TUCS

Mikko-Jussi Laakso | Teemu Rajala | Erkki Kaila | Tapio Salakoski

Visualizable Pseudo Programming Language

Turku Centre *for* Computer Science

TUCS Technical Report
No 838, August 2007

Visualizable Pseudo Programming Language

Mikko-Jussi Laakso
University of Turku, Department of Information Technology

Teemu Rajala
University of Turku, Department of Information Technology

Erkki Kaila
University of Turku, Department of Information Technology

Tapio Salakoski
University of Turku, Department of Information Technology

TUCS Technical Report
No 838, August 2007

Abstract

Selecting a right programming language for introductory CS courses is not an easy task. The pressure to use a commercially successful language can easily be the most significant factor in choosing the language. However, several studies suggest that it is preferable to use language especially designed for teaching. These languages are usually some kind of pseudo languages, defined by the programming educators. The idea of using pseudo language is to keep the syntax of the code as simple as possible so the students can focus on learning the programming concepts, instead of some irrelevant syntactical features.

Pseudo languages are usually subsets of existing programming languages, and thus can be used to teach all the basic programming concepts. This however leads to a fact that programs written in pseudo code are usually not executable as-is, which is one of its main problems. With this in mind we have developed a program visualization tool called VILLE in university of Turku. The tool has a built-in pseudo language that is a subset of Python. Users can add programming examples written in Java and translate them to pseudo language. The programs in pseudo code can also be translated to a runnable Python code. Additionally, users can define their own pseudo language and modify the existing syntaxes.

Keywords: novice programming, pseudo code, program visualization, self-definable, teaching

1. Introduction

As McCracken [10] states, learning to program is one of the most important outcomes of the computer science studies. However, the first steps may prove to be extremely difficult. According to Lister et al. [9], as well as lacking the actual skills for problem solving, the students may as well have deficient prerequisite skills for solving the tasks, including the ability to read and understand the program code. This may partially derive from the choice of programming language used in teaching. While most of the commercially successful languages used nowadays are expressive, flexible and suitable for variety of projects, the excessive syntax in basic programs may be confusing for beginners.

Ala-Mutka [1] states that more it takes to learn the programming language and the use of the environment the harder it is for students to assimilate the general aspects of programming and problem solving. Hence, it is course designer's and teacher's responsibility to choose the programming language that has sufficient enough syntax to learn, but which still holds enough potential to teach more advanced aspects – such as object orientation – and which supports the move on to the more advanced languages.

In this paper we discuss the reasons to choose pseudo language – in this case a slightly modified subset of Python – as a first language in introductory programming course, and present some researches and articles on language selection. We also present VILLE, a multilanguage program visualization tool that supports the use of various programming languages.

2. Related work

There has been lot of discussion on choosing the right language for the first programming courses. It seems of course sensible to select a language most suitable for teaching. However, there is always the pressure to use commercially successful languages [2]. The popularity of language shouldn't nevertheless be the key factor. It is more important to choose the language on the basis of how easily students can understand the basic concepts of the programming paradigm taught.

Grandell et al. [3] used Python as their first language in introductory programming course in high school. In their opinion Python is more suitable than Java as a teaching language, because it has less syntactical baggage. McIver & Conway [11] present seven undesirable features that are common in programming languages usually taught in introductory programming courses, and continue with categorizing seven criteria for choosing an introductory language. Hu [5] tried to find a suitable programming language for novices, and as a result created a language with minimal number of statements called core language. According to article the core language used alongside

with dynamic visualizations maintained an appropriate mix of theory and practice during teaching of a course. Olsen [14] used pseudo code as a design tool in an introductory CS course. Students used pseudo code to define the solution, and then implemented the actual program with C++. Kölling et al. [7] have used object-first principle in teaching programming to novices. The tool they have developed - called BlueJ – has class and object views which students can use to design and develop programs without writing any actual code. Reek [16] has also tried top-down approach in teaching programming.

