INTRODUCTION

Codewitz is an innovative web-based hypermedia project to help the learning and teaching of programming languages. The aim of the project is to develop and produce interactive Learning Objects, ie. Codewitz material for computer assisted learning of programming in eLearning environment. Codewitz material is stored, shared and developed in Codewitz material bank, which is open to all partners to use, develop and produce the material. Furthermore, for teachers of programming and professional programmers Codewitz offers the benefits of an international network of other people who teach programming. For the students the material offers effective learning objects with interactive visualization to help them in the studies of programming languages.

Codewitz was invented in Tampere Polytechnic, Finland, in order to facilitate the learning and teaching of programming. The project has begun with the development Learning Objects for Java and C++ languages. All institutions are invited to join the project. At this point, in December 2003, there are 15 Codewitz partners in 10 countries from Europe and Asia.

In the year 2003 a 3-year sub-project of Codewitz, Codewitz/Minerva received funding from the European Union Socrates programme “Minerva, ODL and ICT in education”. The partner institutions in Codewitz/Minerva project are: University of Applied Sciences in Furtwangen in Germany, Tampere University of Technology in Finland, Reykjavik University in Iceland, Ventspils University College in Latvia, and Technical University of Civil Engineering in Bucharest, Romania. Tampere Polytechnic in Finland is the coordinator.

Codewitz/Minerva project started in October 2003. In order to have a sound foundation on the production of the Codewitz material in the project, a Needs Analysis was agreed to make. The Needs Analysis was made to establish a systematic approach on the issues in programming that the students find difficult to cope with and the teacher difficult to teach. In addition to the questionnaire, an examination of relevant research literature and articles that help the partners to develop a systematic approach in the development of the ideas and plans for implementation of the visualization drills. The literature research was made by Ms. Kirsti Ala-Mutka from Tampere University of Technology and it begins this Needs Analysis.

The students taking part in the programming courses as well as the teachers giving these courses in the Codewitz/Minerva partner institutes answered the questions in the on line –questionnaire. The questionnaire was planned in cooperation with the partners, the questions were identical to all institutions. In addition to the questions similar to all, the partner institutes could add one or two institute specific questions. The survey was conducted on line in November 2003 in all participating institutes.

There were altogether 565 students and 35 teachers participating in the survey (n=600). The amount of answers from students varied between 45 students from one institute to maximum 150 students from another. The amount of answers from teachers varied between 2 and 8 answers per institute.

Ms. Kirsi Immonen from Tampere Polytechnic processed the data of the questionnaire and presented some figures and graphics about the data. With the help of this, and the raw data, the partners analyzed the results concentrating on the results of their own institutes. The guidelines were to analyze the results of the institute, compare them to the average results of the survey and consider the results in the framework of the Codewitz material.

The questionnaire in paper form can be seen as annex 1. The raw data can be found in annex 2, the processed data (including the data of each institute) in annex 3.

PROBLEMS IN LEARNING AND TEACHING PROGRAMMING

- a literature study for developing visualizations in the Codewitz-Minerva project

Kirsti Ala-Mutka, Institute of Software Systems, Tampere University of Technology, Finland
Email: kirsti.ala-mutka@tut.fi

1. Introduction

The art of programming includes knowledge of programming tools and languages, problem-solving skills, and effective strategies for program design and implementation. A common approach in programming education is to first teach the basics of a programming language and then guide students towards effective strategies for the whole programming process. Therefore, the learning of the basic concepts is often emphasized; these form the basis for building more advanced skills. Codewitz project [5] aims at fostering the learning of the basic concepts and structures by creating dynamic visual learning materials for students. In order to create effective visualizations and educational strategies for their use, it is necessary to review the research previously published in the field.

This literature study aims at a brief overview on the problems and on visualization solutions in programming education. The emphasis is on novice programmers that are beginning their programming studies; these courses are often referred in the literature as CS1 courses. An excellent source for the study has been a recent review on the research relating to the educational study of programming, written by Robins et. al. [27]. Another vast source of information is an older collection of research papers on novice programmers, edited by Soloway and Spohrer [29]. Also the research by Milne and Rowe [21], who studied students' and tutors' perceptions of difficulties in object-oriented programming, relates closely to our work. However, their work differs from our needs analysis survey in that the group of respondents was considerably smaller (altogether 66 students and teachers) and the questions concentrated on the C++ language concepts. In addition, many other articles contain important information for the Codewitz-Minerva project and will be mentioned later.

The structure of this study is organized as follows. First, Section 2 recognizes some characteristics and problems of novice programmers. In Section 3, we discuss educational visualization approaches for programming. Section 4 concludes the study and proposes some approaches for the project. Finally, Section 5 lists the references that have been recognized as important resources for the project participants and other interested readers.

2. Difficulties in learning programming

Learning to program is generally considered hard, and programming courses often have high dropout rates. It has even been said, that it takes about 10 years for a novice to become an expert programmer [32]. Educational research has been carried out to recognize the characteristics of novice programmers and to study the learning process and its connections to the different aspects of programming. Lately also differences between procedural and object-oriented education approaches have been studied, as Java and C++ have become common educational languages. We will now explore these issues more closely.

2.1 Characteristics of novice programmers

By definition, novice programmers lack the knowledge and skills of programming experts. Several different separating factors have been studied in the literature and were also reviewed by Robins et al. [27]. Common features for novices seem to be that they are limited to surface knowledge of programs and generally approach programming "line by line" rather than at the level of bigger program structures. Novices spend little time in planning and testing code, and when necessary, try to correct their programs with small local fixes instead of more thoroughly reformulating programs [17]. The knowledge of novices tends to be context specific rather than general [15], and they also often fail to apply correctly the knowledge they have obtained. In fact, an average students does not usually make much progress in an introductory programming course [17]. This was also noticed by the study of McCracken et al. [20], who noticed serious deficiencies in student's programming skills in introductory courses.

Milne and Rowe [21] studied in their recent survey the difficulties of C++ programming by conducting a web-based questionnaire for both students and teachers. One of the most obvious results was that students rated having less difficulties than deduced from teachers' answers. This was suggested to result from the fact that students believe themselves that they have understood the issue, but teachers see the remaining deficiencies in programming courseworks and examinations. This supports the empirical observations of many teachers; programming novices often fail to recognize their own deficiencies.

Also the personal properties of the students affect their performance. There is no literature on significant differences in learning that would be caused by categories like gender or nationality, but general intelligence and mathematical or science abilities seem to be related to success at learning to program [19][4]. In a programming course, different student behaviours in confronting a problematic situation can be recognized. Perkins [25] named two main types: stoppers and movers. In problematic situation stoppers simply stop and abandon all hope of solving the problem on their own, while movers keep trying, modifying their code and use feedback about errors effectively. There are also extreme movers, "tinkerers", who cannot track their program, make changes more or less randomly, and like stoppers do not progress very much in their task.

All in all, there are effective and ineffective novices, i.e. students who learn without excessive effort and those who do not learn without inordinate personal attention [27]. Naturally, students' personal learning strategies and motivation affect their success in learning programming strategies. Robins et al. [27] stated that "Given that knowledge is (assumed to be) uniformly low, it is their preexisting strategies that initially distinguish effective and ineffective novices". Prior knowledge and practices can also be a major source of errors, especially when trying to transfer a step-by-step problem-solving solution directly from a natural language into a program [2]. The differences between the natural language and a programming language can easily cause problems. For example, some novices have understood that the condition in a "while" loop needs to apply continuously rather than it is tested once per iteration.

2.2 Different aspects of programming

Learning programming contains several activities, e.g., learning the language features, program design, and program comprehension. Typical approach in textbooks and programming courses is to start with declarative knowledge about a particular language. However, studies show that it is important to bring also other aspects to the first programming courses.

Several common deficits in novices' understanding of specific programming language constructs are presented in Soloway and Spohrer [29] and collected also by Pane and Myers [23]. For example, variable initialization seems to be more difficult to understand than updating or testing variables. Bugs with especially loops and conditionals are common, and actions that take place "behind the scene", like updating loop variables in "for" loops, are difficult for students. Students have often many misconceptions in their understanding or implementing of recursion. Interestingly, novices were noticed to be more successful at writing recursive functions after learning about iterative functions, but not vice versa [14]. Expressions that are syntactically close to each other or mean different things in different contexts cause often practical difficulties, e.g. "123" and 123, or word "static" in C language. The survey by Milne and Rowe [21] showed that pointers seem to be very problematic, both according to the students and teachers. It was suggested that for this reason also dynamic memory allocation, reference parameters and other issues that need understanding of memory contents and pointers ranked high in the difficulty ratings. Also many other complex language features, like templates, casting, polymorphism and function overloading were considered difficult in the survey.

However, the main source of difficulty does not seem to be the syntax or understanding of concepts, but rather basic program planning [27]. It is important to distinguish between programming knowledge and programming strategies [6]. A student can learn to explain and understand a programming concept, e.g., what does a pointer mean, but still fails to use it appropriately in a program. Winslow [32] noticed that students may know the syntax and semantics of individual statements, but they do not know how to combine these features into valid programs. Even when they know how to solve the problem by hand, they have trouble translating it into an equivalent computer program. To help students to learn both concept knowledge and strategies for their use, Deek et al. [7] developed a problem-solving approach for a programming course. In their approach, language features were introduced to students only in the context to specific problems, little by little. This was shown to have positive effects on students' course results and their programming confidence. More discussion on problem-based learning in computer science courses can be found in Kay et al. [13].

Students have often great difficulties in understanding all the issues relating to the execution of a program. Du Boulay [10] stated that "... it takes quite a long time to learn the relation between a program on the page and the mechanism it describes." Students have difficulties in understanding that each instruction is executed in the state that has been created by the previous instructions. He stated that their should be a simple "notional machine", that simplifies the language and the machine so that all the program transformations can be visible. For example, LOGO language is an example of this kind of approach [24], and was commonly used in 80's in introductory programming education. It is based on the fact that all the instructions concern moving a turtle that is drawing on a display, thus making the program state and transformations clearly visible. However, nowadays most of the universities are required to teach programming languages that are at least close to those used in the industry, and only rarely it is possible to select the approach for an introductory programming course based only on pedagogical reasons.

There is often little correspondence between the ability to write a program and the ability to read one. Programming courses should include them both. In addition, some basic test and debugging strategies should be taught [32]. Robins et al. [27] suggest that one more issue that complicates the learning of programming is the distinction between the model of the program as it was intended, and the model of the program, as it actually is. There are often mistakes in the design and bugs in the code. Also in working life, programmers face daily the need to understand a program that is running in an unexpected way. This requires an ability to track code to build a mental model of the program and predict its behaviour. This is one of skills that could be developed by emphasizing program comprehension and debugging strategies in the programming courses.

2.3 Object-oriented vs. procedural approaches

One of the claims for object-oriented approach has been that it is the natural way of conceptualizing real-world problems. However, studies do not seem to support that [27]. Rist even suggests that object-oriented programming adds the complexity of class structure to a procedural system [26]. Therefore, teachers using object-oriented approaches should not think that object-orientation is particularly easy for students, or that using OO from the very beginning relieves the teacher from teaching procedural programming issues.

Wiedenbeck et al. [31] studied students' comprehension of procedural and object-oriented small programs. They used programs that were functionally similar, but written either in Pascal/C or C++. Students in two comparison groups studied the programs and then answered questions that tested their understanding of different features and states of the program. The results showed that the distributed nature of control flow and "hidden" actions (e.g. constructor or destructor calls) made it more difficult for novices to form a mental representation of an OO program than of a corresponding procedural program. The class structure of the OO program made it a bit easier to understand program entities [30], but especially program flow and data flow issues were easier to understand from a procedural program.

The study by Milne and Rowe [21], recognized some concept difficulties relating to especially object-oriented paradigm. Both students and teachers considered constructors, virtual functions, and function overriding in inheritance among the most difficult issues in object-oriented programming. Interestingly, students saw virtual functions as only moderately difficult as opposed to teachers who considered them as the second biggest problem among the concepts listed in the survey. Also, several respondents that were teaching an OO language stated that they believe that OO languages should only be approached once the student has a thorough grounding in procedural programming.

Barr et al. [1] investigated student programming errors in object-oriented Blue environment. Blue language and environment [16] is especially designed to ease students' learning of object-oriented paradigm with a simple language syntax and graphical support for classes and objects. However, also they noticed that students had more problems with higher-level object-oriented areas than syntax and style. Their findings support the already mentioned superficial approach typical for novice programmers; students could point out syntax and indentation errors in a given program but did not notice serious logical errors. This was also noticed in the inability of creating meaningful comments for the programs, often students just wrote a comment verbally describing the operation of a single code statement.

