Informatics in Education, 2012, Vol. 11, No. 2, 271–282

Problems in Choosing Tools and Methods for Teaching Programming

Daiva VITKUTĖ-ADŽGAUSKIENĖ, Antanas VIDŽIŪNAS
Informatics Faculty, Vytautas Magnus University
Vileikos st. 8, LT-44404 Kaunas, Lithuania
e-mail: d.vitkute@if.vdu.lt, a.vidziunas@if.vdu.lt

Received: November 2011

Abstract. The paper analyses the problems in selecting and integrating tools for delivering basic programming knowledge at the university level. Discussion and analysis of teaching the programming disciplines, the main principles of study programme design, requirements for teaching tools, methods and corresponding languages is presented, based on literature overview and author‘s experience. A pressure from labor market, students and other sources to emphasize practical skills over deeper, long-term programming concepts is described. A model of teaching introductory programming disciplines at a higher logical level, using C#, is presented as a summary of the accomplished analysis, and also taking into account the recommendations of the ACM (Association for Computing Machinery) association for typical teaching programs. Also, design principles for building introductory programming courses, aligned with such teaching approach, are presented. This model has already been trialed at Vytautas Magnus University.

Keywords: teaching programming, learning environment, training-oriented languages, student needs.

1. Introduction

Currently, paradoxical situation is typical of many developed countries, including Lithuania. Despite the fact, that the role of information technology (IT) is growing rapidly in various activity areas, including household activities, and the demand for IT specialists with relatively high salaries is increasing, while the number of students in IT study programs is steadily decreasing. When investigating this problem, different authors (Stephenson, 2011; Meyer, 2012) state, that the main reason for downward trend in popularity of IT and computer science (CS) in universities is curriculum complexity and abstractness, its insufficient links with practical needs and extremely rapid IT evolution.

It should also be noted that the downward trend in popularity is typical not of all IT application areas. Some areas, for example, those dealing with modern software engineering and IT application areas, even show signs of growing in popularity (Computer Software Engineers and Computer Programmers, 2011). Therefore, better alignment of IT study curriculum with student expectations and labor market needs is relevant. Study programs should be regularly reviewed and updated in order to reflect rapid changes in information technology and its implementation strategy for particular country.

In organization of IT studies, Lithuanian universities, like the majority of universities all over the world, are guided by the ACM Curriculum (Curricula recommendations, 2012), where ten out of fourteen basic ACM CS Curriculum (2008) knowledge areas are related to different programming and software engineering problems. Therefore, problems and methods of programming disciplines in IT curriculum require special attention.

Important factor, determining the efficiency of programming studies, is the adequacy of teaching tools, including programming languages, supplementary library collections, software development environments, software templates, textbooks and other means. Universities are using different teaching methods (Van Roy and Haridi, 2004; Kölling and Meyer, 2012), built and formulated on the basis of different programming languages, paradigms and strategies. However, most of them are still looking for best solutions. As this is a significant problem, it is important to widen discussions on how it can be solved using means offered by new programming technologies. Research works related to this problem are usually limited to the analysis of requirements for learning environments used in introductory programming studies (Felleisen and Findler, 2001; Bennedsan and Caspersen, 2008), while insufficient attention is paid to the analysis of teaching tools for specialized programming disciplines, issues of their integration, labor market and student interests.

2. Requirements for Teaching of Programming Fundamentals

Studies of programming fundamentals are aimed at introducing students to basic programming issues and at providing primary practical skills. Until the last decade majority of universities were constructing their programming teaching strategy according the recommendations of ACM Computing Curriculum (2001). This curriculum is based on the mathematical methodology of teaching the programming disciplines (Dijkstra, 1997), going as far as 1976, and suggesting that programming should be considered as a branch of mathematics.

ACM recommendations can be implemented using several different methods (Van Roy *et al.*, 2012). For example, teaching methods based on imperative and functional paradigms have rather old traditions. Study programs, built around this approach, give students a good theoretical understanding of programming principles, but often may lack knowledge on software engineering concepts that are necessary for IT professionals. Therefore universities, profiling their programs in software engineering and other applied areas of IT studies, prefer methods, better interacting with real world need and reflecting concurrency patterns observed in this world.

