*World Transactions on Engineering and Technology Education*
*Vol.2, No.3, 2003*

# Some experiences in teaching introductory programming at the faculty level

**Viljan Mahnic & Matjaz Gams**

Jozef Stefan Institute
Ljubljana, Slovenia

ABSTRACT: The article presents the authors' experiences in teaching programming for nearly 20 years, mostly at faculty level. The authors have been teaching at nearly ten faculties, mostly in Slovenia. The programming languages include versions of Pascal, C, Java, Prolog, and Lisp, Logo and Snobol. The emphasis of the article is on the current teaching of programming at the introductory faculty level. Java and JavaScript seem to be among the leading choices for technically- and non-technically-oriented students. Indeed, over recent years, educators are facing certain changes in teaching programming since programming is becoming more and more related to the Internet and the information society, and is also becoming increasingly integrated with intelligent information services.

## INTRODUCTION

Teaching programming is an important constituent part of computing curricula. Despite the overwhelming increase in the availability of computing resources to secondary schools, many students arrive at universities with little understanding of the programming discipline and the basic principles behind software design. For this reason, the Joint Task Force on Computing Curricula 2001 has chosen to define a separate programming fundamentals knowledge area that enumerates the basic programming skills that all students of computing must acquire in order to prepare themselves for more advanced study [1].

The authors have been teaching programming at faculties, and also at high and elementary schools, for nearly 20 years, primarily in Slovenia, with a couple of foreign courses in European countries. The major faculties are: the Faculty of Computer and Information Science, Faculty of Economics, Faculty of Education, Faculty of Management, Faculty of Maritime and Transport, University of Applied Sciences. During the last two decades, the international exchange of information was mainly concentrated in Europe and the USA. Several professors of the mentioned faculties were also, from time to time, teaching in the USA and European countries; thus the contacts, comparisons and interactions were going on all of the time.

Teaching programming is changing, as are programming languages. Twenty years ago, Pascal dominated programming in Slovenia, as in many European countries [2]. In the USA, on the contrary, C and C-oriented languages were most often used at the introductory level for technically-oriented students. Also, industry in Europe was more oriented towards C. But for teaching a first programming language, it was widely (and with good reason) accepted in Europe that Pascal was much more appropriate than C due to clearer and more understandable uniform concepts [3].

Academia in Europe was eager to exchange arguments, publications and experience regarding the teaching of Wirth's ideas. It was especially important that the language was independent of any commercial factor so that it was standardised and freely available and, at the same time, industry followed theory with reasonable compilers enabling robust real-life commercial applications. Yet with the emergence of Microsoft and object-oriented programming, relations turned around. Suddenly, it was industry that was defining its own standards with Microsoft as the leading force. The gap and distrust remains to this day.

While teaching primarily Pascal as the introductory programming language, several other languages were also under consideration in Slovenia, including:

- Prolog as an appropriate language for mathematically and logically inclined students [4];
- Logo for youngsters;
- Snobol and Lisp for artificial intelligence.

Other languages like Cobol were not taught by the authors, although specific interest groups, eg students of economics, were using specific languages for databases and economic applications. Nevertheless, the dominant programming language in schools at all levels in Slovenia was Pascal.

Around ten years ago, it was becoming clear that Pascal did not follow the object-oriented trend very successfully, and that industrial solutions based on C were overwhelming. C++ and Java were proclaimed as the leading languages in many areas. Another major dilemma was related to programming as a concept – should we, as educators, teach procedural, object-oriented, Internet-oriented or tools- and environment-integrated languages? Commercial languages were often embedded into environments that enabled quick applications for a class of problems. Great hope was devoted to report generators, software engineering and automatic programming. At that time, it was expected that in a decade or so it would be possible to automatically generate programs from specifications. This could mean the end of programming as such or, at minimum, to a large extent. However, not much truly changed. Today, practically every application is still hand-coded. However, nobody in Slovenia teaches Pascal today.

Within this context, the choice of an appropriate programming language for the first course plays an important role. Many authors argue that the first programming language acquired by students should encourage all of those aspects of good style that we wish to impart to them, even if the language is not necessarily widely used [5][6]. In that case, they are more likely to preserve that style in other languages that may be more prevalent, but less appropriate, for an introductory course. It is far more important to instil good habits and then learn a second or third language, rather than to let the marketplace influence the first course. Additionally, the teaching language should satisfy the five criteria proposed by Woodhouse, namely: availability, teacher knowledge, ease of teaching and learning, language utility, and an appropriate structure for problem solving [7].