Another important aspect is the program paradigm used in teaching. Wells and Kurtz [19] suggest that imperative programming paradigm is generally used excessively and students should instead be exposed to several paradigms as early as possible on their studies. To solve this they created a pseudo language that is a superset of many programming paradigms. Brilliant and Wiseman [2] discuss the choice of programming paradigm used, and point out that using non-procedural (non-imperative) language usually evens up the differences between students with different level of computing experience. However they didn't find any advantages using a particular paradigm as a starting point. Van Roy et al. [18] discuss the role of language paradigms in programming teaching, mainly choosing the right paradigm, valuing the terms of concepts and design process over the paradigm, and advantages of using the object oriented paradigm as the first one.

3. Reasons to use pseudo language

Most languages are not designed to be used in teaching programming. The choice of language is often decided because of such characteristics as 'suitability for object-orientation and concurrent programming or pressure from commercial level' (see e.g. [4]). However, the basic syntax of programming language can prove to be very difficult for novices to master. Let's review the common first example in any CS course, an application that outputs 'Hello world!' written in Java:

```
public class HelloWorld{
        public static void main(String[] args){
                System.out.println("Hello world!");
        }
}
```

Figure 1: 'Hello world' example in Java

The example clearly proves that Java has a lot of excessive syntax that has no purpose in learning the basic logic of programming. The class definition, access level modifiers, the complicated syntax of main method declaration and its parameter, and the use of print statement from System class's output stream all are way too complicated subjects to understand when writing your first program. Since most imperative languages have

similar control structures and operational logic, a pseudo programming language with clearer syntax could be more appropriate as a first teaching language. Let's review the above example in basic pseudo language (see appendix A) used in VILLE:

```
print "Hello world!"
```

Figure 2: 'Hello world' example in pseudo language

The output of two examples is the same, but the pseudo version is lacking all the formal definitions. Besides having smaller and cleaner syntax, the pseudo code has some more advantages, including dynamic typing, enforced structural design and easier use of tables. Grandell et al. [3] researched the usage of Python (which pseudo language we're using is a subset of) as a first language in high school introductory programming course with quite auspicious results. There were over 10 % less students failing the course and over 10 % increase in the highest score rates compared to the earlier classes, taught in Java. In addition, students who had earlier experience in programming with other languages found Python easier and more fun

The disadvantage of using such language is that the students normally would like to try to execute their examples, and normally there's no platform for running pseudo code. There are however two workarounds: firstly, since the pseudo code syntax used is a subset of Python, the programs are runnable with a Python interpreter, and secondly, pseudo code is directly runnable in VILLE. Another advantage of such tool is narrowing the gap when transferring forward to other languages, such as Java. This in mind, the basic syntax of pseudo code is quite similar to that in Java and C.

4. VILLE – a tool for using and executing pseudo language

VILLE is a program visualization tool for novice programmers that can be used both in lectures and for independent learning. VILLE has a built-in syntax editor with which users can add new languages to the tool or modify the built-in language syntaxes (Java, Pseudo and C++). Thus teacher can add his programming course's programming examples to VILLE and then visualize their execution in self-defined programming language. More specific description of the system and its features can be found in Rajala et al. [15].

4.1. Key features

Language-independency. One of the most important aspects of VILLE is the possibility to show programming examples in several different programming languages. When user observes program execution in different languages, he sees how similar their

basic functionalities are. It is far more important for novice programmers to learn how different programming concepts actually work, than to focus on syntactical issues of a specific language.

Defining and adding new languages. At present, VILLE supports Java, pseudo code and C++ programming languages. Pseudo code's definition can be altered to suit teachers needs. It is also possible to define and add new programming languages to further extend the language support.

Example collection. VILLE contains predefined set of programming examples, divided in different categories based on their subject. User can create new categories and examples or edit the ones included. By creating and editing examples teacher can illustrate subjects that he thinks are essential in learning to program. Teacher can also make changes to the programming examples during lectures to visualize the effects of the modifications.

Execution row by row. Progress of the program execution is visualized by highlighting rows in the code. In addition to highlighting the program row under execution, VILLE also highlights previously executed row with a different colour. This makes following of the program execution easier.