These tendencies were recognized in ACM Computing Curriculum (2008) where expanded knowledge structure for the fundamentals of programming was presented (Curricula recommendations, 2012) with only half of the study program dedicated to algorithm description and analysis of related control and data structures, while the second half is dedicated to different issues of software engineering. At the same time, universities were encouraged to develop original study programs, answering to the special university or/and region needs.

Different approaches for programming disciplines also require different teaching methods, programming languages, specialized utility libraries, software development environment and other tools. Majority of authors, investigating these problems, recognize that object-oriented languages are best suited for teaching software engineering, however their use for introductory teaching of programming is questionable because of relatively high initial requirements for the knowledge of theoretical background. Discussions on this point are still going on.

3. Principles of Choosing Languages for Teaching

Universities with study programs, oriented towards theoretical backgrounds of CS, prefer programming languages that are specialized for teaching and correspond to the main academic requirements: high level, clean concepts, readable syntax, safety, no redundancy, orientation towards a single programming paradigm and easy transition to other languages. Probably, the most successful project of such a language is the *Pascal* language, created in 1970, which together with its object-oriented modification *Object Pascal* has been an exceptionally popular tool for teaching the fundamentals of programming in European and U.S. universities for two decades. In Lithuania it has also been widely used at the beginning of the last decade. *Eiffel* programming methodology and language (Meyer, 1997), which was demonstrated in public at 1986 and still is popular in many universities, also had very significant influence to teaching principles of object-oriented programming and software engineering concepts. New software development and programming methods that were introduced in Eiffel later were implemented in other popular languages: Java, C++, C# (Meyer, 2012).

Another trend in the development of programming languages, are languages that can be easily mastered and used for practical purposes without special training. The *Basic* programming language is the predecessor of such languages, and was later followed by *ABC, Visual Basic* (*VB*) and *Python*. However these languages are not suitable for teaching programming fundamentals for *CS* and *IT* students due to the lack of strict requirements for the program structure, though they are popular in non-IT curriculums.

One more group of training-oriented languages is aimed at demonstrating the advantages of different programming paradigms and their application possibilities: *Smalltalk* (object-oriented programming), *Lisp* (functional programming), *Prolog* (logic programming) and others. Since application of different programming environments for different paradigms creates organizational problems, integrated tools for paradigm training are being offered. An example of such a solution is shown in Fig. 1 (Reinfelds, 2002; Van Roy and Haridi 2004), where programs are executed in a virtual machine, managed by the

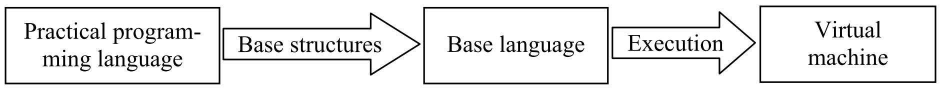

Fig. 1. Structure of a specialized programming concept learning environment.

base language, and all the practical programming languages are defined as extensions of this base language, obtained by adding corresponding syntactical structures. In such a case, a functional or a logical language is suggested as the base language, having more precise theoretical reasoning for different paradigms and, also, simpler syntax. This kind of teaching method together with mathematical specification means for the solution of different tasks, is beneficial of theoretical computer science and mathematics curriculum studies.

It is interesting to note, that a similar concept of program execution in a virtual machine, is implemented and widely used for practical programming in *Java* and *dotNet* technologies. The only difference is in the purpose of the base language. In this case it is a low-level language, ensuring program versatility and their independence of the physical properties of computer equipment. In this case, the virtual machine not only executes instructions written in a practical programming language, but, also, allows the use of utility libraries of the virtual machine. Such a structure is very promising not only with respect to programming technology organization, but, also, with respect to programming training purposes for students specialized in IT.

When considering programming languages, that are used for teaching software engineering concepts, the following features are defined as important: should have language integration and multi-paradigm tools, static typing, multiple inheritance, method and operator overloading, generic classes, multithreading, access control and build-in security methods (Stephenson, 2011). Orientation of utility libraries towards OS or virtual machine environment indicate possibility to use language in multi-language and multi-platform projects. Also, it is important to have friendly software development environment and mechanism for freeing memory of unused objects (garbage collection), thus simplifying the student tasks.

