Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 125–132
www.elsevier.com/locate/entcs

Kick-Start Activation to Novice Programming — A Visualization-Based Approach

Essi Lahtinen[1] and Tuukka Ahoniemi[2]

Department of Software Systems
Tampere University of Technology
Tampere, Finland

Abstract

In the beginning of learning programming students have misconceptions of what programming is. We have used a kick-start activation in the beginning of an introductory programming course (CS1) to set the record straight. A kick-start activation means introducing the deep structure of programming before the surface structure by making the students solve a certain type of problem in the first lecture. The problem is related to a realistic computer program, simple enough for everyone to understand and allow students to participate in debugging. A visualization-based approach helps making the example more concrete for students.
In this article we present the concept kick-start activation and one concrete example. To support the example, we have also developed a visualization using the visualization tool JHAVÉ. We got positive feedback on the example and suggest further development of kick-start activations in order to make the beginning of learning programming more motivating for students.

Keywords: Teaching programming, Novice programmers, Visualizations, Kick-start activation.

1 Introduction

Students who enroll to introductory programming courses (CS1) have plenty of misconceptions about the nature of programming and some students do not know what programming is at all. The course typically starts with the teacher trying to correct the misconceptions by emphasizing that programming is more problem-solving and thinking than typing program code. The concept of an algorithm is introduced, as well as some tools for implementing algorithms and designing programs, such as pseudocode or flow charts.

A classical first example of an algorithm is a recipe in a cook-book. A recipe is a relatively unambiguous, detailed set of instructions. If you follow the instructions carefully you will have a food portion as the result. However, there are problems

[1] Email: essi.lahtinen@tut.fi
[2] Email: tuukka.ahoniemi@tut.fi

doi:10.1016/j.entcs.2008.12.056

with this example. Firstly, it is not at all related to computers. Thus students might feel that the teacher is stating the obvious or even explaining nonsense when he/she is talking about cooking and algorithms instead of programming. Secondly, even though comparing cooking recipes and algorithms gives a clear idea on what an algorithm is, it does not really help to understand what a programmer does. There is no occupation where the job is to develop new recipies. Thus, the underlying idea of programming is not delivered to the students. Thirdly, the methaphor also does not help in explaining the programming process for the student. There are no concrete examples on important phases like designing, testing nor debugging. The student might still continue carrying the misconception of programming being merely the implementation of an algorithm.

We introduce a different way to start the course: *kick-start activation*. In this approach, we get into the deep structure of programming before the surface structure is even introduced. Our target audience is especially the students who do not know anything about programming before the kick-start activation.

In this article we first present the idea of a kick-start activation in Section 2. Then we introduce our example and explain how we use it in Section 3. Section 4 presents the visualization and feedback. Finally, discussion and conclusion are included in Section 5.

2 Criteria for a Kick-Start Activation

In our opinion, to make the opening of the course interesting for students, one needs to get directly into the real problems, i.e., a problem that requires an algorithmic solution. In the case of programming this means skipping the surface structure, such as the syntax of the programming language, and starting from the deep structure of programming, i.e., a problem that the students solve themself. We call this kind of an introduction *kick-start activation* because it is a fast-forward jump-in approach and it engages students in the example since they solve the problem.

Our *main criterion* for the example presented in the kick-start activation is that it has to be based on *a real computer program*. The benefits of a real programming example are:

- In addition to introducing the concepts of algorithms, pseudo code and flow charts one can also introduce
  - problem solving and the phases of programming,
  - the idea of testing algorithms and the idea of testing programs, and
  - what the work of a programmer is like.
- It helps to explain the difference of human thinking and the way the computer works.
- The execution of the algoritm can be explained and demonstrated with a computer.
- One can also show a an implementation in a programming language to give an example. Students can identify the control structures of the pseudo code from the program code. Even if the students do not understand theprogramming language

syntax yet, it gives them a concrete example on what a programming language looks like.

- It can be concretized by a program visualization that the students can run.