3. Visualization approaches for programming education

Visualizations have been used for long time in computer science education, since they have been considered as beneficial for understanding and learning the abstract and complex concepts of the field. Most of the approaches, however, concentrate on algorithm animation and less emphasis has been given to visualizing the basic structure of programs and their execution. However, since the research on algorithm animation has been carried out for a long time, there are results that can be utilized also in designing approaches for visualizing the basic programming structures.

3.1 Using and designing visualizations

A working group on Improving the Educational Impact of Algorithm Visualization studied the use of visualizations in computer science education with three surveys and an extensive literature review. The results of the work are published in Naps et al. [22]. Their work, although primarily aimed at algorithm visualizations, contain many interesting notions that can be applied also for designing and using visualizations in more general contexts. In fact, most of the recommendations of the working group seem to apply to the visualization approaches presented in the next subsection.

According to the surveys, all respondents were confident that visualizations help students' understanding and learning of concepts. Main statement against visualizations was the amount of time that is required to create visualizations, to install and study the technology involved, and to integrate the visualizations to the courses. The workgroup collected eleven best practices for good educational visualizations. Some of them are already included in the present approach of Codewitz materials, e.g. providing multiple views, flexible execution control and explanations on visualizations. Also the possibility for using input data given by the user was seen important, since it engages the user more to following the visualization. In addition to these, there were also some new proposals, e.g. "Support learner-built visualizations", "Support dynamic questions", and "Support dynamic feedback".

As seen from the mentioned best practices, the active engagement of students was considered very important. Naps et al. state "... visualization technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity." They proposed a taxonomy for different forms of learner engagement with visualization technology: 1. No viewing, 2. Viewing, 3. Responding, 4. Changing, 5. Constructing, 6. Presenting. The categories 2-6 can naturally also be combined. The first category means that there are no visualizations, and the last one means that students have a possibility construct a visualization that they should present and explain to the class. The steps in between describe different states of engagement in increasing order. Step 2 Viewing, including step-by-step execution and multiple views, is a basic requirement for all the higher levels. Examples of the steps 3-5 follow:

· 3. Responding. Present questions relating to the visualization, e.g. "What is the value of the variable a in line 3?"

· 4. Changing. Give student a chance to change the behaviour of the program, e.g. with different input. This can be also combined to the responding task, e.g. asking for an input to produce a certain output or to cover all possible execution paths.

· 5. Constructing. Offer students a possibility to create their own programs - possibly according to a given task - and see them visualized.

The strong belief of educators for the benefits of visualization was very interesting, since there are no experimental studies that support the educational effectiveness of visualizations. To support the further research on this field, the workgroup developed a research framework to evaluate the educational effectiveness of visualizations. The basic idea is first to design task specific questions, then use visualizations in teaching and then test student knowledge. When comparing to another group with different teaching approach, it can be measured, whether visualizations have had positive effect to the learning. Similar approach could be used also in Codewitz-Minerva, when evaluating the quality and impact of the visualizations developed in the project.

3.2 Visualizations for basic programming concepts

Recursion was the only basic programming concept that has several visualization approached in the literature, e.g. [10] and [29]. In addition, there are many recursion visualizations that work on high level, not showing the actual code. Several tools have been developed to generally visualize program execution line-by-line, e.g. Teaching Machine [4], Thetis [11], and AnimPascal [31] with their traditional approach. PlanAni [30] is an example of a more recent work with an emphasis on the roles of the variables. However, in this section we concentrate on examples that have been developed to be used as teaching materials that introduce only few concepts at a time.

Rowe and Thorburn [28] developed a web-based tool called Vince to help students understand the execution of C programs. Their tool visualizes the workings of a C program step by step, showing the contents of the computer memory and providing user with explanations of each step. Tool display shows similar issues than present

Codewitz materials, but more closely related to the actual execution model of the functions and the physical structure of the computer memory. The idea for the tool came from the program code debuggers, and this can be seen in the interface implementation. In addition to the example programs designed by the teacher, students can also provide their own programs or modify teacher's program, and then see how these programs are executed. The code examples are organized into prepared tutorials that students can study on their own. The study showed that Vince had had a positive effect on students' learning, and it was considered a good supplement for an introductory programming course.

Although not very graphical, another interesting work is published by Elenbogen et al. [11]. They have developed a set of interactive web exercises for learning various features of C++. Students can modify small program templates and check how their code affects the program behaviour. The questions are set so that they ask students about their understanding of the shown code segments, and students can execute the program themselves to find out the answers. On some exercises the execution is shown only by the results of the functions, but in the example of the classic Towers of Hanoi, also the animated towers are presented. Authors state to be satisfied with their solution that enhances the learning in an introductory course. These applets can be also used as automatically assessed laboratory exercises even on large courses.

Nowadays, in the era of e-learning, great emphasis is given to the reusability and portability of learning materials. A concept of learning objects that cover a certain aspect of learning has been introduced and a final version of a standard description for learning object metadata was reached in 2002 [12]. This forms the basis for the working approach of Boyle and his team in London Metropolitan University [18]. Boyle [3] has been developing materials for visualizing certain programming concepts and describes them as pedagogically rich compound learning objects. Each object is designed towards a certain learning goal. The approach composes together several features visualizing the execution of e.g. a "while" loop and giving students questions and tasks relating to the concept. As opposed to larger visualization or learning support systems, the benefit of this approach is that this kind of objects can be easily taken in use and incorporated to education. Independent learning objects also make it possible to develop common repositories for reusable and portable learning materials. This is close to the basic principles of the whole Codewitz project.

4. Suggestions for the Codewitz project

Learning to program is a complex task and teachers should design their teaching approaches carefully. The traditional approach of concepts first is common and found to be effective, although also different approaches exist. There are plenty of typical novice programming errors, many of which are well collected into the technical report by Pane and Myers [23]. They present features and issues that should be taken into account when designing support systems for programming novices, and their work is useful also in connection to developing Codewitz materials.

However, the biggest problem of novice programmers does not seem to be the understanding of basic concepts but rather learning to apply them. Robins et al. [27] suggest that teachers should focus more on combination and use of these features, especially on the underlying issues of basic program design. For example, building visualization examples of the basic structures and their combinations in different situations could promote students understanding of different strategies and build a mental "library" of different solution schemas for program design.

Comprehending program states has shown to be difficult already with procedural programming and seems to be even more so when using object-oriented paradigm. Therefore, visualizing the execution of small object-oriented programs and showing a full life cycle of objects, would probably help many students. Since pointers and memory contents cause difficulties with many complicated object-oriented features, visualizations could be especially useful for illustrating program flow with these. Also the complicated inheritance features could be explained with dynamic visualizations, e.g., showing how virtual function calls are resolved or what happens when an object of a child class is instantiated.

Existing visualization approaches for basic programming structures are good examples also for Codewitz project. Also the research on algorithm simulation and visualization in educational settings can provide important information for developing Codewitz simulations. For example, the framework developed by Naps et al. [22] could be utilized when designing the research approach for evaluating educational effectiveness of Codewitz materials. Since large program execution visualization systems already exist, Codewitz should keep its concentration on developing independent small-scale examples. Small well documented educational entities (i.e. learning objects) that are easily usable and technically portable can be incorporated to the education in many ways without extensive training for teachers or students.

As opposed to the present materials, the partners of the Codewitz-Minerva project could also develop approaches that support more than just comprehension skills. Since learning problems are often connected to more advanced issues than individual concepts, visual applications could be directed to develop program generation, modification and debugging skills. If small examples, emphasizing few concepts at a time, could be developed to support students' active programming skills, they would also better engage the student in the learning situation. Since success in creating a functional program is a major positive force on students' traditional programming work, also visualizations could have more problem-solving nature instead of only representing concepts.

Finally, teachers should always remember that technical tools and visualizations are just learning aids and materials. It is necessary to consider educational issues when designing these technical aids, but however good the materials are, they can still be used either effectively of ineffectively. Teachers need to always thoroughly design their instructional approach to the issues on the course and how the aiding materials are incorporated to the education. Just giving a web address for students and advising them to use the Codewitz materials by themselves, will not reach all the students who would need the support for their learning. Material usage should be carefully designed and the settings should be well documented so that the project results will contain, in addition to the materials, also tested teaching strategies as examples of the use of them.

---

5. References

[1] Barr, M., Holden, S., Phillips, D. & Greening, T. (1999). An exploration of novice programming errors in an object-oriented environment, SIGCSE Bulletin, 31(4), pp. 42-46.

[2] Bonar, J. & Soloway, E. (1989). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers, in Soloway & Spohrer: Studying the Novice Programmer, pp. 325-354.

[3] Boyle, T. (2003). Design principles for authoring dynamic, reusable learning objects. Australian Journal of Educational Technology, 19(1), pp. 46-58, available at http://www.ascilite.org.au/ajet/ajet19/boyle.html, referenced 2.12.2003.

[4] Bruce-Lockhart, M. & Norvell, T. (2000). Lifting the hood of the computer: program animation with the Teaching Machine. Proceedings of the Canadian Conference on Electrical and Computer Engineering, vol 2, pp. 831-835.

[5] Byrne, P. & Lyons, G. (2001). The effect of student attributes on success in programming, Proceedings of the 6th annual conference on Innovation and technology in computer science education, pp. 49-52.

[6] Codewitz project. http://www.codewitz.net/, referenced 2.12.2003.

[7] Davies, S. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39, pp. 237-267.

[8] Deek, F., Kimmel, H. & McHugh, J. (1998). Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. Journal of Engineering Education, 87, pp. 313-320.

[9] Du Boulay. (1989) . Some difficulties of learning to program. In Soloway & Spohrer: Studying the Novice Programmer, pp. 283-300.

[10] Elenbogen, B.S., Maxim, B.R. & McDonald, C. (2000). Yet, more Web exercises for learning C++, Proc. of the 31st SIGCSE Technical Symposium on Computer Science Education SIGCSE Bulletin, pp. 290-294.

[11] Freund, S. & Roberts, E. (1996). Thetis: an ANSI C programming environment designed for introductory use. Proc. of the 27th SIGCSE Technical Symposium on Computer Science Education, pp. 300-304.

[12] George, C. (2002). Using visualization to aid program construction tasks. Proc. of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp. 191-195.

[13] IEEE Learning Technology Standards Committee. Final LOM Draft Standard, http://ltsc.ieee.org/wg12/20020612-Final-LOM-Draft.html, referenced at 2.12.2003.

[14] Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J. & Crawford, K.(2000). Problems-based learning for foundation computer science courses. Computer Science Education, 10(2), pp. 109-128.

[15] Kessler, C. & Anderson, J. (1989). Learning flow of control: recursive and iterative procedures. In Soloway & Spohrer: Studying the Novice Programmer, pp. 229-260.

[16] Kurland, D., Pea, R., Clement, C. & Mawby, R. (1989). A Study of the development of programming ability and thinking skills in gih school students. In Soloway & Spohrer: Studying the Novice Programmer, pp. 209-228.

[17] Kölling, M. & Rosenberg, J. (1996). Blue - A Language for Teaching Object-Oriented Programming, Proc. of the 27th SIGCSE Technical Symposium on Computer Science Education, pp. 190-194.

[18] Linn, M. & Dalbey, J. (1989). Cognitive consequences of programming instruction. In Soloway & Spohrer: Studying the Novice Programmer, pp. 83-112.

[19] London Metropolitan University. Learning Objects for Introductory Programming. http://www.unl.ac.uk/ltri/learningobjects/index.htm, referenced 2.12.2002.

[20] Mayer, R., Dyck, J. & Vilberg, W. (1989). Learning to program and learning to think: what's the connection? In Soloway & Spohrer: Studying the Novice Programmer, pp. 113-124.

[21] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, SIGCSE Bulletin, 33(4), pp. 125-180.

[22] Milne, I., Rowe, G. (2002). Difficulties in Learning and teaching Programming - Views of Students and Tutors, Education and Information Technologies, 7(1), pp. 55-66.

[23] Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. & Velázquez-Iturbide, J.Á. (2003), Exploring the role of visualization and engagement in computer science education, SIGCSE Bulletin, 35(2), pp. 131-152.

[24] Pane, J. & Myers, B. (1996). Usability Issues in the Design of Novice Programming Systms, School of Computer Science Technical Reports, Carnegie Mellon university, CMU-CS-96-132, available at http://www.cs.cmu.edu/~pane/ftp/CMU-CS-96-132.pdf

[25] Papert, S. & Solomon, C. (1989). Twenty things to do with a computer. In Soloway & Spohrer: Studying the Novice Programmer, pp. 3-27.

[26] Perkins, D., Hanconck, C., Hobbs, R., Martin, F. & Simmons, R. (1989). Conditions of learning in novice programmers. In Soloway & Spohrer: Studying the Novice Programmer, pp. 261-279.

[27] Rist, R. (1996). Teaching Eiffel as a first language. Journal of Object-Oriented Programming, 9, pp. 30-41.

