Nested if in a spreadsheet

Source: Own work

2.3 Looping

Loops are not available directly in a spreadsheet, but in the case of iterative calculations they can be simulated by expanding formulas as for the case of a sum of elements (see Fig. 3).

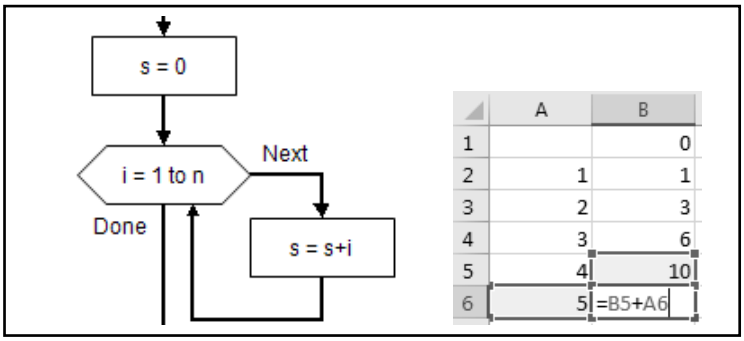

Figure 3. Sum of integers – flowchart and for loop in a spreadsheet

Source: Own work

While a loop is used for two cases of stopping condition for a sum of elements imposed on the value of added elements or on the value of a sum (see Fig.4).

2.4 Sample exam problems

All exam problems belong to one of the two groups:

- for loop together with if branching (vectors and matrices and their elements);
- while loop (sums of series, expansion to series)

Sample exam problems are as follows:

- Create function that calculates the average of matrix elements from the range (a,b);
- Create function that expands to the Taylor series centred at zero (Maclaurin series) cosine function; add only elements greater than eps.

The solution of these problems is very simple. Sample codes have only a few lines (see Fig. 5). General structure of the code can be easily memorized but a solution of each problem requires understanding of the algorithm. Flowgorithm helps to understand how algorithms work especially enabling to follow calculations in an automatic way

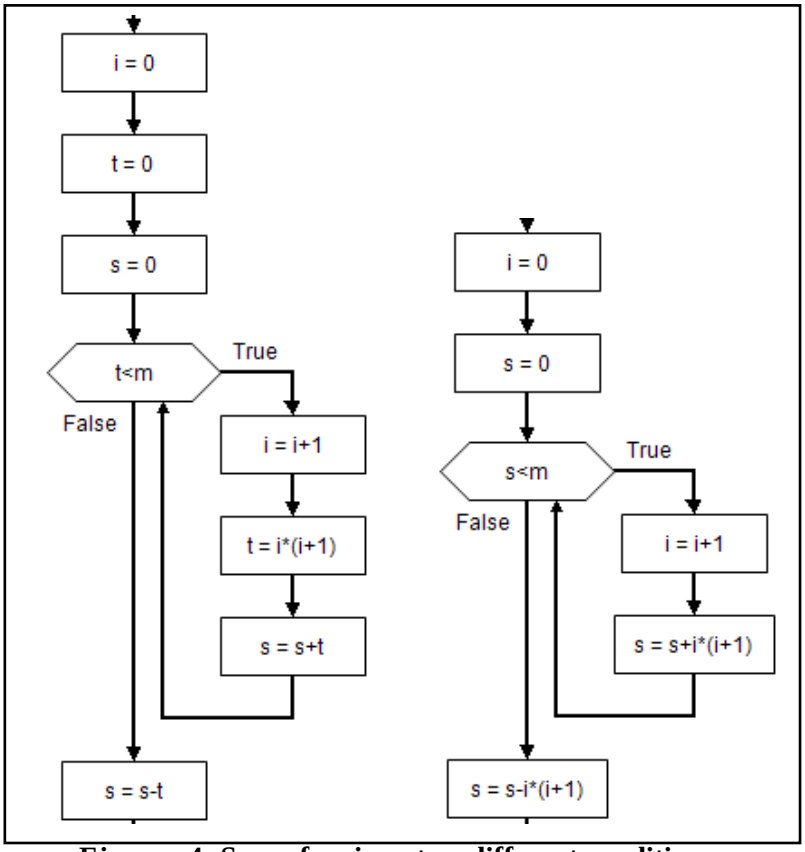

Figure 4. Sum of series – two different conditions

Source: Own work

2.5 e-Learning materials and tools

All educational resources are available on the faculty Moodle platform with materials like quizzes (self-assessment tests) and software animations. There are two books about Mathcad Prime prepared especially for the course. There is also a portal dedicated to Polish version of a book (prime.il.pw.edu.pl). In the forthcoming academic year active software simulations will also be available. All educational materials are very popular among Students but unfortunately mainly just before the exams. Students are definitely reluctant to work in a systematic way.