At the University of Ljubljana, Ljubljana, Slovenia, the introductory programming course is taught at different faculties and several programming languages were used over the last few years. At the Faculty of Computer and Information Science, Oberon was used from 1995 till 2001 [8]. It was then replaced by Java in 2002. On the other hand, in the Faculty of Education, where future teachers of computing are educated, JavaScript was taught as the introductory language.

The aim of this article is to describe the authors' experiences in teaching each of the aforementioned programming languages and provide analyses regarding their strengths and weaknesses.

## OBERON

Oberon was introduced with the aim of exposing students to the object-oriented paradigm, in addition to the usual paradigm of procedural programming. It was chosen as a teaching language because it has all of the desired properties considered necessary for teaching good programming style (ie simple and precise syntax, strong typing, modular program structure supporting data abstraction), and because of its specific approach to OOP, which enables a smooth transition from traditional procedural programming to OOP [6].

In Oberon, object-orientation is achieved through the use of type extension and procedure variables, thereby enabling all object-oriented concepts and techniques to be explained in terms of concepts already known from procedural programming. It was also important that Oberon was based on a Pascal-type of language, thus enabling a smooth transition.

Considering the aforementioned features, both programming paradigms were combined in the design of the introductory programming course: the procedural paradigm and the object-oriented paradigm. The procedural programming paradigm was the foundation upon which the concepts of object-oriented programming were developed. At the beginning of the course, basic data types and control structures were introduced. Then, the concepts of procedures and modules were explained, as well as input and output operations.

Special attention was paid to the concepts of scope (local and global variables, nesting of scopes, plus export and import declarations) and parameters (formal versus actual parameters, value versus variable parameters). After mastering procedures and modules, composite data types and dynamic data structures were discussed, such as arrays, records and lists. Finally, the most important procedural programming techniques, including stepwise refinement, abstract data structure and abstract data types, were presented to students.

In the second part of the course, object-oriented programming concepts and techniques were covered. Extensions of familiar concepts, from procedural programming (namely extension of record types, procedure variables) were described first. OOP concepts were then introduced one by one (ie generic modules, heterogeneous data structures, objects and dynamic binding of procedures, appropriate module organisation, type-bound procedures, inheritance and redefinition of procedures) using an example dealing with different kinds of vehicles (eg automobiles, buses, trucks, etc). The use of Oberon enabled the introduction of typical OO jargon to be deferred as long as possible. All object-oriented techniques and concepts were explained in terms of concepts already known from procedural programming.

In order to obtain students' opinions about the course, students were surveyed each year utilising a questionnaire that consisted of four groups of questions that dealt with the assessment of previous knowledge, a general evaluation of the course content, evaluation of the Oberon programming language, as well as students' opinions about OOP. Results of these surveys have been published elsewhere [9][10].

The results of the surveys confirmed the correctness of the basic considerations used in the design of the course. The previous knowledge of new students was estimated correctly, and the level of difficulty of the course was chosen appropriately. Most students (about 90%) found the course useful or even useful and interesting. The majority of students (more than 60%) also agreed with the incorporation of OOP in the introductory programming course. They felt that the portion devoted to OOP was adequate, and they also supported Oberon's approach to OOP, which treats OOP as an extension of traditional procedural programming.

However, it became evident that the choice of the teaching language is a very delicate problem. There were different opinions among students over what factor should prevail: the support of principles of proper programming or commercial use in practice. While 54% of students agreed that the programming language must primarily support the elements of good programming style, 46% of them strongly advocated the criterion of commercial success. Despite the fact that 61% of students judged Oberon suitable as a teaching language, the survey also revealed its main deficiencies: the unreliable and under-elaborated environment and insufficient commercial use. It should be noted that in the academic years 1995/1996 and 1996/1997, students practised programming using Oberon System 3 and Oberon V4, which implement Oberon as part of the Oberon operating system.

Furthermore, it was very hard to empirically evaluate Oberon's influence on programming style and habits of beginners. In contrast to our belief that teaching Oberon can contribute a lot to understanding of programming concepts and clear program design many students perceived learning Oberon an unnecessary effort because they will never use it in practice. Therefore, we were forced to look for a new teaching language for the introductory programming course.

## JAVA

Java was chosen to replace Oberon because it as a modern programming language that is object-oriented, platform independent and has simpler and better syntax compared to C or C++. Given the fact that it is suitable for both general-purpose business programs, as well as for interactive World Wide Web-based Internet applications, Java became widespread in industry, as well as in academic environments.