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

[29] Rowe, G., Thorburn, G. (2000). VINCE - an on-line tutorial tool for teaching introductory programming, British Journal of Educational Technology, 31(4), pp. 359-11p.

[30] Sajaniemi, J. & Kuittinen, M. (2003). Program Animation Based on the Roles of Variables. Proceedings of the ACM 2003 Symposium on Software Visualization, pp. 7-16.

[31] Satratzemi, M., Dagdilelis, V. & Evagelidis, G. (2001). SIGCSE Bulletin, 33(2), pp. 137-140.

[32] Soloway, E. & Spohrer, J. (1989). Studying the Novice Programmer, Lawrence Erlbaum Associates, Hillsdale, New Jersey. 497 p.

[33] Tung, S.-H., Chang, C.-T., Wong, W.-K. & Jehng, J.-C. (2001). Visual representations for recursion, International Journal of Human-Computer Studies, 54(3), pp. 285-300.

[34] Wiedenbeck, S. & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles, International Journal of Human-Computer Studies, 51(1), pp. 71-87.

[35] Wiedenbeck, S., Ramalingam, V., Sarasamma, S. & Corritore C. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. Interacting with Computers, 11(3), pp. 255-282.

[36] Winslow, L.E. (1996). Programming pedagogy - A psychological overview. SIGCSE Bulleting, 28(3), pp. 17-22.

TAMPERE POLYTECHNIC

Esa Kujansuu, School of Technology and Forestry, Tampere Polytechnic, Finland
Email: esa.kujansuu@tamk.fi

1. Introduction

There are three schools in Tampere Polytechnic: School of Technology and Forestry, Business School and School of Art and Media. Following questionnaire was organised by School of Technology and Forestry. The coordination of Codewitz project is also implemented by School of Technology and Forestry.

In School of Technology and Forestry programming is taught mostly in the degree program of computer systems engineering. In some other degree programs there are basic courses available as well as in optional studies and in Open and Virtual Polytechnic. Computer system engineering program has four specialisation options: Computer Engineering, Software Engineering, Telecommunications Engineering and Electronics.

The students taking part in this survey were all from Computer System Engineering degree program. Every year there are about 600 new students beginning their studies in School of Technology and Forestry. In the degree program of the Computer System Engineering there are every year 125 new students. Computer System engineering is a four-year program.

2. Questionnaire to the students

In this summary we focus on the results of the students from Tampere Polytechnic and on the results of other students in this survey. We also take a look at if there are differences between the results of teachers and students. The comparisons are made according to the average values of the answers.

2.1. Results from Tampere Polytechnic

About one hundred students took part in this survey (N=106). The most of these students were second year students. These students have already learned the basics and they should not have yet forgotten what was difficult last year. About 50% out of these students had earlier programming experience before they started their studies in Tampere Polytechnic. However, the most of these students thought they had low programming skills before studies. The most of the students of Tampere Polytechnic study with the computer at home.

Programming development environment and access to computers were found easy. The level of difficulty of understanding programming structures and learning the syntax was found easy or average. The most difficult areas in the course contents were the design of a program to solve a problem, the division of procedures into functions and classes as well as finding bugs in one’s own programs.

In programming structures or concepts the students considered the basics like variables, selection and loops to be easy. Average difficulty or above they had experienced in the areas of recursion, arrays, parameters, structured data types, abstract data types and input or output functions. The most difficult areas were pointers and references, error handling and the use of language libraries.

The students felt they learn best through the exercises or in practical session, whereas learning was most difficult in lectures. The students thought that example programs and interactive visualizations offer the best help in learning. The programming course book was the least helpful in the learning process. However, only 25% of the total 106 students had seen or used visualization materials. Opinions about the best ways to use these materials varied almost equally to each possible selection. Using them individually during practical sessions had the highest amount of hits (19).

2.2. Differences in the teachers' answers compared to the students' answers

There were only four (4) teachers answering to this questionnaire, but as there are only six (6) full-time programming teachers working in School of Technology and Forestry in Tampere Polytechnic they present the majority of the programming teachers. Because of the tight schedule we were not able to get Business School to join this survey. Teachers who took part in the survey teach a variety of courses from basics to advanced level courses.

In the area of difficult issues in learning programming the answers from teachers were almost equal to the students. Only difference was that the teachers thought these topics would be slightly more difficult than the students in general had indicated, but the emphasis was similar.

In the area of course contents the most significant difference between students and teachers was again the level of difficulty in general, but now there were some differences in the emphasis. The students do not think recursion to be as difficult as the teachers do. Also there is a big difference in the opinion of difficulty of using language libraries. Teachers think the use of language libraries is much easier than the students indicated. In other areas the answers from students and teachers were quite similar.

Both teachers and students think that learning happens best in practical sessions. Also teachers and students agree the least effective learning situation is during lectures, and that the best learning resources are example programs and interactive visualizations.

2.3. Differences between Tampere Polytechnic and general Student Results (Tampere Polytechnic excluded)

The whole amount of answers was 565 and about 19 % of the students' answers were from Tampere Polytechnic. In general, the results of Tampere Polytechnic students follow the results from other universities. There are minor differences in certain topics, but basically the results are similar to the other universities.

In course contents the students of Tampere Polytechnic found dividing functionality into functions slightly more difficult than the average result of other students in this question. In programming structures our students found all aspects of programming structures represented in the questionnaire more difficult than the students of other universities. Nevertheless, the emphasis was similar.

In learning and teaching situations, the students of Tampere Polytechnic had some small differences in comparison to other universities. They felt they learn better than others through exercises and during discussions in small groups, but they felt they learn a bit less than others while studying and working by themselves.

Tampere Polytechnic students found interactive visualizations more helpful than others though only 26 of them said yes when asked if they had seen or used Codewitz materials.

2.4. Conclusions concerning the questionnaire

We can see in the survey that the same topics are difficult for Tampere Polytechnic students as for the other students. We were expecting more differences as most of the students taking part in the survey were second-year students and they had learned the basics structures last year. During the second year of their studies they are learning object-oriented programming.

The most interesting result in Tampere Polytechnic can be found in the situations where to learn programming, as well as in the conception of the difficulty level of the recursion. Surprisingly both teachers and students think they learn less during lectures. This can be caused by the need of abstract thinking in programming which is hard to absorb by only receiving information during lectures. This result can also appear when both groups think the practical programming tasks are the goal instead of the theoretical knowledge of programming.

The teachers in Tampere Polytechnic think recursion is a more difficult topic than the students consider it to be. The students do not think it would be easy, but the teachers rated the difficulty level of recursion higher. This came up probably because recursion is a difficult task for teachers to present during lectures so that students would learn it.

From the questionnaire we should take the most difficult topics for students and develop helpful interactive materials for topics where the average of answers were above average difficulty. These topics are for example pointers and references, error handling and recursion.

Tampere Polytechnic Summary

In this summary list titles are written in the order of difficulty:

Difficult in Course Content

- The design of a Program to solve a problem
- The division of procedures into functions and classes
- Finding bugs in one’s own program

Difficult in Programming Structures

- Pointers, references
- Using language libraries
- Error handling
- Recursion

3. Review of existing material

The following review is based on existing material in the Codewitz Material Bank (CMB) on November 2003. All the materials at the moment in CMB are developed by Tampere Polytechnic. This is also the reason, why we have experience in using the materials in courses. The following review is thus based on the experiences of using materials in teaching. The Codewitz materials have been used in traditional courses (classroom teaching). We have also tested these materials in partly virtual courses in Tampere Polytechnic.

The existing material in the CMB at the moment is planned for the first courses of Java and C++. The majority of the examples and exercises are about understanding loops. According to the questionnaire, students find the difficulty of loops easy or average. The loops are not the most difficult part in the first programming courses, but many students have found loops to be the first difficult topic in learning programming. In a basic programming course arrays normally follow the loops in the teaching schedule. Arrays can mostly only be used by loops. In this sense if the loops are not understood properly, the next topic will be more difficult to learn. Due to this, the amount of the exercises about loop can be justified to be the greatest after the first year of the three-year development period.

As a result of this questionnaire we have to emphasize the topics, which were found the most difficult while learning to become a programmer. These topics were pointers and references, error handling and using language libraries.

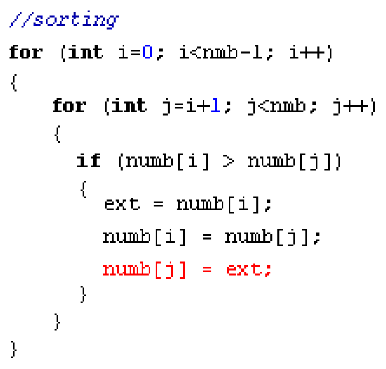
In the following chapters there are excerpts from the material collection.

3.1. Examples

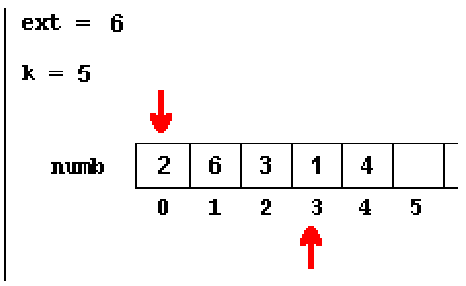
As a result of the use of the examples during lectures and in practical sessions following examples have served best the lecturer’s interests. These examples have a visual graph of the situation, which is difficult to present on the blackboard. These were chosen because they also match quite well with the results of this survey.

Sorting Integer Array

In this example the idea of the sorting algorithm is visible by showing how the assistant variable is used when the values in the array must change place. The idea is shown below as steps of the visual presentation dealing with the changing places of values in array. A student can follow through the whole process of sorting the array as presented here:

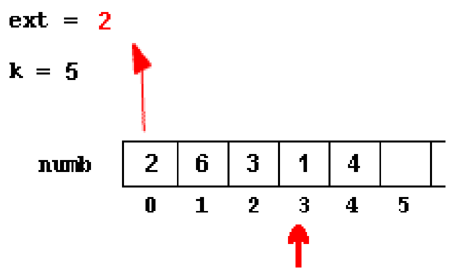


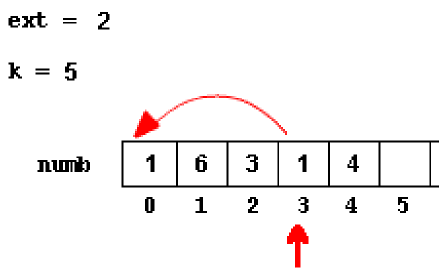
```
//sorting
for (int i=0; i<nmb-1; i++)
{
    for (int j=i+1; j<nmb; j++)
    {
      if (numb[i] > numb[j])
      {
        ext = numb[i];
        numb[i] = numb[j];
        numb[j] = ext;
      }
    }
}
```



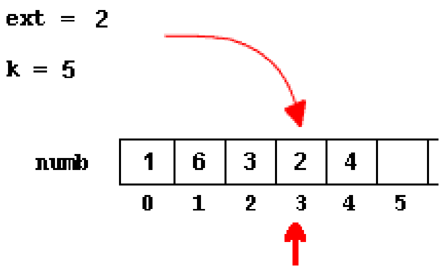
step 1

step 2





step 3 step 4



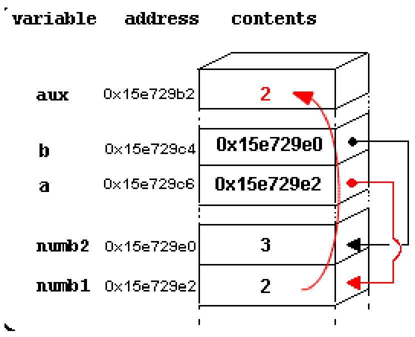
Visualizing the idea of sorting as visualized above has been found helpful among the students and teachers at Tampere Polytechnic. In the future, it could be useful to visualize the most commonly used sorting algorithms like this, because there exists only one example about this topic at the moment.

Pointers as parameters of a function

In this example there are two separate programs. The one shows the use of pointers as parameter variables and as well as the ground for the change of values of main function variables. The other is similar with only one exception: the parameter variables are not pointers. The other program is to show to a student why a function can't affect variables of main function if pointers or references are not used.