3 SURVEYS AND THEIR RESULTS

In order to learn what students' experiences are like in designing algorithms and programming, difficulties with different teaching topics and favourite learning resources two surveys were conducted.

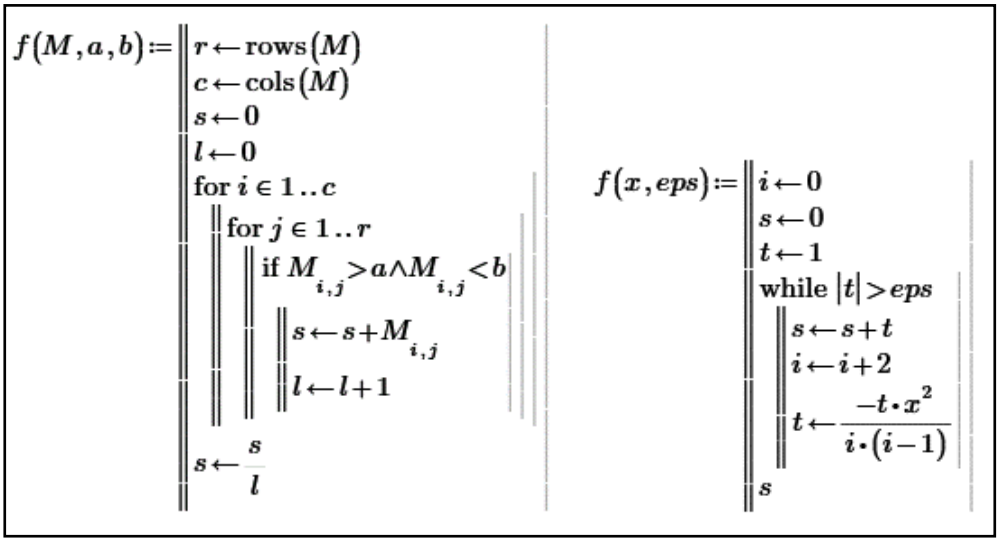

Figure 5. Solution of sample exam problems

Source: Own work

3.1 Surveys methodology

Surveys took place at the very end of semester in January 2017. Participation in the surveys was not compulsory but students were asked to participate in them in order to improve quality of the classes. Anonymous questionnaires were filled by 136 students out of 186 attending classes. The whole process was partly automatic – Google Forms were used to collect the data. For all surveys Cronbach's α coefficients (Cronbach, 1951, Cronbach & Shavelson, 2004) as a lower bound estimate of the reliability of psychometric test were calculated. This coefficient should be at least 0.6.

3.2 First survey

The first survey was based on Konecki's research described in (Konecki, 2014), (Konecki, 2015) and (Konecki & Petrlic, 2014). Likert scale was used for all given questions (1-strongly disagree, 10-strongly agree). Results for questions concerning experiences in designing algorithms and programming (Table 1) are different than obtained by Konecki, whose research was conducted among 190 students of information science. This is mainly due to the facts that civil engineering students do not like algorithms and programming. Cronbach's α is for this test 0.8301.

3.3 Second survey

The second survey was based on another questionnaire (Malik & Coldwell-Neilson, 2016). In the first part of the second survey the five-point Likert scale is used, from very difficult to learn (1) to very easy to learn (5). The answers to the questions regarding difficulties with different teaching topics (Table 2) show that repetition and selection as well as functions belong to the group of very difficult to

learn topics. This was visible during practical tests. Choice of an appropriate loop (for or while), was the biggest problem for students. Cronbach's α is for this test 0.7927.

Table 1.

Reported experiences in designing algorithms and programming

Questionnaire Item	Mean
I have no difficulties in understanding of programming problems that are presented to me	4.000
When solving programming task, I have difficulties in understanding the task itself	5.471
I have difficulties in drawing a diagram or writing a pseudocode of a given programming task's solution	5.434
I have more problems in visualizing and designing a conceptual solution in a pseudocode than in understanding and remembering programming language syntax	5.397
Designing of algorithmic solutions is difficult and not intuitive to me	5.610
The main problem I experience is remembering programming language syntax	5.169
The main problems I experience refer to understanding and visualizing programming tasks and designing their algorithmic solutions	5.518

Source: Own work inspired by Konecki

Table 2.

