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 Baldwin & 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 (Hooshayr, 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)

- **FLOWGORTHM** – 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 VIualization Storyboader) 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 (Hooshayr 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).

![Flowchart of the Babylonian method and Euclidean algorithm](source)

**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).

\[
f(x) = \begin{cases} 
-x & x < -1 \\
1 & x \in [-1, 1] \\
x & x > 1 
\end{cases} = \text{IF}(A1<-1, -A1, \text{IF}(A1>1, A1, 1))
\]

**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).

![Flowchart and spreadsheet](image)

**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.
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.
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 & Petrlík, 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**

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

*Source: Own work inspired by Konecki*

Table 2.

**Teaching topics**

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

*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>
<thead>
<tr>
<th>Learning situations</th>
<th>Mean</th>
<th>Never</th>
<th>Rarely</th>
<th>Sometimes</th>
<th>Often</th>
<th>Always</th>
</tr>
</thead>
<tbody>
<tr>
<td>In lectures</td>
<td>1.889</td>
<td>64</td>
<td>37</td>
<td>24</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>In lab sessions</td>
<td>3.434</td>
<td>6</td>
<td>19</td>
<td>43</td>
<td>46</td>
<td>22</td>
</tr>
<tr>
<td>While studying alone</td>
<td>3.456</td>
<td>6</td>
<td>20</td>
<td>43</td>
<td>40</td>
<td>27</td>
</tr>
<tr>
<td>While working alone on programming coursework</td>
<td>3.485</td>
<td>5</td>
<td>21</td>
<td>41</td>
<td>41</td>
<td>28</td>
</tr>
<tr>
<td>In exercise sessions in small groups</td>
<td>2.397</td>
<td>42</td>
<td>31</td>
<td>38</td>
<td>17</td>
<td>8</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>I found the…</th>
<th>Mean</th>
<th>Useless</th>
<th>Not very useful</th>
<th>Somewhat useful</th>
<th>Useful</th>
<th>Very useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introductory course book</td>
<td>2.449</td>
<td>38</td>
<td>36</td>
<td>35</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Lecture notes</td>
<td>2.073</td>
<td>58</td>
<td>27</td>
<td>36</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td>4.058</td>
<td>3</td>
<td>8</td>
<td>23</td>
<td>46</td>
<td>56</td>
</tr>
<tr>
<td>Example programs</td>
<td>3.926</td>
<td>6</td>
<td>5</td>
<td>29</td>
<td>49</td>
<td>47</td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td>3.250</td>
<td>10</td>
<td>22</td>
<td>50</td>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>Interactive visualizations</td>
<td>3.324</td>
<td>15</td>
<td>15</td>
<td>44</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>Movies (software animations)</td>
<td>4.132</td>
<td>2</td>
<td>9</td>
<td>23</td>
<td>37</td>
<td>65</td>
</tr>
</tbody>
</table>

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


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


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