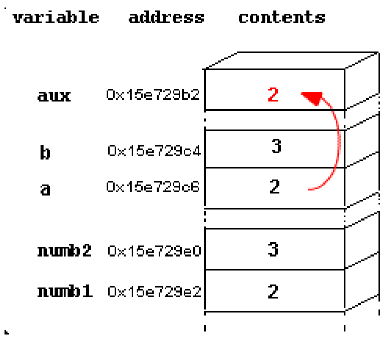
With pointers:

```
void change(int *a, int *b)
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```



Without pointers:

```
void change(int a, int b)
{
    int aux;
    aux = a;
    a = b;
    b = aux;
}
```



This example has proofed to be useful and helpful among students. In the survey pointers were considered to be one the most difficult topics in programming. At the moment there are only three pointer examples in CMB and there are no examples about

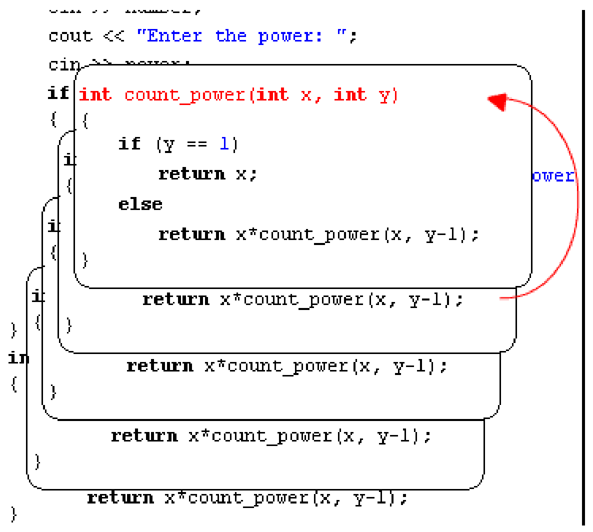
----

`new` operator. We should concentrate on pointers in the future work. Examples of the `new` operator should be taken under development.

Calculating the power using Recursion

Recursion was one of the difficult topics according to the survey. There is only one example in CMB at the moment. There should be more recursion examples available. The focus should be on the examples about the actions, which are not possible to be done without recursion. The following example could be implemented also without recursion.

In this example the goal is to show how the recursive function calls actually work. The method of visualization can be seen below.

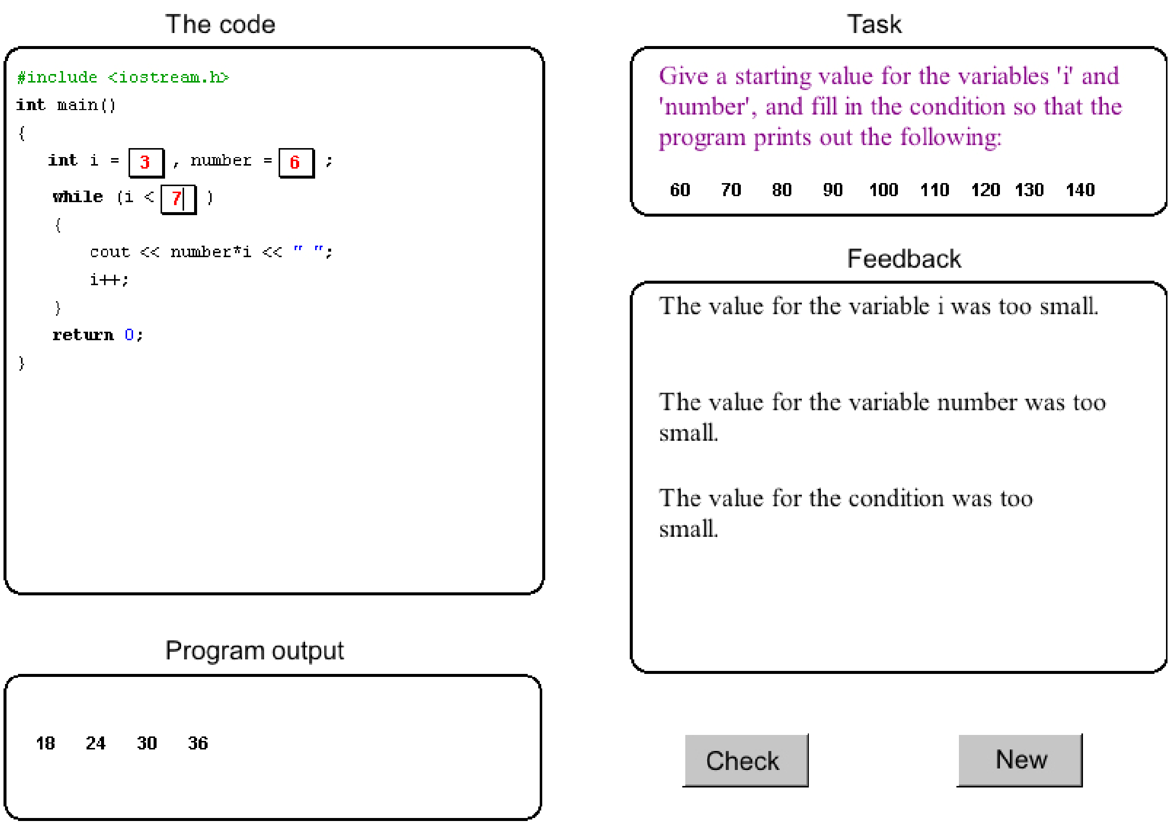


```
cout << "Enter the power: ";
cin [illegible]
int count_power(int x, int y)
{
    if (y == 1)
        return x;
    else
        return x*count_power(x, y-1);
}
        return x*count_power(x, y-1);
        return x*count_power(x, y-1);
        return x*count_power(x, y-1);
        return x*count_power(x, y-1);
}
```

3.2. Exercises

In the exercises students need to fill in the missing values of the program code according to the given task and program output. These exercises have been planned on the loops. On other areas the exercises have not met the goals we had. We at Tampere Polytechnic have found it troublesome to find the correct level of difficulty in exercises. Many of those are too easy whereas some are even too hard for students. The level and the way of giving feedback is also a complex task to figure out. During the years we have had more failures than successes in the topic of making exercises. In the future work of exercises we need to figure out what would be a good way to make exercises or whether we should only concentrate on examples.

The power of exercises has been seen on loops. The students who have had trouble in understanding the work of loops have found these exercises helpful. When students use these exercises they have understood the meaning of condition and they have adopted the idea of how to calculate the amount of loops needed to complete the task. Below is one exercise in which we at Tampere Polytechnic think the level of difficulty is at the most appropriate level.



```
#include <iostream.h>
int main()
{
   int i = 3 , number = 6 ;
    while (i < 7 )
    {
        cout << number*i << " ";
        i++;
    }
    return 0;
}
```

Give a starting value for the variables 'i' and 'number', and fill in the condition so that the program prints out the following:

```
60   70   80   90   100   110   120   130   140
```

The value for the variable i was too small.

The value for the variable number was too small.

The value for the condition was too small.