Features of popular programming languages for evaluating their correspondence to both requirements of introductory programming and software engineering are presented in Table 1 (Voegele, 2012). When selecting a programming language for teaching, it is also necessary to consider university traditions, labor market needs, interests of students and other factors.

4. Labor Market Requirements

Adequacy of programming languages, used for teaching, to labor market requirements can be assessed by analyzing web publications covering different programming language topics and issues for programmers. Such assessment is regularly published in TIOBE programming community website (TIOBE programming community index, 2012). Values of TIOBE index of ten most popular programming languages, presented in Table 2, indicate that about 58% of web publications are dedicated to *C* family languages and *Java*.

High popularity for programming in *C* can be explained by growing demand for various programmable digital devices and system programming needs, where this language is the most suitable. However, *C* is not recommended for introductory training purposes,

Table 1
Summary of programming language features

Property	Eiffel	Obj. Pascal	Java	C#	C++	Python	Perl	VB
Paradigm	Object	Hybrid	Object	Multipa- radigm	Hybrid	Hybrid	Hybrid	Hybrid
Typing	Static	Static	Static	Static	Static	Dynamic	Dynamic	Static
Generics	Yes	No	Yes	Yes	Libraries	No	No	No
Inheritance	Multiple	Multiple	Single, multiple interfaces	Single, multiple inter- faces	Multiple	Multiple	Multiple	None
Method overloading	No	Yes	Yes	Yes	Yes	No	No	No
Operator overloading	Yes	No	No	Yes	Yes	Yes	Yes	No
Garbage collection	Yes	Program- mable	Yes	Yes	Program- mable	Yes	Yes	Yes
Class methods	No	Yes	Yes	Yes	Yes	No	No	No
Orientation of libraries	OS	OS	Virtual machine	Virtual machine	OS	OS	OS	OS
Access control	Selective export	Public, pro- tected, private	Public, pro- tected, private	Public, pro- tected, private, internal	Public, protected, private, “friends”	Name mangling	None	Public, private
Multi- threading	Yes	No	Yes	Yes	Libraries	Yes	No	No
Pointer arithmetic	No	Yes	No	Yes	Yes	No	No	No
Language integration	C, C++, Java	Assemb- ler	C, C++	All .NET lan- guages	C, Assem- bler	C, C++, Java	C, C++	C
Built-in security	No	No	Yes	Yes	No	No	Yes	No

Table 2

TIOBE Index of programming languages for March 2012

Languages	Java	C	C#	C++	Objective C	PHP	VB	JavaScript	Python	Delphi/ Ob. Pascal
Rating (%)	17.1	17.1	8.2	8.0	7.7	5.6	4.4	3.4	3.3	2.7

as the risk of designing insecure programs is very high, the error control system is rather complex, and, also, many alternative structures, scattered across narrowly specialized utility library collections, are used. But some experience of using this language for *IT* specialist is necessary, because *C* is the main tool for language integration (Table 1). Such knowledge could be acquired in special software engineering courses.

The above mentioned shortcomings of *C* programming language are not present in J*ava* and *C#* languages, tailored for the design of complex application systems. However, their use for introductory teaching of programming is often being criticized because of relatively high initial knowledge of the basic principles that is required. Universities try to solve this problem in different ways.

Analysis of IT study program descriptions of the Lithuanian universities, available on the Web, showed that they mainly follow the trend of classical programming teaching methodology. Therefore, attention in introductory programming disciplines is concentrated at the analysis of simple data structures and algorithms using procedural programming tools in *Turbo Pascal*, *C* and *C++* languages. Relevant software engineering issues are addressed only in later stages, using applied software packages and *Java* or *dotNet* technologies. Today this approach becomes more and more irrational, does not correspond to the labor market needs, and is being criticized by various authors (Felleisen and Findler, 2001).