Teaching topics

I found…	Mean	Very difficult to learn	Difficult to learn	Neutral	Easy to learn	Very easy to learn
Arrays	3.345	11	13	48	43	21
Expressions	3.463	4	18	48	43	23
Functions	2.845	19	37	38	30	12
Operators	3.434	7	15	48	44	22
Parameters	3.338	3	23	49	47	14
Repetition	2.904	14	37	45	28	12
Selection	3.074	10	32	50	26	18
Variables	3.346	5	21	52	38	20

Source: Own work inspired by Malik & Coldwell-Neilson

In the second part of the survey the five-point Likert scale is used. Questions related to the learning situation use a scale of never (1) to always (5). The answers to the questions regarding learning situations (Table 3) show, that lectures never or rarely helped in learning programming. Students treat programming as something practical, so they do prefer to learn programming during lab sessions. Cronbach's α is for this test 0.4866.

Also in the last part of the survey the five-point Likert scale is used. Questions relating to the teaching materials use a scale of useless material (1) to very useful material (5). The answers to the questions regarding teaching and learning resources (Table 4) show, that students treat the introductory course book and lecture notes as mainly useless, not very useful or somewhat useful. Software animations (movies), exercise questions and answers and example programs are treated as useful or very useful resources. Students rarely attend lectures and they do prefer to watch in a passive way movies rather than actively read a book. Cronbach's α is for this test 0.6746.

Table 3.

Learning situations

I learned about programming…	Mean	Never	Rarely	Sometimes	Often	Always
In lectures	1.889	64	37	24	8	3
In lab sessions	3.434	6	19	43	46	22
While studying alone	3.456	6	20	43	40	27
While working alone on programming coursework	3.485	5	21	41	41	28
In exercise sessions in small groups	2.397	42	31	38	17	8

Source: Own work inspired by Malik & Coldwell-Neilson

In the next phase of this research self-assessment of the course using Bloom's revisited taxonomy like in Alaoutinen & Smolander (2010) and investigation of test reliability including Guttman's lambda-2 (Guttman, 1945) are planned. Moreover multiple choice tests will be used to evaluate student understanding during computer programming classes (Kuechler & Simkin, 2003).

Table 4.

Teaching and learning resources

I found the…	Mean	Useless	Not very useful	Somew hat useful	Useful	Very useful
Introductory course book	2.449	38	36	35	17	10
Lecture notes	2.073	58	27	36	13	2
Exercise questions and answers	4.058	3	8	23	46	56
Example programs	3.926	6	5	29	49	47
Still pictures of programming structures	3,250	10	22	50	32	22
Interactive visualizations	3.324	15	15	44	35	27
Movies (software animations)	4,132	2	9	23	37	65

Source: Own work inspired by Malik & Coldwell-Neilson

CONCLUSION

This research was inspired by the Cognitive-Affective Theory of Learning with Media (CATLM) created by Moreno and presented in Moreno (2005, 2006). CATLM represents an expansion of the popular Cognitive Theory of Multimedia Learning (CTML) reported by Mayer in his book "Multimedia Learning" (Mayer, 2001) and later by Sorden in "Handbook of Educational Theories" (Sorden, 2013). CATLM assumes that students need to become motivated to make full use of their cognitive resources (Park, Plass, & Brünken, 2014). All tutors in the presented course were specialists in Computational Thinking but perhaps students had not enough motivation for learning which was the reason of problems and bad results.

The question raised five years ago – "how to motivate digital natives to learn" (Wlasak, Jaczewski, Dubilis, & Warda, 2013) is still open. Students are generally against programming. They are absolutely satisfied even by their poor knowledge of IT limited to some basic editing skills. Results of 258 tests and retakes in Mathcad clearly show it.

The examination consisted of twelve problems – ten devoted to calculations and two to programming. The total score is fourteen points – ten from calculations and four from programming. Results of these tests show that students try to avoid

problems in programming and do prefer to gain points from simple calculation problems.