Flexible control of visualization both forwards and backwards. User can move one step at a time, both forwards and backwards in the execution of the program. Examples can also be run automatically with adjustable speed. Moving backwards in the program execution isn't usually possible in similar applications. Additionally, VILLE has an execution slider with which user can use to move to any point of program execution.

Breakpoints. The user can set breakpoints into any program code line and move between them both forwards and backwards. This functionality enables debug-based control and observation of the program execution. More over, backward tracing between breakpoints is not a standard feature in program code debuggers.

Code line explanation. Every code line has an explanation, in which all the program events related to the line are clearly explained. Furthermore, all possible outputs and variable states are shown. Code line explanation is a missing feature in many similar applications.

Role information. Variables' role information is integrated into the code line explanation. According to Sajaniemi and Kuittinen [17[this helps programming learning and enhances understanding of the program.

The parallel view shows the program code execution simultaneously in two different programming languages. This way the user can see how the execution progresses similarly regardless of syntactical differences between the languages.

Call stack. The moving of the program execution between different methods due to function calls and returns is visualized with a call stack. When a method is called, new window is opened in the call stack. The window remains in the call stack until the method is finished. When the execution returns to the caller, possible return value is shown on top of the call stack. The visualization of the execution can be alternatively viewed in parallel view with program code viewed in two languages simultaneously.

Publish examples. With the export feature VILLE's examples can be saved to an example collection. The example collection contains a version of VILLE with example creation and modification functions disabled. Teacher can use the export feature to publish course's programming examples in web for students to use.

Pop-up questions. One useful feature of VILLE is the possibility to create pop-up questions for the programming examples. With the built-in editor teacher can create multiple choice questions and set them to trigger in certain states of the program execution.

4.2. The User Interface of VILLE

For students VILLE offers three different views. In the main view user can browse through programming examples and start the visualization of chosen example. In the visualization view user can follow the execution of the example. The visualization view has two different modes: the call stack mode and the parallel mode. In the call stack mode (Figure 3) in addition to code listing on the left side of the view, method calls are visualized as a stack of windows on the right side of the view. Method calls' return values are shown on top of the stack. On the top left corner are the program execution controls, slider for the speed of automatic execution and drop-down menu where user can choose programming language in which the example is shown. Under the code listing and call stack lie fields that show information about the program state. The left field shows an explanation of program line under execution and the roles of variables on that line. The middle field lists all the output generated by the program and the right field shows the states and values of variables during the execution. At the bottom of the view is an execution slider, which can be used to move around in the program.