The increasing number of recommendations is aimed at shifting the focus from teaching programming paradigms concentrating on main software engineering concepts in introductory programming disciplines (Van Roy *et al.*, 2012). The "outside-in" approach based on "inverted curriculum" ideas (Meyer, 2012), would be an example of a revolutionary approach of this kind. It relies on the assumption that the most effective way to learn software engineering is based on the analysis and experiments related to already existing qualitative software.

5. Requirements for Learning Environments for Special Programming Disciplines

When accomplishing the analysis of requirements for learning environments suitable for the studies of special programming disciplines, it is appropriate to single out the following requirement groups with similar goals: software engineering, system programming and applied programming. It is easiest to define the requirements for the system programming group, as practically all commonly used operating systems are written in *C* language, which was even specifically designed to facilitate the development of such systems. Since systemic programming topics are usually included in the first two years of university study programs, is often recommended, that one of the *C* family languages should be included in learning environments for the fundamentals of programming.

When developing learning environments for software engineering disciplines, the decisive role is dedicated to the means for the design of complex software systems, as well as the means of their support and reengineering. They should be capable of illustrating the requirements of all the software development stages, from building formal specifications to the preparation of documentation. Description of programming languages in Table 1, shows that the most popular in teaching languuages (*Eiffel*, *Java* and *C#*) are dedicated to object-oriented technology. Other languages, that are combining procedural and object-oriented paradigms, are rapidly losing their popularity (C++ and Object Pascal) or aren't used for teaching IT professionals (VB, Python).

The software engineering needs are best of all met by *Java* and *C#* languages, their principal capabilities being very much the same. An important advantage of both two languages is that they use only virtual machine resources adapted to the implemented technology. It thus provides perfect conditions for maximizing the performance of the learning environments, and, also, for designing multiplatform programs, capable of working under different operating systems. The popularity of these languages is increased by the fact that they are well equipped with special libraries for various applied programming needs (computer graphics, databases interfaces and other).

Java technology is better equipped with the freely distributed open source and multiplatform means, however, the *dotNet* technology, represented by *C#,* also has many advantages, including implementation of many promising innovations. For example, it allows indirect description of variable types, has an integrated *LinQ* query language for collection management, allows regular expressions for word processing purposes.

One should also pay attention, that the *dotNet* technology, represented by *C#* language, supports several other programming languages (*VB.Net, C++/CLI, JScript.NE, Python* and others), which provides excellent conditions for software system building using different language modules. These tools help students to realize that programming is not only writing and analysis of algorithms, but, also, design, testing and support of complex systems, using rich sets of standard blocks and specialized design environments.

6. Evaluation of Student Needs

In order to better define strategy for the modernization of programming fundamentals' studies and corresponding learning environments, student need analysis was accomplished at VMU in 2008–2011, covering student needs for basic specialty subject knowledge. A survey was accomplished among second-year students, who already have completed introductory subjects, and have not yet forgotten their motivation for the choice of studies. They were given two groups of questions: what they hoped to learn during the study of initial subjects and where (in what activities) they hope to use the gained knowledge.