Our *second criterion* is that the kick-start activation needs to be simple enough so it can be understood by everyone. We decided that it has to be an example that *relates to everyday life*. Besides that we chose not to use a real programming language nor any terms, pictures, or other details that relate to computers. For example, we did not want `[]`-operators in the algorithm or memory addresses in the pictures. These would just add extra details that are irrelevant at this stage. Instead of using a programming language it is easier to fade out the surface structure of programming by using a natural-language-like pseudo code presentation and flow charts. To concretize the pseudo code and flow chart we developed a visualization that illustrates how the algorithm would be run by a computer if the computer could understand it.

The *third criterion* for a kick-start activation was to make *students take part* in the example. As programming is much more thinking and problem-solving than using the programming language syntax, there are numerous programming related activities that students can try already in the beginning of the course. For instance testing an algorithm is a task that can be given to a student. One practical way of doing this is developing a buggy version of an algorithm that the students can debug.

3 Our Example: Hyphenating Finnish Words

The topic of our kick-start activation was the hyphenation rules of the Finnish language. Word processors have spell checking and automatic hyphenation, i.e., computer programs are hyphenating Finnish words. In addition, every student knows how to spell [3] so the topic is general enough.

The exact rules for hyphenating Finnish are not common knowledge in Finland even if it is easy to hyphenate Finnish for everyone who knows how to speak the language. Fortunately the rules are simple enough to be explained to students in a few sentences. Still, it is non-trivial to build a hyphenation algorithm. The algorithm requires a loop structure to go through the letters of the hyphenated word and a couple of if-statements to choose which hyphenation rule to apply.

For example, the first of three hyphenation rules called *the consonant rule* states the following: *if there is a vowel followed by one or more consonants, a hyphen is placed directly before the last consonant.* The window on the right hand side in Figure 1 presents the algorithm based on the rules. The consonant rule can be identified in the marked area of the figure.

A word is a data structure that can be understood even without knowing the data type `string`. A word can also be drawn like a line of alphabet building blocks (See the window on the left hand side in Figure 1). Introducing the computer

[3] In this situation actually: *in Finland* every student knows how to spell *Finnish*.

memory or other similar details for the student is unnecessary. Drawing the data structure as a line of building blocks actually allows us to visualize the addition of a hyphen: a picture animation where a block with the character '-' slides and slips in between the blocks of the word.

On the lecture, our intention was to highlight that designing and testing the algorithm with pen and paper is a big part of programming. To describe this clearly we used a three step example:

(i) First we quickly designed a hyphenation algorithm. Though it seemed to be correct the hasty design had on purpose produced a buggy solution.

(ii) Then the students tested the algorithm and hopefully found the error. After this we discussed how important it is to understand the problem before you start designing the algorithm.

(iii) Finally, we explained the hyphenation rules deeper for the students and designed a new algorithm properly. The final result was a correctly working algorithm.

The example included two algorithms. We call these *the premature algorithm* (produced in step 1) and *the mature algorithm* (produced in step 3).

The purpose of the testing phase was to activate the students. They were actually performing a programming related task even if they thought they did not know any programming yet. The idea is that the students can use the visualization to run and test the algorithm. The testing could of course be done using only pen and paper, but the visualization is handy in it. We gave a link to the visualization to the students for later use so that they could revise the lecture using the visualization.

4 The Visualization

There are many program visualization tools available for presenting basic programming structures for novice programmers for instance, Jeliot 3 [4] for Java, VIP [12] for C++, and Ville [7] and Planani [9] for multiple different languages. These visualization tools work on program code level, so they assume that the student already understands some programming language and thus are not suitable for our target audience. There is also a visualization tool called RAPTOR [1] where the students can construct flow charts and the tool will visualize them for the student. The RAPTOR flow charts are also close to the program code level, e.g., the tool shows the content of variables and arrays.

We needed a completely syntax-free common purpose visualization tool where we can write the algorithm in a few Finnish sentences and draw the building blocks exactly according to our needs. Thus, the existing program visualization tools did not suit our purposes. However, in the field of algorithm visualizations there was one tool flexible enough: JHAVÉ [6] and its Gaigs support class package. With a bit of imagination we were able to use this algorithm visualization tool slightly unorthodoxically and produce the hyphenation visualization.