```
18   24   30   36
```

In CMB there are both examples and exercises, which we consider not to be successful. We have however stored these materials into CMB. The partners of the Codewitz project can now decide themselves, which are the examples and exercises to be used as learning resource for the students. Not so successful learning objects can also be good examples of useless work. Thus having them in CMB can reduce the time spent on useless work.

UNIVERSITY OF APPLIED SCIENCES IN FURTWANGEN

Harald Gläser and Friedbert Kaspart, Faculty of Computer Science, University of Applied Sciences in Furtwangen, Germany

Email: harald.glaeser@fh-furtwangen.de, friedbert.kaspart@fh-furtwangen.de

1. Introduction

The survey has been carried out in the Faculty of Computer Science at the University of Applied Sciences in Furtwangen (FHF). The Faculty of Computer Science consists of two directions for study: General Computer Science (AI), with focus on software development and Computer Networking (CN) with focus on network technologies. The studies in each direction last 8 semesters, i.e. 4 years. The students graduate to "Diplominformatiker". The 3. and 6. semesters include practical training, normally in the industry. In the 8. semester the students do their diploma theses, which takes 4 to 6 months. The studies of 8 semesters consist of two semesters of basic studies and six semesters of major studies.

The capacity per year and per study direction is 70 students, which means that every semester about 35 students in AI and CN alike begin their studies of computer science. These numbers may however vary, for example this winter there have been 50 beginners in CN.

The structure of the basic education in programming is the same for AI and CN. In the first semester there is an obligatory course in programming in Java (4 hours of lectures per week and 4 hours practice) and in the second semester the students of AI and CN have to take the course programming in C++ (2 hours of lectures per week and 4 hours practice per week). The groups for practice should not exceed 20 participants. This means that several groups for one course are necessary. There are some courses related to programming in the first year, i.e. software engineering and databases.

After the first two semesters the offer of courses in respect of software development is bigger for students of AI than for those of CN. The students of AI in general also choose more software-oriented themes for their practices in the industry. The overlap between AI and CN in the major part of the study is mainly in the development of distributed applications, i.e. middleware.

2. Analyzing the survey

Our survey begins by explaining the situation with our students before they enter the FHF. Then it discusses the problem areas of the students while learning programming. It continues with the discussion of the preferred learning situation. Here a small side step is made to make some remarks about the optimal degree of freedom in the students' learning situation. We examine the relationship between the students' view reflected in the data and the teachers' opinion about the students. We discuss the situation of the computer infrastructure and where the students mainly do programming. Some remarks follow about eventual inconsistencies in the details of the raw data. The analysis is finished by the conclusions, i.e. what did we learn from an e-learning tool like Codewitz.

2.1. "Conditioning" of FHF students before entering university course

Approximately 76% of the students entering a university course at the FHF have already experience with at least one programming language. They consider their programming skills at this beginning to be low or moderate. On the other hand, the rest of the students have no experience with programming languages at all. This results in an extremely inhomogeneous level of the beginners. Any course in this situation should pay attention to this fact. The contents of the course must be aligned according to the needs of the real beginners. Nevertheless the already advanced students should not be bored. During a lecture this cannot be achieved. Due to the different knowledge background (and intellectual capacities), the students have different "understanding velocities". Understanding velocity is the rate, at which a student is able to understand programming concepts. Teaching velocity on the other hand is the rate, at which a teacher is teaching these programming concepts. Best learning is achieved, when the understanding velocity of the student is equal to the teaching velocity of the teacher.

In case the teacher is able to adjust his teaching velocity to the learning velocity of an average student about one half of the students cannot follow, because their understanding velocity is too low. The other half is bored, because their understanding velocity is not used fully by the teaching velocity of the teacher. With the restriction, that the teaching velocity should be adapted to the needs of the beginners, in case of FHF about 76% of the students will be bored, because they are not beginners. We will return to this point in our conclusion.

2.2. Problem areas of students while learning programming

The students of FHF altogether seem to have problems with some very distinct programming concepts. All students have identified the following areas as “difficult”:

- pointers
- recursion
- structured data types
- classes
- error handling
- libraries

These are the more advanced programming concepts in comparison to the more basic concepts as “if” and “loop”. “if”and “loop” have been rated as “easy” as well as variables and arrays. Nevertheless, if the division between beginners and advanced students is made, another picture appears. Advanced students rate the problem to “Understanding how to design a program to solve a certain task” as “average”, whereas the beginners rate it as “difficult”. Similarly, “variables”, “if” and “loops” have been rated significantly more difficult by the beginners in comparison to the advanced students. It is interesting to observe, that this relation is inversed in case of “pointers”, “abstract data types”, “error handling”, “libraries”. This may confirm an old finding, that the advanced concepts of object oriented programming are more easily learned by total novices in programming than by students already accustomed to procedural programming.

	Using program development environment, compiler, editor etc.	Gaining enough access to computers/networks	Understanding programming structures	Learning the programming language syntax	Understanding how to design a program to solve a certain task	Dividing functionality into procedures, functions and/or classes	finding bugs from my own program	variables (lifetime, scope)	selection structures, (if, switch ...)	loop structures (while, for, do-while ...)	recursion	arrays
no prior prog. skills	2,15	2,62	3	2,62	3,62	3,31	3,77	2,46	2,46	2,38	3,38	2,54
with prior prog.skills	2,14	2,12	2,93	2,71	3,24	3,02	3,29	2	1,83	1,83	3,17	2,43

	pointers, references	parameters	structured data types (e.g. records, structs)	abstract data types (e.g. classes)	input/output functions, file handling	error handling (by program flow, exceptions , etc.)	using language libraries (e.g. STL in C++, Java libraries)
no prior prog. skills	3,15	2,69	3,92	3,08	2,92	3,42	3,5
with prior prog.skills	3,74	2,93	3,62	3,69	3,21	3,6	3,69

Table 1: Rating 1 = very easy, rating 2 = easy, rating 3 = average, rating 4 = difficult, rating 5 = very difficult

Every programming concept has been rated at least as “average” (2.5 to 3.5) by at least one group of the students with and without prior programming knowledge.

What has been rated as “very difficult” (<3.5)? “Understanding how to design a program to solve a certain task”, “Finding bugs from my own program”, “structured data types”, have been rated “very difficult” by beginners. This rating was given to “pointers, references”, “abstract data types (classes)”, “error handling” by the students with prior knowledge. “using language libraries (e.g. STL in C++, Java libraries)” was rated difficult by both.

2.3. Preferred learning situation

In the following table the answers of FHF students to the question “When do you feel that you learn issues about programming, personally?” is summarized.

	in lectures	in exercise and	in practical	while studying by	while working by

		discussion sessions in small groups	sessions (in computer room)	myself with teaching materials	myself on programming courseworks
no prior prog. skills	2,69	3,46	3,69	3,31	3,46
with prior prog.skills	2,9	3,57	3,6	3,83	3,71

Table 2: the numbers have to be read as follows: 5= almost always, 4=often, 3=sometimes, 2=a couple of times, 1=never.

The practical sessions have been rated as very effective, regardless whether the students had prior knowledge in programming or not. The lectures have been rated the worse, but still with “sometimes”. The students without prior knowledge seem to have somewhat more difficulties with the lectures than the students with prior knowledge. This strengthens the above argument with the different understanding velocities of these groups of students. The most prominent difference between students with prior knowledge and without prior knowledge appears in situations, where the students work for themselves. This situation has been rated quite good (“sometimes” to “often”) but even higher by the students with prior knowledge. It seems that the absolute novice needs some help in the beginning. After that he is more and more able to learn by himself.

The right degree of freedom

Let us interpret the results found above in terms of degrees of freedom. The situation with the least degree of freedom is the lecture. The highest degree of freedom has the study of teaching material, i.e. books, etc. The optimum seems to be in between. Steering of the learning process is necessary, the lectures serve to define the material to learn. Consequently, we don’t want to skip lectures, but we know that we need more for success. We know that the students use books but we also know that we have to challenge them to read in the book. We ask them to do some programming work and they look in the book or script for the details, which are needed to be able to fulfil the task. We steer the learning process by defining a programming task. Hence the centre of motion (and of motivation) is the challenge, and that the students have the right feeling. The data reflect that the students know: all is necessary and the most important part is the work on programming course work. The Codewitz material has to fit in this environment. The students, which have deficiencies, must be able to orient themselves in the Codewitz material to find the training material, which helps them to fill the knowledge gaps. We as teachers may be the advisors to tell the student which part of the training material to work with in order to cure the deficiencies. When we look for ideas to complete the Codewitz material, we may look through our programming course works and identify parts as components, the construction of which can be trained by Codewitz material.

2.3. Students view versus teachers opinion

When we compare the ratings of the students on how difficult the different topics appear to them, with the opinion of the teachers in respect of the degree of difficulty, there is no significant difference observable. It seems that the statements of the students about their difficulties in respect of the considered topics agree with the teachers' view. This is what we expected. We think, that we know our students very well. The lessons take place in rather small groups. The size of a group for programming exercises is normally not larger than 20 students. The programming assignments are normally defined and attended by the lecturer. As a result, the teachers are in tight contact with the students. Beyond that, written exams are corrected by the teacher, which enables the teachers to have direct feedback on the topics that have been understood by the students and to what degree.

Furthermore, in respect of the opinion what is the most effective learning style for our students we see a consensus between the answers of the students and the teachers on the one hand and our opinion about students on the other.

2.4. Infrastructure and where students do programming

Our university is a small institution and in our faculty the students and teachers know each other. There is a rather familiar atmosphere. Therefore it is possible to leave the PC and workstation rooms open and unattended from the morning till the evening during the working days. In fact, the students themselves take care of the PCs. Hence the students have easy and little regulated access to this infrastructure and this is reflected in the data. On the other hand, the infrastructure is not updated very frequently due to the limited budget. The offered computers are therefore partly rather outdated. The (rare) statements that the access to computers is difficult, reflects in our opinion the last mentioned fact. In the PC rooms there is a broadband Internet access, which is attractive to the students. We observe that more and more students have a notebook of their own, which they bring to the PC rooms in order to get network access. Consequently, more and more students use their own notebooks in nearly all programming practices. We estimate that more than 90 % (eventually all) of the students in the second or higher semester own a PC or a notebook. As we were aware of these facts, we knew that students do a lot of programming work at home on their own computers. Nevertheless we have been astonished how unique the answer was, showing that the programming work is done most frequently at home even in the first semester. The clear conclusion is that our students have to have access to all training material from home. Many of the students also have Internet access from home, so they are able to access training material from the Internet. However, many students have to pay for the time spent in Internet. Therefore the training material has to be designed to be usable offline from home.

2.5. Details of data and reliability

In general the analysis of the data gives reasonable results. The results of the data analysis are in accordance with what is expected and can be explained well with the existing situation. However we avoided to exaggerate with the interpretation of the data or to try to figure out smaller details. We have been for example astonished that some of our students stated to have experience with Codewitz or similar data. We did not yet offer such material to our students, and we did not hear of any experiences of our students with such kind of material. When digging deeper into the raw data we found that there exist records with an entry CN1 for the semester but with an answer different to FHF to the question "what is your university". And this seems to be a contradiction because CN 1 should be specific for our university. So in our opinion it is possible that these answers are caused by some student error or misunderstanding.

3. Conclusions for a learners tool

The extremely inhomogeneous background of the students in a beginners' course causes a broad spectrum of understanding velocities within the students. This makes lectures an inappropriate tool for learning, at least for the “advanced” beginners. A much better learning situation is self-learning – but one must not leave the students with a course book alone. It must be guided self-learning and it must be more guided for the novice beginners than for the “advanced” beginners. In this context computer aided learning tools like Codewitz come into play. With such tools every student can adapt the learning velocity to his/her individual understanding velocity. They cannot replace a course book but can be an additional help during self-learning. It should be integrated in an individual learning environment for every student, consisting of:

a) Course book,
b) computer aided exercise tools (Codewitz)
c) bigger programming tasks
d) discussion sessions with teacher and other students

Point b) is meant as smaller exercises, which addresses a single programming problem e.g. pointers or loops. Point c) addresses the big problem of “Understanding how to design a program to solve a certain task”. This problem goes beyond the scope of a restricted problem area, but brings together all the basic techniques to write a program. Also a computer can aid such exercises. A concept of such a computer-aided tool can be to present “rudimentary” programs, and the student has to make them run, by completing them. This concept has three advantages:

a) It makes the step from learning single programming concepts like “if” and “loop” to writing whole algorithms easier. This can be achieved by using different degrees of completeness of the presented programs. The beginner is confronted with nearly perfect programmes – he has only to add a few things, and the program is running. The more the student advance, the less complete are the programmes he has to make running. In the extreme form, the student has to write the program from the scratch.

b) It addresses the issue “finding bugs”. The programs not only can be incomplete, but also faulty. A beginner must find theses faults to make the program run correctly.

c) It is a known fact, that a programmer can learn a lot by reading programs from other programmers. This is achieved by presenting programs (even if they are rudimentary) to the students. E.g. it is possible to show a special technique for using libraries. The students have rated using libraries “difficult”.

TAMPERE UNIVERSITY OF TECHNOLOGY

Essi Lahtinen, Institute of Software Systems, Tampere University of Technology, Finland
Email: essi.lahtinen@tut.fi

1. Introduction

Tampere University of Technology (TUT) differs from the other universities participating in Codewitz/Minerva-project by the amount of students and the size of teaching groups. There are every year almost 1000 students who take a beginner course in programming and the basic programming lectures are held in lecture rooms for at least 250 students at a time. The teaching is very impersonal and there are not enough resources to keep contact with all the students.