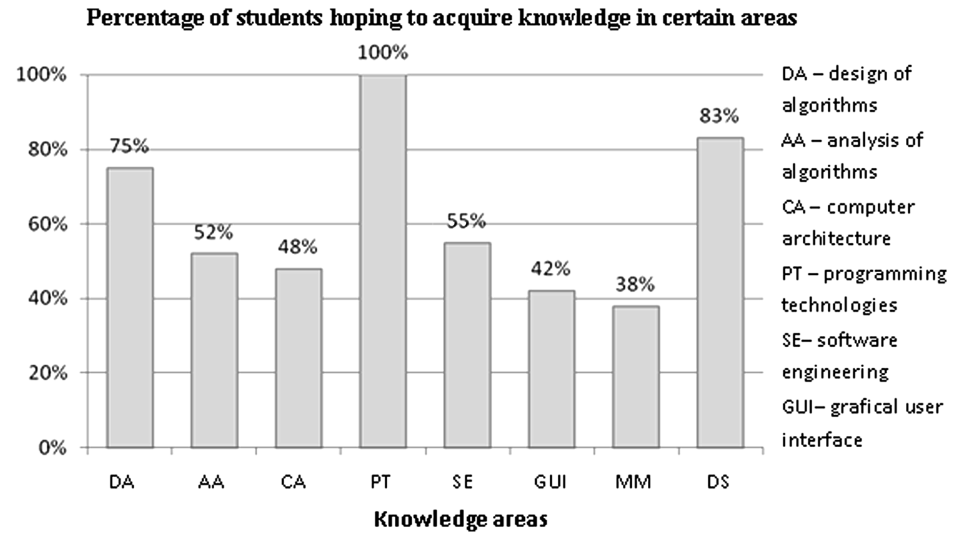

Fig. 2. IT student requests for the knowledge structure of basic specialty subjects.

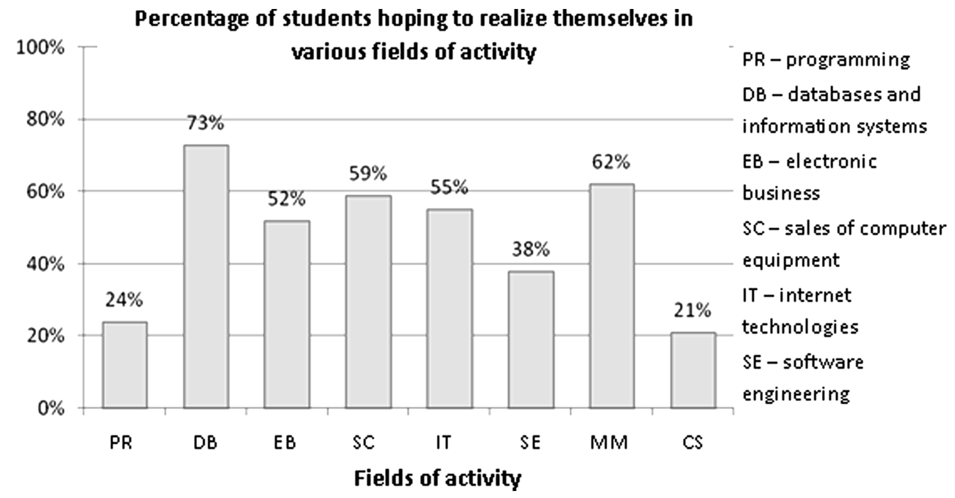

Fig. 3. Plans of second-year students in realizing themselves in different fields of activity.

Survey results are presented in Figs. 2 and 3, showing the percentage of students (118 out of 124 second-year students were interviewed) hoping to gain knowledge of corresponding IT disciplines in the first years of studies, and also the percentage of those willing to implement this knowledge in specific fields.

The first four columns in Fig. 2 show students' attitude to introductory subjects recommended in ACM model programs: basics of algorithm design, algorithms analysis, basics of computer architecture and programming technologies. All the students recognize the importance of programming technology studies, but also wish to acquire the basics of other subjects that are intended for detailed studies in the subsequent years. Even 83% of students would prefer early acquaintance with distributed system architecture and Internet technology, 55% – with software systems engineering, and about 40% – with multimedia products and graphic user interface development tools. Students' desire to advance the study of special subjects and acquire skills in their practical application depends on several factors: use of IT experience acquired outside the university, personal preferences, the need to make better use of personally owned computer hardware and software and use of the gained knowledge in search for employment or additional funding sources for studies. Currently, about 10% of ITS second-year students at VMU are already employed, and in the next years this number increases to 30%–40%. Half of them manage to find jobs related to the speciality of their studies.

Interesting results were obtained from the survey on the plans for applying the acquired knowledge (Fig. 3). All interviewed students understand the importance of detailed studies of programming technologies, but only 24% would like to work in this area. The majority of students (73%) expect to work in database design and information system administration field, and 59% – in computer equipment sales field. Students' career plans are affected by the curricula structure, IT needs in the market, university teaching and research labs activities, and overall scientific orientation of the university. For example, rather small popularity of computer-controlled system field is most probably due to the fact that Vytautas Magnus University is focused on the humanities and social science studies, while information systems management, e-business and Internet areas are popular due to the fact that there is large focus on these areas in VMU study programs, and, also, there is large demand for such professionals in Lithuania. Attention should also be paid to the popularity of multimedia, attracting young people because of the external performance of its products, variety of technology used and diversity of practical application perspectives.