The info screen of JHAVÉ's execution window is normally used for showing

algorithm specific instructions written in HTML. The tool allows the use of images as a part of the HTML page with the <image> tag. This feature let us implement the flow chart animation with a set of fixed images. The images were then presented in the correct order by showing a particular image in each state of the program. With the possibility of using HTML and images in JHAVÉ, one could design many sorts of examples as the technical implementation is limited solely to the creation of the images.

Using JHAVÉ, we implemented two different presentations of *the hyphenation algorithm visualization*: a pseudo code view and a flow chart. Both of these presentations also contain a window with the alphabet building block picture of the hyphenated word. Screenshots can be seen in Figure 1 and Figure 2. There were two different algorithms that we visualized: the premature and mature. Since there are two different presentations of both the algorithms we actually had four different visualizations.

The student can control the visualization using the step and step-back buttons. The execution of the algorithm is visualized by coloring the nodes in the flow chart or the lines of the pseudo code synchronously with the steps. As the program is hyphenating words, the state of the word in each step is visualized in the window with the alphabet building block picture on the left hand side. There are pictures of two words: the original word without the hyphens and the result where the hyphens are added as the algorithm proceeds. The visualization also colors the alphabet building blocks that the algorithm is handling.

4.1 Student Engagement

According to research on the field of visualizations, student engagement is vital for learning when a student uses visualization [11]. Naps et al. [5] present a Visualization Engagement Taxonomy that describes six levels of learner engagement with visualization technology. On top of the lowest level of existing engagement—*Viewing*—are the more active levels: *Responding* and *Changing* an existing visualization and *Constructing* and *Presenting* ones own visualization.

As the algorithm is given fixed in the hyphenation algorithm visualization, the student engagement is enhanced by allowing the student to provide his/her own input word for the algorithm. This corresponds to the level *Change* of the Visualization Engagement Taxonomy [5]. To attain the level *Response* also, the flow of the program is interrupted with pop-up questions querying about the next behavior of the program.

4.2 Student Feedback

We evaluated the visualization with a quantitative survey after the lecture where we used it. We handed in a questionnaire on paper for the students. We received altogether 113 responses. 71 of the respondents (63%) had no programming experience before the course.