In addition to the mass lectures there are exercise sessions in smaller groups, usually with 20-30 students in a group. The exercises are in general held in a normal class room, not in a computer class. The exercises consist of working in small groups and solving programming problems on paper and pen. The students do the practical work on computer by themselves. They have to return homework assignments that will be corrected either automatically or by an assistant depending on the course and assignment. The difference between the group size and the learning situations in TUT and the other universities participating in the survey, should be considered when comparing the results.

There were only 114 answers from students from TUT. This is probably due to the fact that the timing of the survey was not in line with the timing of our basic programming courses. 114 answers are sufficient to receive a rough picture of the learning difficulties of our students. Most of the students who took part in the questionnaire were already studying the course “Programming 2”. The beginner course was only handling “if”-structures at the time of the survey, so the students did not know what most of the terms in the questions meant.

It is difficult to say, to what extent the results of the survey reveal the learning difficulties in our basic teaching, as 70% of the students who took part in the questionnaire had programming experience before starting the studies at TUT. On the other hand, more than half of the students with earlier programming experience have estimated their programming skills to be low or very low. Consequently, only 28% of the students participating in the survey had moderate or better programming skills prior to the studies at TUT.

2. Issues in learning programming

The issues that students in TUT found difficult in learning programming, were more practical than in the other universities. The only issues in which TUT had slightly higher average than the average of all the universities were "using program development environment, compiler, editor etc." and "finding bugs form their own programs". Issues demanding theoretical thinking and/or understanding of concepts (such as understanding programming structures, the language syntax, designing a problem to solve a certain task or dividing the functionality), were found easier by the students in TUT than was the average of the whole group.

Due to the large teaching groups on the basic programming courses, it is not possible to take the students to a computer room to teach there. Instead the teaching concentrates in the theory and the students learn the computer functionalities by themselves when doing the homework assignments. This causes practical difficulties in the beginning.

The third practical question "gaining access to computers/network" was found much easier in TUT than the average, which implies that the practical difficulties must be only because of the lack of personal advice when using computers. This is a thing that cannot be helped with visualized exercises and examples. Furthermore, more then 98% of the students rated their computer using skills at least moderate. 87% of them also has a computer home and most commonly work there so the lack of computers is not the reason for the practical programming problems. The difficulties on the practical issues were not significantly bigger in TUT though there are no exercises in the computer rooms.

As teachers normally, the teachers in TUT thought that most of the issues are more difficult for the students than the students themselves had indicated. The only issue that the students in TUT found more difficult than the teachers in TUT, was learning the language syntax. This might also have some connection to the fact that there is no teaching in the computer rooms, because teaching on computers usually concentrates on compiling the program and correcting the syntax errors. The teachers do not realize the difficulties the students have with the syntax as they do not work with the students in the computer rooms.

3. Programming structures/concepts

Compared to the average of all the students, the TUT students seem to rate their difficulties in understanding the programming structures smaller. So do the TUT teachers compared to all the teachers. This is surprising, because large teaching groups and lack of personal contact to the teacher usually cause more difficulties in learning.

Furthermore, TUT teachers rate the learning difficulties bigger in all of the structures/concepts than the TUT students. The three structures that the TUT students rated almost as difficult as the TUT teachers, were variables (lifetime, scope), arrays and structural data types. Either the students think that these structures are difficult or the teachers do not think they are too difficult. Arrays and structural data types are easy to draw in a teaching situation, which leads to the question whether visualization can help in such an extent?

Maybe the greatest difficulties that the students face with arrays are the indexes. It is very unintuitive to begin indexing from zero instead of one. Indexing is also difficult to visualize, due to the fact that human eye intuitively says the first box in a line is the first and not the zeroth. It is easy to write the indexes under the boxes when one draws a picture of an array, but does it help to give the right impression and to keep it in mind when writing arrays in the code?

Making small visualizations about structural data types is not difficult. To make a visualization that actually uses the benefits of structural data types for something sensible is more difficult because of the amount of code needed for that. The concepts, that were the most difficult for TUT students, were pointers and error handling. Both of these are easy to be visualized with a simulation and there are already visualizations about data structures built with pointers. Pointers are the kind of concepts that require a lot of thinking and concentration when being used. It can be that the students understand the basic idea of a pointer but it is still difficult to use pointers oneself. It is difficult to say, whether there is a need for simpler pointer visualizations than the data structure visualizations are.

The difficulties in error handling relate to the fact that the program execution jumps from one place to another when an exception is being used and it is difficult for the students to follow, what happens during the jump. The next difficult things to learn were recursion, arrays, abstract data types, input and output handling and using language libraries.

4. Learning and teaching situations

When TUT students were asked about the learning situations and materials, the standard deviation in "learning in practical sessions in computer rooms" and the standard deviation in "using interactive simulations as the learning material" are so extensive that the average values are not credible for comparing these issues. The reason for this might be, that the students do not know exactly what is meant with these two questions, as they have not used interactive simulations or practical sessions in computer rooms in their studies.

The students in TUT rank "working by themselves" more useful for learning than the students in other universities. The values from the other learning situations do not differ very much. Because of the large teaching groups in TUT, the programming projects, which the students have to do themselves are usually extensive and require sometimes up to 100 hours of working by themselves. This is why self-working is stressed. The fact that the lectures are held in such big groups might also have influence on this matter.

Only a fraction of students who answered the question about using visualizations in learning thought that the visualizations would be best to be presented in lectures. In the lectures the interactive functions of the visualizations are though not useful. Maybe the students would find them more beneficial in self-learning if there were more of them available in an easier format.

5. Conclusions

The students in TUT need guidance in similar programming concepts as the students in other universities. More guidance than in other universities is needed in practical computer work, but this cannot be helped with Codewitz visualizations.

Maybe the fact that the difficulties in learning programming concepts are rated smaller in TUT than the other universities, is because the students are used to making effort on finding the information themselves due to the fact that they do not have that much personal guidance available. In this case there is certainly use for interactive visualizations because they can support the self-learning when teacher is not available.

REYKJAVIK UNIVERSITY

Ásrun Matthíasdóttir, School of Computer Science, University of Reykjavik, Iceland
Email: asrun@ru.is

1. Introduction

There are three schools operating within Reykjavík University (RU): the School of Law, the School of Computer Science and the School of Business. In addition, the Executive Education program at RU offers various innovations in the fields of management, business and technology. RU has 105 staff members. The total number of students at RU during the academic year 2002-2003 was 1390. The Computer Science Department has 17 full-time faculty members and 20 part-time teachers. 12 staff members are directly involved with open distance learning (ODL). The total number of students registered in the department during the academic year 2002-2003 is 535; 190 of those are ODL students.

From the foundation of the University, the School of Computer Science has emphasized distance education. For students who cannot attend a traditional study program, the School of Computer Science offers Distance Learning or University Study while Working (HMV). In 2002 – 2003 approximately 31% of the students are enrolled in “Distance Learning” and “University Study while working” programs. More than 50% of new students in 2003 are enrolled in “Distance Learning” and “University Study while working” programs; therefore the importance of ODL is growing.

2. Results of questionnaire for RU students

2.1. Students

The survey was conducted electronically in November 2003 with the help of web site. Altogether 151 students from Reykjavík University participated. Only 20% (n=30) of them had used themselves or seen visualization materials from the Codewitz project. Well over half (63%) of the students had programming experience before they started their study in RU and 55% rated their programming skills (regardless which programming language) before starting in RU as very low or low. About half of the students have been studying programming for 1 year or more as figure 1 shows.

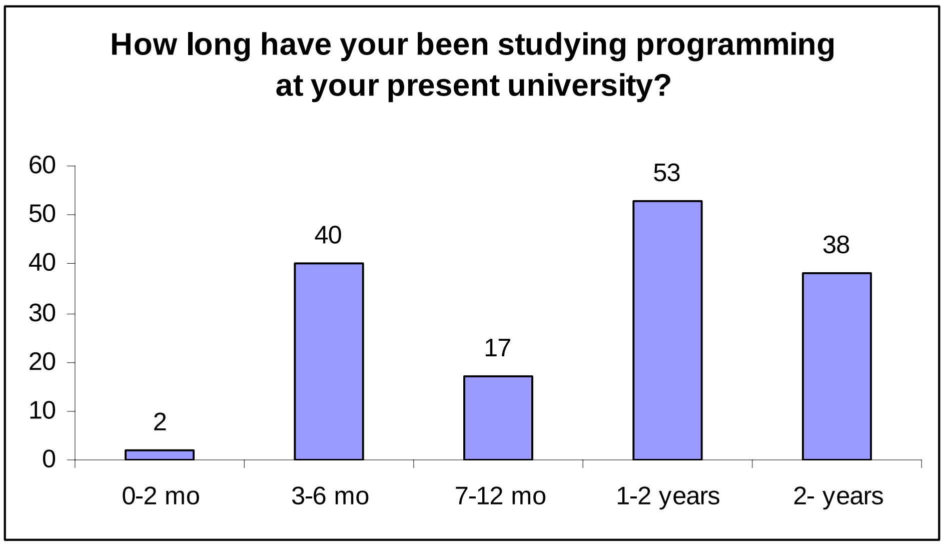


Figure 1

Most (83%) of the RU students describe their computer using skills as good or expert and they do most commonly work on computers at home (48%), at university (26%) or elsewhere (23%) but the library is rarely used (3%).

Students have different attitude toward what is difficult in learning programming and when they were asked the RU students replied as shown in Table 1. Gaining enough access to computers/networks is least difficult and understanding how to design a program, how to solve a certain task and how to find bugs from own program were the most difficult issues for the RU students in learning programming.

Table 1

What kind of issues you feel difficult in learning programming?	**Average**
Using program development environment, compiler, editor etc	2,41
Gaining enough access to computers/networks	2,09
Understanding programming structures	2,77
Learning the programming language syntax	2,62
Understanding how to design a program to solve a certain task	3,03
Dividing functionality into procedures, functions and/or classes	2,80
Finding bugs from my own program	3,00

The students were also asked about their attitude toward programming structures/concept and in Figure 2 we can see the average score (scale 1- 5). The students agree, that pointers, references and recursion have been the most difficult concepts.

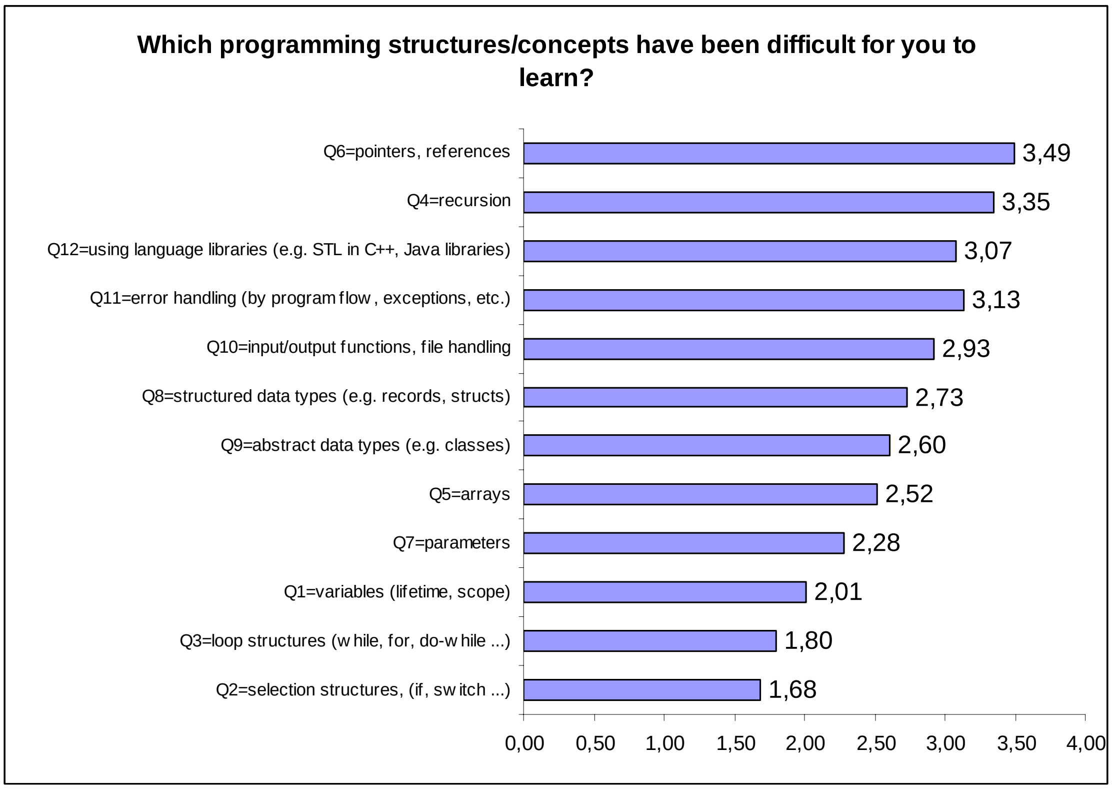


Figure 2

In table 2 we can see that students feel that they learn issues about programming on average most often while working by them self on programming coursework and while studying by themselves with teaching materials.

Table 2

When do you feel that you learn issues about programming, personally? (never, a couple of times, sometimes, often , almost always))	*Average*
in lectures	3,04
in exercise and discussion sessions in small groups	3,57
in practical sessions (in computer room)	3,88
while studying by myself with teaching materials	4,02
while working by myself on programming coursework	4,17

The material is also important in students' study and when asked what helped/would probably help them in learning programming, the RU students did value example programs very high, together with lecture note/copies of transparencies, exercise questions and answers as well as programming course book.

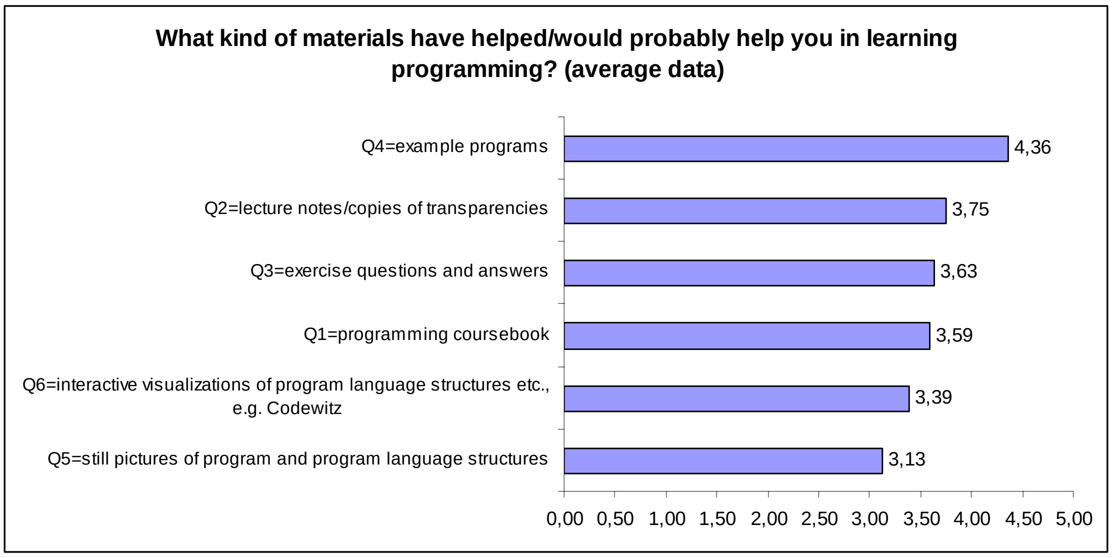


Figure 3

Only 30 RU students had used or seen visualization materials (e.g. Codewitz) on their course. The situation, which the student considers visualization materials to be most useful are when presented on lectures, and when using them self as can be seen in figure 4.

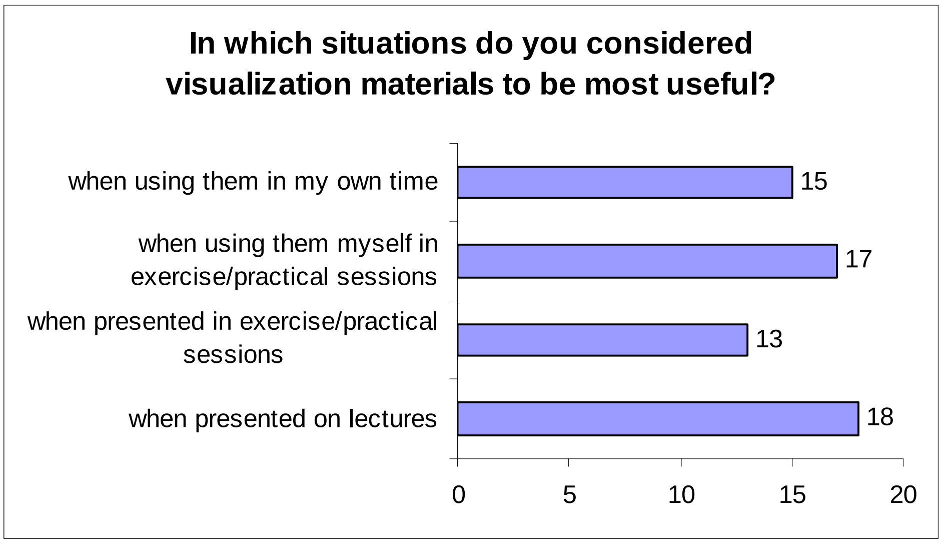


Figure 4

2.2. Teachers at Reykjavik University

Seven teachers at Reykjavik University did answer the teachers' survey and four of them had been teaching computer sciences for 3 years, one for 2 year and 2 for 1 year. As the participants were so few from RU there is no reason to analyse the data further here.

3. Evaluation of the existing Codewitz material

As such a high proportion of the students in Computer Science are Distance Learning and Study while Working students, all lectures at RU are recorded. Students can watch and hear the lectures on intranet and Internet when needed. Students in the traditional study do also benefit from this as the lectures are also available to them. It is our feeling that these lectures have also been widely used by the students in traditional study program. This is one of specialities of RU, having great importance when selecting teaching methods**.** Thus, in this section we are going to evaluate the existing Codewitz material from three points of view, i.e.:

- How does the existing Codewitz material fit in our teaching methods?

- Can we use the existing material in new ways?
- Is there some specific subject not covered but needed for what we are teaching now?

3.1. How does the existing Codewitz material fit in our teaching methods?

One of the questions for the students in the Codewitz survey was:What kind of materials have helped/would probably help you in learning programming? Students answered how much they used different learning methods. They could select several answers and the answers were graded according to the Table 3.

Table 3

Answer	**Grade**
Practically useless	1
Helps only very rarely	2
Sometimes useful	3
Often useful	4
Very useful, I use it a lot	5

Average grade was then calculated to see the importance of each learning method. As can be seen in table 4, the average grade of the students for the learning method “interactive visualizations of program language structures etc., e.g. Codewitz” were high, i.e. nearly the same as other, more conventional methods.

Table 4

Learning method	*Average all students*	*Average RU students*
Q1=programming coursebook	3,35	3,64
Q2=lecture notes/copies of transparencies	3,39	3,75
Q3=exercise questions and answers	3,32	3,63

Q4=example programs	4,19	4,32
Q5=still pictures of program and program language structures	3,15	3,13
Q6=interactive visualizations of program language structures etc., e.g. Codewitz	3,33	3,39

Results viewed in column-chart:

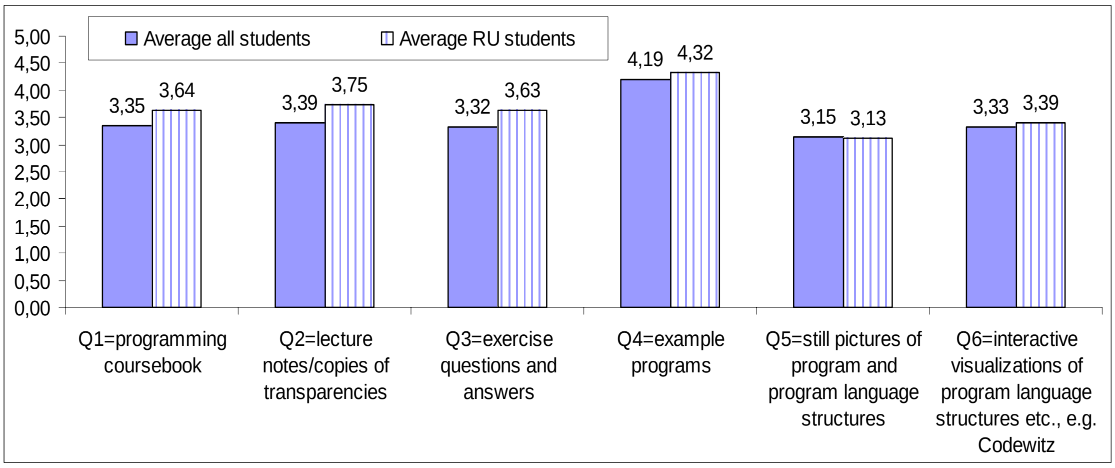


Figure 5

3.2. Can we use the existing material in new ways?

At RU we have high expectations that the Codewitz material will fit well to our teaching methods and it may be especially useful in Distance Learning where it may help us minimizing the “brottfall”

The opportunities that we see are the following:

- Put specific Codewitz material as it is on the net along with recorded lectures.
- Include the Codewitz material in the recorded lectures.
- Make a specific recordings where certain Codewitz material is explained.

Put specific Codewitz material as it is on the net along with recorded lectures.

We have already tried this last semester. It is our experience, confirmed in the students survey, that this helps the students understand the “námsefni”. But if we use this method, we believe that it would be important to introduce the material in the lectures even though it is not showed there.

Include the Codewitz material in the recorded lectures.

We did also try this method last semester. The result was good, the teachers did receive more response from the students and we believe that this method highly increases the value of the lectures.

Make a specific recordings where a certain Codewitz material is explained

In the survey there was a special question referring to this way of using the Codewitz material. There were few students that answered this question but the answers were rather positive. It is our opinion, that we should elaborate this method further.

VENTSPILS UNIVERSITY COLLEGE

Estere Vitola, Aivars Zemitis, School of Economics and Management, Ventspils University College, Latvia
Email: esevito@navigator.lv, zemitis@venta.lv

1.Introduction

Ventspils University College was established on July 23, 1997 by the state – the Republic of Latvia. University offers academic and professional higher education in three study directions: 1) Economics and Management Studies with the specialisation in the following areas: Finance Management and Management Accounting, Marketing Management, Logistics Management, Applications of IT in Business Administration. After 4-year studies in Management Sciences and defending the bachelor's thesis, students obtain an academic degree of a Bachelor of Social Sciences. 2) Translation Studies with specialisation in: English - Russian - Latvian, German - Russian – Latvian. After 4-year translation studies and defending the qualification paper, students obtain a professional diploma of a translator/interpreter. 3) One can study Information Technology since academic year 2002/2003. There are two study programmes: IT Bachelor programme (4 years, 8 terms) and IT College programme (2,5 year, 5 terms). There are 87 students in the first year and 68 students in the second year who acquire IT.