All students interviewed identified at least three intended fields professional activity. This shows that they have a good feeling of IT market dynamics, do not expect to work in a narrow field for the whole lifetime, and are willing to participate in the design of the design of the study program curricula capable of forming their skills and abilities in the best possible way.

7. Goals and Means for Integration of Tools Used for Teaching Programming

Until recent years, Department of Informatics of Vytautas Magnus University was using traditional paradigm-based programming teaching methodology with several programming languages already in the initial phase of studies. Abstract and weakly related to the practical needs introductory programming subjects were the key reasons for low-rate student achievements. Currently, the situation is being changed by integration of teaching tools used for introductory programming and software engineering disciplines, where a higher logical level language is used for many purposes. This is consistent with model *ACM* program recommendations from 2008 to assign at least half of the introductory programming classes for software engineering issues. It is also estimated that a majority of freshmen, although lacking programming experience, are already familiar with elementary concepts of programming from their secondary school studies. Thus, it is reasonable to allocate the university studies of programming basics to the topics that are suitable to get initial experience of building simple object programs with clearly expressed practical orientation. For example:

- overview of programming paradigms, software engineering, virtual machine and *dotNet* technology concepts;
- language lexics, basic data types and internal classes;
- using of integrated software development environment (IDE);
- structure of object-oriented programs and their interface with data sources and users;
- generic collections and their practical applications;
- exceptions and their handling.

After providing the initial knowledge about the software design and principles of processing generic data collections, the second sequential study subject in programming, which is traditionally called "Data types and structures", should be dedicated to detailed analysis of how to create and modify abstract structures (classes) introduced in the first study subject, using object-oriented tools. Such studies may include the following topics:

- user classes and means of their adaptation to applied programming needs;
- inheritance and class families;
- design and using generic classes, interfaces and delegates;
- event driven programs and multithreading;
- queries and their practical application;
- development and deploying of dotNet assemblies.

Detailed studies of algorithm design and analysis, that are obligatory to IT students, are focused in disciplines dedicated for discrete structures and algorithm theory. Such a study organization approach allows the development of coherent theoretical and practical student knowledge needed for further software engineering and applied programming studies in the same learning environment.

The discussed the structure of the introductory programming subjects was also tested for retraining courses organized by Vytautas Magnus University and received positive audience evaluations.

8. Conclusions

Traditional paradigm-based programming teaching methodology using several programming languages is applied at the majority of Lithuanian universities. Since this complicates the absorption process for these subjects, negatively affects the student study-rate achievements, and also reduces the popularity of IT studies in general, the problem of modernizing the initial programming studies is very relevant. This problem is also characteristic of many other universities in the world. Different ways of solving this problem are suggested, mainly based on shifting the focus from teaching programming paradigms towards concentrating on main software engineering concepts at introductory programming disciplines. The problem of integration of tools used for programming disciplines is also relevant. The analysis of different programming languages used for teaching, presented in the paper, shows that Java and C# languages are the most suitable for described approach. Authors, using their experience at Vytautas Magnus University, presented an approach of organizing initial programming studies based on C# language advantages. The discussed approach is also in-line with students' wishes for the structure of specialty subject study program structure, these wishes being identified by carrying out a corresponding survey.

References

Bennedsan, J., Saspersen, M. (2008). *Reflections on the Teaching of Programming*. Springer.

Computer Software Engineers and Computer Programmers [last viewed on May 5, 2011]. `http://www.bls.gov/oco/ocos303.htm`.

Curricula Recommendations [last viewed on March 28, 2012]. `http://www.acm.org/education/curricula-recommendations`.

Deitel, H., Deitel, P. (2000). *C++, How to Programm*, third edn. New York, Prentice Hall.

Dijkstra, E.W. (1997). *A Discipline of Programming*. Prentice Hall.