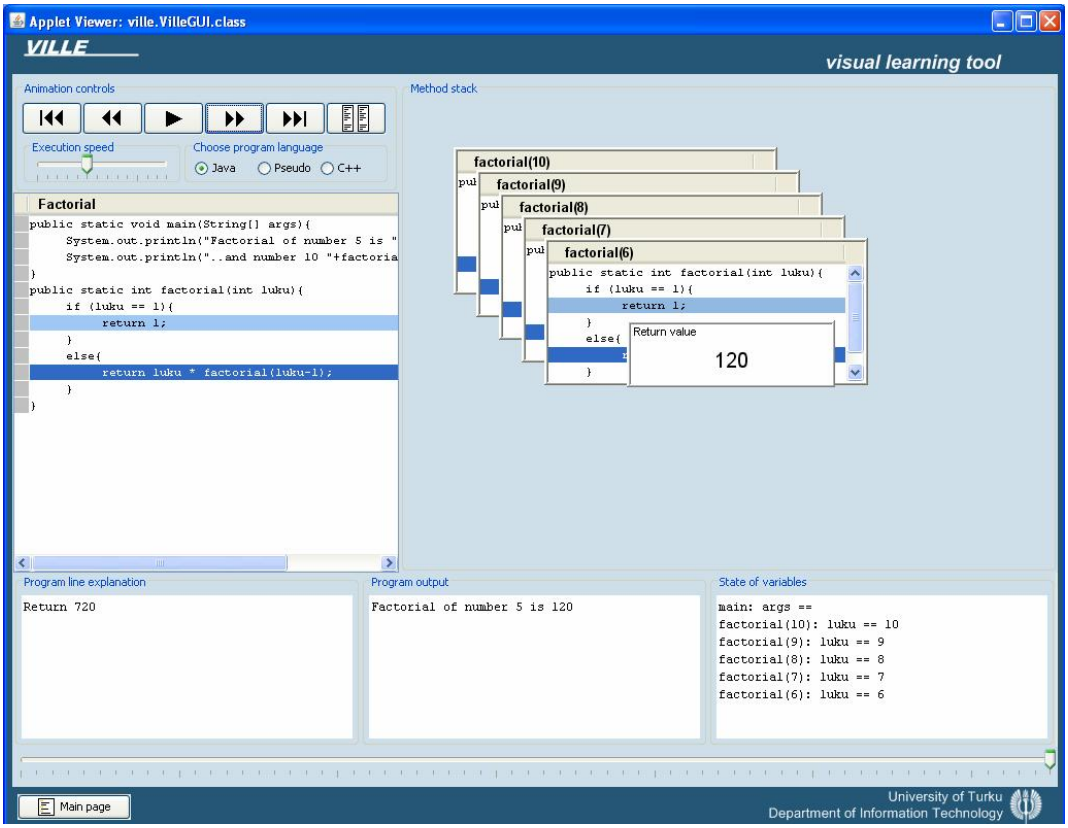

Figure 3: Visualization view of VILLE in call stack mode

The program execution in the visualization view can also be followed in the parallel view (Figure 4). In this case the programming example opens in two parallel frames. User can select the language for both frames and compare the syntaxes and the program execution between the two languages side-by-side. As discussed in chapter 2, pseudo language is often found to be the most suitable first language for students learning to program. However, the gap between pseudo and commercial languages (e.g. Java) can prove to be quite wide. The advantage of the parallel view – besides narrowing the gap mentioned – is proving that the imperative paradigm is implemented quite similarly in all supported languages.