The feedback was generally positive since 53% of the respondents said that the

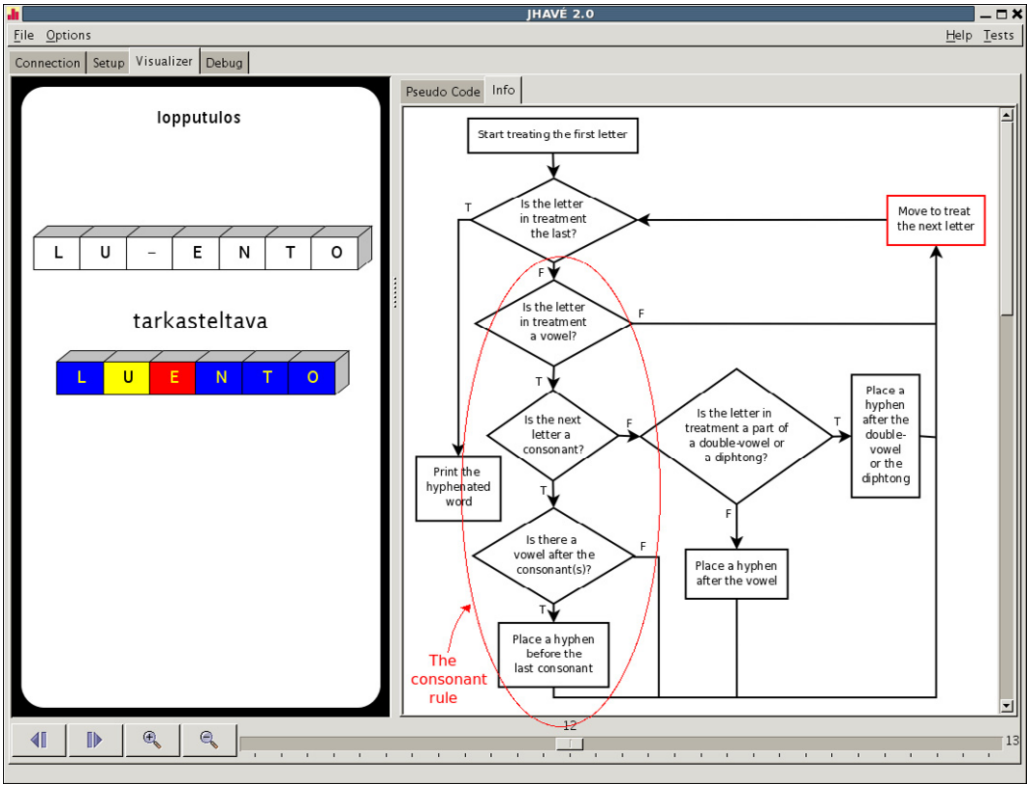

Fig. 1. The flow chart version of the visualization.

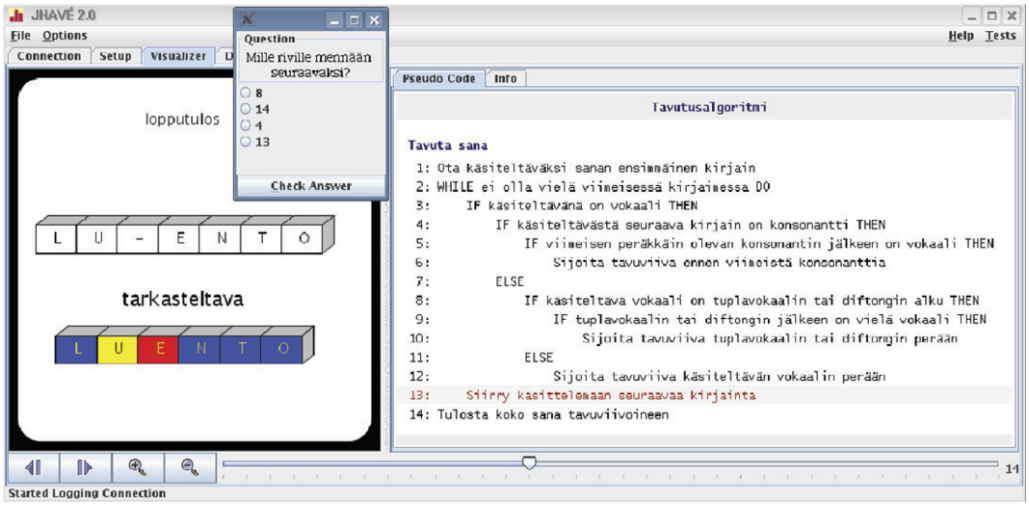

Fig. 2. The pseudo code version of the visualization with an activating pop-up question.

visualization looked nice (agree or totally agree), 86% thought that is was useful for learning (agree or totally agree), and only 5% thought that it disturbed the lecture (agree or totally agree).

We performed a cross tabulation and a χ^2-test for some of the variables and found out that the students with no earlier programming experience thought that the vi-

sualization was more useful for learning than the students who had programmed before coming to the course. This difference is statistically significant ($p < 0,05$). The reason is also obvious: the students with earlier programming experience already had an understanding on how algorithms and flow charts work so they do not need the visualization for understanding the hyphenation algorithm. This result shows that we managed to help the students who were the target audience of the visualization.

After all, the most important feedback was that our students were listening to the hyphenation example intensively on the lecture. Two teachers tried the example and both of them could sense a notable difference in the lecture situation compared to the cook-book example.

5 Discussion and Conclusions

The kick-start activation received positive feedback both from the students and the teachers who used it. We think that our approach was successful because the criteria were designed carefully and there was a visualization tool that aided both presenting the example and understanding it. This example could be used as a source of ideas for other topics to build kick-start activations of.

There are not many program visualization tools available for our target audience—the students who do not know anything about programming yet. In addition to our visualization we have found a system called SICAS [3] that could probably also be used for presenting a kick-start activation. It is based on similar principles and allows students to construct their own flow charts and visualize them. However, currently it is not used the same way we used our visualization.