The content of the course was redesigned in order to exploit fully the strengths of Java with regard to object-oriented programming and the development of Internet applications. Nevertheless, given the fact that Java is a hybrid programming language, the content of the course still combines both programming paradigms: procedural and object-oriented.

After the introduction of basic data types and control structures, the concepts of class, method and object are introduced. Special attention is given to the use of constructors, understanding blocks and scope, organising classes and data hiding. Students are taught how to use pre-written classes and import pre-written constants and methods. Arrays and strings are then described, together with the methods of the *String* and *StringBuffer* classes.

The central part of the course is devoted to inheritance. Basic inheritance concepts, such as extending classes, overriding superclass methods, working with superclasses that have constructors and accessing superclass methods, are explained first. This is followed by more advanced inheritance concepts, ie creating and using abstract classes, using dynamic method binding, creating arrays of sub-class objects, as well as creating and using interfaces and packages.

At the end of the course, some – from the students' viewpoint – more attractive topics are covered, including an introduction to graphics and applets. The content of the course is in a great deal based on the text by Farrell, *Java Programming* [11]. However, the sequence of chapters is somewhere changed, and the last three chapters are omitted because of the time constraints of the course (the introductory programming course in the Faculty of Computer and Information Science lasts 15 weeks, and comprises 45 hours of lectures and 45 hours of laboratory practice.

The students were surveyed again with the aim of verifying whether the decision to use Java and modify the course content was right. The survey revealed that students almost unanimously support the change of the teaching language. The great majority of them (almost 95%) agreed that Java was the right choice, and only 5% advocated the use of another programming language. However, the survey also revealed that students found the modified course content to be more difficult. Table 1 shows the comparison between the answers regarding the difficulty of the introductory programming course in the academic year 1995/1996 (when Oberon was used for the first time) and the academic year 2002/2003 (when Oberon was replaced by Java). The portion of students who found the course too difficult increased from a still acceptable 18.18% in 1995/1996 to almost 35% in 2002/2003.

Table 1: Comparison of students' answers regarding the difficulty of the introductory programming course.

| Statement | Oberon | Java |
|---|---|---|
| The course is too easy | 3.90% | 4.46% |
| The level of difficulty is just right | 77.92% | 60.71% |
| The course is too difficult | 18.18% | 34.82% |

It is thought that increased difficulty is a consequence of the fact that Java's syntax is more complex and some programming concepts are more difficult to understand when compared with Oberon. For example, in Java, a clear distinction is missing between the concepts of class and module. Instead of having two different concepts, the class is used for two different purposes: to specify the structure of objects, as well as to specify programming logic. This requires the static keyword to be used in order to distinguish between instance methods and variables on the one hand, and class methods and variables on the other. Additionally, a significant shift of programming concern is noticed when using Java: programmers must not be focused purely on programming logic, but must also know how to use comprehensive libraries of predefined classes and methods. Finally, rather complicated concepts of event driven programming must be introduced in order to understand particular topics, including applets and computer graphics.

Considering the results of the survey, a reduction in the amount of material covered during the introductory course is planned. Also, some topics are to be moved to subsequent courses that are taught in the second and third semesters.

## JAVASCRIPT AND OTHER LANGUAGES

While Java, Pascal, C and Oberon were chosen as appropriate choices for technically-oriented profiles, it is evident that most students will not program complex applications, and probably will not program at all. For example, an economics student will most likely deal with finances, business, management and organisational duties, and not actually code a program. One choice is to introduce programming concepts like iterations and recursions through program packages like *Derive* or *Mathematica*. Indeed, this was done with a reasonable level of success and students were soon able to solve simple tasks for pedagogical or experimental purposes with the additional possibility to graphically present data and results. These skills are still regarded as being quite useful for professional work and are worth teaching at various faculties.

Some programming languages like Snobol or Smalltalk, and even Logo, were (or are) more or less becoming extinct. Prolog and Lisp were competing for artificial-intelligence tasks. Lisp dominated in the USA, while Prolog was stronger in Europe. However, languages were successful for provoking interesting academic-level thinking, but were not appropriate for applications. While these languages are still used for artificial intelligence and some specialised directions, their emphasis on high-level thinking and programming is not appropriate for introductory teaching programming at the faculty level.

JavaScript was chosen in Slovenia as the most appropriate for the introductory teaching of non-technically-oriented students, primarily because it is similar to Java-type of languages as the most successful modern language, and because it is simpler and easier to use than Java. JavaScript is not a full programming language and is certainly not appropriate to code large non-Internet applications. However, the major strength comes from its relation to HTML and the Internet.