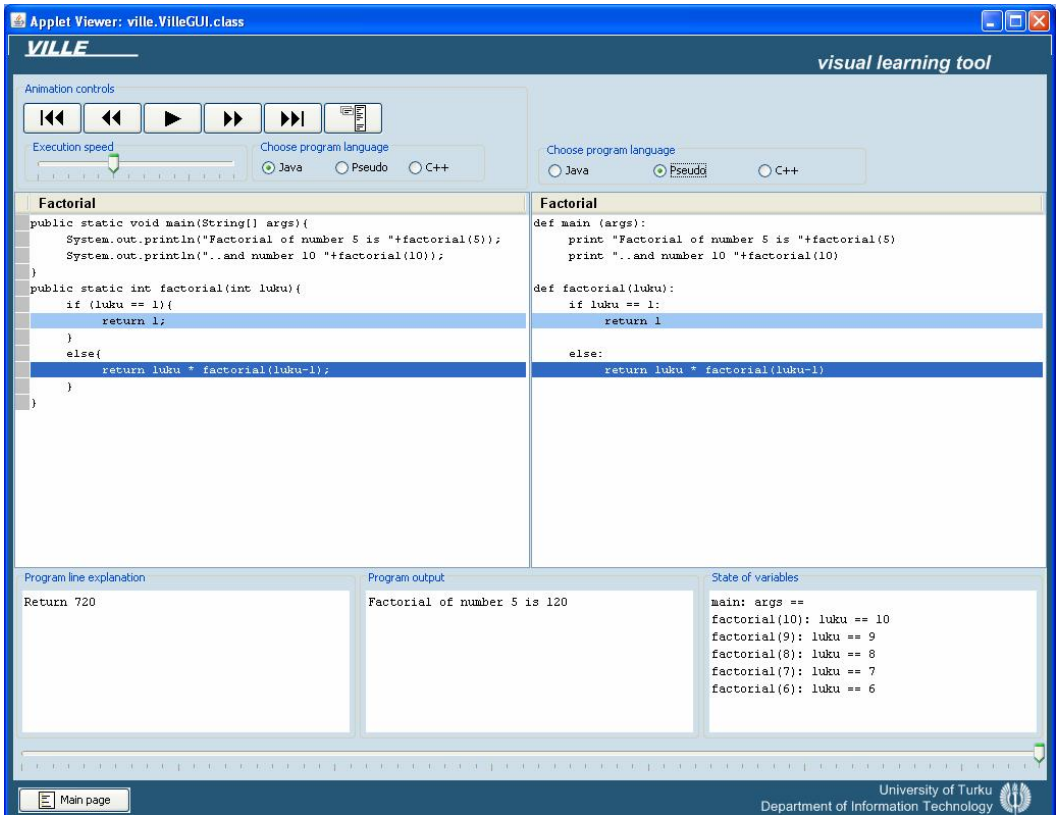

Figure 4: Visualization view of VILLE in parallel mode

Teachers have access to three additional views: the example creation and modification view, the syntax editor and the pop-up question editor. In example creation and editing view teachers can modify or create examples. In the syntax editor view teachers can modify existing syntaxes and create new syntaxes for programming languages. With the pop-up editor teachers can create pop-up questions for the examples that are shown to the students during the execution of examples. It is also possible to organize the examples and categories with simple controls.

Since the examples are executed on row-by-row basis, all supported languages have to have matching syntax lines. Hence, the pseudo language supported is a slightly modified subset of Python (see appendix 1 for pseudo code syntax); this pseudo code can however be translated to executable Python code with VILLE. The syntax editor (Figure 5) view has the syntax lines used in parsing Java code on the left side of the view, and the syntax lines of modifiable languages on the right side.