According to OECD Report "Students, Computers and Learning - Making the Connection"(OECD, 2015) students who use computers at school only moderately score the highest in reading. Moreover, students who do not use computers in maths classes score higher results in mathematics. Perhaps the same observation is valid for algorithmics and programming. Overuse of technology can lead to worse results.

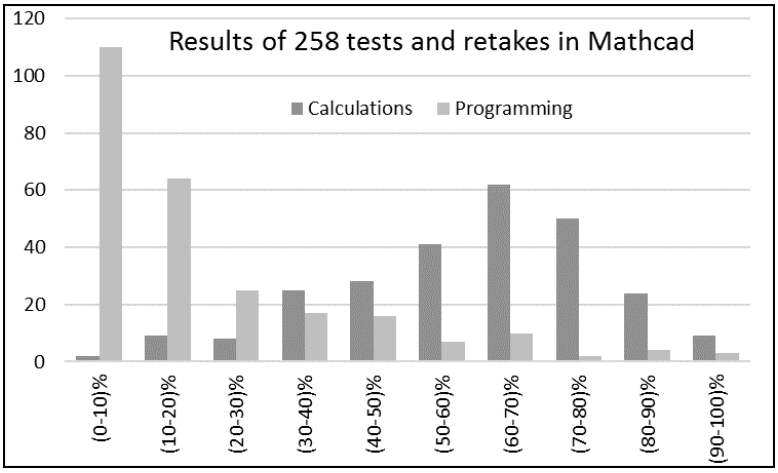

Figure 6. Results of tests in Mathcad

Source: Own work

Flowgorithm proved to be a very effective lecture tool allowing to present algorithms and their results. During laboratories Flowgorithm was used mainly only when students were obliged to do this, which is the result of negative attitude to programming. Flowgorithm enabled to distinguish between programming (creating an algorithm) and coding (representing an algorithm in a particular programming language) and concentrate on algorithms and programming. The next question – how to assure digital natives that computational and algorithmic thinking as well as programming skills are essential for all engineers is also open.

How to use in effective way algorithm animations for teaching and learning is still an open research question (Fleischer & Kucera, 2002), (Végh & Stoffová, 2017). Another important research issue is Technology Acceptance Model (TAM) (Adams, Nelson, & Todd, 1992) used to measure and evaluate perceived usefulness, ease of use, and usage of information technology. TAM can be exercised to measure continuance intention to use MOOCs (Wu & Chen, 2017) and to measure users' acceptance of e-Learning (Tarhini, Hone, Liu, & Tarhini, 2017).

Acknowledgments

The author would like to thank all students who participated in surveys and filled two long questionnaires. Research was conducted within a frame of grant No. 504/03550/1088/40.

REFERENCES

Adams, D. A., Nelson, R. R., & Todd, P. A. (1992). Perceived Usefulness, Ease of Use, and Usage of Information Technology: A Replication. MIS Q., 16(2), 227–247. https://doi.org/10.2307/249577

Ala-Mutka, K. (2004). Problems in Learning and Teaching Programming - a literature study for developing visualizations in the Codewitz-Minerva project (p. 13). Institute of Software Systems, Tampere University of Technology, Finland. Retrieved from https://www.cs.tut.fi/~edge/literature_study.pdf (accessed 27.11.2018)

Alaoutinen, S., & Smolander, K. (2010). Student self-assessment in a programming course using Bloom's Revised Taxonomy. In Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education (pp. 155–159). https://doi.org/10.1145/1822090.1822135

Azemi, A., & Pauley, L. L. (2008). Teaching the introductory computer programming course for engineers using Matlab. In Frontiers in Education Conference, 2008. FIE 2008. 38th Annual (pp. T3B-1–T3B-23). IEEE. https://doi.org/10.1109/FIE.2008.4720302

Azemi, Asad, Bodek, M., & Chinn, G. (2013). Teaching an introductory programming course using hybrid e-learning approach. In Proceedings - Frontiers in Education Conference, FIE (pp. 1911–1913). https://doi.org/10.1109/FIE.2013.6685168

Baldwin, L. P., & Kuljis, J. (2001). Learning programming using program visualization techniques. In Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001 (p. 8 pp.). USA: IEEE. https://doi.org/10.1109/HICSS.2001.926232

Carlisle, M. C. (2009). Raptor: a visual programming environment for teaching object-oriented programming. Journal of Computing Sciences in Colleges, 24(4), 275–281.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: a visual programming environment for teaching algorithmic problem solving. ACM SIGCSE Bulletin, 37(1), 176–180.