The conceptual framework of programming knowledge developed by McGill and Volet [2] suggests that in addition to syntactic and conceptual knowledge a programmer also needs strategic knowledge of programming. Reports on the state of field show that visualizations are often used for only presenting programming concepts [10]. The scope of our visualization is more in the strategic knowledge since it focuses on the programming phases: testing and design.

In the development of the visualization we also emphasized student engagement in the levels of the Visualization Engagement Taxonomy [5]. The visualization is most activating when the student is guided to use it in the three step lesson we described in Section 3. This requires either a teacher to explain the hyphenation problem and the need for debugging the first version of the algorithm or the student to read this from the material by himself. The idea of connecting a visualization to a certain study material is similar to the one presented in an ITiCSE working group report about hypertextbooks [8]. We think that the visualization of the mature version of the algorithm could also be used without the debugging phase just for presenting the concepts algorithm, pseudo code, and flow chart. This way the example would be less challenging and the activation of the student would be left only to the pop-up questions.

The best possibility for activating students would be to make them correct the

bug or build a completely new correct algorithm after finding the bug from the premature version of the algorithm. This can, however, be very challenging for a novice student so we did not try it. It would be an interesting future work idea to build a visualization tool where the student could build the correct algorithm by modifying the flow chart. Another idea for future work is that we could implement different kinds of premature algorithms. There could be easier and more difficult bugs for the debugging task.

6 Acknowledgments

Special thanks to Prof. Thomas Naps (University of Wisconsin, Oshkosh) for his enormously useful guidance with visualizations in common and help with the JHAVÉ visualization tool. Nokia Foundation has partly funded this work.

References

[1] Giordano, J. C. and M. Carlisle, *Toward a more effective visualization tool to teach novice programmers*, in: *SIGITE '06: Proceedings of the 7th conference on Information technology education* (2006), pp. 115–122.

[2] McGill, T. and S. Volet, *A conceptual framework for analyzing students' knowledge of programming*, Journal on research on Computing in Education **29** (1997), pp. 276–297.

[3] Mendes, A. J., A. Gomes, M. Esteves, M. J. Marcelino, C. Bravo and M. A. Redondo, *Using simulation and collaboration in cs1 and cs2*, SIGCSE Bull. **37** (2005), pp. 193–197.

[4] Moreno, A., N. Myller, E. Sutinen and M. Ben-Ari, *Visualizing programs with Jeliot 3*, Proceedings of the International Working Conference on Advanced Visual Interfaces AVI 2004 (2004).

[5] Naps, T., G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. Velazquez-Iturbide, *Exploring the role of visualization and engagement in computer science education*, SIGCSE Bulletin **35** (2003), pp. 131–152.

[6] Naps, T. L., J. R. Eagan and L. L. Norton, *JHAVé - An environment to actively engage students in web-based algorithm visualizations*, ACM SIGCSE Bulletin , Proceedings of the thirty-first SIGCSE technical symposium on Computer science education SIGCSE '00 **32** (2000), pp. 109–113.

[7] Rajala, T., M.-J. Laakso, E. Kaila and T. Salakoski, *VILLE – A language-independent program visualization tool*, in: *Proceedings of The Seventh Koli Calling Conference on Computer Science Education*, 2007.

[8] Rössling, G., T. Naps, M. S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S. H. Rodger, J. Urquiza-Fuentes and J. Ángel Velázquez-Iturbide, *Merging interactive visualizations with hypertextbooks and course management*, in: *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education* (2006), pp. 166–181.

[9] Sajaniemi, J. and M. Kuittinen, *Visualizing roles of variables in program animation*, Information Visualization **3** (2004), pp. 137–153.

[10] Shaffer, C. A., M. Cooper and S. H. Edwards, *Algorithm visualization: a report on the state of the field*, in: *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education* (2007), pp. 150–154.

[11] Stasko, J. T. and C. D. Hundhausen, *Algorithm Visualization*, in: *Computer Science Education Research* (2004), pp. 199–228.

[12] Virtanen, A. T., E. Lahtinen and H.-M. Järvinen, *VIP, a visual interpreter for learning introductory programming with C++*, Proceedings of the Fifth Finnish/Baltic Sea Conference on Computer Science Education (2005), pp. 129–134.