JavaScript programs are part of HTML, and adding JavaScript's scripts adds interactivity and functionality to user's Web pages. Through JavaScript, it is possible to introduce basic knowledge and mental concepts of programming, even though full programming is never taught. This is the main attraction for non-technically-oriented students.

In addition, there are thousands of JavaScript programs on the Internet, and there is rarely a need to code a program on one's own. Rather, interesting programs are modified and adapted to specific needs and wishes. JavaScript is often referred to as a scripting language, with the implications that it is easier to script than to program.

The most important functional ability of JavaScript is that it enables the use if intelligent services, which are the backbone of an information society. Indeed, for real-life applications, programmers might need advanced programs in Java or other full-featured programming languages, but for teaching and academic use, JavaScript was chosen as the most appropriate in terms of attractiveness, simplicity and pedagogical terms.

As expected, the authors' experiences with JavaScript are positive indeed. It is easy to introduce simple programming techniques to even non-technically-oriented students. Students, on the other hand, are quite satisfied when they see that writing simple programs is a truly a simple task in JavaScript. Students are also attracted to the Internet and are quite satisfied that what they do can be directly applied to their Web pages.

## CONCLUSIONS

The top days of massive number of students programming applications are probably gone. Yet most of the real applications today get coded in a way that is strikingly similar to that undertaken a decade or two ago. The times of automatic programming now seem further away than ten years ago. On the other hand, programming skills are needed for technically and non-technically-oriented students, if not for reasons other than for use on the interactive Web.

Teaching introductory programming seems to be Java-oriented, whether that be Java for technically-oriented students or JavaScript for less technically educated ones. There are several other languages, like Prolog, but their influence upon introductory programming is not great. Only later, and for specialised directions like artificial intelligence courses, do other programming languages become relevant.

On the other hand, all the principles of good programming and program methodologies remain nearly intact. The orientation is currently focused on object and agent programming, while at the same time, most of the simple programs are very similar to traditional procedures and routines.

The knowledge of programming substantially varies and is, on average, lower than it was a decade or two ago. Furthermore, programming is no longer related to just writing programs: scripting, environments, tools and support services are related tasks and form a substantial part of the whole event.

## REFERENCES

1. The Joint Task Force on Computing Curricula 2001, IEEE Computer Society and Association for Computing Machinery, Computing Curricula 2001 (draft). 6 March (2000).
2. Gams, M., Bratko, I., Batagelj, V., Reinhardt, R., Martinec, M., Spegel, M. and Tancig, P., Programming language Pascal I. *Informatica*, 3, 43-48 (1984).
3. Wirth, N., *Recollections about the Development of Pascal*. In: Bergin, T.J. and Gibson, R.G., History of Programming Languages II. London: Addison-Wesley (1996).
4. Bratko, I., *Prolog Programming for Artificial Intelligence*, London: Addison Wesley (2001).
5. Burgess, C.J. and Jones, B.F., *Should Software Quality be a Major Issue when Teaching First Year Programming to Software Engineers?* In: King, G., Brebbia, C.A., Ross, M. and Staples, G. (Eds), Software Engineering in Higher Education. Southampton: Computational Mechanics Publications, 75-81 (1994).
6. Mahnic, V. and Vilfan, B., *A First Course in Object-Oriented Programming Using Oberon*. In: Uso, J-L., Mitic, P. and Sucharov, L.J. (Eds), Software Engineering in Higher Education. Southampton: Computational Mechanics Publications, 329-336 (1995).
7. Woodhouse, D., Introductory courses in computing: aims and languages. *Computer Educ.*, 7, **2**, 79-89 (1983).
8. Reiser, M. and Wirth, N., *Programming in Oberon, Steps beyond Pascal and Modula*. Reading: Addison-Wesley (1992).
9. Mahnic, V. *Some Experience in Teaching an Introductory Programming Course Using Oberon*. In: Tasso, C., Adey, R.A. and Pighin, M. (Eds), Software Quality Engineering. Southampton: Computational Mechanics Publications, 27-36 (1997).
10. Mahnic, V. How to teach undergraduates the introductory programming course? *Proc. 8th Electrotechnical and Computer Science Conf. ERK '99*, Portoroz, Slovenia, 411-414 (1999) (in Slovene).
11. Farrell, J., *Java Programming* (2nd edn). Boston: Thomson Course Technology (2003).