Crews, T., & Ziegler, U. (1998). The flowchart interpreter for introductory programming courses. In Frontiers in Education Conference, 1998. FIE '98. 28th Annual (pp. 307–312). https://doi.org/10.1109/FIE.1998.736854

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. Psychometrika, 16(3), 297–334. https://doi.org/10.1007/BF02310555

Cronbach, L. J., & Shavelson, R. J. (2004). My Current Thoughts on Coefficient Alpha and Successor Procedures. Educational and Psychological Measurement, 64(3), 391–418. https://doi.org/10.1177/0013164404266386

Diehl, S. (Ed.). (2002). Software Visualization (Vol. 2269). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/3-540-45875-1 (accessed 27.11.2018)

Dol, S. M. (2015). Fe.g.: An Animated Flowchart with Example to Teach the Algorithm Based Courses in Engineering. In 2015 IEEE Seventh International Conference on Technology for Education (T4E) (pp. 49–52). https://doi.org/10.1109/T4E.2015.3

Fleischer, R., & Kucera, L. (2002). Algorithm Animation for Teaching. In Revised Lectures on Software Visualization, International Seminar (pp. 113–128). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=647382.724788

Gaddis, T. (2015). Starting Out with Programming Logic and Design (4 edition). Boston: Pearson.

Gajewski, R. R. (2014). Engineering Calculations and Their Programming: PTC® MathCAD Prime® 3.0. Warsaw: Oficyna Wydawnicza Politechniki Warszawskiej.

Gajewski, R. R., & Jaczewski, M. (2014). Flipped Computer Science Classes. In Federated Conference on Computer Science and Information System (pp. 795–802). Warsaw.

Gajewski, R., Wlasak, L., & Jaczewski, M. (2013). IS (ICT) and CS in Civil Engineering Curricula: Case Study. In Proceedings of the 2013 Federated Conference on Computer Science and Information Systems (pp. 717–720). Krakow: IEEE.

Giannakos, M. N., Pappas, I. O., Jaccheri, L., & Sampson, D. G. (2016). Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. Education and Information Technologies. https://doi.org/10.1007/s10639-016-9538-1

Goktepe, M. (1988). Design and Implementation of a Tool for Teaching Programming (M.Sc. Thesis). Bilkent University, Ankara.

Gomes, A., & Mendes, A. J. (2007). Learning to program-difficulties and solutions. In International Conference on Engineering Education–ICEE (Vol. 2007).

Guttman, L. (1945). A basis for analyzing test-retest reliability. Psychometrika, 10(4), 255–282. https://doi.org/10.1007/BF02288892

Hooshyar, D., Ahmad, R. B., Nasir, M. H. N. M., Shamshirband, S., & Horng, S.-J. (2015). Flowchart-based programming environments for improving comprehension and problem-solving skill of novice programmers: a survey. International Journal of Advanced Intelligence Paradigms, 7(1), 24–56. https://doi.org/10.1504/IJAIP.2015.070343

Hundhausen, C. D., & Brown, J. L. (2005). What You See Is What You Code: A radically dynamic algorithm visualization development model for

novice learners. In 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05) (pp. 163–170). IEEE.

Hundhausen, C. D., & Douglas, S. A. (2002). Low-fidelity algorithm visualization. Journal of Visual Languages & Computing, 13(5), 449–470.

Konecki, M. (2014). Problems in Programming Education and Means in Their Improvement. In DAAAM International Scientific Book 2014 (pp. 459–470).

Konecki, M. (2015). Algorithmic thinking as a prerequisite of improvements in introductory programming courses. Uporabna Informatika, 23(3), 162–169.

Konecki, M., & Petrlic, M. (2014). Main problems of programming novices and the right course of action. In Central European Conference on Information and Intelligent Systems (pp. 116–123). Varazdin: Faculty of Organization and Informatics Varazdin.

Kuechler, W. L., & Simkin, M. G. (2003). How Well Do Multiple Choice Tests Evaluate Student Understanding in Computer Programming Classes? Journal of Information Systems Education, 14(4), 389–399.

Kuen, K. C. (2011). Learning Programming Concepts Using Flowcharting Software. In Proceedings of the Global Chinese Conference on Computers in Education (GCCCE) 2011. Hangzhou, China.