Felleisen, M., Findler, R.M. (2001). *How to Design Programs: An Introduction to Computing and Programming*. MIT Press.

Reinfelds, J. (2002). Teaching of programming with a programmer's theory of programming. In: *Informatics Curricula, Teaching Methods, and Best Practice* (ICTEM 2002). Kluwer Academic Publishers.

Kölling M. (2012). The problem of teaching object-oriented programming [last viewed on March 24, 2012]. `http://www.bluej.org/papers/1999-09-JOOP2-environments.pdf`.

Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice Hall.

Meyer, B. (2012). *The Outside-In Method of Teaching Introductory Programming* [last viewed on March 24, 2012]. `http://se.ethz.ch/~meyer/publications/teaching/teaching-psi.pdf`.

Stephenson, P. (2011). *Promising New Pedagogical Approaches for Teaching High School Computer Science* [Last viewed on May 5, 2011]. `http://c2474712.cdn.cloudfiles.rackspacecloud.com/CuricPedFinal1.pdf`.

TIOBE Programming Community Index [Last viewed on March 28, 2012]. `http://www.tiobe.com`.

Van Roy, P., Haridi, S. (2004). *Concepts, Techniques and Models of Computer Programming*. Mit Press.

Van Roy, P., Armstrong, J., Flat, M., Magnusson, B. (2012). *The Role of Language Paradigms in Teaching Programming* [Last viewed on March 28, 2012]. *http://www.info.ucl.ac.be/~pvr/sigcse2003panel.pdf.*

Voegele, J. (2012). *Programming Language Comparison* [Last viewed on March 28, 2012]. `http://www.jvoegele.com/software/langcomp.html`.

D. Vitkutė-Adžgauskienė is the dean of the Faculty of Informatics at Vytautas Magnus University, Kaunas, Lithuania. She studied applied mathematics at Kaunas Technology university, got her PhD degree in informatics from Vytautas Magnus University and EMBA from Baltic Management Institute. She has 20 years of teaching experience in informatics, and has also worked for more than 10 years with a mobile operator in Lithuania leading the development of advanced mobile services and solutions. She participated in different international projects, including JEP-4298 (Tempus), RAPIDITY (Phare Multi-Country Programme in Distance Education), ALIPRO (FP6), OpenScout (eContentplus) and others. Her main research interests are: system simulation and control, innovations in mobile solution development, computerized text mining and semantic analysis. Her publication list includes over 30 papers and conference contributions.

A. Vidžiūnas, dr. is associated professor of the Faculty of Informatics at Vytautas Magnus University, Kaunas, Lithuania. He studied electrical engineering and got his PhD degree in informatics at Kaunas Technology University, has over 40 years experience of teaching computer programming in Lithuanian universities. Also he is author and co-author of over 20 books in IT and programming languages. His research interests are programming technologies and programming teaching methods.

Programavimo mokymo metodų ir priemonių parinkimo problemos

Daiva VITKUTĖ-ADŽGAUSKIENĖ, Antanas VIDŽIŪNAS

Straipsnyje nagrinėjamos pagrindinių programavimo žinių mokymui skirtų priemonių parinkimo ir integravimo problemos. Aptariami ir analizuojami šiam tikslui skirtų dalykų mokymo programų universitetinėse studijose parengimo principai, reikalavimai mokymo metodams, priemonėms ir naudojamoms programavimo kalboms. Problemos nagrinėjamos remiantis literatūros analize ir asmenine autorių pedagoginio darbo patirtimi, atsižvelgiant į didėjančius studentų ir darbo rinkos reikalavimus praktinių programavimo įgūdžių vystymui naudojant pažangias ir perspektyvias technologijas bei priemones. Aprašytas įvadinių programavimo dalykų mokymo aukštesniame loginiame lygmenyje naudojant C# kalbos priemones modelis, kuris parengtas remiantis atliktos analizės rezultatais ir ACM (Association for Computing Machinery) asociacijos rekomendacijomis. Taip pat aptarti tokio modelio realizavimui skirtų programavimo dalykų mokymo programų formavimo principai, kurie sėkmingai išbandyti ir patikrinti Vytauto Didžiojo universitete.