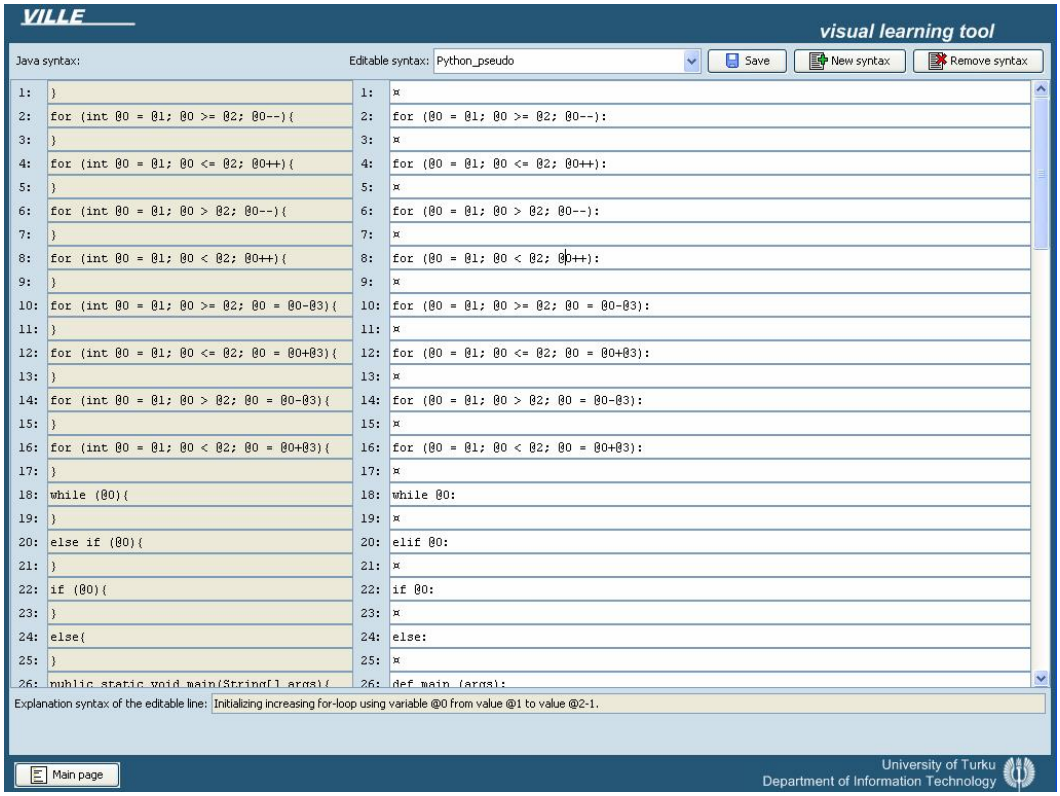

Figure 5: Syntax editor view in VILLE

4.3. Similar tools

There are many tools that visualize program execution. However, there are very few that use pseudo language, and to our knowledge none featuring a possibility to add or modify syntaxes and languages.

Jeliot3 and ALVIS Live are the most similar tools to VILLE from this paper's point of view. There are also some systems which can execute pseudo code but those systems are designed for teaching data structures and algorithms. More detailed analysis, comparison between VILLE and Jeliot3 and references can be found in Rajala et al. [15].

Table 1 presents comparison between VILLE and two popular program visualization tools, Jeliot 3 and ALVIS Live from the language perspective. Jeliot 3 [12] is a tool developed in University of Joensuu. In addition to code highlighting, it has a theatre view that visualizes variable and object states. ALVIS Live [6] is an algorithm animator, which uses pseudo code called SALSA. In ALVIS users can create algorithms by typing the pseudo code or by graphical tools.

	VILLE	Jeliot 3	ALVIS Live
Built-in languages	Java, pseudo code and C++	Java	SALSA pseudo code
Editable syntaxes	Yes	No	No
Add and define new languages	Yes	No	No
Examples	Various built-in, possible to add new ones	Various included with Windows executable version; possible to add new ones	Possible to add new ones
Program code viewed in	Selectable language; alternative view with two languages	Java	SALSA pseudo code
Graphical elements for visualizations	No	Yes	Yes

Table 1: Comparison of VILLE, Jeliot3 and ALVIS Live

From the language perspective, VILLE is the only tool that supports several programming languages. With it user can define additional programming languages and trace the execution of the program code side by side in parallel view with two selected programming languages. More over, teacher can create a pseudo code of his own and programs written with this pseudo code can be executed like any other programming language. Thus, by using VILLE the teacher can emphasize the programming language independency view. These features are absent from Jeliot3 and ALVIS.

Of course, there are differences between all tools when investigating the visualization techniques and methods. For example, in contrast to VILLE, which presents program and variable states primarily in textual form, these two other systems, Jeliot3 and ALVIS, use graphical symbols. These kinds of features are out of the scope of this article.

5. Discussion

From the learners point of view it is much more important to master the principles behind the basic programming concepts regardless of the selected programming language. More over, the syntaxes of the basic programming components are very

similar in all imperative languages. Thus, the use of pseudo language is recommended and reasonable for the basic programming courses. Still, we have to keep in mind that pseudo code is often perceived as non-executable programming language which can be seen as a drawback.

Many different pseudo languages have been defined and proposed for general use. However, the new ones are still created and presented by teachers because the existing ones do not fulfil their requirements. These self defined pseudo languages are very rarely interpretable or executable; however, by using VILLE the teacher can overcome this problem, and execute programs written in his own language. In addition, the execution is visualized step by step and its effects to variable states and program outputs are also presented. The language – with some limitations - can be self defined: the scope of supported features includes all the basic programming components (e.g. control structures, records, tables, strings etc.) covering majority of topics in the first programming courses. VILLE's built-in pseudo code is presented in appendix A.

Our research is going on and VILLE has been tested at University of Turku. The preliminary results indicate that students who used VILLE found it useful while learning programming basics.

6. Conclusions

VILLE is program visualization tool for teaching novice programming for lecture use and for self learning. The tool supports the programming language independent view which demonstrates the similarity between basic programming components in every imperative programming language.

The teacher can define his own pseudo language that suits his teaching needs and those programs written on that language can be executed and visualized in VILLE. In addition, the teacher can organize all course related programming examples in VILLE and publish those in the web for students to acquire.