Malik, S. I., & Coldwell-Neilson, J. (2016). A model for teaching an introductory programming course using ADRI. Education and Information Technologies, 1–32. https://doi.org/10.1007/s10639-016-9474-0

Mayer, R. E. (2001). Multimedia Learning. New York: Cambridge University Press.

Moreno, R. (2005). Instructional Technology: Promise and Pitfalls. In Technology-Based Education: Bringing Researchers and Practitioners Together (pp. 1–19). Information Age Publishing.

Moreno, R. (2006). Does the modality principle hold for different media? A test of the method-affects-learning hypothesis. Journal of Computer Assisted Learning, 22(3), 149–158. https://doi.org/10.1111/j.1365-2729.2006.00170.x

Nickerson, J. V. (1994). Visual Programming (Ph.D. Dissertation). New York University, Computer Science, New York. Retrieved from https://web.stevens.edu/jnickerson/indextopic.htm.

OECD. (2015). Students, Computers and Learning. Making the connection. (p. 204). Retrieved from /content/book/9789264239555-en

Park, B., Plass, J. L., & Brünken, R. (2014). Cognitive and affective processes in multimedia learning. Learning and Instruction, 29, 125–127. https://doi.org/10.1016/j.learninstruc.2013.05.005

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., Paterson, J., (2007). A survey of literature on the teaching of introductory programming. ACM SIGCSE Bulletin, 39(4), 204–223.

Rahmat, M., Shahrani, S., Latih, R., Yatim, N. F. M., Zainal, N. F. A., & Rahman, R. A. (2012). Major Problems in Basic Programming that

Influence Student Performance. Procedia - Social and Behavioral Sciences, 59, 287–296. https://doi.org/10.1016/j.sbspro.2012.09.277

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

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM, 20(6), 373–381.

Sleeman, D. (1986). The Challenges of Teaching Computer Programming. Commun. ACM, 29(9), 840–841. https://doi.org/10.1145/6592.214913

Sorden, S. D. (2013). The Cognitive Theory of Multimedia Learning. In Handbook of Educational Theories. Information Age Publishing.

Tarhini, A., Hone, K., Liu, X., & Tarhini, T. (2017). Examining the moderating effect of individual-level cultural values on users' acceptance of E-learning in developing countries: a structural equation modeling of an extended technology acceptance model. Interactive Learning Environments, 25(3), 306–328. https://doi.org/10.1080/10494820.2015.1122635

Thompson, M. (2012). Evaluating the Use of Flowchart-based RAPTOR Programming in CS0. In Proceedings of the 45th Annual Midwest Instruction and Computing Symposium. Cedar Falls, Iowa: University of Northern Iowa. Retrieved from http://micsymposium.org/mics2012/submissions/mics2012_ submis sion_38.pdf

Végh, L., & Stoffová, V. (2017). Algorithm Animations for Teaching and Learning the Main Ideas of Basic Sorting. Informatics in Education, 16(1), 121–140.

Venit, S., & Drake, E. (2014). Prelude to Programming (6th ed.). Boston: Pearson.

Wing, J. M. (2006). Computational Thinking. Communications of the ACM, 49(3), 33–35.

Wlasak, L., Jaczewski, M., Dubilis, T., & Warda, T. (2013). How to Motivate Digital Natives to Learn? In WCCE 2013 10th IFIP World Conference on Computers in Education (Vol. 3: Book of Abstracts, pp. 78–79). Torun: IFIP.

Wolfram, S. (2016). How to Teach Computational Thinking—Stephen Wolfram Blog. Retrieved from http://blog.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/ (accessed 30 November 2016)

Wu, B., & Chen, X. (2017). Continuance intention to use MOOCs: Integrating the technology acceptance model (TAM) and task technology fit (TTF) model. Computers in Human Behavior, 67, 221–232. https://doi.org/10.1016/ j.chb.2016.10.028

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. ACM Inroads, 2(4), 71. https://doi.org/10.1145/2038876.2038894

Zainal, N. F. A., Shahrani, S., Yatim, N. F. M., Rahman, R. A., Rahmat, M., & Latih, R. (2012). Students' Perception and Motivation

Towards Programming. Procedia - Social and Behavioral Sciences, 59, 277–286. https://doi.org/10.1016/j.sbspro.2012.09.276