IT students develop their programming skills studying object oriented programming as programming language C++. The second year students work at a project in which they have to show their knowledge about C++. Working at the project, each student acquire deeply and without assistance exactly what they need, including a lot of knowledge in areas which are not taught in depth at the university.

There are modern classrooms and excellent study facilities, latest computer technology and language laboratory in Ventspils University College. Each student has his own e-mail address and free Internet access. There is a wide library with hundred places for readers, and several specially fitted multimedia-working places in the University. In the academic year 2003/ 04 Ventspils University College has 800 full – time students and more than 50 lecturers. The students can study both with the help of resources from state budget and their private money resources. There is also possibility to take to take a credit.

2. Analyzis of the survey

2.1. Some remarks

87 students from Ventspils University College participated in the questionnaire. That means 15,6% from all the students (565 in all) have taken part in it.

Bearing in mind that IT studies are being offered just to the second year at Ventspils University College, it is clear why only a few students have studied programming more than one year. The students' answers confirm it - 53 students from 87 are beginners, they had not studied programming longer than six months. In comparison to all other universities, this outcome is lower in Ventspils University College than in other universities.

Programming as a subject is not included in the comprehensive school curriculum in Latvia. It is possible to learn programming in the activities outside the normal school curriculum- in hobby groups, in optional classes or in some schools that have special curriculum.

The answers to the question "Did you have programming experience before you started learning in your university?" testify that 56% of the students answered “No”. In comparison to the total results (40%), this result is higher in our university than the average. Other students entering the university had poor or very poor (62%) and good or very good knowledge (12%). Naturally, the knowledge has improved during the studies. At present only 14% of the students evaluate their knowledge as poor or very poor, 46% as good, but 7,5% feel like experts. It is positive to discover, that the students understand that the level of programming has grown (unfortunately not as high as they wanted). In this point our students also evaluate their computer using skills lower than the students from other universities. Therefore, as there are the first year and second year students among the respondents, the improvement of the second year students might be higher. In individual conversations the first year students confirm that they have difficulties with programming. The tests results show that the student as are not being able to type a simple programme, which consists of self developed function, on their own.

2.2. About course contents

Acquiring programming, students admit dividing functionality into procedures, functions and classes, as the most difficult areas. However, these form the basis of the object-oriented programming. At the same time also other issues seem quite complicated for students. Most frequently they give the answer - "average". The use of programming environment, compilers and editors present the less difficulties.

In learning programming students have the biggest problems with pointers, structured data types (records, structures), abstract data types (classes) as well as error handling and using language libraries. The most common answers are "difficult" or "don't know". It is clear because the first year students have not yet studied these themes.

Acquiring variables and selection structures and loop structures seems easy for students. The most common answers to these questions are "easy".

The answers of our students to these questions are very similar to the answers of the other universities' students. However, the answer "don't know" is more common for our students.

The lecturers' answers to the question "What kind of issues and which programming concepts they perceive learning difficulties for their students?" correspond with the students' own attitude (especially with the ones from Ventspils University). For example, the lecturers, as well as students, suppose that dividing functionality into procedures, functions or classes is the most difficult issue. The lectures and the students with higher experience find all the questions more complicated that the students with less experience.

2.3. About learning and teaching situation.

Most often students feel that they learn issues about programming while they are working by themselves on programming coursework or while they are studying by themselves with teaching materials, more rarely in practical sessions or in exercises and discussions sessions in small groups. According to the results of the questionnaire, the students learn least in the lectures. This was also the opinion of other universities' students and teachers.

From all types of supplementary material the students (both the students from Ventspils University College and other universities) prefer example programs as well as interactive visualisations of program languages structures and programming course book.

The lectures do not always have the same point of view with the students about materials. Both students and lecturers admit, that example programs and interactive visualisations of program languages structures are the most useful ones. On the other hand, lecturers think programming course books are the least useful (the most common answer is "sometimes"). By contrast, students have answered that course books are often useful, some students from Ventspils University even think they are very useful. Different attitudes can also be found in the question concerning exercises, teachers find exercises more useful than the students.

3. Codewitz material

3.1. Use of the material

The Ventspils University College is a partner in the Codewitz project, which produces visualization material for teaching and learning programming languages. Our students have only recently been introduced to the ready-made visualization examples and exercises. Nevertheless, 74% of our students answered to have used visualization materials, in comparison to 44% of the average answer to this question. Students both from Ventspils University College and from other universities considered visualization material to be most helpful, when they use them by themselves or when they are used in the lectures. The lecturers, who are familiar with the Codewitz material (66%) also recognized, that material is most useful when students use material by themselves.

The results of the questionnaire show that the Codewitz project is very important and necessary. From all kind of materials both students and teachers consider interactive visualisations of program language structures as the most effective ones.

Classifying all the questions in the order of importance it becomes evident, that it is essential to cover all parts of programming languages by the Codewitz materials. Adapting Codewitz materials according to the students' preliminary knowledge it is advisable to explain variables and structures -both selection structures (if, switch ...) and loop structures (while, for, do-while ...) well and as simple as possible. Nevertheless, we should keep in mind that the students of the next level need all the other issues as well.

3.2. Proposals for the development of the Codewitz material

It is observed that students sometimes have serious learning problems connected with perceiving programming language use and its structure. Codewitz idea was to help students by providing them with visualisation aids and animated code. Furthermore, these materials are very helpful for the teachers to prepare the lectures. Students of Ventpils University College already find Codewitz materials helpful and think that it is necessary to continue to develop this project. They are very grateful for opportunity to work with such code animations.

Codewitz materials include animations of programming language code. Due to the fact that there are animations, it is possible to follow the code’s performance, which makes it presumable for students to understand the execution of the programme. It is also very helpful and necessary to see the meaning of each executed line below the code. When students have seen the process of programs execution, it is a lot easier to understand the whole meaning of the language.

However, the performance of code's animation is a time-consuming process, which sometimes could not be so essential. In order to prevent that, we would suggest that the rollover images could be included in Codewitz materials – by moving the mouse over the code, the user could get the explanation of each line the mouse points to. Certainly, it does not eliminate the possibility to use animated code as it is already in the Codewitz materials, to make it feasible for the student to understand the code more deeply and follow its execution.

It would also be highly recommended to divide all the Codewitz materials into different parts – for beginners, intermediate and advanced studies.

Beginners' part could include only basic material - introduction to programming language, simple programs for getting acquainted with basic functions, methods and programming style of the language.

Intermediate part would consist of more serious code and material content (for example about classes, files, lists).

Advanced studies would include more complicated materials, and sophisticated code, also about the possibility to transfer other popular languages to current programming language.

In order to understand how the program looks like, it would be necessary to give the opportunity for students to run .exe file.

Understanding the code is part of the programming language studying, other part is the understanding of the meaning of the language code functions. In my opinion, it would not be enough to describe only each line, but descriptions of the language terms and definitions are also needed. This is due to the fact that complete beginners in programming would even require introduction in programming languages describing the meaning of terms like algorithm.

In order to perceive different abstract terms, the animations with real world objects would be very helpful. Maybe it would be necessary to consider the use of Java language in order to make some web based applications.

Other idea is to make applications made with code, which contains errors, and let students to correct it until it works. This could an aid to learn error handling or simply to remember how the code looked in correct program codes.

Not less important is the content of program codes and the design of the page. The codes must be humorous and witty in order to make the studying more fun and less exhausting. It could also make the memorising more effective. We should not forget the importance of planning and design, due to the fact that it is necessary for a student to be competent with the possibilities of the materials.

On the whole, the existing materials are a very good beginning for Codewitz project, but it has to be developed and supplemented.

TECHNICAL UNIVERSITY OF CIVIL ENGINEERING

Ion Mierlus Mazilu, Department of Mathematics, Technical University of Civil Engineering, Bucharest, Romania
Email: mmi@mail.utcb.ro

1. Introduction

Technical University of Civil Engineering (T.U.C.E.B.) Bucharest is the only Romanian university devoted entirely to engineering education in civil engineering and related fields.

T.U.C.E.B. has 7.795 students assigned in six faculties as follows:

1. Faculty of Civil, Industrial and Agricultural Buildings;
2. Faculty of Hydrotechnics;
3. Faculty of Building Services;
4. Faculty of Railways, Roads and Bridges;
5. Faculty of Technological Equipment;
6. Faculty of Geodesy;

in addition, there is a Department of Civil Engineering (English / French) and a Technical College. Each faculty and department has its own informatics laboratories. For the first year of studies there are 20 professors teaching informatics to the students.

The Mathematics Department is responsible for the basics informatics classes. As a result of the high-school education reform, with a view to the E.U. accession, the grades that had a specialization in mathematics and physics have been transformed into mathematics-informatics. Therefore, 60% of the students that attend the T.U.C.E.B. courses are computer literate.

In our university the first year course of Fundamental Informatics and Programming is held during one semester or two. For example:

- the Faculty of Civil, Industrial and Agricultural Buildings and the Faculty of Railways, Roads and Bridges have during first semester one course / one laboratory (each one 2 hours) by week;
- the Economical Engineering section has two semesters one course / one laboratory (each one 2 hours) by week, informatics basics and databases (DBMS – Visual FoxPro);
- the Department of Civil Engineering (English / French) has two semesters one course / one laboratory (each one 2 hours) by week (first semester - Fundamental Informatics, second semester – Programming Language);
- the Faculty of Building Services three semesters course (one hour weekly) / laboratory (two hours weekly);

The general schedule in informatics as follows:

A. Course unit: Programming Languages (Department of Civil Engineering (English / French))

- Programming languages basics.
- Algorithms. Flowcharts.
- Structured programming. Object-oriented programming principles.
- C/C++ language features. Basic knowledge's.
- C/C++ programs structure.
- Data types. Constants. Variables. Statements.
- Operators. Expressions. Evaluation statements.
- Decision statements. Loop statements.
- Arrays and character strings.
- Pointers. Functions.

- Input/Output devices.
- Objects and classes. Constructors and destructors. Inheriting concepts. Derived and virtual classes.

B. Course unit: Computer Science (Department of Civil Engineering (English / French))

- Computer fundamentals. Hardware and software. Working principles.
- Computer networks. Local and wide area networks. Internet. WWW (World Wide Web).
- Software classes. Operating systems: WINDOWS.
- Web Page. Creating a HTML (HyperText Markup Language) Web Page. Microsoft FrontPage.
- Microsoft Office: Text editor (MS-Word), Presentation editor (MS-PowerPoint), Spreadsheets (MS-Excel), Databases (MS-ACCESS).
- Mathematical analysis and algebraic computation. MathCAD.

C. Course unit: Basic Computer Science and Programming Language (Faculty of Civil, Industrial and Agricultural Buildings and the Faculty of Railways, Roads and Bridges)

- Computer fundamentals. Hardware and software.
- Computer networks. Local and wide area networks.
- Software classes. Operating systems WINDOWS.
- Web Page. Creating a HTML document.
- Microsoft Office: Text editor (MS-Word), Presentation editor (MS-PowerPoint), Spreadsheets (MS-Excel), Databases (MS-ACCESS).
- Mathematical analysis and algebraic computation. MathCAD.

- Programming languages basics (C++ / TP / MathCAD) - similar with Department of Civil Engineering course unit.

Furthermore, the students up to first year have access to informatics laboratory classes for others disciplines like:

- Numerical Analysis;
- Materials Strength;
- Seismic Engineering;
- Systems Reliability and so on.

According to the existing number of computers in laboratories, the students during schedule classes must share a computer with one other student. The remaining unused time resulting from schedule classes is limited and at this time the individual study on computer is restricted.

2. Analyzis of the learning and teaching programming

The majority of the students are handling fairly well the usage of a computer (ie. Windows operating system and Microsoft Office modules), as well as specialized programs for engineering disciplines. Meanwhile, the students have difficulties concerning algorithms and programming.

Even if the students know the language syntax (Turbo Pascal or C++) they have difficulties in resolving the algorithm and making the program. This causes also difficulties in the interpretation of the algorithm in pseudo-code or flow-chart and the translation to a programming language.

We consider that the object programming is difficult to be assimilated by the students. In addition, the number of schedule classes is insufficient for this purpose. For these reasons this module is not in the teaching course, with one exception. The exception concerns the students of the first year from Economical Engineering (Faculty of Civil, Industrial and Agricultural Buildings) section that study Visual Fox.

Regarding the procedural programming there are major difficulties for the students that graduate from a high school with insufficient schedule classes of programming.

This student category has difficulties concerning the meaning of:

- Evaluation statements – like $s := s + expression$ or $p := p \cdot expression$ and the use of the neutral element for addition and product;
- Decision statements - in establishing the condition (or compound condition);
- Loop statements – in establishing the condition (or compound condition) and the position of the condition(s) (a priori / posteriori).

A useful tool to be inserting in the Codewitz-project e-learning application is to type the syntax and the semantic of a command when the mouse pointer passes over the typed instruction.

The evaluation of student's knowledge is executed in three steps: two probations during the semester and one in the last week of the semester (final). An evaluation consists of resolving a similar problem that was studied in the classes with the help of computer.

The final probation resides in a test with questions like these sequences:

1. What is the value of the variable i after the following sequence in Turbo Pascal :