VILLE constitutes a good amendment to introductory programming courses by offering a chance to handle basic programming related issues in more abstract way.

References

[1] Ala-Mutka, K. (2005). Ohjelmoinnin opetuksen ongelmia ja ratkaisuja. Tekniikan opetuksen symposium 20.-21.10.2005. Helsinki University of Technology. http://www.dipoli.tkk.fi/ok/p/reflektori/verkkojulkaisu/index.php?p=verkkojulkaisu.

[2] Brilliant, S.S. and Wiseman, T.R. (1996). The first programming paradigm and language dilemma. ACM SIGCSE Bulletin, 28(1):338-342.

[3] Grandell, L., Peltomäki, M., Back, R.-J. and Salakoski, T. (2006). Why Complicate Things? Introducing Programming in High School Using Python. Proceedings of the 8th Australian Conference on Computing Education, Hobart, Australia, 52:71-80.

[4] Hadjerrouit, S. (1998). Java as first programming language: a critical evaluation, ACM SIGCSE Bulletin, 30(2):43-47.

[5] Hu, Minjie. (2004). Teaching Novices Programming with Core Language and Dynamic Visualisation. Proceedings of the NACCQ 2004, Christchurch, New Zealand, 95-104.

[6] Hundhausen, C. D. and Brown, J. L. (2007). What You See Is What You Code: A 'Live' Algorithm Development and Visualization Environment for Novice Learners. Journal of Visual Languages and Computing, 18(1):22-47.

[7] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J. (2003). The BlueJ system and its pedagogy. Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13(4).

[8] Laakso, M.-J., Salakoski, T. and Korhonen, A. (2005). The Feasibility of Automatic Assessment and Feedback. Proceedings of Cognition and Exploratory Learning in Digital Age (CELDA 2005). IEEE Technical Committee on Learning Technology and Japanese Society of Information and Systems in Education. Porto, Portugal, 113-122.

[9] Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J., Sanders, K., Seppälä, O., Simon, B. and Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers, ACM SIGCSE Bulletin, 36(4):119-150.

[10] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students. ACM SIGCSE Bulletin, 33(4):125-140.

[11] McIver, L. and Conway, D. (1996). Seven Deadly Sins of Introductory Programming Language Design. Proceedings, Software Engineering: Education & Practice, 309-316.

[12] Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. Proceedings of the working conference on Advanced visual interfaces, Gallipoli, Italy, 373-376.

[13] Naps, T., Eagan, J. and Norton, L. (2000). JHAVÉ—an environment to actively engage students in Web-based algorithm visualizations. Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, Austin, Texas, United States, 109-113.

[14] Olsen, A.L. (2005). Using Pseudocode to Teach Problem Solving. Journal of Computing Sciences in Colleges, 21(2):231-236.

[15] Rajala, T., Laakso, M., Kaila, E. & Salakoski T. (2007). VILLE – Multilanguage Tool for Teaching Novice Programming. TUCS Technical Report, No 827, June 2007.

[16] Reek, M.M. (1995). A Top-Down Approach to Teaching Programming. ACM SIGCSE Bulletin, 27(1):6-9.

[17] Sajaniemi, J. & Kuittinen, M. (2003). Program Animation Based on the Roles of Variables. *In the Proceedings of the 2003 ACM Symposium on Software Visualization*, San Diego, California, 7–ff.

[18] Van Roy, P., Armstrong, J., Flatt, M. and Magnusson, B. (2003). The role of language paradigms in teaching programming. Proceedings of the 34th SIGCSE technical symposium on Computer science education, Reno, Nevada, USA, 269-270.

[19] Wells, M.B. and Kurtz, B.L. (1989). Teaching multiple programming paradigms: a proposal for a paradigm general pseudocode. Proceedings of the twentieth SIGCSE technical symposium on Computer science education, Louisville, Kentucky, United States, 246-251.

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics

Åbo Akademi University

- Department of Information Technologies

Turku School of Economics

- Institute of Information Systems Sciences

ISBN 978-952-12-1938-2
ISSN 1239-1891