```
i:= 12;
x:= 10;
while x >= 10 do
```

```
   begin
i:= i+1;
x:= x-1
   end;
```

a) –1 b) 0 c)1 d) 2 e) 13 f) unbounded loop

2. Given the sequence

```
x:= true;
y:= false;
z:= x and not( not x or y)
```

what is the value of z ?

a) true b) falsec) incorrect expression in Turbo Pascal

3. What will be the value of the variable A after the following sequence:

```
A:= 9;
i := 4;
case i of
1,2,3 : a := sqr(a);
4,5,6 : a := sqrt(a);
7,8,9: a := a+1
end;
```

a) 3 b) 4 c) 9 d) 81 e) the sequence has no meaning in Turbo Pascal

4. The effect of the instruction SEEK(n,f) is:
    a) the n-th record of the f file is read;
    b) the n-1-th record of the f file is read;
    c) the value n is written in the file f;
    d) the n-th record of the f file is deleted;
    e) the record pointer will be placed on the n-1-th record in the file f

5. In Turbo Pascal we can access directly
    a) only text files;
    b) only files with type
    c) text files and also files with type

6. What is the value of the variable z after the following sequence in Turbo Pascal :

```
a:= 2;
b:= -3;
if not(a>=b) then z:= a+b else z:=a-b;
```

a) -1 b) 5 c) 2 d) -3 e) -5 f) the sequence has no meaning in Turbo Pascal

7. In Turbo Pascal we can compare 2 variables, constants or functions only if their type is:
    a) numerical
    b) char;
    c) boolean;

d) numerical, char or boolean as long as both terms of the expression have the same type

8. In Turbo Pascal, the typical function for the ordinal type are:

    a) EOF(f), FILEPOS(f), FILESIZE(f);

    b) succ(x), ord(x), pred(x);

    c) ln(x), exp(x), abs(x)

9. If we define a variable in a Turbo Pascal subroutine then:

    a) the variable is ”seen” in any procedure or function from that subroutine

    b) the variable is ”seen” in the main program who has that subroutine

    c) the variable is ”seen” only by that subroutine

10. In the body of a function subroutine

    a) there has to be at least one instruction like function_name:= expression;

    b) there can or can not be an instruction function_name:= expression

    c) doesn’t have to be any instruction function_name:= expression

3. Analyzis of the survey

We have compared the results of the Codewitz-project questionnaire with the results of our students at the Fundamental Informatics and Programming course. Mostly, the tendencies are the same, with only small variations. From our experience our students have problems regarding:

- finding an appropriate algorithm for solving the problem;

- representing the algorithm in a logical scheme form
- writing the program from the logical scheme

The programming skills of our first year students are mostly in the category low or moderate, with some exceptions, but their computer using skills are good. Usually they work individually at home, because the number of student is very big compared with the number of computers and laboratories available in the university.

Regarding their difficulties in learning programming the big difficulties appear in understanding programming structures, understanding how to design a program to solve a certain task, dividing functionality into procedures, functions and classes and in finding the program bugs. The programming structures or concepts that are difficult for most of them to understand and learn are recursion, pointers and references, parameters and abstract data types.

The time for individual study in the university laboratories being very limited, most of them are learning and working by themselves. That is why we consider an interactive material to be very useful to them and in order to complete in a most fortunate way the examples that are given to them in lectures or practical sessions. A small number of our students had the opportunity to see the first versions of the Codewitz-project material during lecturer Esa Kujansuu lectures. These students were very interested in this interactive method of learning, which is useful to them not only in practical sessions at the university, but also when they are studying by themselves.

Codewitz-project offers much needed individual help in the most difficult problems of learning to master the basic programming skills. The students of programming get additional visualized help in building an initial knowledge base for their professional studies in computer science and advanced programming. The project is developing visualization drills that guide the students step by step, give them positive, continuous and immediate feedback and, importantly, a feeling of success in their studies.

For the teaching professionals Codewitz-project offers not only the right to use and develop existing material, but also the chance to produce one's own material and get instant feedback on it. The project offers for them a unique way to develop their courses, control of results of the students, and jointly develop approaches to visualize teaching and illustrate the difficult structural problems in programming.

4. Conclusions

Due to our University profile the students are mostly program software users and the procedural programming is used in further activities. We propose the development of some modules for further extension of Codewitz-project with e-learning program:

1. An algorithmically module including:
   - general introduction to algorithms;
   - structural programming concerning the three basics concepts;
   - testing the algorithm with animation on a flow-chart that is most suggestive;
2. A Visual Basic module for macro commands in Excel, Access according to the schedule classes for the advanced students that use these programs for annual projects. These proposals also respond to the necessity of linking the procedural programming with object programming.

CONCLUSIONS

The idea for the Codewitz project emerged from the perception of the programming teachers, that the students have difficulties in learning programming languages and that the more traditional teaching situations and methods, such us mass lectures, cannot tackle these problems. The students, especially the beginners, seemed to have difficulties particularly in understanding the programming structure and the logic of the programming languages. The aim of the Codewitz material is to produce exercises and examples to help the students to learn and the teachers to teach programming.

Even though the partners of the Codewitz project have long experience in teaching programming, the Needs Analysis was decided to be made in order to make sure, that the material to be produced meets the needs of the students and teachers. With the help of the literature study and its references the partners can form a systematic approach to their development of the ideas and plans for the material.

With the help of the questionnaire, which was introduced in identical form at the same time in Codewitz/Minerva partner institutions, we hoped to receive a confirmation to our perceptions about the difficulties the students and teachers face in learning and teaching programming. As can be read from the analyzes of the partners, the results of the survey confirmed the notions the teachers had. The students find that finding bugs in their own programs and dividing functionality are the most difficult areas in learning programming. Taking the specific concepts into consideration, the students found pointers, references and error handling the most difficult to learn. Unsurprisingly, teachers had considered these issues to be the most difficult ones to teach.

In the production of the Codewitz material these most difficult areas and concepts have to given special attention in order to facilitate the task of the students and teachers to learn and teach these issues. However, the more ‘easy’ areas should not be forgotten either, as the good knowledge of the basics, such as loops and selection structures, form the basis for the advanced studies of programming.

One of the interesting findings of the survey is, that the cultural differences between the institutes in Scandinavia, Baltic countries, Western and Eastern Europe seem to be very small at least within this sampling in the field of programming studies and its difficulties. Some differences can be seen in the access to computers for example, but the main tendencies remain the same.

Another, possibly a slightly alarming finding of the survey is, that the students find they learn least in the lectures. The sizes of the classes attending to the lessons vary in the partner institutes, but it does not seem to an effect whether there are 30 or 200 students in the lesson, in any case the teacher has no resources to pay individual attention to students. Guidance, possibility to ask and feedback are available in practical sessions, in which the learning results are higher. The best way to learn is according to the results however individual working with for example home works. When the students were asked about the materials which helps/would help them most, the emphasis was on interactive material. With the help of interactive exercises the students can concentrate on the issues adjusted to their understanding velocity, they must think by themselves and they can get instant feedback whether they solved the exercise correctly or not. In the lessons the velocity of the teaching might be too slow or too fast, there is very little or no interaction and there is a danger that the students take a more passive role instead of really thinking about the issue to be taught. However, lectures are also needed in order to give the theoretical background.

Therefore, Codewitz and other interactive, visualized materials supplement effectively the tools needed for the learning and teaching of programming. With the help of the results we now know, what areas need to be taken into special attention when planning Codewitz material. The survey also provoked some ideas of development and amendment of the material, which are important to explore further. Keeping the results of the survey in mind, the planning, production and development of the material will continue towards better programming skills.